IONA®

Artix™

Artix Configuration Reference

Version 4.0, March 2006

Making Software Work Together™

While the information in this publication is believed to be accurate, 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 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.

Updated: 26-May-2008

Contents

Preface

What is Covered in this Book

The *Artix Configuration Reference* provides a comprehensive reference for the configuration variables in an Artix configuration domain.

Who Should Read this Book

This book is intended for use by system administrators, in conjunction with Configuring and Deploying Artix Solutions. It assumes that the reader is familiar with Artix administration. Anyone involved in designing a large scale Artix solution will also find this book useful.

Knowledge of middleware or messaging transports is not required to understand the general topics discussed in this book. However, if you are using this book as a guide to deploying runtime systems, you should have a working knowledge of the middleware transports that you intend to use in your Artix solutions.

Note: When deploying Artix in a distributed architecture with other middleware, please see the documentation for that middleware product. You may require access to an administrator. For example, a Tuxedo administrator is required to complete a Tuxedo distributed architecture.

How to Use this Book

This book is organized as follows:

- Chapter 1 describes the Artix runtime configuration variables.
- Chapter 2 describes the Artix plug-in namespaces and variables.
- Chapter 3 describes the configuration namespaces and variables used to configure Artix security features.
- Chapter 4 describes the CORBA plug-in configuration namespaces and variables.

The Artix Library

The Artix documentation library is organized in the following sections:

- Getting Started
- Designing and Developing Artix Solutions
- Configuring and Deploying Artix Solutions
- Using Artix Services
- Integrating Artix Solutions
- Integrating with Enterprise Management Systems
- Reference Documentation

Getting Started

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

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

Designing and Developing Artix Solutions

The books in this section go into greater depth about using Artix to solve real-world problems. They describe how Artix uses WSDL to define services, and how to use the Artix APIs to build new services. They include:

- Building Service-Oriented Architectures with Artix provides an overview of service-oriented architectures and describes how they can be implemented using Artix.
- Understanding Artix Contracts describes the components of an Artix contract. Special attention is paid to the WSDL extensions used to define Artix-specific payload formats and transports.
- Developing Artix Applications in C++ discusses the technical aspects of programming applications using the C++ API.
- Developing Advanced Artix Plug-ins in C++ discusses the technical aspects of implementing advanced plug-ins (for example, interceptors) using the C++ API.
- Developing Artix Applications in Java discusses the technical aspects of programming applications using the Java API.

Configuring and Deploying Artix Solutions

This section includes:

- Configuring and Deploying Artix Solutions discusses how to configure and deploy Artix-enabled systems, and provides examples of typical use cases.

Using Artix Services

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

- Artix Locator Guide discusses how to use the Artix locator.
- Artix Session Manager Guide discusses how to use the Artix session manager.
- Artix Transactions Guide, C++ explains how to enable Artix C++ applications to participate in transacted operations.
- Artix Transactions Guide, Java explains how to enable Artix Java applications to participate in transacted operations.
- Artix Security Guide explains how to use the security features of Artix.

Integrating Artix Solutions

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

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

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

Integrating with Enterprise Management Systems

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

- IBM Tivoli Integration Guide explains how to integrate Artix with IBM Tivoli.
- BMC Patrol Integration Guide explains how to integrate Artix with BMC Patrol.
- CA-WSDM Integration Guide explains how to integrate Artix with CA-WSDM.

Reference Documentation

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

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

Getting the Latest Version

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

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

Searching the Artix Library

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

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

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

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

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

Artix Online Help

Artix Designer and the Artix Management Console include comprehensive online help, providing:

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

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

- Select **Help**|**Help Contents** from the menu bar. Sections on Artix Designer and the Artix Management Console appear in the contents panel of the Eclipse help browser.
- Press **F1** for context-sensitive help.

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

Artix Glossary

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

Additional Resources

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

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

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

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

Document Conventions

Typographical conventions

This book uses the following typographical conventions:

`Fixed width` Fixed width (courier font) in normal text represents portions of code and literal names of items such as classes, functions, variables, and data structures. For example, text might refer to the `IT_Bus::AnyType` class.

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

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

`Fixed width italic` Fixed width italic words or characters in code and commands represent variable values you must supply, such as arguments to commands or path names for your particular system. For example:

```
% cd /users/YourUserName
```

Italic Italic words in normal text represent *emphasis* and introduce *new terms*.

Bold Bold words in normal text represent graphical user interface components such as menu commands and dialog boxes. For example: the **User Preferences** dialog.

Keying Conventions

This book uses the following keying conventions:

No prompt When a command's format is the same for multiple platforms, the command prompt is not shown.

`%` A percent sign represents the UNIX command shell prompt for a command that does not require root privileges.

`#` A number sign represents the UNIX command shell prompt for a command that requires root privileges.

`>` The notation > represents the MS-DOS or Windows command prompt.

`...`
`.`
`.`
`.` Horizontal or vertical ellipses in format and syntax descriptions indicate that material has been eliminated to simplify a discussion.

`[]` Brackets enclose optional items in format and syntax descriptions.

`{}` Braces enclose a list from which you must choose an item in format and syntax descriptions.

`|` In format and syntax descriptions, a vertical bar separates items in a list of choices enclosed in `{}` (braces).

In graphical user interface descriptions, a vertical bar separates menu commands (for example, select **File**|**Open**).

CHAPTER 1

Artix Runtime

Artix is based on IONA's highly configurable Adaptive Runtime (ART) infrastructure. This provides a high-speed, robust, and scalable backbone for deploying integration solutions. This chapter explains the configuration settings for the core Artix runtime.

In this chapter

This chapter includes the following:

ORB Plug-ins

Overview

The orb_plugins variable specifies the list of plug-ins that Artix processes load during initialization. A *plug-in* is a class or code library that can be loaded into an Artix application at runtime. These plug-ins enable you to load network transports, payload format mappers, error logging streams, and other features on the fly.

The default orb_plugins entry includes the following:

```
orb_plugins = ["xmlfile_log_stream",
               "iiop_profile",
               "giop",
               "iiop"];
```

All other plug-ins that implement bindings and transports load transparently when the WSDL file is loaded into an application. These plug-ins do not need to be explicitly listed in orb_plugins. Artix determines what plug-ins are required from the content of the WSDL file.

However, plug-ins for other services (for example, for security, locator, session manager, routing, XSLT transformation, logging, and so on) must all be included in the orb_plugins entry.

Artix plug-ins

Each network transport and payload format that Artix interoperates with uses its own plug-in. Many of the Artix services features also use plug-ins. Artix plug-ins include the following:

- "Java plug-ins".
- "Transport plug-ins".
- "Payload format plug-ins".
- "Service plug-ins".
- "Internal ORB plug-ins"

Java plug-ins

Plug-ins written in Java are configured differently from C++ plug-ins. For the most part, only custom plug-ins are written in Java. However, the JMS transport plug-in is also written in Java and requires that you configure it appropriately.

Java plug-in loader

When using a Java plug-in, you must include an entry for the Java plug-in loader in the orb_plugins list, as shown in Example 1.

Example 1: *Including the Java Plug-in Loader*

```
orb_plugins=[..., "java", ...];
```

The java plug-in automatically loads the JMS transport plug-in.

java_plugins variable

In addition to including the java plug-in loader in the orb_plugin list, you must specify the java_plugins configuration variable, which lists the names of the Java plug-ins that are to be loaded. java_plugins is a list like orb_plugins. A plug-in cannot be listed in both variables. Only Java plug-ins should be listed in java_plugins; and Java plug-ins should not be listed in orb_plugins.

For example, if you are using a custom Java plug-in called my_java_handler in your application you would use the configuration similar to the fragment shown in Example 2 to load the plug-ins.

Example 2: *Loading a Java Plug-in*

```
orb_plugins=["xml_log_stream", ... "java", ...];
java_plugins=["my_java_handler"];
```

In addition, you must also specify the plug-in factory class, for example:

```
plugins:my_java_handler:classname="myJavaHandlerFactory"
```

For more details, see “Custom Plug-ins” on page 114.

The following Java plug-in is supplied by Artix, and can be included in your `java_plugins` list:

`java_uddi_proxy` Dynamically locates existing Web services endpoints using the UDDI service.

Transport plug-ins

The Artix transport plug-ins are listed in Table 1.

Table 1: *Artix Transport Plug-ins*

Plug-in	Transport
`at_http`	Provides support for HTTP.
`https`	Provides support for HTTPS.
`iiop`	Provides support for CORBA IIOP.
`iiop_profile`	Provides support for CORBA IIOP profile.
`giop`	Provides support for CORBA GIOP.
`tunnel`	Provides support for the IIOP transport using non-CORBA payloads.
`tuxedo`	Provides support for Tuxedo interoperability.
`mq`	Provides support for IBM WebSphere MQ interoperability, and MQ transactions.
`tibrv`	Provides support for TIBCO Rendezvous interoperability.
`java`	Provides support for Java Message Service (JMS) interoperability (and also for Java UDDI and custom Java plug-ins).

Payload format plug-ins

The Artix payload format plug-ins are listed in Table 2.

Table 2: *Artix Payload Format Plug-ins*

Plug-in	Payload Format
`soap`	Decodes and encodes messages using the SOAP format. See also "SOAP" on page 95.
`G2`	Decodes and encodes messages packaged using the G2++ format.
`fml`	Decodes and encodes messages packaged in FML format.
`tagged`	Decodes and encodes messages packed in variable record length messages or another self-describing message format.
`tibrv`	Decodes and encodes TIBCO Rendezvous messages.
`fixed`	Decodes and encodes fixed record length messages.
`ws_orb`	Decodes and encodes CORBA messages.

Service plug-ins

Artix service feature plug-ins are listed in Table 3.

Table 3: *Artix Service Plug-ins*

Plug-in	Artix Feature
`bus_loader`	In a pure CORBA application, add a `bus_loader` at the end of your plug-in list to start the bus and initialize all `BusPlugin`s. Not needed if your application uses `IT_Bus::init`.

Table 3: *Artix Service Plug-ins (Continued)*

Plug-in	Artix Feature
`bus_response_monitor`	Enables performance logging. Monitors response times of Artix client/server requests. See also "Response Time Collector" on page 83.
`locator_client`	Queries the locator and returns a reference to a target service. See also the Artix Locator Guide.
`locator_endpoint`	Enables endpoints to use the Artix locator service. See also "Locator Endpoint Manager" on page 80.
`ots`	Enables the CORBA OTS transaction system. See also "Bus" on page 49.
`ots_lite`	Enables the OTS Lite transaction system, which supports one-phase commit transactions. See also "Bus" on page 49.
`request_forwarder`	Enables forwarding of write requests from slave replicas to master replicas. See also "Database Environment" on page 56.
`routing`	Enables Artix routing. See "Routing" on page 86.
`service_locator`	Enables the Artix locator. An Artix server acting as the locator service must load this plug-in. See also "Locator Service" on page 77.
`session_manager_service`	Enables the Artix session manager. An Artix server acting as the session manager must load this plug-in. See also "Session Manager" on page 92.

Table 3: *Artix Service Plug-ins (Continued)*

Plug-in	Artix Feature
`session_endpoint_manager`	Enables the Artix session manager. Endpoints wishing to be managed by the session manager must load this plug-in. See also "Session Endpoint Manager" on page 93.
`sm_simple_policy`	Enables the policy mechanism for the Artix session manager. Endpoints wishing to be managed by the session manager must load this plug-in. See also "Session Manager Simple Policy" on page 94.
`service_lifecycle`	Enables service lifecycle for the Artix router. This optimizes performance of the router by cleaning up proxies/routes that are no longer in use. See also "Service Lifecycle" on page 90.
`uddi_proxy`	Dynamically locates existing Web services endpoints using the UDDI service. See also "java_plugins variable" on page 3.
`wsat_protocol`	Enables the WS-Atomic Transaction (WS-AT) system. See also "Bus" on page 49.
`ws_chain`	Enables you to link together a series of services into a multi-part process. See also "Web Services Chain Service" on page 102.
`ws_coordination_service`	Enables the WS-Coordination service, which coordinates two-phase commit transactions. See also "Bus" on page 49.
`wsdl_publish`	Enables Artix endpoints to publish and download Artix WSDL files. See also "WSDL Publishing Service" on page 109.

Table 3: *Artix Service Plug-ins (Continued)*

Plug-in	Artix Feature
`wscolloc`	Enables colocation for applications that share a common binding. For example, using the Artix transformer with an Artix server, you can colocate both processes. Instead of passing messages through the messaging stack, messages are passed directly between the two, improving performance.
`xmlfile_log_stream`	Enables you to view Artix logging output in a file. See also "XML File Log Stream" on page 111.
`xslt`	Enables Artix to process XSLT scripts. See also "Transformer Service" on page 96.

Internal ORB plug-ins

This applies to CORBA integrations only. It is possible to specify whether the default ORB shares settings with an internal ORB. In certain circumstances such as initialization, Orbix creates an internal ORB instance. The `share_variables_with_internal_orb` setting is used to prevent an internal CORBA ORB from loading Artix plug-ins.

For example, if you set an indirect persistence mode policy on an Artix CORBA server, and also use the Artix `locator_endpoint` plug-in. Essentially, in this case, the Artix CORBA endpoint is talking to both Artix and Orbix locators.

Setting share_variables_with_internal_orb to false prevents the internal ORB (IT_POAInternalORB) from sharing the default ORB plug-ins. The default setting is as follows:

```
share_variables_with_internal_orb = "false";

IT_POAInternalORB
{
    orb_plugins = ["iiop_profile", "giop", "iiop"];
}
```

The list of plug-ins available for the internal ORB is specified using the IT_POAInternalORB configuration scope.

Event Log

Overview

The event_log namespace control logging levels in Artix. It contains the following variables:

- event_log:filters
- event_log:filters:bus:pre_filter

event_log:filters

The event_log:filters variable can be set to provide a wide range of logging levels. The default event_log:filters setting displays errors only:

```
event_log:filters = ["*=FATAL+ERROR"];
```

The following setting displays errors and warnings only:

```
event_log:filters = ["*=FATAL+ERROR+WARNING"];
```

Adding INFO_MED causes all of request/reply messages to be logged (for all transport buffers):

```
event_log:filters = ["*=FATAL+ERROR+WARNING+INFO_MED"];
```

The following setting displays typical trace statement output (without the raw transport buffers being printed):

```
event_log:filters = ["*=FATAL+ERROR+WARNING+INFO_HI"];
```

The following setting displays all logging:

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

The default configuration settings enable logging of only serious errors and warnings. For more exhaustive output, select a different filter list at the default scope, or include a more expansive event_log:filters setting in your configuration scope.

event_log:filters:bus:pre_filter

event_log:filters:bus:pre_filter provides filtering of log messages that are sent to the EventLog before they are output to the LogStream. This enables you to minimize the time spent generating log messages that will be ignored. For example:

```
event_log:filters:bus:pre_filter = "WARN+ERROR+FATAL";

event_log:filters = ["IT_BUS=FATAL+ERROR", "IT_BUS.BINDING=*"];
```

In this example, only WARNING, ERROR and FATAL priority log messages are sent to the EventLog. This means that no processing time is wasted generating strings for INFO log messages. The EventLog then only sends FATAL and ERROR log messages to the LogStream for the IT_BUS subsystem.

Note: event_log:filters:bus:pre_filter defaults to * (all messages). Setting this variable to WARN+ERROR+FATAL improves performance significantly.

Message Snoop

Overview

Artix message snoop is a message interceptor that sends input/output messages to the Artix log to enable viewing of the message content. This is a useful debugging tool when developing and testing an Artix system. The `artix:interceptors:message_snoop` namespace includes the following configuration variables:

- `artix:interceptors:message_snoop:enabled`
- `artix:interceptors:message_snoop:log_level`
- `artix:interceptors:message_snoop:log_subsystem`

artix:interceptors:message_snoop:enabled

`artix:interceptors:message_snoop:enabled` specifies whether message snoop is enabled. Message snoop is enabled by default. It is automatically added as the last interceptor before the binding to detect any changes that other interceptors might make to the message. By default, `message_snoop` logs at `INFO_MED` in the `MESSAGE_SNOOP` subsystem.

Message snoop is invoked on every message call, twice in the client and twice in the server (assuming Artix is on both sides). This means that it can impact on performance. More importantly, message snoop involves risks to confidentiality. You can disable message snoop using the following setting:

```
artix:interceptors:message_snoop:enabled = "false";
```

WARNING: For security reasons, it is strongly recommended that message snoop is disabled in production deployments.

artix:interceptors:message_snoop:log_level

artix:interceptors:message_snoop:log_level specifies a message snoop log level globally or for a service port. The following example sets the level globally:

```
artix:interceptors:message_snoop:log_level = "WARNING";
event_log:filters = ["*=WARNING", "IT_BUS=INFO_HI+WARN+ERROR",
   "MESSAGE_SNOOP=WARNING"];
```

The following example sets the level for a service port:

```
artix:interceptors:message_snoop:http://www.acme.com/tests:mySer
   vice:myPort:log_level = "INFO_MED";
event_log:filters = ["*=INFO_MED", "IT_BUS=",
   "MESSAGE_SNOOP=INFO_MED"];
```

artix:interceptors:message_snoop:log_subsystem

artix:interceptors:message_snoop:log_subsystem specifies a specific subsystem globally or for a service port. The following example sets the subsystem globally:

```
artix:interceptors:message_snoop:log_subsystem = "MY_SUBSYSTEM";
event_log:filters = ["*=INFO_MED", "IT_BUS=",
   "MY_SUBSYSTEM=INFO_MED"];
```

The following example sets the subsystem for a service port:

```
artix:interceptors:message_snoop:http://www.acme.com/tests:mySer
   vice:myPort:log_subsystem = "MESSAGE_SNOOP";
event_log:filters = ["*=INFO_MED", "IT_BUS=",
   "MESSAGE_SNOOP=INFO_MED"];
```

If message snoop is disabled globally, but configured for a service/port, it is enabled for that service/port with the specified configuration only. For example:

```
artix:interceptors:message_snoop:enabled = "false";

artix:interceptors:message_snoop:http://www.acme.com/tests:mySer
    vice:myPort:log_level = "WARNING";
artix:interceptors:message_snoop:http://www.acme.com/tests:mySer
    vice:myPort:log_subsystem = "MY_SUBSYSTEM";

event_log:filters = ["*=WARNING", "IT_BUS=INFO_HI+WARN+ERROR",
    "MY_SUBSYSTEM=WARNING"];
```

Setting message snoop in conjunction with log filters is useful when you wish to trace only messages that are relevant to a particular service, and you do not wish to see logging for others (for example, the container, locator, and so on).

Policies

Overview

The policies namespace contains variables that control the publishing of server and client hostnames. These include the following:

- policies:at_http:client:proxy_server
- policies:at_http:server_address_mode_policy:publish_hostname
- policies:at_http:server_address_mode_policy:local_hostname
- policies:http:client_address_mode_policy:local_hostname
- policies:http:server_address_mode_policy:port_range
- policies:iiop:server_address_mode_policy:local_hostname
- policies:iiop:server_address_mode_policy:port_range
- policies:iiop:server_address_mode_policy:publish_hostname
- policies:soap:server_address_mode_policy:local_hostname
- policies:soap:server_address_mode_policy:publish_hostname

policies:at_http:client:proxy_server

policies:at_http:client:proxy_server specifies the URL of the HTTP proxy server (if one exists) along a request/response chain.

Note: Artix does not support the existence of more than one proxy server along a request/response chain.

For example:

```
policies:at_http:client:proxy_server =
    "http://localhost:0/SOAPHTTPProxy";
```

You can specify the HTTP proxy server in different ways. The order of priority is as follows:

1. Context API.
2. WSDL file.

3. Command line configuration, for example:

```
client -ORBpolicies:at_http:client:proxy_server="http://localhost:0/SOAPHTTPProxy"
```

4. This configuration variable.

policies:at_http:server_address_mode_policy:publish_hostname

policies:at_http:server_address_mode_policy:publish_hostname specifies how the server's address is published in dynamically generated Artix service contracts when using the HTTP transport. The possible values are as follows:

canonical	Publishes the fully qualified hostname of the machine in the http:address element of the dynamic WSDL (for example, http://myhost.mydomain.com).
unqualified	Publishes the unqualified local hostname of the machine in the http:address element of the dynamic WSDL. This does not include the domain name with the hostname (for example, http://myhost).
ipaddress	Publishes the IP address associated with the machine in the http:address element of the dynamic WSDL (for example, http://10.1.2.3).

For example:

```
policies:at_http:server_address_mode_policy:publish_hostname="canonical";
```

The following values are deprecated:

false	Publishes the IP address of the running server in the http:address element. This is the default behavior.
true	Publishes the hostname of the machine hosting the running server in the http:address element of the WSDL contract.

policies:at_http:server_address_mode_policy:local_hostname

policies:at_http:server_address_mode_policy:local_hostname specifies the server hostname that is published in dynamically generated Artix contracts. For example:

```
policies:at_http:server_address_mode_policy:local_hostname="207.45.52.34";
```

This variable accepts any valid string value. The specified hostname is published in the http:address element, which describes the server's location. If no hostname is specified, policies:at_http:server_address_mode_policy:publish_hostname is used instead.

policies:http:client_address_mode_policy:local_hostname

policies:http:client_address_mode_policy:local_hostname specifies the outgoing client hostname. This enables you to explicitly specify the hostname that the client binds on, when initiating a TCP connection.

This provides support for *multi-homed* client host machines with multiple hostnames or IP addresses (for example, those using multiple DNS aliases or multiple network interface cards).

For example, if you have a client machine with two network addresses (207.45.52.34 and 207.45.52.35), you can explicitly set this variable to either address:

```
policies:http:client_address_mode_policy:local_hostname =
    "207.45.52.34";
```

This variable accepts any valid string value. By default, the local_hostname variable is unspecified, and the client uses the 0.0.0.0 wildcard address. In this case, the network interface card used is determined by the operating system.

policies:http:server_address_mode_policy:port_range

`policies:http:server_address_mode_policy:port_range` specifies a range of HTTP ports in the following format: *`FromPort`*`:`*`ToPort`*

For example:

```
policies:http:server_address_mode_policy:port_range="4003:4008";
```

> **Note:** The specified `port_range` has no effect when a fixed TCP port is specified for the SOAP address in the WSDL contract. The WSDL setting takes precedence over the configuration file setting.

policies:iiop:server_address_mode_policy:local_hostname

`policies:iiop:server_address_mode_policy:local_hostname` enables you to explicitly specify the host name that the server listens on and publishes in its IORs.

For example, if you have a machine with two network addresses (`207.45.52.34` and `207.45.52.35`), you can explicitly set this variable to either address:

```
policies:iiop:server_address_mode_policy:local_hostname =
    "207.45.52.34";
```

By default, the `local_hostname` variable is unspecified.

policies:iiop:server_address_mode_policy:port_range

`policies:iiop:server_address_mode_policy:port_range` specifies the range of ports that a server uses when there is no well-known addressing policy specified for the port. Specified values take the format of *`FromPort`*`:`*`ToPort`*, for example:

```
policies:iiop:server_address_mode_policy:port_range="4003:4008"
```

policies:iiop:server_address_mode_policy:publish_hostname

policies:iiop:server_address_mode-policy:publish_hostname specifes whether IIOP exports hostnames or IP addresses in published profiles. Defaults to false (exports IP addresses, and does not export hostnames). To use hostnames in object references, set this variable to true, as in the following file-based configuration entry:

```
policies:iiop:server_address_mode_policy:publish_hostname=true
```

policies:soap:server_address_mode_policy:local_hostname

policies:soap:server_address_mode_policy:local_hostname specifies the server hostname that is published in dynamically generated Artix contracts. For example:

```
policies:soap:server_address_mode_policy:local_hostname="207.45.52.34";
```

This variable accepts any valid string value. The specified hostname is published in the soap:address element, which describes the server's location. If no hostname is specified, policies:soap:server_address_mode_policy:publish_hostname is used instead.

policies:soap:server_address_mode_policy:publish_hostname

policies:soap:server_address_mode_policy:publish_hostname specifies how the server's address is published in dynamically generated Artix contracts when using SOAP as a transport. The possible values are as follows:

canonical	Publishes the fully qualified hostname of the machine in the soap:address element of the dynamic WSDL (for example, http://myhost.mydomain.com).
unqualified	Publishes the unqualified local hostname of the machine in the soap:address element of the dynamic WSDL. This does not include the domain name with the hostname (for example, http://myhost).

`ipaddress` Publishes the IP address associated with the machine in the `soap:address` element of the dynamic WSDL (for example, `http://10.1.2.3`).

For example:

```
policies:soap:server_address_mode_policy:publish_hostname="ipaddress";
```

The following values are deprecated:

`false` Publishes the IP address of the running server in the `soap:address` element. This is the default behavior.

`true` Publishes the hostname of the machine hosting the running server in the `soap:address` element of the WSDL contract.

Binding Lists

Overview

When using Artix's CORBA functionality you need to configure how Artix binds itself to message interceptors. The Artix binding namespace contains variables that specify interceptor settings. An interceptor acts on a message as it flows from sender to receiver.

Computing concepts that fit the interceptor abstraction include transports, marshaling streams, transaction identifiers, encryption, session managers, message loggers, containers, and data transformers. Interceptors are based on the "chain of responsibility" design pattern. Artix creates and manages chains of interceptors between senders and receivers, and the interceptor metaphor is a means of creating a virtual connection between a sender and a receiver.

The binding namespace includes the following variables:

- client_binding_list
- server_binding_list

client_binding_list

Artix provides client request-level interceptors for OTS, GIOP, and POA collocation (where server and client are collocated in the same process). Artix also provides and message-level interceptors used in client-side bindings for IIOP, SHMIOP and GIOP.

The binding:client_binding_list specifies a list of potential client-side bindings. Each item is a string that describes one potential interceptor binding. The default value is:

```
binding:client_binding_list = ["OTS+POA_Coloc","POA_Coloc","OTS+GIOP+IIOP","GIOP+IIOP"];
```

Interceptor names are separated by a plus (+) character. Interceptors to the right are "closer to the wire" than those on the left. The syntax is as follows:

- Request-level interceptors, such as GIOP, must precede message-level interceptors, such as IIOP.
- GIOP or POA_coloc must be included as the last request-level interceptor.

- Message-level interceptors must follow the GIOP interceptor, which requires at least one message-level interceptor.
- The last message-level interceptor must be a message-level transport interceptor, such as IIOP or SHMIOP.

When a client-side binding is needed, the potential binding strings in the list are tried in order, until one successfully establishes a binding. Any binding string specifying an interceptor that is not loaded, or not initialized through the orb_plugins variable, is rejected.

For example, if the ots plug-in is not configured, bindings that contain the OTS request-level interceptor are rejected, leaving ["POA_Coloc", "GIOP+IIOP", "GIOP+SHMIOP"]. This specifies that POA collocations should be tried first; if that fails, (the server and client are not collocated), the GIOP request-level interceptor and the IIOP message-level interceptor should be used. If the ots plug-in is configured, bindings that contain the OTS request interceptor are preferred to those without it.

server_binding_list

binding:server_binding_list specifies interceptors included in request-level binding on the server side. The POA request-level interceptor is implicitly included in the binding.

The syntax is similar to client_binding_list. However, in contrast to the client_binding_list, the left-most interceptors in the server_binding_list are “closer to the wire”, and no message-level interceptors can be included (for example, IIOP). For example:

```
binding:server_binding_list = ["OTS",""];
```

An empty string ("") is a valid server-side binding string. This specifies that no request-level interceptors are needed. A binding string is rejected if any named interceptor is not loaded and initialized.

The default server_binding_list is ["OTS", ""]. If the ots plug-in is not configured, the first potential binding is rejected, and the second potential binding ("") is used, with no explicit interceptors added.

Binding Lists for Custom Interceptors

Overview

The `binding:artix` namespace includes variables that configure Artix applications to use custom interceptors.

Artix interceptors are listed in the order that they are invoked on a message when it passes through a messaging chain. For example, if a server request interceptor list is specified as `"interceptor_1+interceptor_2"`, the message is passed into `interceptor_1` as it leaves the binding. When `interceptor_1` processes the message, it is passed into `interceptor_2` for more processing. `interceptor_2` then passes the message along to the application code.

The interceptor chain is specified as a single string, and each interceptor name must be separated by a + character (for example, `"interceptor_1+interceptor_2+interceptor_3"`).

The variables in the `binding:artix` namespace are as follows:

- `client_message_interceptor_list`
- `client_request_interceptor_list`
- `server_message_interceptor_list`
- `server_request_interceptor_list`

These settings apply to all services activated in a single Artix bus. See also "Port level interceptor chains" on page 25.

client_message_interceptor_list

`binding:artix:client_message_interceptor_list` is a string that specifies an ordered list of message-level interceptors for a Java or C++ client application. Each interceptor is separated using a + character, for example:

```
binding:artix:client_message_interceptor_list =
 "interceptor_1+interceptor_2";
```

There is no default value.

client_request_interceptor_list

binding:artix:client_request_interceptor_list is a string that specifies an ordered list of request-level interceptors for a Java or C++ client application. Each interceptor is separated using a + character, for example:

```
binding:artix:client_request_interceptor_list =
 "interceptor_1+interceptor_2";
```

There is no default value.

server_message_interceptor_list

binding:artix:server_message_interceptor_list is a string that specifies an ordered list of message-level interceptors for a Java or C++ server application. Each interceptor is separated using a + character, for example:

```
binding:artix:server_message_interceptor_list =
 "interceptor_1+interceptor_2";
```

There is no default value.

server_request_interceptor_list

binding:artix:server_request_interceptor_list is a string that specifies an ordered list of request-level interceptors for a Java or C++ server application. Each interceptor is separated using a + character, for example:

```
binding:artix:server_request_interceptor_list =
 "interceptor_1+interceptor_2";
```

There is no default value.

Port level interceptor chains

Each of the variables in the `binding:artix` namespace can also be specified at the level of a service port. This more fine-grained approach enables you to configure different interceptor chains for different endpoints in the same application. For example:

```
binding:artix:client_request_interceptor_list:ServiceQname:PortName=
    "interceptor_1+interceptor_2";

binding:artix:server_request_interceptor_list:ServiceQname:PortName=
    "interceptor_1+interceptor_2";

binding:artix:client_message_interceptor_list:ServiceQname:PortName=
    "interceptor_1+interceptor_2";

binding:artix:server_message_interceptor_list:ServiceQname:PortName=
    "interceptor_1+interceptor_2"";
```

The syntax of a *`ServiceQname`* is *`NamespaceURI`*:*`LocalPart`*. The following example shows a service defined as `FooService` with a target namespace of `http://www.myco.com/myservice`:

```
binding:artix:client_request_interceptor_list:http://www.myco.com/myservice:FooService:FooPort=
    "interceptor_1+interceptor_2";
```

Interceptor Factory Plug-in

Overview

An Artix plug-in that implements an interceptor is dynamically loaded when the interceptor name is specified in the binding list (see “Binding Lists for Custom Interceptors” on page 23).

You must either include the interceptor plug-in name in your `orb_plugins` list, or specify an interceptor factory plug-in.

> **Note:** For Java applications, you also have the option of specifying a handler classname. See “Java Handler Class” on page 28.

interceptor_factory:*InterceptorFactoryName*:plugin

`interceptor_factory:`*`InterceptorFactoryName`*`:plugin` specifies the name of the plug-in used by a custom interceptor. The format of this variable is as follows:

```
interceptor_factory:InterceptorFactoryName:plugin="PluginName";
```

For example,

```
interceptor_factory:TestInterceptor:plugin= "test_interceptor";
```

You do not need to add such configuration for the interceptors that are implemented internally by the various Artix plug-ins (for example, `security`, `service_lifecycle`, and `artix_response_time_interceptor`). These are all hard coded already.

C++applications

The following names are used in this syntax:

- The name of the interceptor factory: *`InterceptorFactoryName`*
- If the interceptor is implemented as a plug-in, the name of the plug-in: (*`PluginName`*)
- The name of the shared library that hosts the plug-in: *`SharedLibName`*

You must always specify the mapping between the plug-in name and the shared library name, using the following configuration syntax:

```
plugins:PluginName:shlib_name = "SharedLibName";
```

There are two ways in which a plug-in can be loaded:

- Specify the plug-in name in the ORB plug-ins list, for example:

```
orb_plugins = [ ..., "PluginName", ... ];
```

  Using this approach, the plug-in is loaded during ORB initialization.

- Configure a mapping between an interceptor factory name and the plug-in name as follows:

```
interceptor_factory:InterceptorFactoryName:plugin="PluginName";
```

  Using this approach, the plug-in is loaded when the interceptor list is parsed.

Java applications

For Java applications, the interceptor factory is called a `HandlerFactory`. This can be registered with the Artix bus any of the following ways:

- Write a Java plug-in and register a handler factory inside the plug-in. For details, see Developing Artix Applications in Java.
- Register directly with the Artix bus in your server or client mainline code. If you use this approach, you do not need any additional plug-in configuration, just specify the interceptor factory names in the list.

  The `HandlerFactory` should be registered before registering the servant on the server side, and before creating the client proxy base on the client-side. The public API is:

```
    bus.registerHandlerFactory(new MyHandlerFactory());
```

  For more details, see Developing Artix Applications in Java.

- Alternatively, you can use configuration to dynamically register a Java handler without writing a plug-in, or creating a `HandlerFactory`. For details, see “Java Handler Class” on page 28.

Java Handler Class

Overview

Specifying a Java handler class in configuration enables dynamic creation and registration of a `HandlerFactory` for your handler. On startup, the Java runtime searches the configured list of interceptors for names that are used to identify a classname for a Java handler. The runtime wraps the specified handlers in a `GenericHandlerFactory`, and registers these factories with the Artix bus

Configuring an endpoint to use a Java handler is a two step process. First, specify an implementation class and associate it with a name. Second, add the handler to one of the endpoint's interceptor chains.

handler:handler_name:classname

`handler:`*`HandlerName`*`:classname` specifies the Java implementation class for your handler. This information is used to dynamically create and register a `HandlerFactory` for your handler. This variable has the following syntax:

```
handler:HandlerName:classname="handlerClassname";
```

The value you supply for *`HandlerName`* is the name by which the handler will be referred to in interceptor chains. The value you supply for *`handlerClassname`* is the fully qualified class name of your handler's implementation. For example, if you wrote a handler in a class called `com.acme.myHandler` you would add the following variable to your endpoint's configuration:

```
handler:my_handler_app:classname="com.acme.myHandler";
```

When adding the handler to the endpoint's interceptor chain you would refer to the handler using `my_handler_app`.

Note: If you implemented your handler as an Artix plug-in, you must specify its implementation as described in "Java plug-in classes" on page 115.

handlers and interceptor chains

You must configure your application to load the handlers at the appropriate points in the message chain. This is done using the following configuration variables in the application's configuration scope:

- `binding:artix:client_message_interceptor_list`
- `binding:artix:client_request_interceptor_list`
- `binding:artix:server_message_interceptor_list`
- `binding:artix:server_request_interceptor_list`

The handlers are placed in the list in the order they will be invoked on the message as it passes through the messaging chain. The following example shows an application that uses both client and server handlers.

```
java_interceptors
{
  handler:first_handler:classname="com.acme.myFirstHandler";
  handler:second_handler:classname="com.acme.mySecondHandler";
  ...
  client
  {
    binding:artix:client_request_interceptor_list =
   "first_handler+second_handler";
    binding:artix:client_message_interceptor_list =
   "first_handler+second_handler";
  };
  server
  {
    binding:artix:server_request_interceptor_list=
   "second_handler+first_handler";
    binding:artix:server_message_interceptor_list =
   "second_handler+first_handler";
   };
};
```

For more details, see "Binding Lists for Custom Interceptors" on page 23.

Thread Pool Control

Overview

Variables in the `thread_pool` namespace set policies related to thread control. Thread pools can be configured at several levels, where the more specific configuration settings take precedence over the less specific. They can be set globally for Artix instances in a configuration scope, or they can be set on a per-service basis.

To set the values globally, use the following syntax:

```
thread_pool:VariableName
```

To set the values on a per-service basis, specify the service name (and optionally the service URI) from the Artix contract. The syntax is as follows:

```
thread_pool:VariableName:ServiceURI:ServiceName
```

Threading variables

This namespace includes following variables:

- `thread_pool:initial_threads`
- `thread_pool:high_water_mark`
- `thread_pool:low_water_mark`
- `thread_pool:max_queue_size`
- `thread_pool:stack_size`

The following variable applies to work queues:

- `service:owns_workqueue`

thread_pool:initial_threads

`thread_pool:initial_threads` specifies the number of initial threads in each port's thread pool. Defaults to `2`.

This variable can be set at different levels in your configuration. The following is a global setting:

```
thread_pool:initial_threads = "3";
```

The following setting is at the service name level, which overrides the global setting:

```
thread_pool:initial_threads:SessionManager = "1";
```

The following setting is at the fully-qualified service name level:

```
thread_pool:initial_threads:http://my.tns1/:SessionManager= "1";
```

This overrides the service name level, and is useful when there is a naming clash with service names from two different namespaces.

thread_pool:high_water_mark

`thread_pool:high_water_mark` sets the maximum number of threads allowed in each service's thread pool. Defaults to `25`.

This variable can be set at different levels in your configuration. The following is a global setting:

```
thread_pool:high_water_mark = "10";
```

The following setting is at the service name level, which overrides the global setting:

```
thread_pool:high_water_mark:SessionManager = "10";
```

The following setting is at the fully-qualified service name level:

```
thread_pool:high_water_mark:http://my.tns1/:SessionManager= "10";
```

This overrides the service name level, and is useful when there is a naming clash with service names from two different namespaces.

thread_pool:low_water_mark

thread_pool:low_water_mark sets the minimum number of threads in each service's thread pool. Artix will terminate unused threads until only this number exists. Defaults to 5.

This variable can be set at different levels in your configuration. The following is a global setting:

```
thread_pool:low_water_mark  = "5";
```

The following setting is at the service name level, which overrides the global setting:

```
thread_pool:low_water_mark:SessionManager = "5";
```

The following setting is at the fully-qualified service name level:

```
thread_pool:low_water_mark:http://my.tns1/:SessionManager= "5";
```

This overrides the service name level, and is useful when there is a naming clash with service names from two different namespaces.

thread_pool:max_queue_size

thread_pool:max_queue_size specifies the maximum number of request items that can be queued on the internal work queue. If this limit is exceeded, Artix considers the server to be overloaded, and gracefully closes down connections to reduce the load. Artix rejects subsequent requests until there is free space in the work queue.

Defaults to -1, which means that there is no upper limit on the size of the request queue. In this case, the maximum work queue size is limited by how much memory is available to the process.

The following is a global setting:

```
thread_pool:max_queue_size  = "10";
```

The following setting is at the service name level, which overrides the global setting:

```
thread_pool:max_queue_size:SessionManager = "10";
```

The following setting is at the fully-qualified service name level:

```
thread_pool:max_queue_size:http://my.tns1/:SessionManager= "10";
```

This overrides the service name level, and is useful when there is a naming clash with service names from two different namespaces.

thread_pool:stack_size

`thread_pool:stack_size` specifies the stack size for each thread. The stack size is specified in bytes. The default is the following global setting:

```
thread_pool:stack_size = "1048576";
```

The following setting is at the service name level, which overrides the global setting:

```
thread_pool:stack_size:SessionManager = "1048576";
```

The following setting is at the fully-qualified service name level:

```
thread_pool:stack_size:http://my.tns1/:SessionManager= "1048576";
```

This overrides the service name level, and is useful when there is a naming clash with service names from two different namespaces.

service:owns_workqueue

`service:owns_workqueue` specifies whether the service can own a workqueue. If this variable is set to `true`, the service can own a workqueue, if needed. For example, this means that if your application calls `Service::get_workqueue()`, this creates and returns a work queue specific to that service.

If this variable is set to is false, the service never owns a work queue. It uses the bus work queue. The default value is `true`.

Initial Contracts

Overview

Initial contracts specify the location of the WSDL contracts for Artix services. This provides a uniform mechanism for finding Artix services and contracts, and enables user code to be written in a location transparent way.

Because these variables are in the global scope of `artix.cfg`, every application can access the contracts. These contracts specify a `localhost:0` port, which means that the operating system assigns a TCP/IP port on startup.

To explicitly set a port, copy the relevant WSDL contract to another location, and edit it to include the port. In the application scope, add a `bus:initial_contract:url` entry that points to the edited WSDL file. The `bus:initial_contract:url` namespace includes the following variables:

- `container`
- `locator`
- `peermanager`
- `sessionmanager`
- `sessionendpointmanager`
- `uddi_inquire`
- `uddi_publish`
- `login_service`

container

`bus:initial_contract:url:container` specifies the location of the WSDL contract for the Artix container serivice. For example:

```
bus:initial_contract:url:container =
    "InstallDir/artix/Version/wsdl/container.wsdl";
```

locator

`bus:initial_contract:url:locator` specifies the location of the WSDL contract for the Artix locator serivice. For example:

```
bus:initial_contract:url:locator =
   "InstallDir/artix/Version/wsdl/locator.wsdl";
```

peermanager

`bus:initial_contract:url:peermanager` specifies the location of the WSDL contract for the Artix peer manager. For example:

```
bus:initial_contract:url:peermanager  =
   "InstallDir/artix/Version/wsdl/peer-manager.wsdl";
```

sessionmanager

`bus:initial_contract:url:sessionmanager` specifies the location of the WSDL contract for the Artix session manager. For example:

```
bus:initial_contract:url:sessionmanager  =
   "InstallDir/artix/Version/wsdl/session-manager.wsdl";
```

sessionendpointmanager

`bus:initial_contract:url:sessionendpointmanager` specifies the location of the WSDL contract for the Artix session endpoint manager. For example:

```
bus:initial_contract:url:sessionendpointmanager  =
   "InstallDir/artix/Version/wsdl/session-manager.wsdl";
```

uddi_inquire

`bus:initial_contract:url:uddi_inquire` specifies the location of the WSDL contract for the Artix UDDI inquire service. For example:

```
bus:initial_contract:url:uddi_inquire  =
   "InstallDir/artix/Version/wsdl/uddi/uddi_v2.wsdl";
```

uddi_publish

`bus:initial_contract:url:uddi_publish` specifies the location of the WSDL contract for the Artix UDDI publish service. For example:

```
bus:initial_contract:url:uddi_publish  =
   "InstallDir/artix/Version/wsdl/uddi/uddi_v2.wsdl";
```

login_service

`bus:initial_contract:url:login_service` specifies the location of the WSDL contract for the Artix peer manager. For example:

```
bus:initial_contract:url:login_service  =
   "InstallDir/artix/Version/wsdl/login_service.wsdl";
```

Further information

For more information on finding WSDL contracts, see Configuring and Deploying Artix Solutions.

Initial References

Overview

Initial references provide a uniform mechanism for enabling servers and clients to communicate with services deployed in the Artix container. This enables user code to be written in a location transparent way. The `bus:initial_references` namespace includes the following variables:

- `locator`
- `peermanager`
- `sessionmanager`
- `sessionendpointmanager`
- `uddi_inquire`
- `uddi_publish`
- `login_service`
- `container`

locator

`bus:initial_references:url:locator` specifies the location of an initial endpoint reference for the Artix locator service. For example:

```
bus:initial_references:url:locator = "./locator.ref";
```

For example, the `locator.ref` initial reference file can be generated using the following command:

```
it_container_admin -container ContainerService.url
    -publishreference -service
    {http://ws.iona.com/locator}LocatorService -file locator.ref
```

In this example, `it_container_admin` asks the Artix container service in `ContainerService.url` to publish an endpoint reference to a locator service. The same command can be used when a server or a client obtains an endpoint reference.

peermanager

bus:initial_references:url:peermanager specifies the location of an initial endpoint reference for the Artix peer manager service. For example:

```
bus:initial_references:url:peermanager = "./peermanager.ref";
```

For example, the peermanager.ref initial reference file can be generated using the following command:

```
it_container_admin -container ContainerService.url
   -publishreference -service
   {http://ws.iona.com/peer_manager}PeerManagerService -file
   peermanager.ref
```

In this example, it_container_admin asks the Artix container service in ContainerService.url to publish an endpoint reference to a peer manager service. The same command can be used when a server or a client obtains an endpoint reference.

sessionmanager

bus:initial_references:url:sessionmanager specifies the location of an initial endpoint reference for the Artix session manager service. For example:

```
bus:initial_references:url:sessionmanager =
   "./sessionmanager.ref";
```

For example, the sessionmanager.ref initial reference file can be generated using the following command:

```
it_container_admin -container ContainerService.url
   -publishreference -service
   {http://ws.iona.com/sessionmanager}SessionManagerService
   -file sessionmanager.ref
```

In this example, it_container_admin asks the Artix container service in ContainerService.url to publish an endpoint reference to a session manager service. The same command can be used when a server or a client obtains an endpoint reference.

sessionendpointmanager

`bus:initial_references:url:sessionendpointmanager` specifies the location of an initial endpoint reference for the Artix session endpoint manager service. For example:

```
bus:initial_references:url:sessionendpointmanager =
    "./sessionendpointmanager.ref";
```

For example, the `sessionendpointmanager.ref` initial reference file can be generated using the following command:

```
it_container_admin -container ContainerService.url
   -publishreference -service
   {http://ws.iona.com/sessionmanager}SessionEndpointManagerService
   -file sessionendpointmanager.ref
```

In this example, `it_container_admin` asks the Artix container service in `ContainerService.url` to publish an endpoint reference to a session endpoint manager service. The same command can be used when a server or a client obtains an endpoint reference.

uddi_inquire

`bus:initial_references:url:uddi_inquire` specifies the location of an initial endpoint reference for the Artix UDDI inquire service. For example:

```
bus:initial_references:url:uddi_inquire = "./uddi_inquire.ref";
```

For example, the `uddi_inquire.ref` initial reference file can be generated using the following command:

```
it_container_admin -container ContainerService.url
    -publishreference -service
    {http://www.iona.com/uddi_over_artix}UDDI_InquireService
    -file uddi_inquire.ref
```

In this example, `it_container_admin` asks the Artix container service in `ContainerService.url` to publish an endpoint reference to a UDDI inquire service. The same command can be used when a server or a client obtains an endpoint reference.

uddi_publish

`bus:initial_references:url:uddi_publish` specifies the location of an initial endpoint reference for the Artix UDDI publish service. For example:

```
bus:initial_references:url:uddi_publish = "./uddi_publish.ref";
```

For example, the `uddi_publish.ref` initial reference file can be generated using the following command:

```
it_container_admin -container ContainerService.url
   -publishreference -service
   {http://www.iona.com/uddi_over_artix}UDDI_PublishService
   -file uddi_publish.ref
```

In this example, `it_container_admin` asks the Artix container service in `ContainerService.url` to publish an endpoint reference to a UDDI publish service. The same command can be used when a server or a client obtains an endpoint reference.

login_service

`bus:initial_references:url:login_service` specifies the location of an initial endpoint reference for the Artix login service. For example:

```
bus:initial_references:url:login_service =
   "./login_service.ref";
```

For example, the `login_service.ref` initial reference file can be generated using the following command:

```
it_container_admin -container ContainerService.url
   -publishreference -service
   {http://ws.iona.com/login_service}LoginService -file
   locator.ref
```

In this example, `it_container_admin` asks the Artix container service in `ContainerService.url` to publish an endpoint reference to a login service. The same command can be used when a server or a client obtains an endpoint reference.

container

bus:initial_references:url:container specifies the location of an initial endpoint reference for the Artix container service. For example:

```
bus:initial_references:url:container = "./container.ref";
```

For example, the container.ref initial reference file can be generated using the following command:

```
it_container_admin -container ContainerService.url
   -publishreference -service
   {http://ws.iona.com/container}ContainerService -file
   container.ref
```

In this example, it_container_admin asks the Artix container service in ContainerService.url to publish an endpoint reference to a container service. The same command can be used when a server or a client obtains an endpoint reference.

QName Aliases

Overview

QName aliases are shorthand names for services in Artix configuration files. QNames are specified in the following format:

{*NamespaceURI*}*LocalPart*

For example: `{http://ws.iona.com/locator}LocatorService`. In this case, the `bus:initial_references:url:locator` variable is used as a shorthand instead of a more verbose format, such as `bus:initial_references:url:LocatorService:http://ws.iona.com/locator`.

The `bus:qname_alias` namespace includes the following variables:

- `container`
- `locator`
- `peermanager`
- `sessionmanager`
- `sessionendpointmanager`
- `uddi_inquire`
- `uddi_publish`
- `login_service`

container

`bus:qname_alias:container` specifies the QName alias for the Artix container service. For example:

```
bus:qname_alias:container =
"{http://ws.iona.com/container}ContainerService";
```

locator

bus:qname_alias:locator specifies the QName alias for the Artix locator service. For example:

```
bus:qname_alias:locator =
   "{http://ws.iona.com/locator}LocatorService";
```

peermanager

bus:qname_alias:peermanager specifies the QName alias for the Artix peer manager service. For example:

```
bus:qname_alias:peermanager =
   "{http://ws.iona.com/peer_manager}PeerManagerService";
```

sessionmanager

bus:qname_alias:sessionmanager specifies the QName alias for the Artix session manager service. For example:

```
bus:qname_alias:sessionmanager =
   "{http://ws.iona.com/sessionmanager}SessionManagerService";
```

sessionendpointmanager

bus:qname_alias:sessionendpointmanager specifies the QName alias for the Artix session endpoint manager service. For example:

```
bus:qname_alias:sessionendpointmanager =
   "{http://ws.iona.com/sessionmanager}SessionEndpointManagerService";
```

uddi_inquire

bus:qname_alias:uddi_inquire specifies the QName alias for the Artix UDDI inquire service. For example:

```
bus:qname_alias:uddi_inquire =
   "{http://www.iona.com/uddi_over_artix}UDDI_InquireService";
```

uddi_publish

bus:qname_alias:uddi_publish specifies the QName alias for the Artix UDDI publish service. For example:

```
bus:qname_alias:uddi_publish =
   "{http://www.iona.com/uddi_over_artix}UDDI_PublishService";
```

login_service

bus:qname_alias:login_service specifies the QName alias for the Artix login service. For example:

```
bus:qname_alias:login_service =
   "{http://ws.iona.com/login_service}LoginService";
```

Artix Reference Compatibility

Overview

From Artix 3.0.1, the proprietary references produced by Artix no longer use a hard coded `reference_properties` element name. Instead, Artix references use extension element names that are described in the port definition.

Artix 3.0.1 reference format

For example, when using SOAP, an Artix 3.0.1 stringified reference has the following format:

```
<?xml version='1.0' encoding='utf-8'?>
<m1:reference service="m2:AccountService"
                        wsdlLocation="file:./bank.wsdl"
                        xmlns:xs="http://www.w3.org/2001/XMLSchema"
                        xmlns:m1="http://www.iona.com/bus"
                        xmlns:m2="http://www.iona.com/bus/tests"

   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <port name="AccountPort" binding="m2:AccountBinding">
        <m3:address xsi:type="m3:tAddress"

   location="http://localhost:999/AccountService/AccountPort/"
            xmlns:m3="http://schemas.xmlsoap.org/wsdl/soap/">
        </m3:address>
    </port>
</m1:reference>
```

Pre-Artix 3.0.1 reference format

In earlier versions, stringified references had the following format:

```
<?xml version='1.0' encoding='utf-8'?>
<m1:reference service="m2:AccountService"
                        wsdlLocation="file:./bank.wsdl"
                        xmlns:xs="http://www.w3.org/2001/XMLSchema"
                        xmlns:m1="http://www.iona.com/bus"
                        xmlns:m2="http://www.iona.com/bus/tests"
                        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

    <port name="AccountPort" binding="m2:AccountBinding">
        <reference_properties xsi:type="m3:tAddress"
            location="http://localhost:999/AccountService/AccountPort/"
            xmlns:m3="http://schemas.xmlsoap.org/wsdl/soap/">
        </reference_properties>
    </port>
</m1:reference>
```

Note: This change is wire incompatible with previous versions of Artix.

bus:reference_2.1_compat

`bus:reference_2.1_compat` specifies backward compatability with pre-Artix 3.0.1 versions of an Artix reference. For example:

```
bus:reference_2.1_compat = "true";
```

If this variable is set to `true`, the Artix reference is generated in the pre-Artix 3.0.1 format. If this is not set or set to `false`, Artix references are generated in the Artix 3.0.1 format.

Artix Plug-ins

Artix is built on IONA's Adaptive Runtime architecture (ART), which enables users to configure services as plug-ins to the core product. This chapter explains the configuration settings for Artix-specific plug-ins.

Overview

Each Artix transport, payload format, and service has properties that are configurable as plug-ins to the Artix runtime. The variables used to configure plug-in behavior are specified in the configuration scopes of each Artix runtime instance, and follow the same order of precedence. A plug-in setting specified in the global configuration scope is overridden by a value set in a narrower scope.

For example, if you set `plugins:routing:use_pass_through` to `true` in the global scope, and set it to `false` in the `my_app` scope, all Artix runtimes, except for those running in the `my_app` scope, use `true` for this value. Any Artix instance using the `my_app` scope uses `false` for this value.

In this chapter

This chapter includes the following:

Bus

Overview

The plugins:bus namespace includes the following variables:

- plugins:bus:register_client_context
- plugins:bus:default_tx_provider:plugin

plugins:bus:register_client_context

plugins:bus:register_client_context specifies whether to register a client context. You can enable registration of client contexts as follows:

```
plugins:bus:register_client_context = "true";
```

The client context provides information about the origin of the incoming request (for example, its original IP address). By default, the context is not registered. This avoids any extra overhead associated with obtaining this information and populating the context.

plugins:bus:default_tx_provider:plugin

plugins:bus:default_tx_provider:plugin specifies the default transaction system used by Artix when a new transaction is started by bus.transactions().begin_transaction(). The specified value is the plug-in name of the transaction system provider plug-in. The available values are:

ots_tx_provider	Uses OTS as the transaction provider. Creates either an OTS Lite (single-resource) or OTS Encina (multi-resource) transaction. This is the default setting. For details of the additional configuration used to specify whether OTS Lite or OTS Encina is used, see Chapter 4.
wsat_tx_provider	Uses a WS-Coordination/WS-AtomicTransaction provider. The coordination service can either be run in-process or inside the Artix container.

Selecting a transaction provider

The choice of which transaction provider to use depends on the type of Artix binding your application uses. If most of your communication is over a CORBA binding, use `ots_tx_provider`. If most of your communication uses a SOAP binding, use `wsat_tx_provider`.

In both cases, Artix automatically interposes a transaction context of the correct type when a call is made over a particular binding. For example, if the default provider is OTS, and the application makes an outbound SOAP call, Artix includes a WS-AtomicTransaction SOAP header in the SOAP call. In this case, the transaction is still coordinated by OTS.

Similarly, if the default provider is WSAT, and a CORBA call is made, Artix automatically includes an OTS CORBA service context in the IIOP call. In this case, the transaction is coordinated by a WS-Coordination service.

orb_plugin configuration

The appropriate plug-in for your transaction system must also be loaded. For example, to load the OTS plug-in, include the `ots` plug-in name in the `orb_plugins` list:

```
# Artix configuration file
ots_lite_client_or_server {
    plugins:bus:default_tx_provider:plugin = "ots_tx_provider";
    orb_plugins = [ ..., "ots"];
};
```

For full details of using transaction systems in Artix, see Developing Artix Applications in C++.

CA WSDM Observer

Overview

The plugins:ca_wsdm_observer namespace includes the following variables:

- plugins:ca_wsdm_observer:auto_register
- plugins:ca_wsdm_observer:config_poll_time
- plugins:ca_wsdm_observer:handler_type
- plugins:ca_wsdm_observer:max_queue_size
- plugins:ca_wsdm_observer:min_queue_size
- plugins:ca_wsdm_observer:report_wait_time

plugins:ca_wsdm_observer:auto_register

plugins:ca_wsdm_observer:auto_register specifies whether the Artix CA WSDM observer automatically registers observed services with a WSDM service. The default is:

```
plugins:ca_wsdm_observer:auto_register = "true";
```

If you have a large number of observed services, the runtime performance may be decreased because of equally large register service requests sent to a WSDM service.

You can set this variable to false and manually import service details from WSDL definitions into a WSDM console. However, this only works for SOAP-HTTP non-transient services. This is because WSDM can not import non-SOAP services described in WSDL, while Artix does not publish WSDL for transient services.

plugins:ca_wsdm_observer:config_poll_time

`plugins:ca_wsdm_observer:config_poll_time` specifies how often, in seconds, the observer should poll a WSDM service for configuration updates, use the following variable:

```
plugins:ca_wsdm_observer:config_poll_time
```

The default is `180` seconds (`3` minutes). Configuration updates tell the observer whether transaction monitors have been enabled. If so, the observer copies input/output raw messages, and reports them to a WSDM service if duration or request/response size thresholds have been exceeded.

plugins:ca_wsdm_observer:handler_type

`plugins:ca_wsdm_observer:handler_type` specifies a value that identifies an Artix observer to a WSDM service. It should be above `200`. The default is:

```
plugins:ca_wsdm_observer:handler_type = "217";
```

In addition, if you change the default, you must also update the following file with the new handler type:

```
WSDM-Install-Dir/server/default/conf/WsdmSOMMA_Basic.properties
```

Entries in this file take a format of `observertype.X=ArtixObserver`, where `X` is the handler type value. The default entry is:

```
observertype.217=ArtixObserver
```

plugins:ca_wsdm_observer:max_queue_size

`plugins:ca_wsdm_observer:max_queue_size` specifies the maximum number of service request records that the observer queue can hold. For example:

```
plugins:ca_wsdm_observer:max_queue_size = "600";
```

The default is `500`. New records are dropped when the queue size reaches this value. If `report_wait_time` is not set, this variable is ignored. In this case, reports are sent as soon as the queue size is equal to `max_queue_size`.

plugins:ca_wsdm_observer:min_queue_size

`plugins:ca_wsdm_observer:min_queue_size` specifies how many service request records must be available in a queue before a report is sent to a WSDM service. For example:

```
plugins:ca_wsdm_observer:min_queue_size = "6";
```

The default is `5`. Set this variable if your load is expected to be large. If this variable is too low, the observer may send reports too frequently, and if it is too high, the memory footprint may increase significantly.

plugins:ca_wsdm_observer:report_wait_time

`plugins:ca_wsdm_observer:report_wait_time` specifies how often reports should be sent in seconds. For example:

```
plugins:ca_wsdm_observer:report_wait_time = 10;
```

This variable is an alternative to `min_queue_size`, which instead specifies the frequency of reports on a time basis. This variable should be used with `max_queue_size`.

Client-Side High Availability

Overview

The variables in the `plugins:ha_conf` namespace configure client-side high availability settings:

- `plugins:ha_conf:strategy`
- `plugins:ha_conf:random:selection`

plugins:ha_conf:strategy

`plugins:ha_conf:strategy` specifies whether the client uses `random` or `sequential` endpoint selection. For example, specifying `random` enables client applications to select a random server each time they connect:

```
plugins:ha_conf:strategy="random";
```

Defaults to `sequential`.

plugins:ha_conf:random:selection

`plugins:ha_conf:random:selection` specifies whether the client always selects a random server or only after the client loses connectivity with the first server in the list. Possible values are `always` or `subsequent`.

Specify `always` if you want your clients to be uniformly load-balanced across different servers, for example:

```
plugins:ha_conf:strategy="random";
plugins:ha_conf:random:selection="always";
```

Specify `subsequent` if you want your clients to favour a particular server for their initial connectivity, for example:

```
plugins:ha_conf:strategy="random";
plugins:ha_conf:random:selection="subsequent";
```

Defaults to `always`.

Container

Overview

The plugins:container namespace includes the following variables:

- plugins:container:deployfolder
- plugins:container:deployfolder:readonly

plugins:container:deployfolder

plugins:container:deployfolder specifies the location of a local folder where deployment descriptor files are saved to, and where they are read from on restart. For example:

```
plugins:container:deployfolder="../etc";
```

At startup, the container looks in the configured deployment folder and deploys the contents of the folder.

By default, this folder enabled for dynamic read/write deployment. This means that the container adds and removes files from the deployment folder dynamically as services are deployed or removed from the container.

plugins:container:deployfolder:readonly

plugins:container:deployfolder:readonly specifies whether the local folder used to store deployment descriptor file is a read-only folder. This can be used as an initialization folder to predeploy the same required set of services after every restart.

This variable should be used in conjunction with plugins:container:deployfolder. For example, the following configuration enables a read-only persistent deployment folder:

```
plugins:container:deployfolder:readonly="true";
```

Database Environment

Overview

The variables in the plugins:artix:db namespace configure database environment and service replication settings:

- plugins:artix:db:allow_minority_master
- plugins:artix:db:auto_demotion
- plugins:artix:db:db_open_retry_attempts
- plugins:artix:db:download_files
- plugins:artix:db:election_timeout
- plugins:artix:db:env_name
- plugins:artix:db:home
- plugins:artix:db:iiop:port
- plugins:artix:db:inter_db_open_sleep_period
- plugins:artix:db:max_buffered_msgs
- plugins:artix:db:max_msg_buffer_size
- plugins:artix:db:max_ping_retries
- plugins:artix:db:ping_lifetime
- plugins:artix:db:ping_retry_interval
- plugins:artix:db:priority
- plugins:artix:db:replica_name
- plugins:artix:db:replicas
- plugins:artix:db:roundtrip_timeout
- plugins:artix:db:sync_retry_attempts
- plugins:artix:db:wsdl_port

plugins:artix:db:allow_minority_master

`plugins:artix:db:allow_minority_master` specifies whether a lone slave can promote itself to a master if it sees that the current master is unavailable. This is only allowed when the replica cluster has two members. This variable defaults to `false` (not allowed). If it is set to `true`, a slave that cannot reach its partner replica will promote itself to master, even though it only has fifty per cent of the votes (one out of two).

WARNING: This variable must be used with caution. If it is set to `true`, and the two replicas in the cluster become separated due to a network partition, they are both promoted to master. This can be very problematic because both replicas could make database updates, and resolving those updates later could be very difficult, if not impossible.

It is recommended that high availability clusters have an odd number of members, and the recommended minimum number is three. It is only possible to use a cluster with two members if you specify the following configuration:

```
plugins:artix:db:allow_minority_master=true;
```

plugins:artix:db:auto_demotion

`plugins:artix:db:auto_demotion` specifies whether a master automatically demotes itself to a slave when it loses contact with the majority of the replica cluster. Defaults to `true`.

The problem of duplicate masters is crucial for any election-based high availability system. Every effort must be taken to ensure that only one master exists at any one time, because database updates made to multiple masters can be extremely difficult to resolve.

The most common cause of duplicate masters to appear is a network partition. This is a split in the network that leaves the current master on one side and a majority of slaves on the other side. Because the slaves have the majority of votes, they elect a master on their side.

When this variable is set to `true`, duplicate masters should never exist. If a master loses contact with the majority of the replica set, it will automatically demote itself to slave.

WARNING: This variable must be used with caution. If it is set to `false`, there is a chance that duplicate masters may appear after a network partition. If this happens, and the partition is repaired (allowing the masters to see each other), both masters will self-demote to a slave, hold an election to determine who is most up-to-date, and re-elect a master. If this occurs, any updates made on a demoted master when it was separated from the replicas will be lost.

plugins:artix:db:db_open_retry_attempts

`plugins:artix:db:db_open_retry_attempts` specifies the number of attempts made by a slave to open its new database.

When a slave starts for the first time and synchronizes with an existing master, it may take some time for a slave to receive the master's database over the wire, especially if the database is large. If the slave gets `no such file or directory` errors when starting up, it may help to increase this value. Defaults to `5`.

plugins:artix:db:download_files

`plugins:artix:db:download_files` specifies whether fresh slaves download the entire database from the master before starting up. Defaults to `true`. Before starting up, fresh slaves have no database files on their local filesystem.

There may be circumstances where fresh slaves should not download the entire database before starting up. For example, if the database very large, it may be desirable to allow Berkeley DB to synchronize the databases instead.

plugins:artix:db:election_timeout

plugins:artix:db:election_timeout specifies the time spent attempting to elect a new master. If a master can not be found in this time, a new election is started. Defaults to 2000 milliseconds (2 seconds). You should not often need to change this setting.

plugins:artix:db:env_name

plugins:artix:db:env_name specifies the filename for the Berkeley DB environment file. The value specified must be the same for all replicas. Defaults to db_env. You should not need to change this setting.

plugins:artix:db:home

plugins:artix:db:home specifies the directory where Berkeley DB stores all the files for the service databases. Each service should have a dedicated folder for its data stores. This is especially important for replicated services.

Defaults to *ReplicaConfigScope*_db (for example, rep1_db), where *ReplicaConfigScope* is the inner-most replica configuration scope. You should not need to explicitly set this variable. If this directory does not already exist, it will be created in the current working directory.

plugins:artix:db:iiop:port

plugins:artix:db:iiop:port specifies the IIOP port that the replica service starts on, and is used for communications between replicas. Defaults to 0.

This variable must be set in a sub-scope for each replica specified in the plugins:artix:db:replicas list. The following example shows a sub-scope for the rep1 replica:

```
rep1{
    plugins:artix:db:priority = 80;
    plugins:artix:db:iiop:port = 2000;
  };
```

plugins:artix:db:inter_db_open_sleep_period

plugins:artix:db:inter_db_open_sleep_period specifies the amount of time spent sleeping between failed database open attempts on the slave side. This variable is related to plugins:artix:db:db_open_retry_attempts.

Defaults to 2000 milliseconds (2 seconds).

plugins:artix:db:max_buffered_msgs

plugins:artix:db:max_buffered_msgs specifies the maximum number of batch messages stored in the message buffer of a high availabilty database. All messages are sent and the buffer is flushed when this limit is reached. Defaults to 10. This feature helps to reduce the traffic between replicas.

plugins:artix:db:max_msg_buffer_size

plugins:artix:db:max_msg_buffer_size specifies the maximum size of the message buffer of a high availabilty database. All messages are sent and the buffer is flushed when this limit is reached. Defaults to 10240. This feature helps to reduce the traffic between replicas.

plugins:artix:db:max_ping_retries

plugins:artix:db:max_ping_retries specifies how many failed pings between replicas can happen before the remote replica is considered unreachable. The replica is then marked as unavailable until it can be pinged again.

Defaults to 1. This means that if one ping fails, the replica is marked as UNAVAIL, and no attempt is made to send it any database update or election packets until it becomes available again.

For more details, see plugins:artix:db:ping_lifetime.

plugins:artix:db:ping_lifetime

`plugins:artix:db:ping_lifetime` specifies the amount of time that the servant pinging replicas waits for before returning. Defaults to `10000` milliseconds (`10` seconds).

Replicas monitor each other using inter-replica pings. These pings are optimized to minimize the amount of network traffic between replicas. This optimization is based on specifying long-lived pings.

If the server process dies before returning, the caller gets an immediate notification of the failure of the ping. However, if the server machine dies, the notification occurs when `plugins:artix:db:roundtrip_timeout` expires. This is because the server-side TCP/IP stack can not notify the caller of connection failure if the host machine dies unexpectedly.

plugins:artix:db:ping_retry_interval

`plugins:artix:db:ping_retry_interval` specifies the number of milliseconds between inter-replica ping attempts. Defaults to `2000` milliseconds (`2` seconds).

For more details, see `plugins:artix:db:ping_lifetime`.

plugins:artix:db:priority

`plugins:artix:db:priority` specifies the replica priority. The higher the priority the more likely the replica is to be elected as master. This variable should be set if you are using replication.

There is no guarantee that the replica with the highest priority is elected master. The first consideration for electing a master is who has the most current database. Setting a priority of `0` means that the replica is never elected master. Defaults to `1`.

This variable must be set in a sub-scope for each replica. See the example for `plugins:artix:db:iiop:port`.

plugins:artix:db:replica_name

plugins:artix:db:replica_name specifies a simple string name for the replica. It indicates the replica in the plugins:artix:db:replicas list that this configuration refers to.

This variable must be set if plugins:artix:db:replicas is set, otherwise a DBException/BAD_CONFIGURATION is thrown. Each replica must have its own unique name, and must be present in the list.

Defaults to the replica's innermost configuration scope (for example, rep1). This value is automatically inferred and does not need to be explicitly set, unless you wish to use a different replica name.

plugins:artix:db:replicas

plugins:artix:db:replicas specifies a cluster of replica services. This variable takes a list of replicas specified using the following syntax:

ReplicaName=*HostName*:*PortNum*

For example, the following entry configures a cluster of three replicas spread across three machines named jimi, noel, and mitch.

```
plugins:artix:db:replicas=["rep1=jimi:2000", "rep2=mitch:3000",
"rep3=noel:4000"];
```

Defaults to an empty list.

Note: It is recommended that you set *ReplicaName* to the same value as the replica's configuration scope (see plugins:artix:db:replica_name).

plugins:artix:db:roundtrip_timeout

plugins:artix:db:roundtrip_timeout specifies the amount of time that a replica waits for a response from a ping sent to another replica. Defaults to 20000 milliseconds (20 seconds).

If this variable is not set, some failed pings may take a long time to return (for example, if the target machine loses power). When a machine fails, the TCP/IP stack on the machine can not terminate the connection. The client still waits for a reply, and thinks that the connection is still valid.

The client only sees that the connection dies when TCP/IP times out and marks the connection as terminated. The variable prevents this situation from occurring.

Note: This variable must be set to a larger value than plugins:artix:db:ping_lifetime. Otherwise, valid pings would be regarded as having timed out when they are still in progress.

plugins:artix:db:sync_retry_attempts

plugins:artix:db:sync_retry_attempts specifies the maximum number of times that the slave sends a synchronization request to the master. This is used when a slave starts for the first time and synchronizes with an existing master.

Slave synchronization is performed by the slave sending a request to the master to write a small piece of data to its database, and then the slave waiting for this data to appear. When the data appears on the slave side, the slave knows it is processing live records from the master and is up-to-date and synchronized. Defaults to 5. You should rarely need to change this setting.

plugins:artix:db:wsdl_port

plugins:artix:db:wsdl_port specifies the WSDL port name for the replica that is used in the service's WSDL contract.

Defaults to the replica's innermost configuration scope (for example, rep1). This value is automatically inferred and does not need to be explicitly set, unless you wish to use a different WSDL port name.

FTP

Overview

The plugins:ftp namespace contains variables for File Transfer Protocol. These include the following:

- plugins:ftp:policy:client:filenameFactory
- plugins:ftp:policy:client:replyFileLifecycle
- plugins:ftp:policy:connection:connectMode
- plugins:ftp:policy:connection:connectTimeout
- plugins:ftp:policy:connection:receiveTimeout
- plugins:ftp:policy:connection:scanInterval
- plugins:ftp:policy:connection:useFilenameMaskOnScan
- plugins:ftp:policy:credentials:name
- plugins:ftp:policy:credentials:password
- plugins:ftp:policy:server:filenameFactory
- plugins:ftp:policy:server:requestFileLifecycle

plugins:ftp:policy:client:filenameFactory

plugins:ftp:policy:client:filenameFactory specifies the name of the class that implements the client's filename factory. This generates the filenames used for storing request messages on the FTP server, and determines the name of the associated replies.

This classname must be listed on the endpoint's classpath. The default setting is:

```
plugins:ftp:policy:client:filenameFactory="com.iona.jbus.transpo
    rts.ftp.policy.client.DefaultFilenameFactory";
```

plugins:ftp:policy:client:replyFileLifecycle

`plugins:ftp:policy:client:replyFileLifecycle` specifies the name of the class that implements the client's reply lifecycle policy. The reply lifecycle policy is responsible for instructing the Artix runtime whether a reply file must be deleted or moved to a different FTP server location.

This classname must be listed on the endpoint's classpath. The default setting is:

```
plugins:ftp:policy:client:replyFileLifecycle="com.iona.jbus.tran
    sports.ftp.policy.client.DefaultReplyFileLifecycle";
```

plugins:ftp:policy:connection:connectMode

`plugins:ftp:policy:connection:connectMode` specifies the connection mode used to connect to the FTP daemon. Valid values are `passive` and `active`. The default is:

```
plugins:ftp:policy:connection:connectMode="passive";
```

plugins:ftp:policy:connection:connectTimeout

`plugins:ftp:policy:connection:connectTimeout` specifies a timeout value in milliseconds for establishing a connection with a remote FTP daemon. The default is:

```
plugins:ftp:policy:connection:connectTimeout="-1";
```

plugins:ftp:policy:connection:receiveTimeout

`plugins:ftp:policy:connection:receive:Timeout` specifies a receive timeout value in milliseconds for the FTP daemon filesystem scanner. The receive timeout will occur when the following condition is met:

```
CurrentTime - StartReplyScanningTime >=
    plugins:ftp:policy:connection:receiveTimeout
```

It is recommended that the receive timeout value is greater than `plugins:ftp:policy:connection:scanInterval` * `1000`. If this value is set to `0`, it is guaranteed that there will be at least one scan of the remote FTPD filesystem before the timeout. The default is:

```
plugins:ftp:policy:connection:receiveTimeout="-1";
```

plugins:ftp:policy:connection:scanInterval

`plugins:ftp:policy:connection:scanInterval` specifies the interval, in seconds, at which the request and reply locations are scanned for updates. The default is:

```
plugins:ftp:policy:connection:scanInterval="5";
```

plugins:ftp:policy:connection:useFilenameMaskOnScan

`plugins:ftp:policy:connection:useFilenameMaskOnScan` specifies whether the Artix runtime uses a filename mask when calling the FTP daemon with a FTP `LIST` command (for example, `LIST myrequests*`).

Some FTP daemons do not implement support for listing a subset of files based on a filename mask. To enable interoperability with such servers, this variable must be set to `false`. However, if you know that an FTP daemon supports a filtered `LIST` command, setting this variable to `true` increases FTP transport performance. The default is:

```
plugins:ftp:policy:connection:useFilenameMaskOnScan="false";
```

plugins:ftp:policy:credentials:name

`plugins:ftp:policy:credentials:name` specifies the FTP daemon user name. This variable along with `plugins:ftp:policy:credentails:password` must have credentails that allows the Artix runtime to list, add, move and remote files from the filesystem location provided using FTP WSDL extensors. The default is:

```
plugins:ftp:policy:credentials:name="anonymous";
```

plugins:ftp:policy:credentials:password

`plugins:ftp:policy:credentials:password` specifies the FTP daemon user password. The default is:

```
plugins:ftp:policy:credentials:password="anonymous@anonymous.net";
```

plugins:ftp:policy:server:filenameFactory

`plugins:ftp:policy:server:filenameFactory` specifies the name of the class that implements the client's filename factory. The filename factory is responsible for identifying which requests to dispatch, and how to name reply messages.

This classname must be listed on the endpoint's classpath. The default setting is:

```
plugins:ftp:policy:server:filenameFactory="com.iona.jbus.transpo
    rts.ftp.policy.server.DefaultFilenameFactory";
```

plugins:ftp:policy:server:requestFileLifecycle

`plugins:ftp:policy:server:requestFileLifecycle` specifies the name of the class that implements the server's request lifecycle policy. The request lifecycle policy is responsible for instructing the Artix runtime whether a request file must be deleted or moved to a different FTP server location.

This classname must be listed on the endpoint's classpath. The default setting is:

```
plugins:ftp:policy:server:requestFileLifecycle="com.iona.jbus.tr
    ansports.ftp.policy.server.DefaultRequestFileLifecycle";
```

JMS

Overview

The variables in the plugins:jms namespace configure settings for interoperability with the Java Message Service. These include the following:

- plugins:jms:policies:binding_establishment:backoff_ratio
- plugins:jms:policies:binding_establishment:initial_iteration_delay
- plugins:jms:policies:binding_establishment:backoff_ratio
- plugins:jms:pooled_session_high_water_mark
- plugins:jms:pooled_session_low_water_mark

plugins:jms:policies:binding_establishment:backoff_ratio

plugins:jms:policies:binding_establishment:backoff_ratio specifies the degree to which delays between reconnection retries increase from one retry to the next. This is used when Artix tries to reconnect to the Java Message Service after a connection is dropped (for example, if JMS becomes unavailable, or a network error occurs).

The successive delays between retries use the following geometric progression:

Retry Number	Delay
1	initial_iteration_delay X backoff_ratio ^{0}
2	initial_iteration_delay X backoff_ratio ^{1}
n	initial_iteration_delay X backoff_ratio $^{(n-1)}$

For example, if the initial_iteration_delay is 1000 milliseconds, and the backoff_ratio is 2:

- The first retry waits 1000 milliseconds.
- The second retry waits 1000 x 2 milliseconds.
- The third retry waits 1000 x 2 x 2 milliseconds.

  ....
- The nth retry waits 1000 x 2^(n-1).

The data type is long, and values must be greater than or equal to 0. Defaults to 2:

```
plugins:jms:policies:binding_establishment:backoff_ratio="2";
```

In your code, in the event of an initial failure, or an inability to make a connection after the configured retries have been exhausted, a method call will receive a RemoteException, which wraps a TransportException.

plugins:jms:policies:binding_establishment:initial_iteration_delay

plugins:jms:policies:binding_establishment:initial_iteration_delay specifies the amount of time, between the first and second attempts to establish a connection with a JMS broker.

The data type is long, and values must be greater than or equal to 0. Defaults to 1000 milliseconds:

```
plugins:jms:policies:binding_establishment:initial_iteration_delay="1000";
```

plugins:jms:policies:binding_establishment:max_binding_iterations

plugins:jms:policies:binding_establishment:max_binding_iterations specifies the limit on the number of times that an Artix client tries to reconnect to a JMS broker. To disable reconnecting to the Java Message Service, set this variable to 0.

The data type is long, and values must be greater than or equal to 0. Defaults to 5:

```
plugins:jms:policies:binding_establishment:max_binding_iterations="5";
```

plugins:jms:pooled_session_high_water_mark

`plugins:jms:pooled_session_high_water_mark` specifies the limit on the number of temporary JMS queues. The high water mark minus the low water mark equals the number of soft references that are stored.

Temporary queues that are stored as soft references will only be garbage collected if memory becomes an issue for the client. However, any temporary queue that is reaped will potentially be replaced by another queue later. The default value is:

```
plugins:jms:pooled_session_high_water_mark = "500";
```

For example, by default, there are 520 temporary queues—500 soft references and 20 strong references (see `plugins:jms:pooled_session_low_water_mark`).

Note: Setting the high water mark value too high could cause problems with the JMS broker that the client is not aware of.

plugins:jms:pooled_session_low_water_mark

`plugins:jms:pooled_session_low_water_mark` specifies the number of temporary JMS queues that are stored as strong references. This is the number of queues that remain in memory.

Temporary queues stored as strong references will never be garbage collected, unless the client times out. In the event of a timeout, the temporary queue is reaped to avoid it being used by another invocation. However, any temporary queue that is reaped will potentially be replaced by another queue later. The default value is:

```
plugins:jms:pooled_session_low_water_mark = "20";
```

For example, by default, there are 520 temporary queues—20 strong references and 500 soft references (see `plugins:jms:pooled_session_high_water_mark`).

JMX

Overview

The plugins:bus_management namespace includes variables that specify JMX monitoring of the Artix runtime. JMX stands for Java Management Extensions. These variables include:

- plugins:bus_0:enabled
- plugins:bus_management:connector:enabled
- plugins:bus_management:connector:port
- plugins:bus_management:connector:registry:required
- plugins:bus_management:connector:url:publish
- plugins:bus_management:connector:url:file
- plugins:bus_management:http_adaptor:enabled
- plugins:bus_management:http_adaptor:port

plugins:bus_0:enabled

plugins:bus_management:enabled specifies whether the Artix runtime can be managed locally using JMX MBeans. The default setting is false. To enable local JMX monitoring, set this variable to true:

```
plugins:bus_management:enabled="true";
```

This setting enables a local access to JMX runtime MBeans. The bus_management plug-in wraps runtime components into open dynamic MBeans and registers them with a local MBeanServer.

plugins:bus_management:connector:enabled

`plugins:bus_management:connector:enabled` specifies whether the Artix runtime can be managed remotely using JMX MBeans. The default setting is `false`. To enable remote JMX monitoring, set the following variables to `true`:

```
plugins:bus_management:enabled="true";
plugins:bus_management:connector:enabled="true";
```

These settings allow for both local and remote access.

Remote access is performed through JMX Remote, using an RMI Connector on a default port of `1099`. When the configuration has been set, you can use the following default JNDI-based JMXServiceURL to connect remotely:

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

plugins:bus_management:connector:port

`plugins:bus_management:connector:port` specifies a port for remote JMX access. For example, given the following setting:

```
plugins:bus_management:connector:port="2000";
```

You can then use the following JMXServiceURL:

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

plugins:bus_management:connector:registry:required

`plugins:bus_management:connector:registry:required` specifies whether the connector uses a stub-based JMXServiceURL. For example, the following settings enable stub-based access:

```
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.

plugins:bus_management:connector:url:publish

plugins:bus_management:connector:url:publish specifies whether publishing the JMXServiceURL to a local file is enabled. To enable this, specify the following:

```
plugins:bus_management:connector:url:publish="true";
```

plugins:bus_management:connector:url:file

plugins:bus_management:connector:url:file specifies a filename for publishing the JMXServiceURL to a local file. For example, the following settings override the default filename:

```
plugins:bus_management:connector:url:publish="true";
plugins:bus_management:connector:url:file="../../service.url";
```

plugins:bus_management:http_adaptor:enabled

plugins:bus_management:http_adaptor:enabled specifies whether the default HTTP adaptor console supplied by the JMX reference implementation is enabled. To enable this adaptor, specify the following:

```
plugins:bus_management:http_adaptor:enabled="true";
```

plugins:bus_management:http_adaptor:port

plugins:bus_management:http_adaptor:port specifies a port for the default HTTP adaptor console supplied by the JMX reference implementation. For example:

```
plugins:bus_management:http_adaptor:port="7659";
```

To access the HTTP adaptor on this port, specify http://localhost:7659 in your browser.

Local Log Stream

Overview

The variables in the plugins:local_log_stream namespace configure text-based logging. By default, Artix is configured to log messages in an XML format. You can change this behavior using the local_log_stream plug-in.

The plugins:local_log_stream namespace contains the following variables:

- plugins:local_log_stream:buffer_file
- plugins:local_log_stream:filename
- plugins:local_log_stream:filename_date_format
- plugins:local_log_stream:log_elements
- plugins:local_log_stream:log_thread_id
- plugins:local_log_stream:milliseconds_to_log
- plugins:local_log_stream:rolling_file

plugins:local_log_stream:buffer_file

plugins:local_log_stream:buffer_file specifies whether the output stream is sent to a buffer before it writes to a local log file. To specify this behavior, set this variable to true:

```
plugins:local_log_stream:buffer_file = "true";
```

When set to true, by default, the buffer is output to a file every 1000 milliseconds when there are more than 100 messages logged. This log interval and number of log elements can also be configured.

plugins:local_log_stream:filename

`plugins:local_log_stream:filename` sets the output stream to the specified local text file. For example:

```
plugins:local_log_stream:filename = "/var/adm/mylocal.log";
```

If you do not specify a file name, logging is sent to `stdout`.

plugins:local_log_stream:filename_date_format

`plugins:local_log_stream:filename_date_format` specifies the format of the date in a text-based rolling log file. The specified date conforms to the format rules of the ANSI C `strftime()` function. For example:

```
plugins:local_log_stream:rolling_file="true";
plugins:local_log_stream:filename="my_log";
plugins:local_log_stream:filename_date_format="_%Y_%m_%d";
```

On the 31st January 2006, this results in a log file named `my_log_2006_01_31`.

plugins:local_log_stream:log_elements

`plugins:local_log_stream:log_elements` specifies the number of log messages that must be in the buffer before they are output to a log file. The default is 100 messages.

For example, the following configuration writes the log output to a log file if there are more than 20 log messages in the buffer.

```
plugins:local_log_stream:log_elements = "20";
```

plugins:local_log_stream:log_thread_id

`plugins:local_log_stream:log_thread_id` specifies whether the thread ID is logged in the log message or not, for example:

```
plugins:local_log_stream:log_thread_id = "true";
```

The default is `true`.

plugins:local_log_stream:milliseconds_to_log

`plugins:local_log_stream:milliseconds_to_log` specifies how often in milliseconds that the log buffer is output to a log file. The default is `1000` milliseconds.

For example, the following configuration writes the log output to a log file every 400 milliseconds.

```
plugins:local_log_stream:milliseconds_to_log = "400";
```

plugins:local_log_stream:rolling_file

`plugins:local_log_stream:rolling_file` is a boolean which specifies that the logging plug-in creates a new log file each day to prevent the log file from growing indefinitely. In this model, the stream appends the current date to the configured filename. This produces a complete filename, for example:

```
/var/adm/artix.log.02172006
```

A new file begins with the first event of the day and ends at 23:59:59 each day. The default behavior is `true`. To disable rolling file behavior, set this variable to false. For example:

```
plugins:local_log_stream:rolling_file = "false";
```

Locator Service

Overview

The locator service plug-in, service_locator, is configured by the variables in the plugins:locator namespace:

- plugins:locator:peer_timeout
- plugins:locator:persist_data
- plugins:locator:selection_method
- plugins:locator:service_group
- plugins:locator:wsdl_port

plugins:locator:peer_timeout

plugins:locator:peer_timeout specifies the amount of time, in milliseconds, that the locator plug-in waits between keep-alive pings of the endpoints that are registered with it. The default and minimum setting is 10000 milliseconds (10 seconds).

The locator uses a third-party peer manager to ping its endpoints. For more details, see "Peer Manager" on page 82.

plugins:locator:persist_data

plugins:locator:persist_data enables persistence in the locator. This variable specifies whether the locator uses a persistent database to store references. For example:

```
plugins:locator:persist_data="true";
```

Defaults to false, which means that the locator uses an in-memory map to store references. When replicating the locator you must set persist_data to true. If you do not, replication does not work.

plugins:locator:selection_method

plugins:locator:selection_method specifies the load balancing selection method used by the locator.

When plugins:locator:persist_data is set to true, the locator to switches from round robin to random load balancing.

You can change the default behavior of the locator to always use random load balancing by setting the following:

```
plugins:locator:selection_method = “random”;
```

plugins:locator:service_group

plugins:locator:service_group specifies an arbitrary group name for an Artix service or bus. For example, you can use this to query the locator for a specified group of services.

There are no restrictions on assigning services to groups in different processes. Services in the same process can belong to different groups, or to no group. Services in different processes can belong to the same group. By default, a service belongs to no group. Specifying a group in a configuration file takes precedence over specifying a group in a WSDL file.

Specifying a group for a service

The following example defines a QName alias named corba_svc, and assigns this to a group named CORBAGroup.

```
bus:qname_alias:corba_svc =
   "{http://demo.iona.com/advanced/LocatorQuery}CORBAService";
plugins:locator:service_group:corba_svc = "CORBAGroup";
```

Specifying a group for a bus

You can also define a global group for all services in the current bus. All services that do not have a group definition in WSDL or configuration then belong to the global group by default.

```
plugins:locator:service_group = "DefaultGroupName";
```

plugins:locator:wsdl_port

plugins:locator:wsdl_port specifies a locator WSDL port for a locator replica service. This allows the locator to specify the WSDL port that it uses when registering its own servant. This feature enables forwarding of write requests from a slave to a master locator. The following is an example setting:

```
plugins:locator:wsdl_port=Locator1;
```

Defaults to the replica’s locator configuration scope name (for example, Locator1). This value is automatically inferred and does not need to be explicitly set, unless you wish to use a different WSDL port name.

Locator Endpoint Manager

Overview

The locator endpoint manager plug-in, `locator_endpoint`, is configured by the following variables:

- plugins:locator_endpoint:exclude_endpoints
- plugins:locator_endpoint:include_endpoints
- plugins:locator_endpoint:peer_timeout

plugins:locator_endpoint:exclude_endpoints

`plugins:locator_endpoint:exclude_endpoints` specifies endpoints to be exluded from the locator. For example, if do not you want to register the container service, but want to register all the endpoints that are activated in that container, use the following setting:

```
plugins:locator_endpoint:exclude_endpoints =
    ["{http://ws.iona.com/container}ContainerService"];
```

You can also wildcard your service names. This enables you to filter based on a specified namespace. For example:

```
plugins:locator_endpoint:exclude_endpoints =
    ["{http://www.sample.com/finance}*"];
```

plugins:locator_endpoint:include_endpoints

`plugins:locator_endpoint:include_endpoints` specifies endpoints to be included in the locator. For example, if you only want to register the session manager, but not any of the endpoints that it manages, use the following setting:

```
plugins:locator_endpoint:include_endpoints =
    ["{http://ws.iona.com/sessionmanager}SessionManagerService"];
```

You can also wildcard your service names. This enables you to filter based on a namespace. For example:

```
plugins:locator_endpoint:include_endpoints =
    ["{http://www.sample.com/finance}*"];
```

Note: Combining the `exclude_endpoints` and `include_endpoints` variables is ambiguous. If you do this, the application will fail to initialize.

plugins:locator_endpoint:peer_timeout

`plugins:locator:peer_timeout` specifies the amount of time, in milliseconds, that the locator endpoint plug-in waits between keep-alive pings back to the locator. The default and minimum setting is `10000` milliseconds (10 seconds).

The locator service endpoint uses a third-party peer manager to ping back to the locator. For more details, see "Peer Manager" on page 82.

Peer Manager

Overview

The peer manager is used by the locator and session manager to ping their endpoints, and verify that they are still running. The `peer_manager` plug-in is transparently loaded by the following plug-ins:

- `service_locator`
- `locator_endpoint`
- `session_manager_service`
- `session_endpoint_manager`

The `peer_manager` includes the following configuration variables:

- `plugins:peer_manager:timeout_delta`

plugins:peer_manager:timeout_delta

`plugins:peer_manager:timeout_delta` specifies the time allowed for failover detection in milliseconds. The default is `2000`. For example, increasing this to `10000` ensures that only a real failure results in an endpoint being removed from the locator's list of endpoints.

Response Time Collector

Overview

The Artix response time collector plug-in configures settings for Artix performance logging. The response time collector plug-in periodically collects data from the response monitor plug-in and logs the results. See Configuring and Deploying Artix Solutions for full details of Artix performance logging.

The response time collector plug-in includes the following variables:

- plugins:it_response_time_collector:client-id.
- plugins:it_response_time_collector:filename.
- plugins:it_response_time_collector:log_properties.
- plugins:it_response_time_collector:period.
- plugins:it_response_time_collector:server-id.
- plugins:it_response_time_collector:syslog_appID.
- plugins:it_response_time_collector:system_logging_enabled.

plugins:it_response_time_collector:client-id

plugins:it_response_time_collector:client-id specifies a client ID that is reported in your log messages. For example:

```
plugins:it_response_time_collector:client-id = "my_client_app";
```

This setting enables management tools to recognize log messages from client applications. This setting is optional; and if omitted, it is assumed that that a server is being monitored.

plugins:it_response_time_collector:filename

plugins:it_response_time_collector:filename specifies the location of the performance log file for a C++ application. For example:

```
plugins:it_response_time_collector:filename =
"/var/log/my_app/perf_logs/treasury_app.log";
```

plugins:it_response_time_collector:log_properties

`plugins:it_response_time_collector:log_properties` specifies the Apache Log4J details. Artix Java applications use Apache Log4J instead of the log filename used for C++. For example:

```
plugins:it_response_time_collector:log_properties = ["log4j.rootCategory=INFO, A1",
"log4j.appender.A1=com.iona.management.logging.log4jappender.TimeBasedRollingFileAppender",
"log4j.appender.A1.File="/var/log/my_app/perf_logs/treasury_app.log",
"log4j.appender.A1.MaxFileSize=512KB",
"log4j.appender.A1.layout=org.apache.log4j.PatternLayout",
"log4j.appender.A1.layout.ConversionPattern=%d{ISO8601} %-80m %n"
];
```

plugins:it_response_time_collector:period

`plugins:it_response_time_collector:period` specifies how often an application should log performance data. For example, the following setting specifies that an application should log performance data every 90 seconds:

```
plugins:it_response_time_collector:period = "90";
```

If you do not specify the response time period, it defaults to 60 seconds.

plugins:it_response_time_collector:server-id

`plugins:it_response_time_collector:server-id` specifies a server ID that will be reported in your log messages. This server ID is particularly useful in the case where the server is a replica that forms part of a cluster.

In a cluster, the server ID enables management tools to recognize log messages from different replica instances. For example:

```
plugins:it_response_time_collector:server-id = "my_server_app1";
```

This setting is optional; and if omitted, the server ID defaults to the ORB name of the server. In a cluster, each replica must have this value set to a unique value to enable sensible analysis of the generated performance logs.

plugins:it_response_time_collector:syslog_appID

`plugins:it_response_time_collector:syslog_appID` specifies an application name that is prepended to all syslog messages. If you do not specify an ID, it defaults to `iona`. For example:

```
plugins:it_response_time_collector:syslog_appID = "treasury";
```

plugins:it_response_time_collector:system_logging_enabled

`plugins:it_response_time_collector:system_logging_enabled` specifies whether system logging is enabled. For example:

```
plugins:it_response_time_collector:system_logging_enabled = "true";
```

This enables you to configure the collector to log to a syslog daemon or Windows event log.

Routing

Overview

The routing plug-in uses the following variables:

- plugins:routing:proxy_cache_size
- plugins:routing:reference_cache_size
- plugins:routing:wsdl_url
- plugins:routing:use_bypass
- plugins:routing:use_pass_through

plugins:routing:proxy_cache_size

plugins:routing:proxy_cache_size specifies the maximum number of proxified server references in the router. This is the number of references that have been converted into a proxy and are ready for invocation.

plugins:routing:proxy_cache_size works in conjunction with plugins:routing:reference_cache_size. Having a smaller setting for proxy_cache_size enables the router to conserve memory, while still being ready for invocations. This is because proxified references use more resources than unproxified references (for example, for client connections and bindings). The default setting is:

```
plugins:routing:proxy_cache_size="50";
```

The router caches references on a least recently used basis in the following order: proxified, unproxified. A proxified reference is demoted to an unproxified reference when the proxy_cache_size limit is reached. Unproxified references are promoted to proxies upon invocation.

For example, take a SOAP-HTTP client and CORBA server banking system with 1,500 accounts. By default, the 50 most recently used accounts are present in the router as proxified references. The next 1450 most recently used are unproxified references.

Note: Router proxification is available for the following bindings and transports: CORBA, SOAP, HTTP, and IIOP Tunnel.

plugins:routing:reference_cache_size

plugins:routing:reference_cache_size specifies the maximum number of unproxified server references in the router. This refers to the number of references that must be proxified before they can be invoked on.

plugins:routing:reference_cache_size works in conjunction with plugins:routing:proxy_cache_size. Having a larger setting for reference_cache_size enables the router to conserve memory, while still being ready for invocations. Unproxified references use less resources than proxies (for example, for client connections and bindings). The default setting is unbounded:

```
plugins:routing:reference_cache_size="-1";
```

The router caches transient references on a least recently used basis in the following order: proxified, unproxified. Unproxified references are promoted to proxies upon invocation. For an example, see plugins:routing:proxy_cache_size.

plugins:routing:wsdl_url

plugins:routing:wsdl_url specifies the URL to search for Artix contracts that contain the routing rules for your application. This value can point to WSDL in any location, it does not need to be on the local machine.

This value can be either a single URL or a list of URLs. If your application is using the routing plug-in, you must specify a value for this variable. The following example is from a default artix.cfg file:

```
plugins:routing:wsdl_url="../wsdl/router.wsdl";
```

The following example specifies multiple routes:

```
plugins:routing:wsdl=["route1.wsdl", "../route2.wsdl",
                      "/artix/routes/route3"];
```

Contract names must be relative to the location from which the Artix router is started. In this example, the router expects that `route1.wsdl` is located in the directory in which it was started, and `route2.wsdl` was located one directory level higher.

Note: This variable does not accept a mixture of back slashes and forward slashes. You must specify locations using only “`\`” or “`/`”.

plugins:routing:use_bypass

`plugins:routing:use_bypass` specifies a special optimization for CORBA-only routes. It enables you to use CORBA location forwarding to connect CORBA clients directly to CORBA servers, bypassing the Artix `routing` plug-in.

When the client sends the first request to the router, the router sends back a CORBA location forwarding reply, which tells the client to connect directly to the server at the end of the route. The client sends this and all subsequent requests directly to the server, bypassing the router completely. This feature is disabled by default. To enable bypass mode, use the following setting:

```
plugins:routing:use_bypass="true";
```

Routes that must examine the content of each request cannot support bypass mode because the requests do not go through the router. The following types of route support bypass mode:

- Straight source-destination routes.
- Failover: This is achieved by co-operation between CORBA and the router. If a server fails, the forwarded CORBA client automatically falls back to the original IOR, the router. The router then re-forwards the client to a healthy server.
- Load balancing: Load cannot be balanced per-operation using bypass. The router forwards each client to a different server, but when a client is forwarded all its requests go to the same server. If the server fails, the client is re-forwarded to the next healthy server in the round-robin, like failover.

plugins:routing:use_bypass and plugins:routing:use_pass_through can both be set together. Bypass is used for CORBA-only applications, while pass-through applies in all other cases. Bypass gives best performance because the router effectively disappears. However, pass-through may be preferable in the following cases:

- Bypass is disabled for per-operation, fan-out, and transport-attribute routes.
- Bypassed clients must be able to connect directly to the destination servers. Bypass is not suitable if the router is being used as part of a firewall, or as a connection concentrator.

plugins:routing:use_pass_through

plugins:routing:use_pass_through specifies whether the router receives a message and sends it directly to the destination without parsing. This only applies when the source and destination use the same binding.

The default is true. The router copies the message buffer directly from the source endpoint to the destination endpoint (if both use the same binding). This disables reference proxification for same-protocol routes (for example, HTTP-to-HTTP).

However, if you want all connections to go through the router, set this variable to false. This means that all references are used across the router.

Note: Some attributes are carried in the message body, instead of by the transport. Such attributes are always propagated when the pass-through optimization is in effect, regardless of attribute propagation rules.

WARNING: Do *not* enable pass-through in a secure router. When pass-through is enabled, the authentication and authorization steps are skipped. Therefore, you must always set plugins:routing:use_pass_through to false in a secure router. See IONA Security Advisory, ISA130905.

Service Lifecycle

Overview

The service lifecycle plug-in enables garbage collection of old or unused proxy services. Dynamic proxy services are used when the Artix router bridges services that have patterns such as callback, factory, or any interaction that passes references to other services. When the router encounters a reference in a message, it proxifies the reference into one that a receiving application can use. For example, an IOR from a CORBA server cannot be used by a SOAP client, so a new route is dynamically created for the SOAP client.

However, dynamic proxies persist in the router memory and can have a negative effect on performance. You can overcome this by using service garbage collection to clean up old proxy services that are no longer used. This cleans up unused proxies when a threshold has been reached on a least recently used basis.

The Artix plugins:service_lifecycle namespace has the following variable:

plugins:service_lifecycle:max_cache_size

plugins:service_lifecycle:max_cache_size

plugins:service_lifecycle:max_cache_size specifies the maximum cache size of servants managed by the service_lifecycle plug-in. For example:

```
plugins:service_lifecycle:max_cache_size = "30";
```

To enable service lifecycle, you must also add the service_lifecycle plug-in to the orb_plugins list, for example:

```
orb_plugins = ["xmlfile_log_stream", "service_lifecycle",
    "routing"];
```

When writing client applications, you must make allowances for the garbage collection service; in particular, ensure that exceptions are handled appropriately.

For example, a client may attempt to proxify to a service that has already been garbage collected. To prevent this, do either of the following:

- Handle the exception, get a new reference, and continue. However, in some cases, this may not be possible if the service has state.
- Set `max_cache_size` to a reasonable limit to ensure that all your clients can be accommodated. For example, if you always expect to support `20` concurrent clients, each with a transient service session, you might wish to configure the `max_cache_size` to `30`.

You must not impact any clients, and ensure that a service is no longer needed when it is garbage collected. However, if you set `max_cache_size` too high, this may use up too much router memory and have a negative impact on performance. For example, a suggested range for this setting is `30-100`.

Note: For a more scalable approach to managing proxies, see `plugins:routing:proxy_cache_size` and `plugins:routing:reference_cache_size`. This uses a single default servant (instead of the multiple servants used by service lifecycle), thereby minimizing the impact on router resources.

Session Manager

Overview

The session manager, session_manager_service, is configured by the following variable:

- plugins:session_manager_service:peer_timeout

plugins:session_manager_service:peer_timeout

plugins:session_manager_service:peer_timeout specifies the amount of time, in milliseconds, that the session manager plug-in waits between keep-alive pings of the endpoints registered with it. The default and minimum setting is 10000 milliseconds (10 seconds).

The session manager uses a third-party peer manager to ping its endpoints For more details, see "Peer Manager" on page 82.

Session Endpoint Manager

Overview

The session endpoint manager plug-in, `session_endpoint_manager`, is configured by the following variables:

- `plugins:session_endpoint_manager:default_group`
- `plugins:session_endpoint_manager:header_validation`
- `plugins:session_endpoint_manager:peer_timeout`

plugins:session_endpoint_manager:default_group

`plugins:session_endpoint_manager:default_group` specifies the default group name for all endpoints that are instantiated using the configuration scope.

plugins:session_endpoint_manager:header_validation

`plugins:session_endpoint_manager:header_validation` specifies whether or not a server validates the session headers passed to it by clients. Default value is `true`.

plugins:session_endpoint_manager:peer_timeout

`plugins:session_endpoint_manager:peer_timeout` specifies the amount of time, in milliseconds, the session endpoint manager plug-in waits between keep-alive pings back to the session manager. The default and minimum setting is `10000` milliseconds (10 seconds).

The session endpoint manager uses a third-party peer manager to ping back to the session manager. For more details, see "Peer Manager" on page 82.

Session Manager Simple Policy

Overview

The session manager's simple policy plug-in, sm_simple_policy, is configured by the following variables:

- plugins:sm_simple_policy:max_concurrent_sessions
- plugins:sm_simple_policy:min_session_timeout
- plugins:sm_simple_policy:max_session_timeout

plugins:sm_simple_policy:max_concurrent_sessions

plugins:sm_simple_policy:max_concurrent_sessions specifies the maximum number of concurrent sessions the session manager will allocate. Default value is 1.

plugins:sm_simple_policy:min_session_timeout

plugins:sm_simple_policy:min_session_timeout specifies the minimum amount of time, in seconds, allowed for a session's timeout setting. Zero means the unlimited. Default is 5.

plugins:sm_simple_policy:max_session_timeout

plugins:sm_simple_policy:max_session_timeout specifies the maximum amount of time, in seconds, allowed for a session's timesout setting. Zero means the unlimited. Default is 600.

SOAP

Overview

The soap plug-in includes the following configuration settings:

- plugins:soap:encoding
- plugins:soap:write_xsi_type

plugins:soap:encoding

plugins:soap:encoding specifies the character encoding used when the SOAP plug-in writes service requests or notification broadcasts to the wire. The valid settings are fully qualified IANA codeset names (Internet Assigned Numbers Authority). The default value is UTF-8. By default, this variable is not listed in the artix.cfg file.

For a listing of valid codesets visit the IANA's website (http://www.iana.org/assignments/character-sets).

plugins:soap:write_xsi_type

plugins:soap:write_xsi_type specifies whether to write the types of message parts in the log file. When set to true, this identifies each of the types associated with the message parts in the log file.

This only affects the content of the log file, giving you more information on the type contained in each message part. This variable for very useful for debugging purposes.

Transformer Service

Overview

The Artix transformer service uses Artix endpoints that are configured in its configuration scope using the `artix:endpoint:endpoint_list`. For each endpoint that uses the transformer, you must specify an operation map with the corresponding *`endpoint_name`* from the endpoint list. The `artix:endpoint` namespace contains the following variables:

- `artix:endpoint:endpoint_list`
- `artix:endpoint:endpoint_name:wsdl_location`
- `artix:endpoint:endpoint_name:wsdl_port`

The transformer service, `xslt`, has the following configuration settings:

- `plugins:xslt:servant_list`
- `plugins:xslt:endpoint_name:operation_map`

artix:endpoint:endpoint_list

`artix:endpoint:endpoint_list` specifies a list of endpoint names that are used to identify the defined endpoints. Each name in the list represents an endpoint configured with the other variables in this namespace. The endpoint names in this list are used by the Web service chain plug-in and the Artix transformer. For example:

```
artix:endpoint:endpoint_list = ["corba", "tunnel"];
```

artix:endpoint:*endpoint_name*:wsdl_location

`artix:endpoint:`*`endpoint_name`*`:wsdl_location` specifies the location of the Artix contract defining this endpoint. For example:

```
artix:endpoint:corba:wsdl_location="C:\myDir/test/wsdl/simple_service.wsdl";
```

artix:endpoint:*endpoint_name*:wsdl_port

`artix:endpoint:`*`endpoint_name`*`:wsdl_port` specifes the `port` that defines the physical representation of the endpoint. Use the following format:

```
[{service_qname}]service_name[/port_name]
```

For example:

```
artix:endpoint:my_endpoint:wsdl_port="{http://www.mycorp.com/}MyService/MyPort";
```

plugins:xslt:servant_list

`plugins:xslt:servant_list` specifies a list of endpoints that are instaniated as servants by the transformer. For example:

```
plugins:xslt:servant_list=["endpoint_one", "endpoint_two" ...]
```

plugins:xslt:*endpoint_name*:operation_map

`plugins:xslt:`*`endpoint_name`*`:operation_map` specifies a list of XSLT operations and scripts to be used in processing the recieved XML messages. This list of scripts is used by each servant to process requests. Each endpoint specified in the servant list has a corresponding operation map entry. The operation map is specified as a list using the syntax .

```
plugins:xslt:endpoint_name:operantion_map = ["wsdlOp1@filename1"
    , "wsdlOp2@filename2", ..., "wsdlOpN@filenameN"];
```

Each entry specifies a logical operation defined in the service contract by an `operation` element, and the XSLT script to run when a request is made on the operation. You must specify an XSLT script for every operation defined. If you do not, the transformer raises an exception when the unmapped operation is invoked.

plugins:xslt:*endpoint_name*:trace_filter

plugins:xslt:*endpoint_name*:trace_filter specifies optional debug settings for the output of the XSLT engine. For example:

```
plugins:xslt:endpoint_name:trace_filter =
    "INPUT+TEMPLATE+ELEMENT+GENERATE+SELECT";
```

These settings are described as follows:

INPUT	Traces the XML input passed to the XSLT engine.
TEMPLATE	Traces template matches in the XSLT script.
ELEMENT	Traces element generation.
GENERATE	Traces generation of text and attributes.
SELECT	Traces node selections in the XSLT script.

Tuxedo

Overview

The Tuxedo plug-in includes the following variable:

- plugins:tuxedo:server

plugins:tuxedo:server

plugins:tuxedo:server is a boolean that specifies if the Artix process is a Tuxedo server and must be started using tmboot. The default is:

```
plugins:tuxedo:server = "false";
```

Web Services Addressing

Overview

The plugins:messaging_port plug-in specifies variables that support WS-Addressing (WS-A) and WS-ReliableMessaging (WS-RM). These include:

- plugins:messaging_port:base_replyto_url
- plugins:messaging_port:supports_wsa_mep
- plugins:messaging_port:wsrm_enabled

See also Web Services Reliable Messaging.

plugins:messaging_port:base_replyto_url

plugins:messaging_port:base_replyto_url specifies a base URI for a WS-Addressing reply-to endpoint. The scope of a reply-to endpoint is at the proxy level, and in Artix, two proxies can not share the same endpoint. This means that each proxy has its own reply-to endpoint. For example, if the base URI is specified as:

```
plugins:messaging_port:base_replyto_url=
    "http://localhost:0/WSATestClient/BaseReplyTo/";
```

And if two proxies are instantiated, the first proxy will have a reply-to endpoint whose URI is as follows:

```
"http://localhost:2356/WSATestClient/BaseReplyTo/ReplyTo0001";
```

Similarly, the second proxy will have a reply-to endpoint whose URI is as follows:

```
"http://localhost:2356/WSATestClient/BaseReplyTo/ReplyTo0002";
```

The WS-A reply-to endpoint can be set at the Artix bus-level (like the earlier example) or at a WSDL port-level, for example:

```
plugins:messaging_port:base_replyto_url:http://www.iona.com/bus/
    tests:SOAPHTTPService:SOAPHTTPPort=
    "http://localhost:0/WSATestClient/BaseReplyTo/";
```

plugins:messaging_port:supports_wsa_mep

`plugins:messaging_port:supports_wsa_mep` specifies whether a WS-Addressing Message Exchange Pattern (MEP) is enabled. You can specify this setting either at the Artix bus-level or a specific WSDL port level. Port-specific configuration overrides bus-specific configuration.

Bus-specific configuration

To enable WS-A at bus level, use the following setting:

```
plugins:messaging_port:supports_wsa_mep = "true";
```

WSDL port-specific configuration

To enable WS-A at a specific WSDL port level, you must specify the WSDL service QName and the WSDL port name, for example:

```
plugins:messaging_port:supports_wsa_mep:http://www.iona.com/bus/
    tests:SOAPHTTPService:SOAPHTTPPort="true";
```

plugins:messaging_port:wsrm_enabled

`plugins:messaging_port:wsrm_enabled` specifies whether WS-RemoteAddressing is enabled. WS-RM can be enabled either at the bus-level or a specific WSDL port level. Port-specific configuration overrides bus-specific configuration.

Bus-specific configuration

To enable WS-RM for a specific bus, use the following setting:

```
plugins:messaging_port:wsrm_enabled = "true";
```

WSDL port-specific configuration

To enable WS-RM at a specific WSDL port level, specify the WSDL service QName and also the WSDL port name, for example:

```
plugins:messaging_port:wsrm_enabled:http://www.iona.com/bus/test
    s:SOAPHTTPService:SOAPHTTPPort="true";
```

Web Services Chain Service

Overview

The Web services chain service refers back to the Artix endpoints configured in its configuration scope using `artix:endpoint:endpoint_list`. For each endpoint that will be part of the chain, you specify a service chain with the corresponding `endpoint_name` from the endpoint list.

The Web service chain service, `ws_chain`, uses the following configuration variables:

- `plugins:chain:endpoint_name:operation_name:service_chain`
- `plugins:chain:init_on_first_call`
- `plugins:chain:servant_list`

plugins:chain:*endpoint_name*:*operation_name*:service_chain

`plugins:chain:endpoint_name:operation_name:service_chain` specifies the chain followed by requests made on the operation specified by `opereration_name`. The operation must be defined as part of the endpoint specified by `endpoint_name`.

Service chains are specified using the following syntax:

```
["operation1@port1","operation2@port2", ..., "operationN@portN"]
```

Each operation and port entry correspond to an `operation` and a `port` in the endpoint's Artix contract. The request is passed through each service in the order specified. The final operation in the list returns the response back to the endpoint.

plugins:chain:init_on_first_call

plugins:chain:init_on_first_call specifies whether to instantiate proxy services when a call is made. Defaults to false. This means that proxies are instantiated when the chain servant starts.

The chain invokes on other services, and for this reason, must instantiate proxies. This can be done when the chain servant starts (variable set to false), or later, when a call is made (variable set to true).

You might not be able to properly instantiate proxies when the servant is started because the servant to call is not started. For example, this applies when using the Artix locator or UDDI.

plugins:chain:servant_list

plugins:chain:servant_list specifies a list of services in the Web service chain. Each name in the list must correspond to a service specified in the configuration scope. The following simple example shows a list that contains one service:

```
bus:qname_alias:my_client =
    "{http://www.iona.com/xslt}my_client_service";
bus:initial_contract:url:client = "../../etc/my_transformation.wsdl";

...

plugins:chain:servant_list = ["my_client"];
```

Web Services Reliable Messaging

Overview

The plugins:wsrm plug-in specifies variables that support WS-ReliableMessaging (WS-RM). These include:

- plugins:wsrm:acknowledgement_interval
- plugins:wsrm:acknowledgement_uri
- plugins:wsrm:base_retransmission_interval
- plugins:wsrm:disable_exponential_backoff_retransmission_interval
- plugins:wsrm:max_messages_per_sequence
- plugins:wsrm:max_unacknowledged_messages_threshold

See also Web Services Addressing.

plugins:wsrm:acknowledgement_interval

plugins:wsrm:acknowledgement_interval specifies the interval at which the WS-RM destination sends asynchronous acknowledgements. This is in addition to the synchronous acknowledgements that are sent upon the receipt of an incoming message. The default value is 3000 milliseconds.

Bus-specific configuration

The following example shows how to set the acknowledgement interval for a specific bus

```
plugins:wsrm:acknowledgement_interval = "2500";
```

WSDL port-specific configuration

The following example shows how to set the acknowledgement interval for a specific WSDL port:

```
plugins:wsrm:acknowledgement_interva:http://www.iona.com/bus/tes
    ts:SOAPHTTPService:SOAPHTTPPort = "2500";
```

plugins:wsrm:acknowledgement_uri

`plugins:wsrm:acknowledgement_uri` specifies the endpoint at which the WS-RM source receives acknowledgements. This is also known as `wsrm:AcksTo`. The default value is the WS-A anonymous URI:

```
http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
```

Bus-specific configuration

The following example shows how to configure the acknowledgement endpoint URI for a specific bus:

```
plugins:wsrm:acknowledgement_uri =
    "http://localhost:0/WSASource/DemoAcksTo/";
```

WSDL port-specific configuration

The following example shows how to configure the acknowledgement endpoint URI for a specific WSDL port:

```
plugins:wsrm:acknowledgement_uri:http://www.iona.com/bus/tests:
SOAPHTTPService:SOAPHTTPPort =
    "http://localhost:0/WSASource/DemoAcksTo/";
```

plugins:wsrm:base_retransmission_interval

`plugins:wsrm:base_retransmission_interval` specifies the interval at which a WS-RM source retransmits a message that has not yet been acknowledged. The default value is `2000` milliseconds.

Bus-specific configuration

The following example shows how to set the base retransmission interval for a specific bus:

```
plugins:wsrm:base_retransmission_interval = "3000";
```

WSDL port-specific configuration

The following example shows how to set the base retransmission interval for a specific WSDL port:

```
plugins:wsrm:base_retransmission_interval:http://www.iona.com/bu
    s/tests:SOAPHTTPService:SOAPHTTPPort = "3000";
```

plugins:wsrm:disable_exponential_backoff_retransmission_interval

`plugins:wsrm:disable_exponential_backoff_retransmission_interval` determines if successive retransmission attempts for an unacknowledged message are performed at exponential intervals or not. The default value is `false`, which means that they are attempted at exponential intervals.

If the value is `true` (exponential backoff disabled), the retransmission of unacknowledged messages is performed at the base retransmission interval.

Bus-specific configuration

The following example shows how to set the exponential backoff for retransmission for a specific bus:

```
plugins:wsrm:disable_exponential_backoff_retransmission_interval
    = "true";
```

WSDL port-specific configuration

The following example shows how to set the exponential backoff for retransmission for a specific WSDL port:

```
plugins:wsrm:disable_exponential_backoff_retransmission_interval
    :http://www.iona.com/bus/tests:SOAPHTTPService:SOAPHTTPPort =
    "true";
```

plugins:wsrm:max_messages_per_sequence

`plugins:wsrm:max_messages_per_sequence` specifies the maximum number of user messages that are permitted in a WS-RM sequence. The default is unlimited; this is sufficient is for most situations.

When this attribute is set, the RM endpoint creates a new RM sequence when the limit is reached and after receiving all the acknowledgements for the messages previously sent. The new message is then sent using the new sequence.

Bus-specific configuration

The following example shows how to set the maximum number of messages for a specific bus

```
plugins:wsrm:max_messages_per_sequence = "1";
```

WSDL port-specific configuration

The following example shows how to set the maximum number of messages for a specific WSDL port:

```
plugins:wsrm:max_messages_per_sequence:http://www.iona.com/bus/t
    ests:SOAPHTTPService:SOAPHTTPPort = "1";
```

plugins:wsrm:max_unacknowledged_messages_threshold

`plugins:wsrm:max_unacknowledged_messages_threshold` specifies the maximum permissible number of unacknowledged messages at the WS-RM source. When the WS-RM source reaches this limit, it sends the last message with a `wsrm:AckRequested` header indicating that a WS-RM acknowledgement should be sent by the WS-RM destination as soon as possible.

In addition, when the WS-RM source has reached this limit, it does not accept further messages from the application source. This means that the caller thread (making the invocation on the proxy) is blocked until the number of unacknowledged messages drops below the threshold.

The default value is `-1` (no limit on number of unacknowledged messages).

Bus-specific configuration

The following example shows how to set the max unacknowledged messages threshold for a specific bus:

```
plugins:wsrm:max_unacknowledged_messages_threshold = "50";
```

WSDL port-specific configuration

The following example shows how to set the max unacknowledged messages threshold for a specific WSDL port:

```
plugins:wsrm:max_unacknowledged_messages_threshold:http://www.io
    na.com/bus/tests:SOAPHTTPService:SOAPHTTPPort = "50";
```

WSDL Publishing Service

Overview

The WSDL publishing service, wsdl_publish, includes the following configuration variables:

- plugins:wsdl_publish:publish_port
- plugins:wsdl_publish:hostname
- plugins:wsdl_publish:processor

Although all three variables are optional, it is recommended that you define plugins:wsdl_publish:publish_port and plugins:wsdl_publish:hostname in production environments.

plugins:wsdl_publish:publish_port

plugins:wsdl_publish:publish_port specifies the port on which the WSDL publishing service can be contacted.

The default value is 0, which specifies that wsdl_publish will use a port supplied by the operating system at runtime. You can get the wsdl_publish URL from the bus.

plugins:wsdl_publish:hostname

`plugins:wsdl_publish:hostname` specifies how the hostname is constructed in the `wsdl_publish` URL. This is the URL that the `wsdl_publish` plug-in uses to retrieve WSDL contracts.

By default, the unqualified local hostname is used. The possible values are as follows:

`canonical`	The fully qualified hostname of the machine (for example `http://myhost.mydomain.com`).
`unqualified`	The unqualified local hostname of the machine. This does not include the domain name with the hostname (for example, `http://myhost`). This is the default.
`ipaddress`	The IP address associated with the machine (for example `http://10.1.2.3`).

Note: For details of how the address is published in dynamically generated WSDL contracts, see `policies:at_http:server_address_mode_policy:publish_hostname` and `policies:soap:server_address_mode_policy:publish_hostname`.

plugins:wsdl_publish:processor

`plugins:wsdl_publish:processor` specifies the type of preprocessing done before publishing a WSDL contract. The possible values are as follows:

`artix`	Strip out server-side artifacts. This is the default setting.
`standard`	Strip out server side artifacts and IONA proprietary extensors.
`none`	Disable preprocessing.

XML File Log Stream

Overview

The XML file log stream plug-in, `xmlfile_log_stream`, enables you to view logging output in an XML file. It includes the following variables:

- `plugins:xmlfile_log_stream:buffer_file`
- `plugins:xmlfile_log_stream:filename`
- `plugins:xmlfile_log_stream:filename_date_format`
- `plugins:xmlfile_log_stream:log_elements`
- `plugins:xmlfile_log_stream:log_thread_id`
- `plugins:xmlfile_log_stream:milliseconds_to_log`
- `plugins:xmlfile_log_stream:rolling_file`
- `plugins:xmlfile_log_stream:use_pid`

plugins:xmlfile_log_stream:buffer_file

`plugins:xmlfile_log_stream:buffer_file` specifies whether the output stream is sent to a buffer before it writes to a local log file. To specify this behavior, set this variable to `true`:

```
plugins:xmlfile_log_stream:buffer_file = "true";
```

When set to `true`, by default, the buffer is output to a file every 1000 milliseconds when there are more than 100 messages logged. This log interval and number of log elements can also be configured.

plugins:xmlfile_log_stream:filename

`plugins:xmlfile_log_stream:filename` specifies the filename for your log file, for example:

```
plugins:xmlfile_log_stream:filename = "artix_logfile.xml";
```

If you do not specify a file name, logging is sent to `stdout`.

plugins:xmlfile_log_stream:filename_date_format

`plugins:xmlfile_log_stream:filename_date_format` specifies the format of the date in an XML-based rolling log file. The specified date conforms to the format rules of the ANSI C `strftime()` function. For example:

```
plugins:xmlfile_log_stream:rolling_file="true";
plugins:xmlfile_log_stream:filename="my_log";
plugins:xmlfile_log_stream:filename_date_format="_%Y_%m_%d";
```

On the 31st January 2006, this results in a log file named `my_log_2006_01_31`.

plugins:xmlfile_log_stream:log_elements

`plugins:xmlfile_log_stream:log_elements` specifies the number of log messages that must be in the buffer before they are output to a log file. The default is 100 messages.

For example, the following configuration writes the log output to a log file if there are more than 20 log messages in the buffer.

```
plugins:xmlfile_log_stream:log_elements = "20";
```

plugins:xmlfile_log_stream:log_thread_id

`plugins:xmlfile_log_stream:log_thread_id` specifies whether the thread ID is logged in the log message or not, for example:

```
plugins:xmlfile_log_stream:log_thread_id = "true";
```

The default is `true`.

plugins:xmlfile_log_stream:milliseconds_to_log

`plugins:xmlfile_log_stream:milliseconds_to_log` specifies how often in milliseconds that the log buffer is output to a log file. The default is `1000` milliseconds.

For example, the following configuration writes the log output to a log file every 400 milliseconds.

```
plugins:xmlfile_log_stream:milliseconds_to_log = "400";
```

plugins:xmlfile_log_stream:rolling_file

`plugins:xmlfile_log_stream:rolling_file` is a boolean which specifies that the logging plug-in creates a new log file each day to prevent the log file from growing indefinitely. In this model, the stream appends the current date to the configured filename. This produces a complete filename, for example:

```
/var/adm/artix.log.02172005
```

A new file begins with the first event of the day and ends at 23:59:59 each day. The default behavior is `true`. To disable rolling file behavior, set this variable to false. For example:

```
plugins:xmlfile_log_stream:rolling_file = "false";
```

plugins:xmlfile_log_stream:use_pid

`plugins:xmlfile_log_stream:use_pid` specifies that the logging plug-in uses a optional process identifier. The default is `false`. To enable the process identifier, set this variable to `true`:

```
plugins:xmlfile_log_stream:use_pid = "true";
```

Custom Plug-ins

Overview

When you write a custom plug-in for Artix, using either C++ or Java, you must provide some configuration to the Artix runtime so that Artix can locate the libraries and initial settings required to properly instantiate the plug-in. This information is provided in the Artix configuration file used by your application. Typically, you would place the information in the global scope so that more than one of your applications can use the plug-in.

C++ plug-in libraries

When writing custom C++ plug-ins, you build your plug-in as a shared library that the bus loads at runtime. In the Artix configuration file, you need to provide the name of the shared library that loads the plug-in. You can do this using the following configuration variable:

```
plugins:PluginName:shlib_name
```

The plug-in name provided must correspond to the plug-in name that is listed in the `orb_plugins` list.

Example 3 shows an example of configuring a custom plug-in called `my_filter` that is implemented by the shared library `my_filter.dll`.

Example 3: *Custom C++ Plug-in Configuration*

```
plugins:my_filter:shlib_name="my_filter"
...
my_app
{
  orb_plugins=["my_filter" ...];
  ...
}
```

Java plug-in classes

Java plug-ins are loaded using the plug-in factory that you implemented for the custom plug-in. In an Artix configuration file, you must provide that name for the plug-in factory class. You can do this using the following configuration variable:

`plugins:`*`PluginName`*`:Classname`

The plug-in name provided must correspond to the plug-in name listed in the `orb_plugins` list. Example 4 shows an example of configuring a custom plug-in called `my_java_filter` that has the factory class `myJavaFilterFactory`.

Example 4: *Custom Java Plug-in Configuration*

```
plugins:my_java_filter:Classname="myJavaFilterFactory"
...
my_app
{
  orb_plugins=[ ..., "java"];
  java_plugins=["my_java_filter"];
  ...
}
```

Specifying a classloading environment

If you want a custom plug-in to use an Artix classloader environment, specify the `plugins:`*`PluginName`*`:CE_Name` variable. The classloader environment name is specified as a unique string.

You must also use the `ce:`*`CE_Name`*`:FileName` variable to specify the location of the XML file that describes the classloader environment. *`CE_Name`* is the classloader environment name used when configuring the plug-in.

The following example shows the configuration for loading a custom plug-in using a classloader environment.

```
plugins:my_app:CE_Name="my_app_ce";
ce:my_app_ce:FileName="\artix_ces\my_app_ce.xml";
```

For more details, see *Developing Artix Applications in Java*.

Prerequisite plug-ins

In addition to providing a pointer to the plug-in's implementation, you can also provide a list of plug-ins that your plug-in requires to be loaded. You can provide this information using the following configuration variable:

`plugins:`*`PluginName`*`:prerequisite_plugins`.

The prerequisite plug-ins are specified as a list of plug-in names similar to that specified in the `orb_plugins` list. When you provide this list the bus ensures that the required plug-ins are loaded whenever your plug-in is loaded.

Example 5 shows configuring some prerequisite plug-ins for a custom plug-in called `my_filter`.

Example 5: *Custom Prerequisite Plug-in Configuration*

```
plugins:my_filter:prerequisite_plugins = ["my_plugin_1",
    "my_plugin_2", "my_plugin_3", "my_plugin4"];
```

The syntax is the same for Java and C++ applications.

Artix Security

This appendix describes variables used by the IONA Security Framework. The Artix security infrastructure is highly configurable.

In this appendix

This appendix discusses the following topics:

Applying Constraints to Certificates

Certificate constraints policy

You can use the CertConstraintsPolicy to apply constraints to peer X.509 certificates by the default CertificateValidatorPolicy. These conditions are applied to the owner's distinguished name (DN) on the first certificate (peer certificate) of the received certificate chain. Distinguished names are made up of a number of distinct fields, the most common being Organization Unit (OU) and Common Name (CN).

Configuration variable

You can specify a list of constraints to be used by CertConstraintsPolicy through the policies:iiop_tls:certificate_constraints_policy or policies:certificate_constraints_policy configuration variables. For example:

```
policies:iiop_tls:certificate_constraints_policy =
   ["CN=Johnny*,OU=[unit1|IT_SSL],O=IONA,C=Ireland,ST=Dublin,L=Ea
   rth","CN=Paul*,OU=SSLTEAM,O=IONA,C=Ireland,ST=Dublin,L=Earth",
"CN=TheOmnipotentOne"];
```

Constraint language

These are the special characters and their meanings in the constraint list:

*	Matches any text. For example: an* matches ant and anger, but not aunt
[ ]	Grouping symbols.
|	Choice symbol. For example: OU=[unit1|IT_SSL] signifies that if the OU is unit1 or IT_SSL, the certificate is acceptable.
=, !=	Signify equality and inequality respectively.

Example

This is an example list of constraints:

```
policies:iiop_tls:certificate_constraints_policy = [
   "OU=[unit1|IT_SSL],CN=Steve*,L=Dublin",
"OU=IT_ART*,OU!=IT_ARTtesters,CN=[Jan|Donal],ST=
Boston" ];
```

This constraint list specifies that a certificate is deemed acceptable if and only if it satisfies one or more of the constraint patterns:

```
If
    The OU is unit1 or IT_SSL
    And
    The CN begins with the text Steve
    And
    The location is Dublin
Then the certificate is acceptable
Else  (moving on to the second constraint)
If
    The OU begins with the text IT_ART but isn't IT_ARTtesters
    And
    The common name is either Donal or Jan
    And
    The State is Boston
Then the certificate is acceptable
Otherwise the certificate is unacceptable.
```

The language is like a boolean OR, trying the constraints defined in each line until the certificate satisfies one of the constraints. Only if the certificate fails all constraints is the certificate deemed invalid.

Note that this setting can be sensitive about white space used within it. For example, "`CN =`" might not be recognized, where "`CN=`" is recognized.

Distinguished names

For more information on distinguished names, see the *Security Guide*.

bus:initial_contract

The bus:initial_contract namespace contains the following configuration variable:

- url:isf_service

url:isf_service

Specifies the location of the Artix security service's WSDL contract. This variable is needed by applications that connect to the Artix security service through a protocol specified in the physical part of the security service's WSDL contract (the alternative would be to connect over IIOP/TLS using a CORBA object reference).

This variable is used in conjunction with the policies:asp:use_artix_proxies configuration variable.

bus:security

The variables in the `bus:security` are intended for use with the `it_container_admin` utility, in order to facilitate communication with a secure Artix container. The `bus:security` namespace contains the following configuration variables:

- enable_security
- user_name
- user_password

enable_security

The `bus:security:enable_security` variable is a boolean variable that enables a client to send WSSE username and password credentials. When `true`, the client sends WSSE username and password credentials with every SOAP request message (whether or not the connection is secured by SSL/TLS); when `false`, the feature is disabled.

There are essentially two different ways of initializing the WSSE username and password credentials on the client side:

- *From the configuration file*—you can set the WSSE credentials in the Artix configuration using the related `user_name` and `user_password` configuration variables. For example:

```
# Artix Configuration File
bus:security:enable_security = "true";
bus:security:user_name = "Username";
bus:security:user_password = "Password";
```

- *From the command line*—if you omit the `bus:security:user_name` and `bus:security:user_password` settings from the Artix configuration, the client program will prompt you for the username and password credentials as it starts up. For example:

```
Please enter login :
Please enter password :
```

user_name

Initializes a WSSE username. This variable is intended for use in conjunction with the `bus:security:enable_security` variable as part of the configuration for the `it_container_admin` utility.

user_password

Initializes a WSSE password. This variable is intended for use in conjunction with the `bus:security:enable_security` variable as part of the configuration for the `it_container_admin` utility.

initial_references

The `initial_references` namespace contains the following configuration variables:

- IT_SecurityService:reference
- IT_TLS_Toolkit:plugin

IT_SecurityService:reference

This configuration variable specifies the location of the Artix security service. Clients of the security service need this configuration setting in order to locate and connect to the security service through the IIOP/TLS protocol.

Note: This variable is *not* relevant to clients that connect to a HTTPS-based security service.

The most convenient way to initialize this variable is to use a `corbaloc` URL. The corbaloc URL typically has the following format:

```
corbaloc:it_iiops:1.2@Hostname:Port/IT_SecurityService
```

Where *`Hostname`* is the name of the host where the security service is running and *`Port`* is the IP port where the security service is listening for incoming connections.

If the security service is configured as a cluster, you need to use a multi-profile corbaloc URL, which lists the addresses of all the services in the cluster. For example, if you configure a cluster of three services—with addresses `security01:5001`, `security02:5002`, and `security03:5003`—you would set the corbaloc URL as follows:

```
corbaloc:it_iiops:1.2@security01:5001,it_iiops:1.2@security02:500
   2,it_iiops:1.2@security03:5003/IT_SecurityService
```

IT_TLS_Toolkit:plugin

This configuration variable enables you to specify the underlying SSL/TLS toolkit to be used by Artix. It is used in conjunction with the `plugins:baltimore_toolkit:shlib_name`, `plugins:schannel_toolkit:shlib_name` (Windows only) and `plugins:systemssl_toolkit:shlib_name` (z/OS only) configuration variables to implement SSL/TLS toolkit replaceability.

The default is the Baltimore toolkit.

For example, to specify that an application should use the Schannel SSL/TLS toolkit, you would set configuration variables as follows:

```
initial_references:IT_TLS_Toolkit:plugin = "schannel_toolkit";
plugins:schannel_toolkit:shlib_name = "it_tls_schannel";
```

password_retrieval_mechanism

The configuration variables in the `password_retrieval_mechanism` namespace are intended to be used *only* by the Artix services. The following variables are defined in this namespace:

- inherit_from_parent
- use_my_password_as_kdm_password

inherit_from_parent

If an application forks a child process and this variable is set to `true`, the child process inherits the parent's X.509 certificate password through the environment.

Note: This variable is intended for use *only* by the standard Artix services.

use_my_password_as_kdm_password

This variable should be set to `true` only in the scope of the KDM plug-in's container. From a security perspective it is dangerous to do otherwise as the password could be left in cleartext within the process.

The KDM is a locator plug-in and so it is natural that it should use the locator's identity as its identity. However, it requires a password to encrypt its security information. By default the KDM requests such a password from the user during locator startup and this is separate from the locator password. The locator password would be used if this variable is set to `true`.

Note: This variable is intended for use *only* by the standard Artix services.

plugins:asp

The plugins:asp namespace contains the following variables:

- authentication_cache_size
- authentication_cache_timeout
- authorization_realm
- default_password
- enable_security_service_cert_authentication
- enable_security_service_load_balancing
- security_type
- security_level

authentication_cache_size

The maximum number of credentials stored in the authentication cache. If this size is exceeded, any new authentication tokens acquired by calling the Artix security service are *not* stored in the cache. The cache can shrink again if some of the cached credentials expire (either because the individual token expiry time is exceeded or the plugins:asp:authentication_cache_timeout is exceeded).

A value of -1 (the default) means unlimited size. A value of 0 means disable the cache. The value must lie within the range -1 to 2^31-1.

Note: This variable does not affect CORBA credentials. For details of how to configure the CORBA cache, see "plugins:gsp" on page 137.

authentication_cache_timeout

The time (in seconds) after which a credential expires. Expired credentials are removed from the cache and must re-authenticate with the Artix security service on the next call from that user.

A value of -1 means an infinite time-out. A value of 0 means disable the cache. The value must lie within the range -1 to 2^31-1.

Default is 500 seconds.

Note: This variable does not affect CORBA credentials. For details of how to configure the CORBA cache, see "plugins:gsp" on page 137.

authorization_realm

Specifies the Artix authorization realm to which an Artix server belongs. The value of this variable determines which of a user's roles are considered when making an access control decision.

For example, consider a user that belongs to the ejb-developer and corba-developer roles within the Engineering realm, and to the ordinary role within the Sales realm. If you set plugins:asp:authorization_realm to Sales for a particular server, only the ordinary role is considered when making access control decisions (using the action-role mapping file).

The default is IONAGlobalRealm.

default_password

When the client credentials originate either from a CORBA Principal (embedded in a SOAP header) or from a certificate subject, the default_password variable specifies the password to use on the server side. The plugins:asp:default_password variable is used to get around the limitation that a PRINCIPAL identity and a CERT_SUBJECT are propagated without an accompanying password.

The artix_security plug-in uses the received client principal together with the password specified by plugins:asp:default_password to authenticate the user through the Artix security service.

The default value is the string, `default_password`.

enable_security_service_cert_authentication

When this parameter is set to `true`, the client certificate is retrieved from the TLS connection. If no other credentials are available, the client certificate is then sent to the Artix security service for authentication.

The client certificate has the lowest precedence for authentication. Hence, if any other credentials are presented by the client (for example, if the client sends a WSSE username and password), these alternative credentials are sent to the Artix security service instead of the certificate credentials.

Default is `false`.

enable_security_service_load_balancing

A boolean variable that enables load balancing over a cluster of security services. If an application is deployed in a domain that uses security service clustering, the application should be configured to use *client load balancing* (in this context, *client* means a client of the Artix security service). See also `policies:iiop_tls:load_balancing_mechanism`.

Default is `false`.

security_type

(Obsolete) From Artix 3.0 onwards, this variable is ignored.

security_level

Specifies the level from which security credentials are picked up. The following options are supported by the `artix_security` plug-in:

`MESSAGE_LEVEL`	Get security information from the transport header. This is the default.
`REQUEST_LEVEL`	Get the security information from the message header.

plugins:at_http

The plugins:at_http configuration variables are provided to facilitate migration from legacy Artix applications (that is, Artix releases prior to version 3.0). The plugins:at_http namespace contains variables that are similar to the variables from the old (pre-version 3.0) plugins:http namespace. One important change made in 3.0, however, is that an application's own certificate must now be provided in PKCS#12 format (where they were previously supplied in PEM format).

If the variables from the plugins:at_http namespace are used, they take precedence over the analogous variables from the principal_sponsor:https and policies:https namespaces.

The plugins:at_http namespace contains the following variables:

- client:client_certificate.
- client:client_private_key_password.
- client:trusted_root_certificates.
- client:use_secure_sockets.
- server:server_certificate.
- server:server_private_key_password.
- server:trusted_root_certificates.
- server:use_secure_sockets.

client:client_certificate

This variable specifies the full path to the PKCS#12-encoded X.509 certificate issued by the certificate authority for the client. For example:

```
plugins:at_http:client:client_certificate =
   "C:\aspen\x509\certs\key.cert.p12"
```

client:client_private_key_password

This variable specifies the password to decrypt the contents of the PKCS#12 certificate file specified by client:client_certificate.

client:trusted_root_certificates

This variable specifies the path to a file containing a concatenated list of CA certificates in PEM format. The client uses this CA list during the TLS handshake to verify that the server's certificate has been signed by a trusted CA.

client:use_secure_sockets

The effect of the `client:use_secure_sockets` variable depends on the type of URL specifying the remote service location:

- `https://`*`host`*`:`*`port`* URL format—the client always attempts to open a secure connection. That is, the value of `plugins:at_http:client:use_secure_sockets` is effectively ignored.
- `http://`*`host`*`:`*`port`* URL format—whether the client attempts to open a secure connection or not depends on the value of `plugins:at_http:client:use_secure_sockets`, as follows:
    - `true`—the client attempts to open a secure connection (that is, HTTPS running over SSL or TLS). If no port is specified in the `http` URL, the client uses port 443 for secure HTTPS.
    - `false`—the client attempts to open an insecure connection (that is, plain HTTP).

If `plugins:at_http:client:use_secure_sockets` is `true` and the client decides to open a secure connection, the `at_http` plug-in then automatically loads the `https` plug-in.

Note: If `plugins:at_http:client:use_secure_sockets` is `true` and the client decides to open a secure connection, Artix uses the following client secure invocation policies by default:

```
    policies:client_secure_invocation_policy:requires =
["Confidentiality","Integrity", "DetectReplay",
"DetectMisordering", "EstablishTrustInTarget"];
```

```
    policies:client_secure_invocation_policy:supports =
["Confidentiality", "Integrity", "DetectReplay",
"DetectMisordering", "EstablishTrustInTarget",
"EstablishTrustInClient"];
```

You can optionally override these defaults by setting the client secure invocation policy explicitly in configuration.

server:server_certificate

This variable specifies the full path to the PKCS#12-encoded X.509 certificate issued by the certificate authority for the server. For example:

```
plugins:at_http:server:server_certificate =
   "c:\aspen\x509\certs\key.cert.p12"
```

server:server_private_key_password

This variable specifies the password to decrypt the contents of the PKCS#12 certificate file specified by `server:server_certificate`.

server:trusted_root_certificates

This variable specifies the path to a file containing a concatenated list of CA certificates in PEM format. The server uses this CA list during the TLS handshake to verify that the client's certificate has been signed by a trusted CA.

server:use_secure_sockets

The effect of the `server:use_secure_sockets` variable depends on the type of URL advertising the service location:

- `https://`*`host`*`:`*`port`* URL format—the server accepts only secure connection attempts. That is, the value of `plugins:at_http:server:use_secure_sockets` is effectively ignored.
- `http://`*`host`*`:`*`port`* URL format—whether the server accepts secure connection attempts or not depends on the value of `plugins:at_http:server:use_secure_sockets`, as follows:
    - `true`—the server accepts secure connection attempts (that is, HTTPS running over SSL or TLS). If no port is specified in the `http` URL, the server uses port 443 for secure HTTPS.
    - `false`—the server accepts insecure connection attempts (that is, plain HTTP).

If `plugins:at_http:server:use_secure_sockets` is set and the server accepts a secure connection, the `at_http` plug-in then automatically loads the `https` plug-in.

Note: If `plugins:at_http:server:use_secure_sockets` is set and the server accepts a secure connection, Artix uses the following server secure invocation policies by default:

```
    policies:target_secure_invocation_policy:requires =
["Confidentiality","Integrity", "DetectReplay",
"DetectMisordering", "EstablishTrustInClient"];

    policies:target_secure_invocation_policy:supports =
["Confidentiality", "Integrity", "DetectReplay",
"DetectMisordering", "EstablishTrustInTarget",
"EstablishTrustInClient"];
```

You can optionally override these defaults by setting the target secure invocation policy explicitly in configuration.

server:use_secure_sockets:container

The effect of the `server:use_secure_sockets:container` variable is similar to the effect of the `server:use_secure_sockets` variable, except that only the `ContainerService` service is affected. Using this variable, it is possible to enable HTTPS security specifically for the `ContainerService` service without affecting the security settings of other services deployed in the container.

plugins:atli2_tls

The plugins:atli2_tls namespace contains the following variable:

- use_jsse_tk

use_jsse_tk

(Java only) Specifies whether or not to use the JSSE/JCE architecture with the CORBA binding. If true, the CORBA binding uses the JSSE/JCE architecture to implement SSL/TLS security; if false, the CORBA binding uses the Baltimore SSL/TLS toolkit.

The default is false.

plugins:csi

The `policies:csi` namespace includes variables that specify settings for Common Secure Interoperability version 2 (CSIv2):

- `ClassName`
- `shlib_name`

ClassName

`ClassName` specifies the Java class that implements the `csi` plugin. The default setting is:

```
plugins:csi:ClassName = "com.iona.corba.security.csi.CSIPlugin";
```

This configuration setting makes it possible for the Artix core to load the plugin on demand. Internally, the Artix core uses a Java class loader to load and instantiate the `csi` class. Plugin loading can be initiated either by including the `csi` in the `orb_plugins` list, or by associating the plugin with an initial reference.

shlib_name

`shlib_name` identifies the shared library (or DLL in Windows) containing the `csi` plugin implementation.

```
plugins:csi:shlib_name = "it_csi_prot";
```

The `csi` plug-in becomes associated with the `it_csi_prot` shared library, where `it_csi_prot` is the base name of the library. The library base name, `it_csi_prot`, is expanded in a platform-dependent manner to obtain the full name of the library file.

plugins:gsp

The plugins:gsp namespace includes variables that specify settings for the Generic Security Plugin (GSP). This provides authorization by checking a user's roles against the permissions stored in an action-role mapping file. It includes the following:

- accept_asserted_authorization_info
- action_role_mapping_file
- assert_authorization_info
- authentication_cache_size
- authentication_cache_timeout
- authorization_realm
- ClassName
- enable_authorization
- enable_gssup_sso
- enable_user_id_logging
- enable_x509_sso
- enforce_secure_comms_to_sso_server
- enable_security_service_cert_authentication
- sso_server_certificate_constraints
- use_client_load_balancing

accept_asserted_authorization_info

If false, SAML data is not read from incoming connections. Default is true.

action_role_mapping_file

Specifies the action-role mapping file URL. For example:

```
plugins:gsp:action_role_mapping_file =
   "file:///my/action/role/mapping";
```

assert_authorization_info

If false, SAML data is not sent on outgoing connections. Default is true.

authentication_cache_size

The maximum number of credentials stored in the authentication cache. If this size is exceeded the oldest credential in the cache is removed.

A value of -1 (the default) means unlimited size. A value of 0 means disable the cache.

authentication_cache_timeout

The time (in seconds) after which a credential is considered *stale*. Stale credentials are removed from the cache and the server must re-authenticate with the Artix security service on the next call from that user. The cache timeout should be configured to be smaller than the timeout set in the is2.properties file (by default, that setting is is2.sso.session.timeout=600).

A value of -1 (the default) means an infinite time-out. A value of 0 means disable the cache.

authorization_realm

authorization_realm specifies the iSF authorization realm to which a server belongs. The value of this variable determines which of a user's roles are considered when making an access control decision.

For example, consider a user that belongs to the ejb-developer and corba-developer roles within the Engineering realm, and to the ordinary role within the Sales realm. If you set plugins:gsp:authorization_realm to Sales for a particular server, only the ordinary role is considered when making access control decisions (using the action-role mapping file).

ClassName

ClassName specifies the Java class that implements the gsp plugin. This configuration setting makes it possible for the Artix core to load the plugin on demand. Internally, the Artix core uses a Java class loader to load and instantiate the gsp class. Plugin loading can be initiated either by including the csi in the orb_plugins list, or by associating the plugin with an initial reference.

enable_authorization

A boolean GSP policy that, when true, enables authorization using action-role mapping ACLs in server.

Default is true.

enable_gssup_sso

Enables SSO with a username and a password (that is, GSSUP) when set to true.

enable_user_id_logging

A boolean variable that enables logging of user IDs on the server side. Default is false.

Up until the release of Orbix 6.1 SP1, the GSP plug-in would log messages containing user IDs. For example:

```
[junit] Fri, 28 May 2004 12:17:22.0000000 [SLEEPY:3284]
   (IT_CSI:205) I - User alice authenticated successfully.
```

In some cases, however, it might not be appropriate to expose user IDs in the Orbix log. From Orbix 6.2 onward, the default behavior of the GSP plug-in is changed, so that user IDs are *not* logged by default. To restore the pre-Orbix 6.2 behavior and log user IDs, set this variable to true.

enable_x509_sso

Enables certificate-based SSO when set to `true`.

enforce_secure_comms_to_sso_server

Enforces a secure SSL/TLS link between a client and the login service when set to `true`. When this setting is true, the value of the SSL/TLS client secure invocation policy does *not* affect the connection between the client and the login service.

Default is `true`.

enable_security_service_cert_authentication

A boolean GSP policy that enables X.509 certificate-based authentication on the server side using the Artix security service.

Default is `false`.

sso_server_certificate_constraints

A special certificate constraints policy that applies *only* to the SSL/TLS connection between the client and the SSO login server. For details of the pattern constraint language, see "Applying Constraints to Certificates" on page 119.

use_client_load_balancing

A boolean variable that enables load balancing over a cluster of security services. If an application is deployed in a domain that uses security service clustering, the application should be configured to use *client load balancing* (in this context, *client* means a client of the Artix security service). See also `policies:iiop_tls:load_balancing_mechanism`.

Default is `true`.

plugins:http

The `plugins:http` namespace contains the following variables:

- `client:client_certificate`
- `client:client_certificate_chain`
- `client:client_private_key`
- `client:client_private_key_password`
- `client:trusted_root_certificates`
- `client:use_secure_sockets`
- `server:server_certificate`
- `server:server_certificate_chain`
- `server:server_private_key`
- `server:server_private_key_password`
- `server:trusted_root_certificates`
- `server:use_secure_sockets`

client:client_certificate

This variable specifies the full path to the PEM-encoded X.509 certificate issued by the certificate authority for the client. For example:

```
plugins:http:client:client_certificate =
   "c:\aspen\x509\certs\key.cert.pem"
```

This setting is ignored if `plugins:http:client:use_secure_sockets` is `false`.

client:client_certificate_chain

(Optional) This variable specifies the full path to the PEM-encoded X.509 certificate chain for the client. For example:

```
plugins:http:client:client_certificate_chain =
   "c:\aspen\x509\certs\key.cert.pem"
```

This setting is ignored if `plugins:http:client:use_secure_sockets` is `false`.

client:client_private_key

This variable specifies a PEM file containing the client certificate's encrypted private key. This private key enables the client to respond to a challenge from a server during an SSL/TLS handshake.

This setting is ignored if `plugins:http:client:use_secure_sockets` is `false`.

client:client_private_key_password

This variable specifies the password to decrypt the contents of the `client_private_key` file.

This setting is ignored if `plugins:http:client:use_secure_sockets` is `false`.

client:trusted_root_certificates

This variable specifies the path to a file containing a concatenated list of CA certificates in PEM format. The client uses this CA list during the TLS handshake to verify that the server's certificate has been signed by a trusted CA.

This setting is ignored if `plugins:http:client:use_secure_sockets` is `false`.

client:use_secure_sockets

This variable specifies whether the client wants to open a HTTPS connection (that is, HTTP running over SSL or TLS) or an insecure connection (that is, plain HTTP).

Valid values are `true`, for HTTPS, and `false`, for HTTP. The default is `false`.

server:server_certificate

This variable specifies the full path to the PEM-encoded X.509 certificate issued by the certificate authority for the server. For example:

```
plugins:http:server:server_certificate =
   "c:\aspen\x509\certs\key.cert.pem"
```

This setting is ignored if `plugins:http:server:use_secure_sockets` is `false`.

server:server_certificate_chain

(Optional) This variable specifies the full path to the PEM-encoded X.509 certificate chain for the server. For example:

```
plugins:http:server:server_certificate_chain =
   "c:\aspen\x509\certs\key.cert.pem"
```

This setting is ignored if `plugins:http:server:use_secure_sockets` is `false`.

server:server_private_key

This variable specifies a PEM file containing the server certificate's encrypted private key. This private key enables the server to respond to a challenge from a client during an SSL/TLS handshake.

This setting is ignored if `plugins:http:server:use_secure_sockets` is `false`.

server:server_private_key_password

This variable specifies the password to decrypt the contents of the `server_private_key` file.

This setting is ignored if `plugins:http:server:use_secure_sockets` is `false`.

server:trusted_root_certificates

This variable specifies the path to a file containing a concatenated list of CA certificates in PEM format. The server uses this CA list during the TLS handshake to verify that the client's certificate has been signed by a trusted CA.

This setting is ignored if `plugins:http:server:use_secure_sockets` is `false`.

server:use_secure_sockets

This variable specifies whether the server accepts HTTPS connection attempts (that is, HTTP running over SSL or TLS) or insecure connection attempts (that is, plain HTTP) from a client.

Valid values are `true`, for HTTPS, and `false`, for HTTP. The default is `false`.

plugins:https

The plugins:https namespace contains the following variable:

- ClassName

ClassName

(Java only) This variable specifies the class name of the https plug-in implementation. For example:

```
plugins:https:ClassName = "com.iona.corba.https.HTTPSPlugIn";
```

plugins:iiop_tls

The `plugins:iiop_tls` namespace contains the following variables:

- buffer_pool:recycle_segments
- buffer_pool:segment_preallocation
- buffer_pools:max_incoming_buffers_in_pool
- buffer_pools:max_outgoing_buffers_in_pool
- delay_credential_gathering_until_handshake
- enable_iiop_1_0_client_support
- incoming_connections:hard_limit
- incoming_connections:soft_limit
- outgoing_connections:hard_limit
- outgoing_connections:soft_limit
- tcp_listener:reincarnate_attempts
- tcp_listener:reincarnation_retry_backoff_ratio
- tcp_listener:reincarnation_retry_delay

buffer_pool:recycle_segments

(Java only) When this variable is set, the `iiop_tls` plug-in reads this variable's value instead of the `plugins:iiop:buffer_pool:recycle_segments` variable's value.

buffer_pool:segment_preallocation

(Java only) When this variable is set, the `iiop_tls` plug-in reads this variable's value instead of the `plugins:iiop:buffer_pool:segment_preallocation` variable's value.

buffer_pools:max_incoming_buffers_in_pool

(C++ only) When this variable is set, the `iiop_tls` plug-in reads this variable's value instead of the `plugins:iiop:buffer_pools:max_incoming_buffers_in_pool` variable's value.

buffer_pools:max_outgoing_buffers_in_pool

(C++ only) When this variable is set, the `iiop_tls` plug-in reads this variable's value instead of the `plugins:iiop:buffer_pools:max_outgoing_buffers_in_pool` variable's value.

delay_credential_gathering_until_handshake

(Windows and Schannel only) This client configuration variable provides an alternative to using the `principal_sponsor` variables to specify an application's own certificate. When this variable is set to `true` and `principal_sponsor:use_principal_sponsor` is set to `false`, the client delays sending its certificate to a server. The client will wait until the server *explicitly* requests the client to send its credentials during the SSL/TLS handshake.

This configuration variable can be used in conjunction with the `plugins:schannel:prompt_with_credential_choice` configuration variable.

enable_iiop_1_0_client_support

This variable enables client-side interoperability of Artix SSL/TLS applications with legacy IIOP 1.0 SSL/TLS servers, which do not support IIOP 1.1.

The default value is `false`. When set to `true`, Artix SSL/TLS searches secure target IIOP 1.0 object references for legacy IIOP 1.0 SSL/TLS tagged component data, and attempts to connect on the specified port.

Note: This variable will not be necessary for most users.

incoming_connections:hard_limit

Specifies the maximum number of incoming (server-side) connections permitted to IIOP. IIOP does not accept new connections above this limit. Defaults to -1 (disabled).

When this variable is set, the `iiop_tls` plug-in reads this variable's value instead of the `plugins:iiop:incoming_connections:hard_limit` variable's value.

Please see the chapter on ACM in the *CORBA Programmer's Guide* for further details.

incoming_connections:soft_limit

Specifies the number of connections at which IIOP should begin closing incoming (server-side) connections. Defaults to -1 (disabled).

When this variable is set, the `iiop_tls` plug-in reads this variable's value instead of the `plugins:iiop:incoming_connections:soft_limit` variable's value.

Please see the chapter on ACM in the *CORBA Programmer's Guide* for further details.

outgoing_connections:hard_limit

When this variable is set, the `iiop_tls` plug-in reads this variable's value instead of the `plugins:iiop:outgoing_connections:hard_limit` variable's value.

outgoing_connections:soft_limit

When this variable is set, the `iiop_tls` plug-in reads this variable's value instead of the `plugins:iiop:outgoing_connections:soft_limit` variable's value.

tcp_listener:reincarnate_attempts

(Windows only)

`plugins:iiop_tls:tcp_listener:reincarnate_attempts` specifies the number of times that a Listener recreates its listener socket after recieving a SocketException.

Sometimes a network error may occur, which results in a listening socket being closed. On Windows, you can configure the listener to attempt a reincarnation, which enables new connections to be established. This variable only affects Java and C++ applications on Windows. Defaults to 0 (no attempts).

tcp_listener:reincarnation_retry_backoff_ratio

(Windows only)

`plugins:iiop_tls:tcp_listener:reincarnation_retry_delay` specifies a delay between reincarnation attempts. Data type is `long`. Defaults to `0` (no delay).

tcp_listener:reincarnation_retry_delay

(Windows only)

`plugins:iiop_tls:tcp_listener:reincarnation_retry_backoff_ratio`sp ecifies the degree to which delays between retries increase from one retry to the next. Datatype is `long`. Defaults to `1`.

plugins:java_server

In the context of Artix security, the variables in the plugins:java_server namespace are used only to configure the Artix security service. To deploy the security service, Artix exploits IONA's *generic server* (which is a feature originally developed for Orbix). The Artix security service is deployed into the following container hierarchy:

- *Generic server*—a simple container, originally developed for the Orbix product, which enables you to deploy CORBA services implemented in C++.
- *Java server plug-in*—a JNI-based adapter that plugs into the generic server, enabling you to deploy CORBA services implemented in Java.
- *JVM created by the Java server plug-in*—once it is loaded, the Java server plug-in creates a JVM instance to host a Java program.
- *Artix security service Java code*—you instruct the Java server plug-in to load the security service core (which is implemented in Java) by specifying the appropriate class to the plugins:java_server:class variable.

In addition to the configuration variables described in this section, you must also include the following setting in your configuration:

```
generic_server_plugin = "java_server";
```

Which instructs the generic server to load the Java server plug-in.

The plugins:java_server namespace contains the following variables:

- class
- classpath
- jni_verbose
- shlib_name
- system_properties
- X_options

class

In the context of the Artix security service, this variable specifies the entry point to the core security service (the core security service is a pure Java program). There are two possible values:

- com.iona.jbus.security.services.SecurityServer—creates an Artix bus instance that takes its configuration from the bus sub-scope of the current configuration scope. This entry point is suitable for a security service that is accessed through a WSDL contract (for example, a HTTPS-based security service).
- com.iona.corba.security.services.SecurityServer—a CORBA-based implementation of the security service, which does *not* create an Artix bus instance. This entry point is suitable for running an IIOP/TLS-based security service.

classpath

Specifies the CLASSPATH for the JVM instance created by the Java server plug-in. For the Artix security service, this CLASSPATH must point at the JAR file containing the implementation of the security service. For example:

```
plugins:java_server:classpath =
    "C:\artix_40/lib/artix/security_service/4.0/security_service-
    rt.jar";
```

The Java server plug-in ignores the contents of the CLASSPATH environment variable.

jni_verbose

A boolean variable that instructs the JVM to output JNI-level diagnostics, which can be helpful for troubleshooting. When true, the JVM-generated diagnostic messages are sent to the Artix logging stream; when false, the diagnostic messages are suppressed.

shlib_name

Specifies the abbreviated name of the shared library that implements the `java_server` plug-in. This variable must always be set as follows:

```
plugins:java_server:shlib_name = "it_java_server";
```

system_properties

Specifies a list of Java system properties to the JVM created by the Java server plug-in. For example, the Artix security service requires the following Java system property settings:

```
plugins:java_server:system_properties =
    ["org.omg.CORBA.ORBClass=com.iona.corba.art.artimpl.ORBImpl",
    "org.omg.CORBA.ORBSingletonClass=com.iona.corba.art.artimpl.O
    RBSingleton",
    "is2.properties=%{INSTALL_DIR}/%{PRODUCT_NAME}/%{PRODUCT_VERS
    ION}/demos/security/full_security/etc/is2.properties.FILE",
    "java.endorsed.dirs=%{INSTALL_DIR}/%{PRODUCT_NAME}/%{PRODUCT_
    VERSION}/lib/endorsed"];
```

Where each item in the list specifies a Java system property, as follows:

<*PropertyName*>=<*PropertyValue*>

X_options

Specifies a list of non-standard, `-X`, options to the JVM created by the Java server plug-in. In contrast to the way these options are specified to the `java` command-line tool, you must omit the `-X` prefix in the `X_options` list.

For example:

```
plugins:java_server:X_options = ["rs"];
```

To find out more about the non-standard JVM options, type `java -X -help` at the command line (using Sun's implementation of the JVM).

plugins:kdm

The plugins:kdm namespace contains the following variables:

- cert_constraints
- iiop_tls:port
- checksums_optional

cert_constraints

Specifies the list of certificate constraints for principals attempting to open a connection to the KDM server plug-in. See "Applying Constraints to Certificates" on page 119 for a description of the certificate constraint syntax.

To protect the sensitive data stored within it, the KDM applies restrictions on which entities are allowed talk to it. A security administrator should choose certificate constraints that restrict access to the following principals:

- The locator service (requires read-only access).
- The kdm_adm plug-in, which is normally loaded into the itadmin utility (requires read-write access).

All other principals should be blocked from access. For example, you might define certificate constraints similar to the following:

```
plugins:kdm:cert_constraints =
   ["C=US,ST=Massachusetts,O=ABigBank*,CN=Secure admin*",
   "C=US,ST=Boston,O=ABigBank*,CN=Orbix2000 Locator Service*"]
```

Your choice of certificate constraints will depend on the naming scheme for your subject names.

iiop_tls:port

Specifies the well known IP port on which the KDM server listens for incoming calls.

checksums_optional

When equal to `false`, the secure information associated with a server must include a checksum; when equal to `true`, the presence of a checksum is optional. Default is `false`.

plugins:kdm_adm

The plugins:kdm_adm namespace contains the following variable:

- cert_constraints

cert_constraints

Specifies the list of certificate constraints that are applied when the KDM administration plug-in authenticates the KDM server. See "Applying Constraints to Certificates" on page 119 for a description of the certificate constraint syntax.

The KDM administration plug-in requires protection against attack from applications that try to impersonate the KDM server. A security administrator should, therefore, choose certificate constraints that restrict access to trusted KDM servers only. For example, you might define certificate constraints similar to the following:

```
plugins:kdm_adm:cert_constraints =
    ["C=US,ST=Massachusetts,O=ABigBank*,CN=IT_KDM*"];
```

Your choice of certificate constraints will depend on the naming scheme for your subject names.

plugins:login_client

The plugins:login_client namespace contains the following variables:

- wsdl_url

wsdl_url

Specifies the location of the login service WSDL to the login_client plug-in. The value of this variable can either be a relative pathname or an URL. The login_client requires access to the login service WSDL in order to obtain details of the physical contract (for example, host and IP port).

plugins:login_service

The plugins:login_service namespace contains the following variables:

- wsdl_url

wsdl_url

Specifies the location of the login service WSDL to the login_service plug-in. The value of this variable can either be a relative pathname or an URL. The login_service requires access to the login service WSDL in order to obtain details of the physical contract (for example, host and IP port).

plugins:schannel

The plugins:schannel namespace contains the following variable:

- prompt_with_credential_choice

prompt_with_credential_choice

(Windows and Schannel only) Setting both this variable and the plugins:iiop_tls:delay_credential_gathering_until_handshake variable to true on the client side allows the user to choose which credentials to use for the server connection. The choice of credentials offered to the user is based on the trusted CAs sent to the client in an SSL/TLS handshake message.

If prompt_with_credential_choice is set to false, runtime chooses the first certificate it finds in the certificate store that meets the applicable constraints.

The certificate prompt can be replaced by implementing an IDL interface and registering it with the ORB.

plugins:security

The plugins:security namespace contains the following variable:

- direct_persistence
- iiop_tls:addr_list
- iiop_tls:host
- iiop_tls:port
- log4j_to_local_log_stream
- share_credentials_across_orbs

direct_persistence

A boolean variable that specifies whether or not the security service runs on a fixed IP port (for an IIOP/TLS-based security service). You must always set this variable to true in the security service's configuration scope, because the security service *must* run on a fixed port.

iiop_tls:addr_list

When the security service is configured as a cluster, you must use this variable to list the addresses of all of the security services in the cluster.

The first entry, *not* prefixed by a + sign, must specify the address of the current security service instance. The remaining entries, prefixed by a + sign, must specify the addresses of the other services in the cluster (the + sign indicates that an entry affects only the contents of the generated IOR, not the security service's listening port).

For example, to configure the first instance of a cluster consisting of three security service instances—with addresses security01:5001, security02:5002, and security03:5003—you would initialize the address list as follows:

```
plugins:security:iiop_tls:addr_list = ["security01:5001",
    "+security02:5002", "+security03:5003"];
```

iiop_tls:host

Specifies the hostname where the security service is running. This hostname will be embedded in the security service's IOR (for an IIOP/TLS-based security service).

iiop_tls:port

Specifies the fixed IP port where the security service listens for incoming connections. This IP port also gets embedded in the security service's IOR (for an IIOP/TLS-based security service).

log4j_to_local_log_stream

Redirects the Artix security service's log4j output to the local log stream. In the Artix security service's configuration scope, you can set the `plugins:security:log4j_to_local_log_stream` variable to one of the following values:

- `true`—the security service log4j output is sent to the local log stream. This requires that the `local_log_stream` plug-in is present in the `orb_plugins` list.
- `false`—*(default)* the log4j output is controlled by the `log4j.properties` file (whose location is specified in the `is2.properties` file).

When redirecting log4j messages to the local log stream, you can control the log4j logging level using Artix event log filters. You can specify Artix event log filters with the following setting in the Artix configuration file:

```
event_log:filters = ["IT_SECURITY=LoggingLevels"];
```

The `IT_SECURITY` tag configures the logging levels for the Artix security service (which includes the redirected log4j stream). log4j has five logging levels: `DEBUG`, `INFO`, `WARN`, `ERROR`, and `FATAL`. To select a particular log4j logging level (for example, `WARN`), replace *`LoggingLevels`* by that logging level plus all of the higher logging levels (for example, `WARN+ERROR+FATAL`).

For example, you can configure the Artix security service to send log4j logging to the local log stream, as follows:

```
# Artix Configuration File
security_service
{
    orb_plugins = ["local_log_stream", "iiop_profile", "giop",
   "iiop_tls"];
    plugins:security:log4j_to_local_log_stream = "true";

    # Log all log4j messages at level WARN and above
    event_log:filters = ["IT_SECURITY=WARN+ERROR+FATAL"];
    ...
};
```

share_credentials_across_orbs

Enables own security credentials to be shared across ORBs. Normally, when you specify an own SSL/TLS credential (using the principal sponsor or the principal authenticator), the credential is available only to the ORB that created it. By setting the `plugins:security:share_credentials_across_orbs` variable to `true`, however, the own SSL/TLS credentials created by one ORB are automatically made available to any other ORBs that are configured to share credentials.

See also `principal_sponsor:csi:use_existing_credentials` for details of how to enable sharing of CSI credentials.

Default is `false`.

plugins:wsdl_publish

The plugins:wsdl_publish namespace contains the following variables:

- enable_secure_wsdl_publish

enable_secure_wsdl_publish

A boolean variable that enables certain security features of the WSDL publishing service that are required whenever the WSDL publishing service is configured to use the HTTPS protocol. Set this variable to true, if the WSDL publishing service is confided to use HTTPS; otherwise, set it to false.

Default is false.

For example, to configure the WSDL publishing service to use HTTPS, you should include the following in your program's configuration scope:

```
# Artix Configuration File
secure_server
{
    orb_plugins = [ ... , "wsdl_publish", "at_http", "https"];

    plugins:wsdl_publish:publish_port = "2222";
    plugins:wsdl_publish:enable_secure_wsdl_publish = "true";
    plugins:at_http:server:use_secure_sockets = "true";

    # Other HTTPS-related settings
    ...
};
```

The plugins:at_http:server:use_secure_sockets setting is needed to enable HTTPS for the WSDL publishing service.

Note: You must set *both* plugins:wsdl_publish:enable_secure_wsdl_publish and plugins:at_http:server:use_secure_sockets to true, when enabling HTTPS for the WSDL publish plug-in.

policies

The policies namespace defines the default CORBA policies for an ORB. Many of these policies can also be set programmatically from within an application. SSL/TLS-specific variables in the policies namespace include:

- allow_unauthenticated_clients_policy
- certificate_constraints_policy
- client_secure_invocation_policy:requires
- client_secure_invocation_policy:supports
- max_chain_length_policy
- mechanism_policy:accept_v2_hellos
- mechanism_policy:ciphersuites
- mechanism_policy:protocol_version
- session_caching_policy
- target_secure_invocation_policy:requires
- target_secure_invocation_policy:supports
- trusted_ca_list_policy

allow_unauthenticated_clients_policy

A generic variable that sets this policy both for iiop_tls and https. To set this policy specifically for the IIOP/TLS protocol, set the policies:iiop_tls:allow_unauthenticated_clients_policy variable, which takes precedence.

A boolean variable that specifies whether a server will allow a client to establish a secure connection without sending a certificate. Default is false.

This configuration variable is applicable *only* in the special case where the target secure invocation policy is set to require NoProtection (a semi-secure server).

certificate_constraints_policy

A generic variable that sets this policy both for iiop_tls and https. To set this policy specifically for the IIOP/TLS protocol, set the policies:iiop_tls:certificate_constraints_policy variable, which takes precedence.

A list of constraints applied to peer certificates—see "Applying Constraints to Certificates" on page 119. If a peer certificate fails to match any of the constraints, the certificate validation step will fail.

The policy can also be set programmatically using the IT_TLS_API::CertConstraintsPolicy CORBA policy. Default is no constraints.

client_secure_invocation_policy:requires

A generic variable that sets this policy both for iiop_tls and https. To set this policy specifically for the IIOP/TLS protocol, set the policies:iiop_tls:client_secure_invocation_policy:requires variable, which takes precedence.

Specifies the minimum level of security required by a client. The value of this variable is specified as a list of association options—see the *Artix Security Guide* for more details about association options.

In accordance with CORBA security, this policy cannot be downgraded programmatically by the application.

client_secure_invocation_policy:supports

A generic variable that sets this policy both for iiop_tls and https. To set this policy specifically for the IIOP/TLS protocol, set the policies:iiop_tls:client_secure_invocation_policy:supports variable, which takes precedence.

Specifies the initial maximum level of security supported by a client. The value of this variable is specified as a list of association options—see the *Artix Security Guide* for more details about association options.

This policy can be upgraded programmatically using either the QOP or the EstablishTrust policies.

max_chain_length_policy

A generic variable that sets this policy both for iiop_tls and https. To set this policy specifically for the IIOP/TLS protocol, set the policies:iiop_tls:max_chain_length_policy variable, which takes precedence.

max_chain_length_policy specifies the maximum certificate chain length that an ORB will accept. The policy can also be set programmatically using the IT_TLS_API::MaxChainLengthPolicy CORBA policy. Default is 2.

Note: The max_chain_length_policy is not currently supported on the z/OS platform.

mechanism_policy:accept_v2_hellos

A generic variable that sets this policy both for iiop_tls and https. To set this policy for a specific protocol, set policies:iiop_tls:mechanism_policy:accept_v2_hellos or policies:https:mechanism_policy:accept_v2_hellos respectively for IIOP/TLS or HTTPS.

The accept_v2_hellos policy is a special setting that facilitates interoperability with an Artix application deployed on the z/OS platform. When true, the Artix application accepts V2 client hellos, but continues the handshake using either the SSL_V3 or TLS_V1 protocol. When `false`, the Artix application throws an error, if it receives a V2 client hello. The default is `false`.

For example:

```
policies:mechanism_policy:accept_v2_hellos = "true";
```

mechanism_policy:ciphersuites

A generic variable that sets this policy both for `iiop_tls` and `https`. To set this policy for a specific protocol, set `policies:iiop_tls:mechanism_policy:ciphersuites` or `policies:https:mechanism_policy:ciphersuites` respectively for IIOP/TLS or HTTPS.

`mechanism_policy:ciphersuites` specifies a list of cipher suites for the default mechanism policy. One or more of the cipher suites shown in Table 4 can be specified in this list.

Table 4: *Mechanism Policy Cipher Suites*

Null Encryption, Integrity and Authentication Ciphers	Standard Ciphers
RSA_WITH_NULL_MD5	RSA_EXPORT_WITH_RC4_40_MD5
RSA_WITH_NULL_SHA	RSA_WITH_RC4_128_MD5
	RSA_WITH_RC4_128_SHA
	RSA_EXPORT_WITH_DES40_CBC_SHA
	RSA_WITH_DES_CBC_SHA
	RSA_WITH_3DES_EDE_CBC_SHA

If you do not specify the list of cipher suites explicitly, all of the null encryption ciphers are disabled and all of the non-export strength ciphers are supported by default.

mechanism_policy:protocol_version

A generic variable that sets this policy both for `iiop_tls` and `https`. To set this policy for a specific protocol, set `policies:iiop_tls:mechanism_policy:protocol_version` or `policies:https:mechanism_policy:protocol_version` respectively for IIOP/TLS or HTTPS.

`mechanism_policy:protocol_version` specifies the list of protocol versions used by a security capsule (ORB instance). The list can include one or more of the values `SSL_V3` and `TLS_V1`. For example:

```
policies:mechanism_policy:protocol_version=["TLS_V1", "SSL_V3"];
```

session_caching_policy

A generic variable that sets this policy both for `iiop_tls` and `https`. To set this policy specifically for the IIOP/TLS protocol, set the `policies:iiop_tls:session_caching_policy` variable, which takes precedence.

`session_caching_policy` specifies whether an ORB caches the session information for secure associations when acting in a client role, a server role, or both. The purpose of session caching is to enable closed connections to be re-established quickly. The following values are supported:

`CACHE_NONE`(default)

`CACHE_CLIENT`
`CACHE_SERVER`
`CACHE_SERVER_AND_CLIENT`

The policy can also be set programmatically using the `IT_TLS_API::SessionCachingPolicy` CORBA policy.

target_secure_invocation_policy:requires

A generic variable that sets this policy both for `iiop_tls` and `https`. To set this policy specifically for the IIOP/TLS protocol, set the `policies:iiop_tls:target_secure_invocation_policy:requires` variable, which takes precedence.

`target_secure_invocation_policy:requires` specifies the minimum level of security required by a server. The value of this variable is specified as a list of association options.

Note: In accordance with CORBA security, this policy cannot be downgraded programmatically by the application.

target_secure_invocation_policy:supports

A generic variable that sets this policy both for `iiop_tls` and `https`. To set this policy specifically for the IIOP/TLS protocol, set the `policies:iiop_tls:target_secure_invocation_policy:supports` variable, which takes precedence.

`supports` specifies the maximum level of security supported by a server. The value of this variable is specified as a list of association options. This policy can be upgraded programmatically using either the `QOP` or the `EstablishTrust` policies.

trusted_ca_list_policy

A generic variable that sets this policy both for iiop_tls and https. To set this policy for a specific protocol, set policies:iiop_tls:trusted_ca_list_policy or policies:https:trusted_ca_list_policy respectively for IIOP/TLS or HTTPS.

trusted_ca_list_policy specifies a list of filenames, each of which contains a concatenated list of CA certificates in PEM format. The aggregate of the CAs in all of the listed files is the set of trusted CAs.

For example, you might specify two files containing CA lists as follows:

```
policies:trusted_ca_list_policy =
    ["install_dir/asp/version/etc/tls/x509/ca/ca_list1.pem",
    "install_dir/asp/version/etc/tls/x509/ca/ca_list_extra.pem"];
```

The purpose of having more than one file containing a CA list is for administrative convenience. It enables you to group CAs into different lists and to select a particular set of CAs for a security domain by choosing the appropriate CA lists.

policies:asp

The policies:asp namespace contains the following variables:

- enable_security
- enable_security
- enable_sso
- load_balancing_policy
- use_artix_proxies

enable_authorization

A boolean variable that specifies whether Artix should enable authorization using the Artix Security Framework. Default is true.

Note: From Artix 4.0 onwards, the default value of policies:asp:enable_authorization is true. For versions of Artix prior to 4.0, the default value of policies:asp:enable_authorization is false.

enable_security

A boolean variable that specifies whether Artix should enable security using the Artix Security Framework. When this variable is set to false, all security features that depend on the artix_security plug-in (that is, authentication and authorization using the Artix security service) are disabled. Default is true.

Note: From Artix 4.0 onwards, the default value of policies:asp:enable_security is true. For versions of Artix prior to 4.0, the default value of policies:asp:enable_security is false.

enable_sso

This configuration variable is obsolete and has no effect.

load_balancing_policy

When client load balancing is enabled, this variable specifies how often the Artix security plug-in reconnects to a node in the security service cluster. There are two possible values for this policy:

- `per-server`—*(the default)* after selecting a particular security service from the cluster, the client remains connected to that security service instance for the rest of the session.
- `per-request`—for each new request, the Artix security plug-in selects and connects to a new security service node (in accordance with the algorithm specified by `policies:iiop_tls:load_balancing_mechanism`).

> **Note:** The process of re-establishing a secure connection with every new request imposes a significant performance overhead. Therefore, the `per-request` policy value is *not* recommended for most deployments.

This policy is used in conjunction with the `plugins:asp:enable_security_service_load_balancing` and `policies:iiop_tls:load_balancing_mechanism` configuration variables.

Default is `per-server`.

use_artix_proxies

A boolean variable that specifies whether a client of the Artix security service connects to the security service through a WSDL contract or through a CORBA object reference. The `policies:asp:use_artix_proxies` variable can have the following values:

- `true`—connect to the security service through a WSDL contract. The location of the security service WSDL contract can be specified using the `bus:initial_contract:url:isf_service` configuration variable.
- `false`—connect to the security service through a CORBA object reference. The object reference is specified by the `initial_references:IT_SecurityService:reference` configuration variable.

Default is `false`.

policies:bindings

The policies:bindings namespace contains the following variables:

- corba:gssup_propagation
- corba:token_propagation
- soap:gssup_propagation
- soap:token_propagation

corba:gssup_propagation

A boolean variable that can be used in a SOAP-to-CORBA router to enable the transfer of incoming SOAP credentials into outgoing CORBA credentials.

The CORBA binding extracts the username and password credentials from incoming SOAP/HTTP invocations and inserts them into an outgoing GSSUP credentials object, to be transmitted using CSI authentication over transport. The domain name in the outgoing GSSUP credentials is set to a blank string. Default is false.

corba:token_propagation

A boolean variable that can be used in a SOAP-to-CORBA router to enable the transfer of an SSO token from an incoming SOAP request into an outgoing CORBA request.

The CORBA binding extracts the SSO token from incoming SOAP/HTTP invocations and inserts the token into an outgoing IIOP request, to be transmitted using CSI identity assertion.

soap:gssup_propagation

A boolean variable that can be used in a CORBA-to-SOAP router to enable the transfer of incoming CORBA credentials into outgoing SOAP credentials.

The SOAP binding extracts the username and password from incoming IIOP invocations (where the credentials are embedded in a GIOP service context and encoded according to the CSI and GSSUP standards), and inserts them into an outgoing SOAP header, encoded using the WSSE standard.

Default is `false`.

soap:token_propagation

A boolean variable that can be used in a CORBA-to-SOAP router to enable the transfer of an SSO token from an incoming CORBA request into an outgoing SOAP request.

The SOAP binding extracts the SSO token from an incoming IIOP request and inserts the token into the header of an outgoing SOAP/HTTP request.

policies:csi

The policies:csi namespace includes variables that specify settings for Common Secure Interoperability version 2 (CSIv2):

- attribute_service:backward_trust:enabled
- attribute_service:client_supports
- attribute_service:target_supports
- auth_over_transport:authentication_service
- auth_over_transport:client_supports
- auth_over_transport:server_domain_name
- auth_over_transport:target_requires
- auth_over_transport:target_supports

attribute_service:backward_trust:enabled

(Obsolete)

attribute_service:client_supports

attribute_service:client_supports is a client-side policy that specifies the association options supported by the CSIv2 attribute service (principal propagation). The only assocation option that can be specified is IdentityAssertion. This policy is normally specified in an intermediate server so that it propagates CSIv2 identity tokens to a target server. For example:

```
policies:csi:attribute_service:client_supports =
   ["IdentityAssertion"];
```

attribute_service:target_supports

`attribute_service:target_supports` is a server-side policy that specifies the association options supported by the CSIv2 attribute service (principal propagation). The only assocation option that can be specified is `IdentityAssertion`. For example:

```
policies:csi:attribute_service:target_supports =
   ["IdentityAssertion"];
```

auth_over_transport:authentication_service

(Java CSI plug-in only) The name of a Java class that implements the `IT_CSI::AuthenticateGSSUPCredentials` IDL interface. The authentication service is implemented as a callback object that plugs into the CSIv2 framework on the server side. By replacing this class with a custom implementation, you could potentially implement a new security technology domain for CSIv2.

By default, if no value for this variable is specified, the Java CSI plug-in uses a default authentication object that always returns `false` when the `authenticate()` operation is called.

auth_over_transport:client_supports

`auth_over_transport:client_supports` is a client-side policy that specifies the association options supported by CSIv2 authorization over transport. The only assocation option that can be specified is `EstablishTrustInClient`. For example:

```
policies:csi:auth_over_transport:client_supports =
   ["EstablishTrustInClient"];
```

auth_over_transport:server_domain_name

The iSF security domain (CSIv2 authentication domain) to which this server application belongs. The iSF security domains are administered within an overall security technology domain.

The value of the `server_domain_name` variable will be embedded in the IORs generated by the server. A CSIv2 client about to open a connection to this server would check that the domain name in its own CSIv2 credentials matches the domain name embedded in the IOR.

auth_over_transport:target_requires

`auth_over_transport:target_requires` is a server-side policy that specifies the association options required for CSIv2 authorization over transport. The only assocation option that can be specified is `EstablishTrustInClient`. For example:

```
policies:csi:auth_over_transport:target_requires =
   ["EstablishTrustInClient"];
```

auth_over_transport:target_supports

`auth_over_transport:target_supports` is a server-side policy that specifies the association options supported by CSIv2 authorization over transport. The only assocation option that can be specified is `EstablishTrustInClient`. For example:

```
policies:csi:auth_over_transport:target_supports =
   ["EstablishTrustInClient"];
```

policies:external_token_issuer

The policies:external_token_issuer namespace contains the following variables:

- client_certificate_constraints

client_certificate_constraints

To facilitate interoperability with Artix on the mainframe, the Artix security service can be configured to issue security tokens based on a username only (no password required). This feature is known as the *external token issuer*. Because this feature could potentially open a security hole in the Artix security service, the external token issuer is made available *only* to those applications that present a certificate matching the constraints specified in policies:external_token_issuer:client_certificate_constraints. For details of how to specify certificate constraints, see "Applying Constraints to Certificates" on page 119.

For example, by inserting the following setting into the security service's configuration scope in the Artix configuration file, you would effectively disable the external token issuer (recommended for deployments that do not need to interoperate with the mainframe).

```
# DISABLE the security service's external token issuer.
# Note: The empty list matches no certificates.
#
policies:external_token_issuer:client_certificate_constraints =
    [];
```

This configuration variable must be set in the security server's configuration scope, otherwise the security server will not start.

policies:https

The policies:https namespace contains variables used to configure the https plugin. It contains the following variables:

- mechanism_policy:accept_v2_hellos
- mechanism_policy:ciphersuites
- mechanism_policy:protocol_version
- trusted_ca_list_policy

mechanism_policy:accept_v2_hellos

This HTTPS-specific policy overides the generic policies:mechanism_policy:accept_v2_hellos policy.

The accept_v2_hellos policy is a special setting that facilitates HTTPS interoperability with certain Web browsers. Many Web browsers send SSL V2 client hellos, because they do not know what SSL version the server supports.

When true, the Artix server accepts V2 client hellos, but continues the handshake using either the SSL_V3 or TLS_V1 protocol. When false, the Artix server throws an error, if it receives a V2 client hello. The default is true.

Note: This default value is deliberately different from the policies:iiop_tls:mechanism_policy:accept_v2_hellos default value.

For example:

```
policies:https:mechanism_policy:accept_v2_hellos = "true";
```

mechanism_policy:ciphersuites

Specifies a list of cipher suites for the default mechanism policy. One or more of the following cipher suites can be specified in this list:

Table 5: *Mechanism Policy Cipher Suites*

Null Encryption, Integrity and Authentication Ciphers	Standard Ciphers
RSA_WITH_NULL_MD5	RSA_EXPORT_WITH_RC4_40_MD5
RSA_WITH_NULL_SHA	RSA_WITH_RC4_128_MD5
	RSA_WITH_RC4_128_SHA
	RSA_EXPORT_WITH_DES40_CBC_SHA
	RSA_WITH_DES_CBC_SHA
	RSA_WITH_3DES_EDE_CBC_SHA

If you do not specify the list of cipher suites explicitly, all of the null encryption ciphers are disabled and all of the non-export strength ciphers are supported by default.

mechanism_policy:protocol_version

This HTTPS-specific policy overides the generic `policies:mechanism_policy:protocol_version` policy.

Specifies the list of protocol versions used by a security capsule (ORB instance). Can include one or more of the following values:

```
TLS_V1
SSL_V3
```

The default setting is `SSL_V3` and `TLS_V1`.

For example:

```
policies:https:mechanism_policy:protocol_version = ["TLS_V1",
   "SSL_V3"];
```

trusted_ca_list_policy

Contains a list of filenames (or a single filename), each of which contains a concatenated list of CA certificates in PEM format. The aggregate of the CAs in all of the listed files is the set of trusted CAs.

For example, you might specify two files containing CA lists as follows:

```
policies:trusted_ca_list_policy =
   ["ASPInstallDir/asp/6.0/etc/tls/x509/ca/ca_list1.pem",
   "ASPInstallDir/asp/6.0/etc/tls/x509/ca/ca_list_extra.pem"];
```

The purpose of having more than one file containing a CA list is for administrative convenience. It enables you to group CAs into different lists and to select a particular set of CAs for a security domain by choosing the appropriate CA lists.

policies:iiop_tls

The policies:iiop_tls namespace contains variables used to set IIOP-related policies for a secure environment. These setting affect the iiop_tls plugin. It contains the following variables:

- allow_unauthenticated_clients_policy
- buffer_sizes_policy:default_buffer_size
- buffer_sizes_policy:max_buffer_size
- certificate_constraints_policy
- client_secure_invocation_policy:requires
- client_secure_invocation_policy:supports
- client_version_policy
- connection_attempts
- connection_retry_delay
- load_balancing_mechanism
- max_chain_length_policy
- mechanism_policy:accept_v2_hellos
- mechanism_policy:ciphersuites
- mechanism_policy:protocol_version
- server_address_mode_policy:local_domain
- server_address_mode_policy:local_hostname
- server_address_mode_policy:port_range
- server_address_mode_policy:publish_hostname
- server_version_policy
- session_caching_policy
- target_secure_invocation_policy:requires
- target_secure_invocation_policy:supports
- tcp_options_policy:no_delay
- tcp_options_policy:recv_buffer_size
- tcp_options_policy:send_buffer_size
- trusted_ca_list_policy

allow_unauthenticated_clients_policy

A boolean variable that specifies whether a server will allow a client to establish a secure connection without sending a certificate. Default is `false`.

This configuration variable is applicable *only* in the special case where the target secure invocation policy is set to require `NoProtection` (a semi-secure server).

buffer_sizes_policy:default_buffer_size

When this policy is set, the `iiop_tls` plug-in reads this policy's value instead of the `policies:iiop:buffer_sizes_policy:default_buffer_size` policy's value.

`buffer_sizes_policy:default_buffer_size` specifies, in bytes, the initial size of the buffers allocated by IIOP. Defaults to 16000. This value must be greater than 80 bytes, and must be evenly divisible by 8.

buffer_sizes_policy:max_buffer_size

When this policy is set, the `iiop_tls` plug-in reads this policy's value instead of the `policies:iiop:buffer_sizes_policy:max_buffer_size` policy's value.

`buffer_sizes_policy:max_buffer_size` specifies the maximum buffer size permitted by IIOP, in kilobytes. Defaults to 512. A value of -1 indicates unlimited size. If not unlimited, this value must be greater than 80.

certificate_constraints_policy

A list of constraints applied to peer certificates—see the discussion of certificate constraints in the Artix security guide for the syntax of the pattern constraint language. If a peer certificate fails to match any of the constraints, the certificate validation step will fail.

The policy can also be set programmatically using the `IT_TLS_API::CertConstraintsPolicy` CORBA policy. Default is no constraints.

client_secure_invocation_policy:requires

Specifies the minimum level of security required by a client. The value of this variable is specified as a list of association options—see the *Artix Security Guide* for more details about association options.

In accordance with CORBA security, this policy cannot be downgraded programmatically by the application.

client_secure_invocation_policy:supports

Specifies the initial maximum level of security supported by a client. The value of this variable is specified as a list of association options—see the *Artix Security Guide* for more details about association options.

This policy can be upgraded programmatically using either the `QOP` or the `EstablishTrust` policies.

client_version_policy

`client_version_policy` specifies the highest IIOP version used by clients. A client uses the version of IIOP specified by this variable, or the version specified in the IOR profile, whichever is lower. Valid values for this variable are: `1.0`, `1.1`, and `1.2`.

For example, the following file-based configuration entry sets the server IIOP version to 1.1.

```
policies:iiop:server_version_policy="1.1";
```

The following `itadmin` command set this variable:

```
itadmin variable modify -type string -value "1.1"
   policies:iiop:server_version_policy
```

connection_attempts

`connection_attempts` specifies the number of connection attempts used when creating a connected socket using a Java application. Defaults to `5`.

connection_retry_delay

connection_retry_delay specifies the delay, in seconds, between connection attempts when using a Java application. Defaults to 2.

load_balancing_mechanism

Specifies the load balancing mechanism for the client of a security service cluster (see also plugins:gsp:use_client_load_balancing and plugins:asp:enable_security_service_load_balancing). In this context, a client can also be an *Artix* server. This policy only affects connections made using IORs that contain multiple addresses. The iiop_tls plug-in load balances over the addresses embedded in the IOR.

The following mechanisms are supported:

- random—choose one of the addresses embedded in the IOR at random (this is the default).
- sequential—choose the first address embedded in the IOR, moving on to the next address in the list only if the previous address could not be reached.

max_chain_length_policy

This policy overides policies:max_chain_length_policy for the iiop_tls plugin.

The maximum certificate chain length that an ORB will accept.

The policy can also be set programmatically using the IT_TLS_API::MaxChainLengthPolicy CORBA policy. Default is 2.

Note: The max_chain_length_policy is not currently supported on the z/OS platform.

mechanism_policy:accept_v2_hellos

This IIOP/TLS-specific policy overides the generic policies:mechanism_policy:accept_v2_hellos policy.

The `accept_v2_hellos` policy is a special setting that facilitates interoperability with an Artix application deployed on the z/OS platform. Artix security on the z/OS platform is based on IBM's System/SSL toolkit, which implements SSL version 3, but does so by using SSL version 2 hellos as part of the handshake. This form of handshake causes interoperability problems, because applications on other platforms identify the handshake as an SSL version 2 handshake. The misidentification of the SSL protocol version can be avoided by setting the `accept_v2_hellos` policy to `true` in the non-z/OS application (this bug also affects some old versions of Microsoft Internet Explorer).

When `true`, the Artix application accepts V2 client hellos, but continues the handshake using either the SSL_V3 or TLS_V1 protocol. When `false`, the Artix application throws an error, if it receives a V2 client hello. The default is `false`.

Note: This default value is deliberately different from the `policies:https:mechanism_policy:accept_v2_hellos` default value.

For example:

```
policies:iiop_tls:mechanism_policy:accept_v2_hellos = "true";
```

mechanism_policy:ciphersuites

This policy overides `policies:mechanism_policy:ciphersuites` for the `iiop_tls` plugin.

Specifies a list of cipher suites for the default mechanism policy. One or more of the following cipher suites can be specified in this list:

Table 6: *Mechanism Policy Cipher Suites*

Null Encryption, Integrity and Authentication Ciphers	Standard Ciphers
RSA_WITH_NULL_MD5	RSA_EXPORT_WITH_RC4_40_MD5
RSA_WITH_NULL_SHA	RSA_WITH_RC4_128_MD5
	RSA_WITH_RC4_128_SHA
	RSA_EXPORT_WITH_DES40_CBC_SHA

Table 6: *Mechanism Policy Cipher Suites*

Null Encryption, Integrity and Authentication Ciphers	Standard Ciphers
	`RSA_WITH_DES_CBC_SHA`
	`RSA_WITH_3DES_EDE_CBC_SHA`

If you do not specify the list of cipher suites explicitly, all of the null encryption ciphers are disabled and all of the non-export strength ciphers are supported by default.

mechanism_policy:protocol_version

This IIOP/TLS-specific policy overides the generic `policies:mechanism_policy:protocol_version` policy.

Specifies the list of protocol versions used by a security capsule (ORB instance). Can include one or more of the following values:

`TLS_V1`
`SSL_V3`
`SSL_V2V3` *(Deprecated)*

The default setting is `SSL_V3` and `TLS_V1`.

For example:

```
policies:iiop_tls:mechanism_policy:protocol_version = ["TLS_V1",
   "SSL_V3"];
```

The `SSL_V2V3` value is now *deprecated*. It was previously used to facilitate interoperability with Artix applications deployed on the z/OS platform. If you have any legacy configuration that uses `SSL_V2V3`, you should replace it with the following combination of settings:

```
policies:iiop_tls:mechanism_policy:protocol_version = ["SSL_V3",
   "TLS_V1"];
policies:iiop_tls:mechanism_policy:accept_v2_hellos = "true";
```

server_address_mode_policy:local_domain

(Java only) When this policy is set, the `iiop_tls` plug-in reads this policy's value instead of the `policies:iiop:server_address_mode_policy:local_domain` policy's value.

server_address_mode_policy:local_hostname

(Java only) When this policy is set, the iiop_tls plug-in reads this policy's value instead of the policies:iiop:server_address_mode_policy:local_hostname policy's value.

server_address_mode_policy:local_hostname specifies the hostname advertised by the locator daemon, and listened on by server-side IIOP.

Some machines have multiple hostnames or IP addresses (for example, those using multiple DNS aliases or multiple network cards). These machines are often termed *multi-homed hosts*. The local_hostname variable supports these type of machines by enabling you to explicitly specify the host that servers listen on and publish in their IORs.

For example, if you have a machine with two network addresses (207.45.52.34 and 207.45.52.35), you can explicitly set this variable to either address:

```
policies:iiop:server_address_mode_policy:local_hostname =
    "207.45.52.34";
```

By default, the local_hostname variable is unspecified. Servers use the default hostname configured for the machine with the Orbix configuration tool.

server_address_mode_policy:port_range

(Java only) When this policy is set, the iiop_tls plug-in reads this policy's value instead of the policies:iiop:server_address_mode_policy:port_range policy's value.

server_address_mode_policy:port_range specifies the range of ports that a server uses when there is no well-known addressing policy specified for the port.

server_address_mode_policy:publish_hostname

When this policy is set, the iiop_tls plug-in reads this policy's value instead of the policies:iiop:server_address_mode_policy:publish_hostname policy's value.

server_address_mode-policy:publish_hostname specifes whether IIOP exports hostnames or IP addresses in published profiles. Defaults to false (exports IP addresses, and does not export hostnames). To use hostnames in object references, set this variable to true, as in the following file-based configuration entry:

```
policies:iiop:server_address_mode_policy:publish_hostname=true
```

The following itadmin command is equivalent:

```
itadmin variable create -type bool -value true
   policies:iiop:server_address_mode_policy:publish_hostname
```

server_version_policy

When this policy is set, the iiop_tls plug-in reads this policy's value instead of the policies:iiop:server_version_policy policy's value.

server_version_policy specifies the GIOP version published in IIOP profiles. This variable takes a value of either 1.1 or 1.2. Orbix servers do not publish IIOP 1.0 profiles. The default value is 1.2.

session_caching_policy

This policy overides policies:session_caching_policy for the iiop_tls plugin.

target_secure_invocation_policy:requires

This policy overides `policies:target_secure_invocation_policy:requires` for the `iiop_tls` plugin.

Specifies the minimum level of security required by a server. The value of this variable is specified as a list of association options—see the *Artix Security Guide* for more details about association options.

In accordance with CORBA security, this policy cannot be downgraded programmatically by the application.

target_secure_invocation_policy:supports

This policy overides `policies:target_secure_invocation_policy:supports` for the `iiop_tls` plugin.

Specifies the maximum level of security supported by a server. The value of this variable is specified as a list of association options—see the *Artix Security Guide* for more details about association options.

This policy can be upgraded programmatically using either the `QOP` or the `EstablishTrust` policies.

tcp_options_policy:no_delay

When this policy is set, the `iiop_tls` plug-in reads this policy's value instead of the `policies:iiop:tcp_options_policy:no_delay` policy's value.

`tcp_options_policy:no_delay` specifies whether the `TCP_NODELAY` option should be set on connections. Defaults to `false`.

tcp_options_policy:recv_buffer_size

When this policy is set, the iiop_tls plug-in reads this policy's value instead of the policies:iiop:tcp_options_policy:recv_buffer_size policy's value.

tcp_options_policy:recv_buffer_size specifies the size of the TCP receive buffer. This variable can only be set to 0, which coresponds to using the default size defined by the operating system.

tcp_options_policy:send_buffer_size

When this policy is set, the iiop_tls plug-in reads this policy's value instead of the policies:iiop:tcp_options_policy:send_buffer_size policy's value.

tcp_options_policy:send_buffer_size specifies the size of the TCP send buffer. This variable can only be set to 0, which coresponds to using the default size defined by the operating system.

trusted_ca_list_policy

This policy overides the policies:trusted_ca_list_policy for the iiop_tls plugin.

Contains a list of filenames (or a single filename), each of which contains a concatenated list of CA certificates in PEM format. The aggregate of the CAs in all of the listed files is the set of trusted CAs.

For example, you might specify two files containing CA lists as follows:

```
policies:trusted_ca_list_policy =
    ["ASPInstallDir/asp/6.0/etc/tls/x509/ca/ca_list1.pem",
    "ASPInstallDir/asp/6.0/etc/tls/x509/ca/ca_list_extra.pem"];
```

The purpose of having more than one file containing a CA list is for administrative convenience. It enables you to group CAs into different lists and to select a particular set of CAs for a security domain by choosing the appropriate CA lists.

policies:security_server

The policies:security_server namespace contains the following variables:

- client_certificate_constraints

client_certificate_constraints

Restricts access to the Artix security server, allowing only clients that match the specified certificate constraints to open a connection to the security service. For details of how to specify certificate constraints, see "Applying Constraints to Certificates" on page 119.

For example, by inserting the following setting into the security service's configuration scope in the Artix configuration file, you can allow access by clients presenting the administrator.p12 and iona_utilities.p12 certificates (demonstration certificates).

```
# Allow access by demonstration client certificates.
# WARNING: These settings are NOT secure and must be customized
#          before deploying in a real system.
#
policies:security_server:client_certificate_constraints =
    ["C=US,ST=Massachusetts,O=ABigBank*,CN=Orbix2000 IONA
    Services (demo cert), OU=Demonstration Section -- no warranty
    --", "C=US,ST=Massachusetts,O=ABigBank*,CN=Abigbank Accounts
    Server*", "C=US,ST=Massachusetts,O=ABigBank*,CN=Iona
    utilities - demo purposes"];
```

The effect of setting this configuration variable is slightly different to the effect of setting policies:iiop_tls:certificate_constraints_policy. Whereas policies:iiop_tls:certificate_constraints_policy affects *all* services deployed in the current process, the policies:security_server:client_certificate_constraints variable affects only the Artix security service. This distinction is significant when the login server is deployed into the same process as the security server. In this case, you would typically want to configure the login server such that it does *not* require clients to present an X.509 certificate (this is the default), while the security server *does* require clients to present an X.509 certificate.

This configuration variable must be set in the security server's configuration scope, otherwise the security server will not start.

principal_sponsor

The principal_sponsor namespace stores configuration information to be used when obtaining credentials. the CORBA binding provides an implementation of a principal sponsor that creates credentials for applications automatically.

Use of the PrincipalSponsor is disabled by default and can only be enabled through configuration.

The PrincipalSponsor represents an entry point into the secure system. It must be activated and authenticate the user, before any application-specific logic executes. This allows unmodified, security-unaware applications to have Credentials established transparently, prior to making invocations.

In this section

The following variables are in this namespace:

- use_principal_sponsor
- auth_method_id
- auth_method_data
- callback_handler:ClassName
- login_attempts

use_principal_sponsor

use_principal_sponsor specifies whether an attempt is made to obtain credentials automatically. Defaults to false. If set to true, the following principal_sponsor variables must contain data in order for anything to actually happen.

auth_method_id

`auth_method_id` specifies the authentication method to be used. The following authentication methods are available:

`pkcs12_file` The authentication method uses a PKCS#12 file.

`pkcs11` Java only. The authentication data is provided by a smart card.

`security_label` Windows and Schannel only. The authentication data is specified by supplying the common name (CN) from an application certificate's subject DN.

For example, you can select the `pkcs12_file` authentication method as follows:

```
principal_sponsor:auth_method_id = "pkcs12_file";
```

auth_method_data

`auth_method_data` is a string array containing information to be interpreted by the authentication method represented by the `auth_method_id`.

For the `pkcs12_file` authentication method, the following authentication data can be provided in `auth_method_data`:

`filename` A PKCS#12 file that contains a certificate chain and private key—*required*.

`password` A password for the private key—*optional*.

It is bad practice to supply the password from configuration for deployed systems. If the password is not supplied, the user is prompted for it.

`password_file` The name of a file containing the password for the private key—*optional*.

This option is not recommended for deployed systems.

For the `pkcs11` (smart card) authentication method, the following authentication data can be provided in `auth_method_data`:

`provider` A name that identifies the underlying PKCS #11 toolkit used by Orbix to communicate with the smart card.

The toolkit currently used by Orbix has the provider name `dkck132.dll` (from Baltimore).

`slot` The number of a particular slot on the smart card (for example, `0`) containing the user's credentials.

`pin` A PIN to gain access to the smart card—*optional*.

It is bad practice to supply the PIN from configuration for deployed systems. If the PIN is not supplied, the user is prompted for it.

For the `security_label` authentication method on Windows, the following authentication data can be provided in `auth_method_data`:

`label` (Windows and Schannel only.) The common name (CN) from an application certificate's subject DN

For example, to configure an application on Windows to use a certificate, `bob.p12`, whose private key is encrypted with the `bobpass` password, set the `auth_method_data` as follows:

```
principal_sponsor:auth_method_data =
    ["filename=c:\users\bob\bob.p12", "password=bobpass"];
```

The following points apply to Java implementations:

- If the file specified by `filename=` is not found, it is searched for on the classpath.
- The file specified by `filename=` can be supplied with a URL instead of an absolute file location.
- The mechanism for prompting for the password if the password is supplied through `password=` can be replaced with a custom mechanism, as demonstrated by the `login` demo.

- There are two extra configuration variables available as part of the principal_sponsor namespace, namely principal_sponsor:callback_handler and principal_sponsor:login_attempts. These are described below.
- These Java-specific features are available subject to change in future releases; any changes that can arise probably come from customer feedback on this area.

callback_handler:ClassName

callback_handler:ClassName specifies the class name of an interface that implements the interface com.iona.corba.tls.auth.CallbackHandler. This variable is only used for Java clients.

login_attempts

login_attempts specifies how many times a user is prompted for authentication data (usually a password). It applies for both internal and custom CallbackHandlers; if a CallbackHandler is supplied, it is invoked upon up to login_attempts times as long as the PrincipalAuthenticator returns SecAuthFailure. This variable is only used by Java clients.

principal_sponsor:csi

The principal_sponsor:csi namespace stores configuration information to be used when obtaining CSI (Common Secure Interoperability) credentials. It includes the following:

- use_existing_credentials
- use_principal_sponsor
- auth_method_data
- auth_method_id

use_existing_credentials

A boolean value that specifies whether ORBs that share credentials can also share CSI credentials. If true, any CSI credentials loaded by one credential-sharing ORB can be used by other credential-sharing ORBs loaded after it; if false, CSI credentials are not shared.

This variable has no effect, unless the plugins:security:share_credentials_across_orbs variable is also true.

Default is false.

use_principal_sponsor

use_principal_sponsor is a boolean value that switches the CSI principal sponsor on or off.

If set to true, the CSI principal sponsor is enabled; if false, the CSI principal sponsor is disabled and the remaining principal_sponsor:csi variables are ignored. Defaults to false.

auth_method_data

auth_method_data is a string array containing information to be interpreted by the authentication method represented by the auth_method_id.

For the GSSUPMech authentication method, the following authentication data can be provided in auth_method_data:

username The username for CSIv2 authorization. This is optional. Authentication of CSIv2 usernames and passwords is performed on the server side. The administration of usernames depends on the particular security mechanism that is plugged into the server side see auth_over_transport:authentication_service.

password The password associated with username. This is optional. It is bad practice to supply the password from configuration for deployed systems. If the password is not supplied, the user is prompted for it.

domain The CSIv2 authentication domain in which the username/password pair is authenticated.

When the client is about to open a new connection, this domain name is compared with the domain name embedded in the relevant IOR (see policies:csi:auth_over_transport:server_domain_name). The domain names must match.

Note: If domain is an empty string, it matches any target domain. That is, an empty domain string is equivalent to a wildcard.

If any of the preceding data are omitted, the user is prompted to enter authentication data when the application starts up.

For example, to log on to a CSIv2 application as the administrator user in the US-SantaClara domain:

```
principal_sponsor:csi:auth_method_data =
    ["username=administrator", "domain=US-SantaClara"];
```

When the application is started, the user is prompted for the administrator password.

Note: It is currently not possible to customize the login prompt associated with the CSIv2 principal sponsor. As an alternative, you could implement your own login GUI by programming and pass the user input directly to the principal authenticator.

auth_method_id

`auth_method_id` specifies a string that selects the authentication method to be used by the CSI application. The following authentication method is available:

`GSSUPMech`	The Generic Security Service Username/Password (GSSUP) mechanism.

For example, you can select the GSSUPMech authentication method as follows:

```
principal_sponsor:csi:auth_method_id = "GSSUPMech";
```

principal_sponsor:https

The principal_sponsor:https namespace provides configuration variables that enable you to specify the *own credentials* used with the HTTPS transport. The variables in the principal_sponsor:https namespace (which are specific to the HTTPS protocol) have precedence over the analogous variables in the principal_sponsor namespace.

Use of the PrincipalSponsor is disabled by default and can only be enabled through configuration.

The PrincipalSponsor represents an entry point into the secure system. It must be activated and authenticate the user, before any application-specific logic executes. This allows unmodified, security-unaware applications to have Credentials established transparently, prior to making invocations.

In this section

The following variables are in this namespace:

- use_principal_sponsor
- auth_method_id
- auth_method_data

use_principal_sponsor

use_principal_sponsor specifies whether an attempt is made to obtain credentials automatically. Defaults to false. If set to true, the following principal_sponsor:https variables must contain data in order for anything to actually happen:

- auth_method_id
- auth_method_data

auth_method_id

auth_method_id specifies the authentication method to be used. The following authentication methods are available:

pkcs12_file The authentication method uses a PKCS#12 file

For example, you can select the pkcs12_file authentication method as follows:

```
principal_sponsor:auth_method_id = "pkcs12_file";
```

auth_method_data

auth_method_data is a string array containing information to be interpreted by the authentication method represented by the auth_method_id.

For the pkcs12_file authentication method, the following authentication data can be provided in auth_method_data:

filename A PKCS#12 file that contains a certificate chain and private key—*required*.

password A password for the private key—*optional*.

It is bad practice to supply the password from configuration for deployed systems. If the password is not supplied, the user is prompted for it.

password_file The name of a file containing the password for the private key—*optional*.

This option is not recommended for deployed systems.

For example, to configure an application on Windows to use a certificate, bob.p12, whose private key is encrypted with the bobpass password, set the auth_method_data as follows:

```
principal_sponsor:auth_method_data =
   ["filename=c:\users\bob\bob.p12", "password=bobpass"];
```

CHAPTER 4

CORBA

When using the CORBA transport, Artix behaves like an Orbix C++ application. This means that you can specify the Orbix configuration variables that apply to the CORBA-based plug-ins used by Artix.

Note: The variables described in this chapter only apply when Artix is using the CORBA transport.

In this chapter

The following CORBA-based variables are discussed in this chapter:

plugins:codeset

The variables in this namespace specify the codesets used by the CORBA portion of Artix. This is useful when internationalizing your environment. This namespace includes the following variables:

- `char:ncs`
- `char:ccs`
- `wchar:ncs`
- `wchar:ccs`
- `always_use_default`

char:ncs

`char:ncs` specifies the native codeset to use for narrow characters. The default setting is determined as follows:

Table 7: *Defaults for the native narrow codeset*

Platform/Locale	Language	Setting
non-MVS, Latin-1 locale	C++	ISO-8859-1
MVS	C++	EBCDIC
ISO-8859-1/Cp-1292/US-ASCII locale	Java	ISO-8859-1
Shift_JS locale	Java	UTF-8
EUC-JP locale	Java	UTF-8
other	Java	UTF-8

char:ccs

`char:ccs` specifies the list of conversion codesets supported for narrow characters. The default setting is determined as follows:

Table 8: *Defaults for the narrow conversion codesets*

Platform/Locale	Language	Setting
non-MVS, Latin-1 locale	C++	
MVS	C++	IOS-8859-1
ISO-8859-1/Cp-1292/US-ASCII locale	Java	UTF-8
Shift_JIS locale	Java	Shift_JIS, euc_JP, ISO-8859-1
EUC-JP locale	Java	euc_JP, Shift_JIS, ISO-8859-1
other	Java	file encoding, ISO-8859-1

wchar:ncs

`wchar:ncs` specifies the native codesets supported for wide characters. The default setting is determined as follows:

Table 9: *Defaults for the wide native codesets*

Platform/Locale	Language	Setting
non-MVS, Latin-1 locale	C++	UCS-2, UCS-4
MVS	C++	UCS-2, UCS-4
ISO-8859-1/Cp-1292/US-ASCII locale	Java	UTF-16
Shift_JIS locale	Java	UTF-16

Table 9: *Defaults for the wide native codesets*

Platform/Locale	Language	Setting
EUC-JP locale	Java	UTF-16
other	Java	UTF-16

wchar:ccs

`wchar:ccs` specifies the list of conversion codesets supported for wide characters. The default setting is determined as follows:

Table 10: *Defaults for the narrow conversion codesets*

Platform/Locale	Language	Setting
non-MVS, Latin-1 locale	C++	UTF-16
MVS	C++	UTF-16
ISO-8859-1/Cp-1292/US-ASCII locale	Java	UCS-2
Shift_JIS locale	Java	UCS-2, Shift_JIS,euc_JP
EUC-JP locale	Java	UCS-2, euc_JP, Shift_JIS
other	Java	file encoding, UCS-2

always_use_default

`always_use_default` specifies that hardcoded default values will be used and any `codeset` variables will be ignored if they are in the same configuration scope or higher.

plugins:giop

This namespace contains the plugins:giop:message_server_binding_list configuration variable, which is one of the variables used to configure bidirectional GIOP. This feature allows callbacks to be made using a connection opened by the client, instead of requiring the server to open a new connection for the callback.

message_server_binding_list

plugins:giop:message_server_binding_list specifies a list message inceptors that are used for bidirectional GIOP. On the client-side, the plugins:giop:message_server_binding_list must be configured to indicate that an existing outgoing message interceptor chain may be re-used for an incoming server binding, similarly by including an entry for BiDir_GIOP, for example:

```
plugins:giop:message_server_binding_list=["BiDir_GIOP","GIOP" ];
```

Further information

For details of all the steps involved in setting bidirectional GIOP, see the *Orbix Administrator's Guide*.

plugins:giop_snoop

The variables in this namespace configure settings for the GIOP Snoop tool. This tool intercepts and displays GIOP message content. Its primary roles are as a protocol-level monitor and a debug aid.

The GIOP Snoop plug-in implements message-level interceptors that can participate in client and/or server side bindings over any GIOP-based transport.

The variables in the giop_snoop namespace include the following:

- filename
- rolling_file
- verbosity

filename

plugins:giop_snoop:filename specifies a file for GIOP Snoop output. By default, output is directed to standard error (stderr). This variable has the following format:

```
plugins:giop_snoop:filename = "<some-file-path>";
```

A *month/day/year* time stamp is included in the output filename with the following general format:

```
<filename>.MMDDYYYY
```

rolling_file

plugins:giop_snoop:rolling_file prevents the GIOP Snoop output file from growing indefinitely. This setting specifies to open and then close the output file for each snoop message trace, instead of holding the output files open. This enables administrators to control the size and content of output files. This setting is enabled with:

```
plugins:giop_snoop:rolling_file = "true";
```

verbosity

`plugins:giop_snoop:verbosity` is used to control the verbosity levels of the GIOP Snoop output. For example:

```
plugins:giop_snoop:verbosity = "1";
```

GIOP Snoop verbosity levels are as follows:

1	LOW
2	MEDIUM
3	HIGH
4	VERY HIGH

plugins:http

The variables in this namespace configure the HTTP transport.This namespace contains the following variables:

- connection:max_unsent_data
- incoming_connections:hard_limit
- incoming_connections:soft_limit
- ip:send_buffer_size
- ip:receive_buffer_size
- ip:reuse_addr
- outgoing_connections:hard_limit
- outgoing_connections:soft_limit
- pool:max_threads
- pool:min_threads
- tcp_connection:keep_alive
- tcp_connection:no_delay
- tcp_connection:linger_on_close
- tcp_listener:reincarnate_attempts

connection:max_unsent_data

connection:max_unsent_data specifies, in bytes, the upper limit for the amount of unsent data associated with an individual connection. Defaults to 512Kb.

incoming_connections:hard_limit

incoming_connections:hard_limit specifies the maximum number of incoming (server-side) connections permitted to HTTP. HTTP does not accept new connections above this limit. Defaults to -1 (disabled).

incoming_connections:soft_limit

incoming_connections:soft_limit sets the number of connections at which HTTP begins closing incoming (server-side) connections. Defaults to -1 (disabled).

ip:send_buffer_size

ip:send_buffer_size specifies the SO_SNDBUF socket options to control how the IP stack adjusts the size of the output buffer. Defaults to 0, meaning the that buffer size is static.

ip:receive_buffer_size

ip:receive_buffer_size specifies the SO_RCVBUF socket options to control how the IP stack adjusts the size of the input buffer. Defaults to 0, meaning the that buffer size is static.

ip:reuse_addr

ip:reuse_addr specifies whether a process can be launched on an already used port. The default on Windows false. An exception indicating that the address is already in use will be thrown.

The default on UNIX is true. This allows a process to listen on the same port.

outgoing_connections:hard_limit

outgoing_connections:hard_limit sets the maximum number of outgoing (client-side) connections permitted to HTTP. HTTP does not allow new outgoing connections above this limit. Defaults to -1 (disabled).

outgoing_connections:soft_limit

`outgoing_connections:soft_limit` specifies the number of connections at which HTTP begins closing outgoing (client-side) connections. Defaults to -1 (disabled).

pool:max_threads

`pool:max_threads` specifies the maximum number of threads reserved from the `WorkQueue` to support tasks working on behalf of the ATLI transport. Defaults to 5.

pool:min_threads

`pool:min_threads` specifies the minimum number of threads reserved from the `WorkQueue` to support tasks working on behalf of the ATLI transport. Defualts to 1.

tcp_connection:keep_alive

`tcp_connection:keep_alive` specifies the setting of `SO_KEEPALIVE` on sockets used to maintain HTTP connections. If set to `TRUE`, the socket will send a *'keepalive probe'* to the remote host if the conneciton has been idle for a preset period of time. The remote system, if it is still running, will send an `ACK` response. Defaults to `TRUE`.

tcp_connection:no_delay

`tcp_connection:no_deplay` specifies if `TCP_NODELAY` is set on the sockets used to maintain HTTP connections. If set to false, small data packets are collected and sent as a group. The algorithm used allows for no more than a 0.2 msec delay between collected packets. Defaults to `TRUE`.

tcp_connection:linger_on_close

tcp_connection:linger_on_close specifies the setting of SO_LINGER on all tcp connections to ensure that tcp buffers get cleared once a socket is closed. Defaults to TRUE.

tcp_listener:reincarnate_attempts

tcp_listnener:reincarnate_attempts specifies the number of times that a Listener recreate its listener socket after recieving a SocketException. This configuration varaible only effects Java applications. Defaults to 1.

plugins:iiop

The variables in this namespace configure active connection management, IIOP buffer management. For more information about active connection management, see the *Orbix Administrator's Guide*.

This namespace contains the following variables:

- `connection:max_unsent_data`
- `incoming_connections:hard_limit`
- `incoming_connections:soft_limit`
- `ip:send_buffer_size`
- `ip:receive_buffer_size`
- `ip:reuse_addr`
- `outgoing_connections:hard_limit`
- `outgoing_connections:soft_limit`
- `pool:max_threads`
- `pool:min_threads`
- `tcp_connection:keep_alive`
- `tcp_connection:no_delay`
- `tcp_connection:linger_on_close`
- `tcp_listener:reincarnate_attempts`
- `tcp_listener:reincarnation_retry_backoff_ratio`
- `tcp_listener:reincarnation_retry_delay`

connection:max_unsent_data

`plugins:iiop:connection:max_unsent_data` specifies the upper limit for the amount of unsent data associated with an individual connection. Defaults to 512k.

incoming_connections:hard_limit

`plugins:iiop:incoming_connections:hard_limit` specifies the maximum number of incoming (server-side) connections permitted to IIOP. IIOP does not accept new connections above this limit. Defaults to -1 (disabled).

incoming_connections:soft_limit

`plugins:iiop:incoming_connections:soft_limit` sets the number of connections at which IIOP begins closing incoming (server-side) connections. Defaults to -1 (disabled).

ip:send_buffer_size

`plugins:iiop:ip:send_buffer_size` specifies the `SO_SNDBUF` socket options to control how the IP stack adjusts the size of the output buffer. Defaults to 0, meaning the that buffer size is static.

ip:receive_buffer_size

`plugins:iiop:ip:receive_buffer_size` specifies the `SO_RCVBUF` socket options to control how the IP stack adjusts the size of the input buffer. Defaults to 0, meaning the that buffer size is static.

ip:reuse_addr

`plugins:iiop:ip:reuse_addr` specifies whether a process can be launched on an already used port. The default on Windows `false`. An exception indicating that the address is already in use will be thrown.

The default on UNIX is `true`. This allows a process to listen on the same port.

outgoing_connections:hard_limit

plugins:iiop:outgoing_connections:hard_limit sets the maximum number of outgoing (client-side) connections permitted to IIOP. IIOP does not allow new outgoing connections above this limit. Defaults to -1 (disabled).

outgoing_connections:soft_limit

plugins:iiop:outgoing_connections:soft_limit specifies the number of connections at which IIOP begins closing outgoing (client-side) connections. Defaults to -1 (disabled).

pool:max_threads

plugins:iiop:pool:max_threads specifies the maximum number of threads reserved from the WorkQueue to support tasks working on behalf of the ATLI transport. Defaults to 5.

pool:min_threads

plugins:iiop:pool:min_threads specifies the minimum number of threads reserved from the WorkQueue to support tasks working on behalf of the ATLI transport. Defualts to 1.

tcp_connection:keep_alive

plugins:iiop:tcp_connection:keep_alive specifies the setting of SO_KEEPALIVE on sockets used to maintain IIOP connections. If set to TRUE, the socket will send a *'keepalive probe'* to the remote host if the conneciton has been idle for a preset period of time. The remote system, if it is still running, will send an ACK response. Defaults to TRUE.

tcp_connection:no_delay

`plugins:iiop:tcp_connection:no_deplay` specifies if `TCP_NODELAY` is set on the sockets used to maintain IIOP connections. If set to false, small data packets are collected and sent as a group. The algorithm used allows for no more than a 0.2 msec delay between collected packets. Defaults to `TRUE`.

tcp_connection:linger_on_close

`plugins:iiop:tcp_connection:linger_on_close` specifies the setting of `SO_LINGER` on all tcp connections to ensure that tcp buffers get cleared once a socket is closed. Defaults to `TRUE`.

tcp_listener:reincarnate_attempts

(C++/Windows only)

`plugins:iiop:tcp_listener:reincarnate_attempts` specifies the number of attempts that are made to reincarnate a listener before giving up, logging a fatal error, and shutting down the ORB. Datatype is `long`. Defaults to `0` (no attempts).

Sometimes an network error may occur, which results in a listening socket being closed. On Windows, you can configure the listener to attempt a reincarnation. This enables new connections to be established.

tcp_listener:reincarnation_retry_backoff_ratio

(C++/Windows only)

`plugins:iiop:tcp_listener:reincarnation_retry_delay` specifies a delay between reincarnation attempts. Data type is `long`. Defaults to `0` (no delay).

tcp_listener:reincarnation_retry_delay

(C++/Windows only)

`plugins:iiop:tcp_listener:reincarnation_retry_backoff_ratio` specifies the degree to which delays between retries increase from one retry to the next. Datatype is `long`. Defaults to `1`.

plugins:naming

The variables in this namespace configure the naming service plugin. The naming service allows you to associate abstract names with CORBA objects, enabling clients to locate your objects.

This namespace contains the following variables:

- destructive_methods_allowed
- direct_persistence
- iiop:port
- lb_default_initial_load
- lb_default_load_timeout
- nt_service_dependencies

destructive_methods_allowed

destructive_methods_allowed specifies if users can make destructive calls, such as destroy(), on naming service elements. The default value is true, meaning the destructive methods are allowed.

direct_persistence

direct_persistence specifies if the service runs using direct or indirect persistence. The default value is false, meaning indirect persistence.

iiop:port

iiop:port specifies the port that the service listens on when running using direct persistence.

lb_default_initial_load

`lb_default_initial_load` specifies the default initial load value for a member of an active object group. The load value is valid for a period of time specified by the timeout assigned to that member. Defaults to `0.0`. For more information, see the *Orbix Administrator's Guide*.

lb_default_load_timeout

`lb_default_load_timeout` specifies the default load timeout value for a member of an active object group. The default value of -1 indicates no timeout. This means that the load value does not expire. For more information, see the *Orbix Administrator's Guide*.

nt_service_dependencies

`nt_service_dependencies` specifies the naming service's dependencies on other NT services. The dependencies are listed in the following format:

```
IT ORB-name domain-name
```

This variable only has meaning if the naming service is installed as an NT service.

plugins:ots

The variables in this namespace configure the object transaction service (OTS) generic plugin. The generic OTS plugin contains client and server side transaction interceptors and the implementation of `CosTransactions::Current`. For details of this plugin, refer to the *CORBA OTS Guide*.

The `plugins:ots` namespace contains the following variables:

- `default_ots_policy`
- `default_transaction_policy`
- `default_transaction_timeout`
- `interposition_style`
- `jit_transactions`
- `ots_v11_policy`
- `propagate_separate_tid_optimization`
- `rollback_only_on_system_ex`
- `support_ots_v11`
- `transaction_factory_name`

default_ots_policy

`default_ots_policy` specifies the default `OTSPolicy` value used when creating a POA. Set to one of the following values:

```
requires
forbids
adapts
```

If no value is specified, no `OTSPolicy` is set for new POAs.

default_transaction_policy

`default_transaction_policy` specifies the default `TransactionPolicy` value used when creating a POA.

Set to one of the following values:

- requires corresponds to a TransactionPolicy value of Requires_shared.
- allows corresponds to a TransactionPolicy value of Allows_shared.

If no value is specified, no TransactionPolicy is set for new POAs.

default_transaction_timeout

default_transaction_timeout specifies the default timeout, in seconds, of a transaction created using CosTransactions::Current. A value of zero or less specifies no timeout. Defaults to 30 seconds.

interposition_style

interposition_style specifies the style of interposition used when a transaction first visits a server. Set to one of the following values:

- standard: A new subordinator transaction is created locally and a resource is registered with the superior coordinator. This subordinate transaction is then made available through the Current object.
- proxy: (default) A locally constrained proxy for the imported transaction is created and made available though the Current object.

Proxy interposition is more efficient, but if you need to further propagate the transaction explicitly (using the Control object), standard interposition must be specified.

jit_transactions

jit_transactions is a boolean which determines whether to use just-in-time transaction creation. If set to true, transactions created using Current::begin() are not actually created until necessary. This can be used in conjunction with an OTSPolicy value of SERVER_SIDE to delay creation of a transaction until an invocation is received in a server. Defaults to false.

ots_v11_policy

ots_v11_policy specifies the effective OTSPolicy value applied to objects determined to support CosTransactions::TransactionalObject, if support_ots_v11 is set to true.

Set to one of the following values:

- adapts
- requires

propagate_separate_tid_optimization

propagate_separate_tid_optimization specifies whether an optimization is applied to transaction propagation when using C++ applications. Must be set for both the sender and receiver to take affect. Defaults to true.

rollback_only_on_system_ex

rollback_only_on_system_ex specifies whether to mark a transaction for rollback if an invocation on a transactional object results in a system exception being raised. Defaults to true.

support_ots_v11

support_ots_v11 specifies whether there is support for the OMG OTS v1.1 CosTransactions::TransactionalObject interface. This option can be used in conjunction with ots_v11_policy. When this option is enabled, the OTS interceptors might need to use remote _is_a() calls to determine the type of an interface. Defaults to false.

transaction_factory_name

transaction_factory_name specifies the initial reference for the transaction factory. This option must match the corresponding entry in the configuration scope of your transaction service implementation. Defaults to TransactionFactory.

plugins:ots_lite

The variables in this namespace configure the Lite implementation of the object transaction service. The `ots_lite` plugin contains an implementation of `CosTransacitons::TransactionFactory` which is optimized for use in a single resource system. For details, see the *CORBA Programmer's Guide*.

This namespace contains the following variables:

- `orb_name`
- `otid_format_id`
- `superior_ping_timeout`
- `transaction_factory_name`
- `transaction_timeout_period`
- `use_internal_orb`

orb_name

`orb_name` specifies the ORB name used for the plugin's internal ORB when `use_internal_orb` is set to `true`. The ORB name determines where the ORB obtains its configuration information and is useful when the application ORB configuration needs to be different from that of the internal ORB. Defaults to the ORB name of the application ORB.

otid_format_id

`otid_format_id` specifies the value of the `formatID` field of a transaction's identifier (`CosTransactions::otid_t`). Defaults to `0x494f4e41`.

superior_ping_timeout

`superior_ping_timeout` specifies, in seconds, the timeout between queries of the transaction state, when standard interposition is being used to recreate a foreign transaction. The interposed resource periodically queries the recovery coordinator, to ensure that the transaction is still alive when the timeout of the superior transaction has expired. Defaults to `30`.

transaction_factory_name

`transaction_factory_name` specifies the initial reference for the transaction factory. This option must match the corresponding entry in the configuration scope of your generic OTS plugin to allow it to successfully resolve a transaction factory. Defaults to `TransactionFactory`.

transaction_timeout_period

`transaction_timeout_period` specifies the time, in milliseconds, of which all transaction timeouts are multiples. A low value increases accuracy of transaction timeouts, but increases overhead. This value is added to all transaction timeouts. To disable all timeouts, set to `0` or a negative value. Defaults to `1000`.

use_internal_orb

`use_internal_orb` specifies whether the ots_lite plugin creates an internal ORB for its own use. By default, `ots_lite` creates POAs in the application's ORB. This option is useful if you want to isolate the transaction service from your application ORB. Defaults to `false`.

plugins:ots_encina

The plugins:ots_encina namespace stores configuration variables for the Encina OTS plugin. The ots_encina plugin contains an implementation of IDL interface CosTransactions::TransactionFactory that supports the recoverable 2PC protocol. For details, see the *CORBA OTS Guide*.

This namespace contains the following variables:

- agent_ior_file
- allow_registration_after_rollback_only
- backup_restart_file
- direct_persistence
- direct_persistence
- global_namespace_poa
- iiop:port
- initial_disk
- initial_disk_size
- log_threshold
- log_check_interval
- max_resource_failures
- namespace_poa
- orb_name
- otid_format_id
- resource_retry_timeout
- restart_file
- trace_comp
- trace_file
- trace_on
- transaction_factory_name
- transaction_factory_ns_name
- transaction_timeout_period
- use_internal_orb
- use_raw_disk

agent_ior_file

agent_ior_file specifies the file path where the management agent object's IOR is written. Defaults to an empty string.

allow_registration_after_rollback_only

allow_registration_after_rollback_only (C++ only) specifies whether registration of resource objects is permitted after a transaction is marked for rollback.

- true specifies that resource objects can be registered after a transaction is marked for rollback.
- false (default) specifies that resource objects cannot be registered once a transaction is marked for rollback.

This has no effect on the outcome of the transaction.

backup_restart_file

backup_restart_file specifies the path for the backup restart file used by the Encina OTS to locate its transaction logs. If unspecified, the backup restart file is the name of the primary restart file—set with restart_file—with a .bak suffix. Defaults to an empty string.

direct_persistence

direct_persistence specifies whether the transaction factory object can use explicit addressing—for example, a fixed port. If set to true, the addressing information is picked up from plugins:ots_encina. For example, to use a fixed port, set plugins_ots_encina:iiop:port. Defaults to false.

global_namespace_poa

global_namespace_poa specifies the top-level transient POA used as a namespace for OTS implementations. Defaults to iOTS.

iiop:port

iiop:port specifies the port that the service listens on when using direct persistence.

initial_disk

initial_disk specifies the path for the initial file used by the Encina OTS for its transaction logs. Defaults to an empty string.

initial_disk_size

initial_disk_size specifies the size of the initial file used by the Encina OTS for its transaction logs. Defaults to 2.

log_threshold

log_threshold specifies the percentage of transaction log space, which, when exceeded, results in a management event. Must be between 0 and 100. Defaults to 90.

log_check_interval

log_check_interval specifies the time, in seconds, between checks for transaction log growth. Defaults to 60.

max_resource_failures

max_resource_failures specifies the maximum number of failed invocations on CosTransaction::Resource objects to record. Defaults to 5.

namespace_poa

namespace_poa specifies the transient POA used as a namespace. This is useful when there are multiple instances of the plugin being used; each instance must use a different namespace POA to distinguish itself. Defaults to Encina.

orb_name

orb_name specifies the ORB name used for the plugin's internal ORB when use_internal_orb is set to true. The ORB name determines where the ORB obtains its configuration information, and is useful when the application ORB configuration needs to be different from that of the internal ORB. Defaults to the ORB name of the application ORB.

otid_format_id

otis_format_id specifies the value of the formatID field of a transaction's identifier (CosTransactions::otid_t). Defaults to 0x494f4e41.

resource_retry_timeout

resource_retry_timeout specifies the time, in seconds, between retrying a failed invocation on a resource object. A negative value means the default is used. Defaults to 5.

restart_file

restart_file specifies the path for the restart file used by the Encina OTS to locate its transaction logs. Defaults to an empty string.

trace_comp

`trace_comp` sets the Encina trace levels for the component *comp*, where *comp* is one of the following:

```
bde
log
restart
tran
tranLog_log
tranLog_tran
util
vol
```

Set this variable to a bracket-enclosed list that includes one or more of the following string values:

- `event`: interesting events.
- `entry`: entry to a function.
- `param`: parameters to a function.
- `internal_entry`: entry to internal functions.
- `internal_param`: parameters to internal functions.
- `global`.

Defaults to `[]`.

trace_file

`trace_file` specifies the file to which Encina level tracing is written when enabled via `trace_on`. If not set or set to an empty string, Encina level transactions are written to standard error. Defaults to an empty string.

trace_on

`trace_on` specifies whether Encina level tracing is enabled. If set to `true`, the information that is output is determined from the trace levels (see `trace_comp`). Defaults to `false`.

transaction_factory_name

transaction_factory_name specifies the initial reference for the transaction factory. This option must match the corresponding entry in the configuration scope of your generic OTS plugin to allow it to successfully resolve a transaction factory. Defaults to TransactionFactory.

transaction_factory_ns_name

transaction_factory_ns_name specifies the name used to publish the transaction factory reference in the naming service. Defaults to an empty string.

transaction_timeout_period

transaction_timeout_period specifies the time, in milliseconds, of which all transaction timeouts are multiples. A low value increases accuracy of transaction timeouts, but increases overhead. This value multiplied to all transaction timeouts. To disable all timeouts, set to 0 or a negative value. Defaults to 1000.

use_internal_orb

use_internal_orb specifies whether the ots_encina plugin creates an internal ORB for its own use. By default the ots_encina plugin creates POA's in the application's ORB. This option is useful if you want to isolate the transaction service from your application ORB. Defaults to false.

use_raw_disk

use_raw_disk specifies whether the path specified by initial_disk is of a raw disk (true) or a file (false). If set to false and the file does not exist, the Encina OTS plugin tries to create the file with the size specified in initial_disk_size. Defaults to false.

plugins:poa

This namespace contains variables to configure the CORBA POA plug-in. It contains the following variables:

- root_name

root_name

root_name specifies the name of the root POA, which is added to all fully-qualified POA names generated by that POA. If this variable is not set, the POA treats the root as an anonymous root, effectively acting as the root of the location domain.

poa:FQPN

The poa namespace includes variables that allow you to use direct persistence and well-known addressing for POAs (Portable Object Adaptors). These variables specify the policy for individual POAs by specifying the fully qualified POA name for each POA. They take the form:

```
poa:FQPN:Variable
```

For example to set the well-known address for a POA whose fully qualified POA name is helloworld you would set the variable poa:helloworld:well_known_address.

The following variables are in this namespace:

- direct_persistent
- well_known_address

direct_persistent

direct_persistent specifies if a POA runs using direct persistence. If this is set to true the POA generates IORs using the well-known address that is specified in the well_known_address varaible. Defaults to false. For an example of how this works, see well_known_address.

well_known_address

well_known_address specifies the address used to generate IORs for the associated POA when that POA's direct_persistent varaible is set to true.

For example, to run your server using direct persistence, and well known addressing, add the following to your configuration:

```
poa:helloworld:direct_persistent = "true";
poa:helloworld:well_known_address = "helloworld_port";
helloworld_port:iiop:port = "9202";
```

This corresponds to the following WSDL:

```
<service name="CorbaService">
   <port binding="corbatm:CorbaBinding" name="CorbaPort">
      <corba:address location="file:../../hello_world_service.ior"/>
          <corba:policy poaname="helloworld"/>
    </port>
</service>
```

Using these configuration variables, all object references created by the `helloworld` POA will now be direct persistent containing the well known IIOP address of port `9202`.

If your POA name is different, the configuration variables must be modified. The scheme used is the following:

```
poa:FQPN:direct_persistent=BOOL;
poa:FQPN:well_known_address=Address_Prefix;
Address_Prefix:iiop:port=LONG;
```

`FQPN` is the fully qualified POA name. This introduces the restriction that your POA name can only contain printable characters, and may not contain white space.

`Address_Prefix` is the string that gets passed to the well-known addressing POA policy. Specify the actual port used using the *`Address_Prefix`*`:iiop:port` variable. You can also use `iiop_tls` instead of `iiop`.

Core Policies

Configuration variables for core policies include:

- non_tx_target_policy
- rebind_policy
- routing_policy_max
- routing_policy_min
- sync_scope_policy
- work_queue_policy

non_tx_target_policy

non_tx_target_policy specifies the default NonTxTargetPolicy value for use when a non-transactional object is invoked within a transaction. Set to one of the following values:

permit	Maps to the NonTxTargetPolicy value PERMIT.
prevent	Maps to the NonTxTargetPolicy value PREVENT.(default)

rebind_policy

rebind_policy specifies the default value for RebindPolicy. Can be one of the following:

TRANSPARENT(default)

NO_REBIND
NO_RECONNECT

routing_policy_max

routing_policy_max specifies the default maximum value for RoutingPolicy. You can set this to one of the following:

ROUTE_NONE(default)

ROUTE_FORWARD
ROUTE_STORE_AND_FORWARD

routing_policy_min

routing_policy_min specifies the default minimum value for RoutingPolicy. You can set this to one of the following:

ROUTE_NONE(default)

ROUTE_FORWARD
ROUTE_STORE_AND_FORWARD

sync_scope_policy

sync_scope_policy specifies the default value for SyncScopePolicy. You can set this to one of the following:

SYNC_NONE

SYNC_WITH_TRANSPORT(default)

SYNC_WITH_SERVER
SYNC_WITH_TARGET

work_queue_policy

work_queue_policy specifies the default WorkQueue to use for dispatching GIOP Requests and LocateRequests when the WorkQueuePolicy is not effective. You can set this variable to a string that is resolved using ORB.resolve_initial_references().

For example, to dispatch requests on the internal multi-threaded work queue, this variable should be set to IT_MultipleThreadWorkQueue. Defaults to IT_DirectDispatchWorkQueue. For more information about WorkQueue policies, see the *CORBA Programmer's Guide*.

CORBA Timeout Policies

Orbix supports standard CORBA timeout policies, to enable clients to abort invocations. IONA also provides proprietary policies, which enable more fine-grained control. Configuration variables for standard CORBA timeout policies include:

- `relative_request_timeout`
- `relative_roundtrip_timeout`

relative_request_timeout

`relative_request_timeout` specifies how much time, in milliseconds, is allowed to deliver a request. Request delivery is considered complete when the last fragment of the GIOP request is sent over the wire to the target object. There is no default value.

The timeout period includes any delay in establishing a binding. This policy type is useful to a client that only needs to limit request delivery time.

relative_roundtrip_timeout

`relative_roundtrip_timeout` specifies how much time, in milliseconds, is allowed to deliver a request and its reply. There is no default value.

The timeout countdown starts with the request invocation, and includes:

- Marshalling in/inout parameters.
- Any delay in transparently establishing a binding.

If the request times out before the client receives the last fragment of reply data, the request is cancelled using a GIOP `CancelRequest` message and all received reply data is discarded.

For more information about standard CORBA timeout policies, see the *CORBA Programmer's Guide*.

IONA Timeout Policies

This section lists configuration variables for the IONA-specific timeout policies, which enable more fine-grained control than the standard CORBA policies. IONA-specific variables in the `policies` namespace include:

- `relative_binding_exclusive_request_timeout`
- `relative_binding_exclusive_roundtrip_timeout`
- `relative_connection_creation_timeout`

relative_binding_exclusive_request_timeout

`relative_binding_exclusive_request_timeout` specifies how much time, in milliseconds, is allowed to deliver a request, exclusive of binding attempts. The countdown begins immediately after a binding is obtained for the invocation. There is no default value.

relative_binding_exclusive_roundtrip_timeout

`relative_binding_exclusive_roundtrip_timeout` specifies how much time, in milliseconds, is allowed to deliver a request and receive its reply, exclusive of binding attempts. There is no default value.

relative_connection_creation_timeout

`relative_connection_creation_timeout` specifies how much time, in milliseconds, is allowed to resolve each address in an IOR, within each binding iteration. Default is 8 seconds.

An IOR can have several `TAG_INTERNET_IOP` (IIOP transport) profiles, each with one or more addresses, while each address can resolve via DNS to multiple IP addresses. Furthermore, each IOR can specify multiple transports, each with its own set of profiles.

This variable applies to each IP address within an IOR. Each attempt to resolve an IP address is regarded as a separate attempt to create a connection.

policies:giop

The variables in this namespace set policies that control the behavior of bidirectional GIOP. This feature allows callbacks to be made using a connection opened by the client, instead of requiring the server to open a new connection for the callback. The `policies:giop` namespace includes the following variables:

- "bidirectional_accept_policy".
- "bidirectional_export_policy".
- "bidirectional_gen3_accept_policy".
- "bidirectional_offer_policy".

bidirectional_accept_policy

`bidirectional_accept_policy` specifies the behavior of the accept policy used in bidirectional GIOP. On the server side, the `BiDirPolicy::BiDirAcceptPolicy` for the callback invocation must be set to `ALLOW`. You can set this in configuration as follows:

```
policies:giop:bidirectional_accept_policy="ALLOW";
```

This accepts the client's bidirectional offer, and uses an incoming connection for an outgoing request, as long the policies effective for the invocation are compatible with the connection.

bidirectional_export_policy

`bidirectional_export_policy` specifies the behavior of the export policy used in birdirectional GIOP. A POA used to activate a client-side callback object must have an effective `BiDirPolicy::BiDirExportPolicy` set to `BiDirPolicy::ALLOW`. You can set this in configuration as follows:

```
policies:giop:bidirectional_export_policy="ALLOW";
```

Alternatively, you can do this programmatically by including this policy in the list passed to `POA::create_POA()`.

bidirectional_gen3_accept_policy

bidirectional_gen3_accept_policy specifies whether interoperability with Orbix 3.x is enabled. Set this variable to ALLOW to enable interoperability with Orbix 3.x:

```
policies:giop:bidirectional_gen3_accept_policy="ALLOW";
```

This allows an Orbix 6.x server to invoke on an Orbix 3.x callback reference in a bidirectional fashion.

bidirectional_offer_policy

bidirectional_offer_policy specifies the behavior of the offer policy used in bidirectional GIOP. A bidirectional offer is triggered for an outgoing connection by setting the effective BiDirPolicy::BiDirOfferPolicy to ALLOW for an invocation. You can set this in configuration as follows:

```
policies:giop:bidirectional_offer_policy="ALLOW";
```

Further information

For more information on all the steps involved in setting bidirectional GIOP, see the *Orbix Administrator's Guide*.

policies:giop:interop_policy

The `policies:giop:interop_policy` child namespace contains variables used to configure interoperability with previous versions of IONA products. It contains the following variables:

- `allow_value_types_in_1_1`
- `enable_principal_service_context`
- `ignore_message_not_consumed`
- `negotiate_transmission_codeset`
- `send_locate_request`
- `send_principal`

allow_value_types_in_1_1

`allow_value_types_in_1_1` relaxes GIOP 1.1 complaince to allow `valuetypes` to be passed by Java ORBs using GIOP 1.1. This functionality can be important when interoperating with older ORBs that do not support GIOP 1.2. To relax GIOP 1.1 compliance, set this variable to `true`.

enable_principal_service_context

`enable_principal_service_context` specifies whether to permit a prinicipal user identifier to be sent in the service context of CORBA requests. This is used to supply an ORB on the mainframe with a user against which basic authorization can take place.

Typically, on the mid-tier, you may want to set the principal to a user that can be authorized on the mainframe. This can be performed on a per-request basis in a portable interceptor. See the *CORBA Programmer's Guide* for how to write portable interceptors.

To enable principal service contexts, set this variable to `true`:

```
policies:giop:interop_policy:enable_principal_service_context="true";
```

ignore_message_not_consumed

`ignore_message_not_consumed` specifies whether to raise `MARSHAL` exceptions when interoperating with ORBs that set message size incorrectly, or with earlier versions of Orbix if it sends piggyback data. The default value is `false`.

The `MARSHAL` exception is set with one of the following minor codes:

- `REQUEST_MESSAGE_NOT_CONSUMED`
- `REPLY_MESSAGE_NOT_CONSUMED`

negotiate_transmission_codeset

`negotiate_transmisission_codeset` specifies whether to enable codeset negotiation for wide characters used by some third-party ORBs, previous versions of Orbix, and OrbixWeb. Defaults to true.

If this variable is set to `true`, native and conversion codesets for `char` and `wchar` are advertised in `IOP::TAG_CODE_SETS` tagged components in published IORs. The transmission codesets are negotiated by clients and transmitted using an `IOP::CodeSets` service context.

If the variable is `false`, negotiation does not occur and Orbix uses transmission codesets of UTF-16 and ISO-Latin-1 for `wchar` and `char` types, respectively. Defaults to `true`.

send_locate_request

`send_locate_request` specifies whether GIOP sends `LocateRequest` messages before sending initial `Request` messages. Required for interoperability with Orbix 3.0. Defaults to `true`.

send_principal

`send_principal` specifies whether GIOP sends `Principal` information containing the current user name in GIOP 1.0 and GIOP 1.1 requests. Required for interoperability with Orbix 3.0 and Orbix for OS/390. Defaults to `false`.

policies:http

This namespace contains variables used to set HTTP-related policies. It contains the following variables:

- buffer_sizes_policy:default_buffer_size
- buffer_sizes_policy:max_buffer_size
- keep-alive:enabled
- server_address_mode_policy:port_range

buffer_sizes_policy:default_buffer_size

buffer_sizes_policy:default_buffer_size specifies, in bytes, the initial size of the buffers allocated by HTTP. Defaults to 4096. This value must be greater than 80 bytes, and must be evenly divisible by 8.

buffer_sizes_policy:max_buffer_size

buffer_sizes_policy:max_buffer_size specifies, in bytes, the maximum buffer size permitted by HTTP. Defaults to -1 which indicates unlimited size. If not unlimited, this value must be greater than 80.

keep-alive:enabled

keep-alive:enabled specifies if the server uses persistent connections in response to an incomming Connection:keep-alive header. If set to true, the server honors the connection setting from the client. If set to false, the server always ignores the connection setting from the client.

If no connection setting is sent from the client and this variable is set to `true`, the server responds with `Connection:close` for HTTP 1.0 requests and `Connection:keep-alive` for HTTP 1.1 requests. Defaults to `false`.

Note: Setting this variable to true does not prevent the server from ultimately choosing to ignore the keep-alive setting for other reasons. For example, if an explicit per client service limit is reached, the server responds with a `Connection:close`, regardless of this variable's setting.

server_address_mode_policy:port_range

`server_address_mode_policy:port_range` specifies the range of ports that a server uses when there is no well-known addressing policy specified for the port.

policies:iiop

The policies:iiop namespace contains variables used to set IIOP-related policies. It contains the following variables:

- client_address_mode_policy:local_hostname
- client_address_mode_policy:port_range
- client_version_policy
- buffer_sizes_policy:default_buffer_size
- buffer_sizes_policy:max_buffer_size
- server_address_mode_policy:local_hostname
- server_address_mode_policy:port_range
- server_address_mode_policy:publish_hostname
- server_version_policy
- tcp_options_policy:no_delay
- tcp_options_policy:recv_buffer_size
- tcp_options_policy:send_buffer_size

client_address_mode_policy:local_hostname

client_address_mode_policy:local_hostname specifies the host name that is used by the client.

This variable enables support for *multi-homed* client hosts. These are client machines with multiple hostnames or IP addresses (for example, those using multiple DNS aliases or multiple network interface cards). The local_hostname variable enables you to explicitly specify the host name that the client listens on.

For example, if you have a client machine with two network addresses (207.45.52.34 and 207.45.52.35), you can explicitly set this variable to either address:

```
policies:iiop:client_address_mode_policy:local_hostname =
   "207.45.52.34";
```

By default, the local_hostname variable is unspecified, and the client uses the 0.0.0.0 wildcard address. In this case, the network interface card used is determined by the operating system.

client_address_mode_policy:port_range

(C++ only) client_address_mode_policy:port_range specifies the range of ports that a client uses when there is no well-known addressing policy specified for the port. Specified values take the format of *from_port*:*to_port*, for example:

```
policies:iiop:client_address_mode_policy:port_range="4003:4008"
```

client_version_policy

client_version_policy specifies the highest GIOP version used by clients. A client uses the version of GIOP specified by this variable, or the version specified in the IOR profile, whichever is lower. Valid values for this variable are: 1.0, 1.1, and 1.2.

For example, the following file-based configuration entry sets the server IIOP version to 1.1.

```
policies:iiop:server_version_policy="1.1";
```

The following itadmin command set this variable:

```
itadmin variable modify -type string -value "1.1"
   policies:iiop:server_version_policy
```

buffer_sizes_policy:default_buffer_size

`buffer_sizes_policy:default_buffer_size` specifies, in bytes, the initial size of the buffers allocated by IIOP. Defaults to 16000. This value must be greater than 80 bytes, and must be evenly divisible by 8.

buffer_sizes_policy:max_buffer_size

`buffer_sizes_policy:max_buffer_size` specifies the maximum buffer size permitted by IIOP, in kilobytes. Defaults to -1, which indicates unlimited size. If not unlimited, this value must be greater than 80.

server_address_mode_policy:local_hostname

`server_address_mode_policy:local_hostname` specifies the server host name that is advertised by the locator daemon, and listened on by server-side IIOP.

This variable enables support for *multi-homed* server hosts. These are server machines with multiple hostnames or IP addresses (for example, those using multiple DNS aliases or multiple network interface cards). The `local_hostname` variable enables you to explicitly specify the host name that the server listens on and publishes in its IORs.

For example, if you have a machine with two network addresses (`207.45.52.34` and `207.45.52.35`), you can explicitly set this variable to either address:

```
policies:iiop:server_address_mode_policy:local_hostname =
    "207.45.52.34";
```

By default, the `local_hostname` variable is unspecified. Servers use the default hostname configured for the machine with the Orbix configuration tool.

server_address_mode_policy:port_range

server_address_mode_policy:port_range specifies the range of ports that a server uses when there is no well-known addressing policy specified for the port. Specified values take the format of *from_port*:*to_port*, for example:

```
policies:iiop:server_address_mode_policy:port_range="4003:4008"
```

server_address_mode_policy:publish_hostname

server_address_mode-policy:publish_hostname specifes whether IIOP exports hostnames or IP addresses in published profiles. Defaults to false (exports IP addresses, and does not export hostnames). To use hostnames in object references, set this variable to true, as in the following file-based configuration entry:

```
policies:iiop:server_address_mode_policy:publish_hostname=true
```

The following itadmin command is equivalent:

```
itadmin variable create -type bool -value true
   policies:iiop:server_address_mode_policy:publish_hostname
```

server_version_policy

server_version_policy specifies the GIOP version published in IIOP profiles. This variable takes a value of either 1.1 or 1.2. Orbix servers do not publish IIOP 1.0 profiles. The default value is 1.2.

tcp_options_policy:no_delay

tcp_options_policy:no_delay specifies whether the TCP_NODELAY option should be set on connections. Defaults to false.

tcp_options_policy:recv_buffer_size

`tcp_options_policy:recv_buffer_size` specifies the size of the TCP receive buffer. This variable can only be set to `0`, which coresponds to using the default size defined by the operating system.

tcp_options_policy:send_buffer_size

`tcp_options_policy:send_buffer_size` specifies the size of the TCP send buffer. This variable can only be set to `0`, which coresponds to using the default size defined by the operating system.

policies:invocation_retry

The policies:invocation_retry namespace contains variables that determine how a CORBA ORB reinvokes or rebinds requests that raise the following exceptions:

- TRANSIENT with a completion status of COMPLETED_NO (triggers transparent reinvocations).
- COMM_FAILURE with a completion status of COMPLETED_NO (triggers transparent rebinding).

This namespace contains the following variables:

- backoff_ratio
- initial_retry_delay
- max_forwards
- max_rebinds
- max_retries

backoff_ratio

backoff_ratio specifies the degree to which delays between invocation retries increase from one retry to the next. Defaults to 2.

initial_retry_delay

initial_retry_delay specifies the amount of time, in milliseconds, between the first and second retries. Defaults to 100.

Note: The delay between the initial invocation and first retry is always 0.

max_forwards

max_forwards specifies the number of forward tries allowed for an invocation. Defaults to 20. To specify unlimited forward tries, set to -1.

max_rebinds

max_rebinds specifies the number of transparent rebinds attempted on receipt of a COMM_FAILURE exception. Defaults to 5.

Note: This setting is valid only if the effective RebindPolicy is TRANSPARENT; otherwise, no rebinding occurs. For more information, see "rebind_policy" on page 236.

max_retries

max_retries specifies the number of transparent reinvocations attempted on receipt of a TRANSIENT exception. Defaults to 5.

For more information about proprietary IONA timeout policies, see the *CORBA Programmer's Guide*.

Index