



Artix 5.6.3

A decorative graphic consisting of several overlapping, wavy blue lines that curve and flow across the lower half of the page, creating a sense of motion and depth.

Configuration
Reference, C++
Runtime

Micro Focus
The Lawn
22-30 Old Bath Road
Newbury, Berkshire RG14 1QN
UK

<http://www.microfocus.com>

Copyright © Micro Focus 2015. All rights reserved.

MICRO FOCUS, the Micro Focus logo and Micro Focus Licensing are trademarks or registered trademarks of Micro Focus IP Development Limited or its subsidiaries or affiliated companies in the United States, United Kingdom and other countries.

All other marks are the property of their respective owners.

2015-02-18

Contents

Preface	vii
Contacting Micro Focus	viii
Artix Runtime	1
ORB Plug-ins	1
Binding Lists	7
Event Log	12
Initial Contracts	16
Initial References	18
JVM Options	22
Message Snoop	22
Multi-threading	24
Policies	27
QName Aliases	34
Reference Compatibility	35
Artix Plug-ins	39
AmberPoint	40
Bus	40
CA WSDM Observer	42
Client-Side High Availability	44
Container	45
Database Environment	45
FTP	52
JMS	54
JMX	57
Local Log Stream	59
Log4J Log Stream	61
Locator Service	61
Locator Endpoint Manager	63
Monitoring	64
Peer Manager	65
Performance Logging	66
Remote Logging	67
Remote Method Invocation	68
Routing	69
Service Lifecycle	72
Session Manager	73
Session Endpoint Manager	74
Session Manager Simple Policy	74
SOAP	75
SOAP 1.2	77
Transformer Service	77
Tuxedo	80
Web Services Addressing	80
Web Services Chain Service	82
Web Services Reliable Messaging	84
WSDL Publishing Service	90
XML File Log Stream	91
Custom Plug-ins	93

Artix Security	95
Applying Constraints to Certificates.....	96
bus: initial_contract	97
bus: security	98
initial_references.....	99
password_retrieval_mechanism.....	99
plugins: asp	100
plugins: at_http	102
plugins: atli2_tls	105
plugins: csi.....	105
plugins: gsp	106
plugins: https	109
plugins: iiop_tls	109
plugins: java_server	111
plugins: login_client.....	114
plugins: login_service	114
plugins: security	114
plugins: security_cluster	116
plugins: wsdl_publish.....	116
plugins: wss	117
policies	118
policies: asp	123
policies: bindings	126
policies: csi	127
policies: external_token_issuer	129
policies: https	129
policies: iiop_tls	134
policies: security_server.....	141
policies: soap: security	141
principal_sponsor	142
principal_sponsor: csi	145
principal_sponsor: http	147
principal_sponsor: https	148
principal_sponsor: iiop_tls.....	149
principal_sponsor: wsse	151
CORBA	153
plugins: codeset.....	153
plugins: giop	155
plugins: giop_snoop	156
plugins: http and https.....	157
plugins: iiop	159
plugins: naming	162
plugins: ots	163
plugins: ots_lite	165
plugins: ots_encina	166
plugins: poa	170
poa: FQPN.....	170
Core Policies	172
CORBA Timeout Policies	173
Artix Timeout Policies.....	174
policies: giop	174
policies: giop: interop_policy	175
policies: http	177
policies: iiop	178
policies: invocation_retry	181

Index.....183

Preface

What is Covered in this Book

The *Artix Configuration Reference, C++ Runtime* provides a comprehensive reference of Artix configuration variables in a C++ runtime environment. These variables are stored in an Artix `.cfg` configuration file.

Who Should Read this Book

This book is intended for use by system administrators, in conjunction with *Configuring and Deploying Artix Solutions, C++ Runtime*. 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:

- “[Artix Runtime](#)” describes the configuration variables for the core Artix runtime (for example, logging and multi-threading).
- “[Artix Plug-ins](#)” describes the configuration variables for specific Artix plug-ins (for example, Artix locator, SOAP, or JMS).
- “[Artix Security](#)” describes the variables used to configure Artix security features (for example, passwords and certificates).
- “[CORBA](#)” describes the variables used to configure CORBA plug-ins (for example, IIOP and OTS).

The Artix Documentation Library

For information on the organization of the Artix library, the document conventions used, and finding additional resources, see *Using the Artix Library*.

Contacting Micro Focus

Our Web site gives up-to-date details of contact numbers and addresses.

Further Information and Product Support

Additional technical information or advice is available from several sources.

The product support pages contain a considerable amount of additional information, such as:

- The WebSync service, where you can download fixes and documentation updates.
- The Knowledge Base, a large collection of product tips and workarounds.
- Examples and Utilities, including demos and additional product documentation.

To connect, enter <http://www.microfocus.com> in your browser to go to the Micro Focus home page.

Note:

Some information may be available only to customers who have maintenance agreements.

If you obtained this product directly from Micro Focus, contact us as described on the Micro Focus Web site, <http://www.microfocus.com>. If you obtained the product from another source, such as an authorized distributor, contact them for help first. If they are unable to help, contact us.

Information We Need

However you contact us, please try to include the information below, if you have it. The more information you can give, the better Micro Focus SupportLine can help you. But if you don't know all the answers, or you think some are irrelevant to your problem, please give whatever information you have.

- The name and version number of all products that you think might be causing a problem.
- Your computer make and model.
- Your operating system version number and details of any networking software you are using.
- The amount of memory in your computer.
- The relevant page reference or section in the documentation.
- Your serial number. To find out these numbers, look in the subject line and body of your Electronic Product Delivery Notice email that you received from Micro Focus.

Contact information

Our Web site gives up-to-date details of contact numbers and addresses.

Additional technical information or advice is available from several sources.

The product support pages contain considerable additional information, including the WebSync service, where you can download fixes and documentation updates. To connect, enter <http://www.microfocus.com> in your browser to go to the Micro Focus home page.

If you are a Micro Focus SupportLine customer, please see your SupportLine Handbook for contact information. You can download it from our Web site or order it in printed form from your sales representative. Support from Micro Focus may be available only to customers who have maintenance agreements.

You may want to check these URLs in particular:

- <http://www.microfocus.com/products/corba/artix.aspx> (trial software download and Micro Focus Community files)
- <https://supportline.microfocus.com/productdoc.aspx> (documentation updates and PDFs)

To subscribe to Micro Focus electronic newsletters, use the online form at:

<http://www.microfocus.com/Resources/Newsletters/infocus/newsletter-subscription.asp>

Artix Runtime

Artix is based on the 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.

This chapter describes the following topics:

- [ORB Plug-ins](#)
- [Binding Lists](#)
- [Event Log](#)
- [Initial Contracts](#)
- [Initial References](#)
- [JVM Options](#)
- [Message Snoop](#)
- [Multi-threading](#)
- [Policies](#)
- [QName Aliases](#)
- [Reference Compatibility](#)

ORB Plug-ins

The `orb_plugins` variable specifies the list of plug-ins that Artix processes load during initialization. A *plug-in* is a DLL or shared 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

Java plug-ins are configured differently from C++ plug-ins. For example, 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 a plug-in factory class, for example:

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

Artix Java plug-ins

The following Java plug-ins are also 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
<code>at_http</code>	Provides support for HTTP.
<code>https</code>	Provides support for HTTPS.
<code>iiop</code>	Provides support for CORBA IIOP.
<code>iiop_profile</code>	Provides support for CORBA IIOP profile.
<code>giop</code>	Provides support for CORBA GIOP.
<code>tunnel</code>	Provides support for the IIOP transport using non-CORBA payloads.
<code>tuxedo</code>	Provides support for Tuxedo interoperability.
<code>mq</code>	Provides support for IBM WebSphere MQ interoperability, and MQ transactions.
<code>java</code>	Provides support for Java Message Service (JMS) interoperability (and also for other 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
<code>soap</code>	Decodes and encodes messages using the SOAP format. See also " SOAP ".
<code>G2</code>	Decodes and encodes messages packaged using the G2++ format.
<code>fml</code>	Decodes and encodes messages packaged in FML format.
<code>tagged</code>	Decodes and encodes messages packed in variable record length messages or another self-describing message format.
<code>fixed</code>	Decodes and encodes fixed record length messages.

Table 2: *Artix Payload Format Plug-ins (Continued)*

Plug-in	Payload Format
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 <code>bus_loader</code> at the end of your plug-in list to start the bus and initialize all <code>BusPluginS</code> . Not needed if your application uses <code>IT_Bus::init</code> .
bus_response_monitor	Enables performance logging. Monitors response times of Artix client/server requests. See also “Performance Logging” on page 66 .
locator_client	Queries the locator and returns a reference to a target service. See also the <i>Artix Locator Guide, C++</i> .
locator_endpoint	Enables endpoints to use the Artix locator service. See also “Locator Endpoint Manager” on page 63 .
ots	Enables the CORBA OTS transaction system. See also “Bus” on page 40 .
ots_lite	Enables the OTS Lite transaction system, which supports one-phase commit transactions. See also “Bus” on page 40 .
request_forwarder	Enables forwarding of write requests from slave replicas to master replicas. See also “Database Environment” on page 45 .
routing	Enables Artix routing. See “Routing” on page 69 .
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 61 .

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

Plug-in	Artix Feature
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 73 .
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 74 .
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 74 .
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 72 .
uddi_proxy	Dynamically locates existing Web services endpoints using the UDDI service. See also "java_plugins variable" on page 2 .
wsat_protocol	Enables the WS-Atomic Transaction (WS-AT) system. See also "Bus" on page 40 .
ws_chain	Enables you to link together a series of services into a multi-part process. See also "Web Services Chain Service" on page 82 .
ws_coordination_service	Enables the WS-Coordination service, which coordinates two-phase commit transactions. See also "Bus" on page 40 .

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

Plug-in	Artix Feature
ws_coloc	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 through the messaging stack, messages are passed directly, which improves performance. See also “Colocation request-level interceptors” on page 11.
wSDL_publish	Enables Artix endpoints to publish and download Artix WSDL files. See also “WSDL Publishing Service” on page 90.
wSRM	Enables Web Services Reliable Messaging. See also “Web Services Reliable Messaging” on page 84.
wSRM_db	Enables Web Services Reliable Messaging persistence. Automatically loads the wSRM plug-in. See also “Web Services Reliable Messaging” on page 84.
xmlfile_log_stream	Enables you to view Artix logging output in a file. See also “XML File Log Stream” on page 91.
xslt	Enables Artix to process XSLT scripts. See also “Transformer Service” on page 77.

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.

Binding Lists

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 colocation (where server and client are collocated in the same process). Artix also provides 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 colocations` 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

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 10.

client_message_interceptor_list

`binding:artix:client_message_interceptor_list` is a string that specifies an ordered list of message-level interceptors for a 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 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 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 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";
```

Colocation request-level interceptors

The Artix support for colocation enables an Artix client proxy to talk directly to a collocated Artix service, without incurring any marshalling or transport overhead. Collocated means that the client proxy and the service belong to the same Artix bus. Instead of passing messages through the messaging stack, messages are passed directly between the two, thereby improving performance.

colocation request-level configuration

Because the collocated layer bypasses the binding and transport layer, you can specify colocation request-level interceptors directly along the invocation path. For example:

```
binding:artix:client_request_interceptor_list:http://www.myco.com/myservice:FooService:FooPort= "A+B+C+ws_coloc";
binding:artix:server_request_interceptor_list:http://www.myco.com/myservice:FooService:FooPort= "ws_coloc+C+B+A";
```

When configuring colocation, you must ensure the following:

- The service must be collocated with the client proxy, otherwise, the `ws_coloc` interceptors have no effect, and the invocation is treated as remote.
- `ws_coloc` must be specified as the last client request-level interceptor and the first server request-level interceptor. This enables other request-level interceptors to be used with colocation, and also enables the use of Artix contexts. Any interceptors specified after the `ws_coloc` interceptor in the client chain, or before the `ws_coloc` interceptor in the server chain, will be ignored.

Using this approach, an existing Artix messaging port-based service (for example, a SOAP/HTTP or CORBA service) can be configured to add colocation quality-of-service without any change to the WSDL contracts.

Note: You do not need to specify the `ws_coloc` plug-in on your `orb_plugins` list. When `ws_coloc` is specified in the request-level interceptor chain, the `ws_coloc` plug-in is loaded automatically.

Interceptor Factory Plug-in

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 9](#)).

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

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.

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.

Event Log

The `event_log` namespace controls logging levels in Artix. It includes the following variables:

- `event_log:filters`
- `event_log:filters:bus:pre_filter`
- `event_log:filter_sensitive_info`
- `event_log:log_service_names:active`

- `event_log:log_service_names:services`

For details on HTTP trace logging, see [policies:http:trace_requests:enabled](#)

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.

[Table 4](#) shows the full syntax used by the `event_log:filters` variable to specify Artix logging severity levels.

Table 4: *Artix Logging Severity Levels*

Severity Level	Description
INFO_LO [W]	Low verbosity informational messages.
INFO_MED [IUM]	Medium verbosity informational messages.
INFO_HI [GH]	High verbosity informational messages.
INFO [_ALL]	All informational messages.
WARN [ING]	Warning messages.
ERR [OR]	Error messages.
FATAL [_ERROR]	Fatal error messages.
*	All messages.

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.

event_log:filter_sensitive_info

`event_log:filter_sensitive_info` specifies whether sensitive information such as plain-text passwords are printed in the log. For example, to enable filtering of `WS-S` plain-text passwords, specify the following configuration setting:

```
event_log:filter_sensitive_info =
  ["event_log:filter_sensitive_info:wss_password"];
event_log:filter_sensitive_info:wss_password =
  ["#PasswordText$%'!$%>", "</", "*"];
```

This setting changes the characters in the log of a `WS-S` plain-text password to `*` characters.

This variable can also be used to filter other types of sensitive logging information, and multiple filters can be enabled in a single setting. The general format for this configuration setting is as follows:

```
event_log:filter_sensitive_info = ["foo"];
foo = [ "Start", "End", "#"];
```

In this general format, the first line provides the list of pattern names to consider for replacement, and the second line provides the actual pattern in the following syntax:

```
["Start_Pattern", "End_Pattern", "Replacement_Character"];
```

This replaces anything in the log between `Start_pattern` and `End_pattern` with the `#` character.

Because Artix configuration files do not support the escaped `"` character in configuration, any pattern that has the `"` character should instead replace this character with the following:

```
$%'!$%
```

You must specify two single quotes and not a double quote. These are then treated as the " character during the filtering of logging information.

event_log:log_service_names:active

`event_log:log_service_names:active` specifies whether to enable logging for specific services. You can use Artix service subsystems to log for Artix services, such as the locator, and also for services that you have developed. This can be useful if you are running many services, and need to filter services that are particularly noisy.

Using service-based logging involves extra configuration and performance overhead, and is disabled by default. To enable logging for specific services, set this variable as follows:

```
event_log:log_service_names:active = "true";
```

For more details, see [event_log:log_service_names:services](#).

event_log:log_service_names:services

`event_log:log_service_names:services` specifies the specific service names that you wish to enable logging for. This variable is specified as follows:

```
event_log:log_service_names:services = ["ServiceName1",  
    "ServiceName2", ... ];
```

Each service name must be specified in the following format:

```
"{NamespaceURI}LocalPart"
```

For example:

```
"{http://www.my-company.com/bus/tests}SOAPHTTPService"
```

To enable logging for specific services, perform the following steps:

1. Set the following variables:

```
event_log:log_service_names:active = "true";  
event_log:log_service_names:services = ["ServiceName1",  
    "ServiceName2"];
```

2. Set your event log filters as appropriate, for example:

```
event_log:filters = ["IT_BUS=FATAL+ERROR",  
    "ServiceName1=WARN+ERROR+FATAL",  
    "ServiceName2=ERROR+FATAL",  
  
    "ServiceName2.IT_BUS.BINDING.CORBA=INFO+WARN+ERROR+FATAL",  
    "L"  
];
```

For more details, see [event_log:log_service_names:active](#)

Further information

For more detailed information on logging, see *Configuring and Deploying Artix Solutions*.

Initial Contracts

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

Because variables in the `bus:initial_contract` namespace are in the global scope of `artix.cfg`, every application can access them. Contracts for Artix services 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 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`

In addition, the following variable enables you to specify a well-known directory where contracts are stored:

- `initial_contract_dir`

container

`bus:initial_contract:url:container` specifies the location of the WSDL contract for the Artix container service. 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 service. 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";
```

initial_contract_dir

`bus:initial_contract_dir` specifies a well-known directory for accessing service contracts. This enables you to configure multiple documents without explicitly setting every document in configuration. If you specify a well-known directory, you only need to copy the WSDL documents to this directory before the application uses them. For example:

```
bus:initial_contract_dir=["."];
```

The value "." means use the directory from where the application was started. You can specify multiple directories as follows:

```
bus:initial_contract_dir = [".", "../..etc"];
```

Further information

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

Initial References

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
-publihreference -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
-publihreference -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
-publihreference -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}SessionEndpointManager  
Service -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
-publihreference -service
{http://www.iona.com/uddi_over_artix}UDDI_PublishServi
ce -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
-publihreference -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
-publihreference -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.

JVM Options

You can use the `jvm_options` configuration variable to pass parameters into a Java Virtual Machine (JVM) that is started in an Artix process.

`jvm_options`

`jvm_options` specifies parameters that are passed to a JVM that is started in an Artix process. This configuration variable takes the following syntax:

```
jvm_options=["-Dname=Value, -Dname=Value, ...", "..."];
```

For example:

```
jvm_options = ["-Xdebug",  
              "-Xrunjdp:transport=dt_socket,address=8787,server=y,suspend=y",  
              "-verbose:class"];
```

This example passes in parameters to debug an Artix Java service that is deployed in an Artix container. These JVM options enable Java Platform Debugging Architecture (JPDA) on port 8787.

Further information

For details on using JPDA, see <http://docs.oracle.com/javase/6/docs/technotes/guides/jpda/>.

Message Snoop

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/test
s:myService: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/test
s:myService: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/test
s:myService:myPort:log_level = "WARNING";
artix:interceptors:message_snoop:http://www.acme.com/test
s:myService: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).

Multi-threading

Variables in the `thread_pool` namespace control multi-threading. Thread pools can be configured globally for Artix instances in a configuration scope, or configured on a per-service basis.

The `thread_pool` 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 automatic work queues:

- `service:owns_workqueue`

The following variables configure threading for custom transports and transports such as HTTP, JMS, and MQ:

- `policy:messaging_transport:client_concurrency`
- `policy:messaging_transport:concurrency`
- `policy:messaging_transport:max_threads`
- `policy:messaging_transport:min_threads`

`thread_pool:initial_threads`

`thread_pool:initial_threads` specifies the number of initial threads in each service's thread pool. Defaults to 5.

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 level of a fully-qualified service name, which overrides the global setting:

```
service:http://my.tns1/:SessionManager:thread_pool:initial_threads = "3";
```

thread_pool:high_water_mark

`thread_pool:high_water_mark` specifies 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 level of a fully-qualified service name, which overrides the global setting:

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

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 level of a fully-qualified service name, which overrides the global setting:

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

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 level of a fully-qualified service name, which overrides the global setting:

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

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 level of a fully-qualified service name, which overrides the global setting:

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

service:owns_workqueue

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

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

This variable can be set at different levels in your configuration. The following is a global setting, which means that all services in a bus have their own work queue:

```
service:owns_workqueue = "true";
```

The following setting is at the level of a fully-qualified service name, which overrides the global setting, and means that only the specified service has its own work queue:

```
service:http://my.tns1/:SessionManager:owns_workqueue = "true";
```

policy:messaging_transport:client_concurrency

`policy:messaging_transport:client_concurrency` specifies the number of `ClientTransport` instances created per `WSDLPort` instance. This controls multi-threading on the client side. The default value is 1.

This variable applies to Artix transports that use a `MULTI_THREADED` client policy (see *Developing Advanced Artix Plug-ins in C++*).

In general, requests from transports such as HTTP must block until the previous reply has been received. If there are multiple invocations blocking on a proxy, these must be queued and effectively serialized. This variable enables the transport mechanism to use a pool of underlying connections, and thereby scale it up.

For example, the Artix HTTP and JMS transports implement this threading model. You can specify this variable to the configuration scope where you start your client with these transports.

policy:messaging_transport:concurrency

`policy:messaging_transport:concurrency` specifies the number of threads in the messaging port's thread pool, when the multi-threaded policy is in effect. The default is 1.

This variable configures the thread pool for a transport that uses a combination of the `MESSAGING_PORT_DRIVEN` and `MULTI_THREADED` policies (see *Developing Advanced Artix Plug-ins in C++*).

For example, the Artix HTTP and JMS transports implement this threading model. You can specify this variable to the scope where you start your server with these transports.

policy:messaging_transport:max_threads

`policy:messaging_transport:max_threads` specifies the maximum number of threads in the messaging port's thread pool, when the multi-instance policy is in effect. The default is 1.

This variable configures the thread pool for a transport that uses a combination of the `MESSAGING_PORT_DRIVEN` and `MULTI_INSTANCE` policies (see *Developing Advanced Artix Plug-ins in C++*).

For example, the Artix MQ transport implements this threading model. You can specify this variable to the scope where you start your server with the MQ transport.

policy:messaging_transport:min_threads

`policy:messaging_transport:min_threads` specifies the minimum number of threads in the messaging port's thread pool, when the multi-instance policy is in effect. The default is 1.

This variable configures the thread pool for a transport that uses a combination of the `MESSAGING_PORT_DRIVEN` and `MULTI_INSTANCE` policies (see *Developing Advanced Artix Plug-ins in C++*).

For example, the Artix MQ transport implements this threading model. You can specify this variable to the scope where you start your server with the MQ transport.

Policies

The `policies` namespace contain variables that control a range of runtime settings. For example, publishing host names, HTTP buffers, and trace logging.

Transport policies

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:buffer:prealloc_shared`
- `policies:http:buffer:prealloc_size`
- `policies:http:client_address_mode_policy:local_hostname`
- `policies:http:server_address_mode_policy:local_hostname`
- `policies:http:server_address_mode_policy:port_range`

- `policies:http:trace_requests:enabled`
- `policies:iiop:client_address_mode_policy:local_hostname`
- `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`

Bus policies

These include the following:

- `policies:bus:resolved_endpoint:max_retries`

Other policies

For information on `policy:messaging_port` variables, see [“Multi-threading” on page 24](#).

`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
  -BUSCONFIG_policies:at_http:client:proxy_server="http://localhost:0/SOAPHTTPPP
  roxy"
```

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:

<code>canonical</code>	Publishes the fully qualified hostname of the machine in the <code>http:address</code> element of the dynamic WSDL (for example, <code>http://myhost.mydomain.com</code>).
------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------

unqualified	Publishes the unqualified local hostname of the machine in the <code>http:address</code> element of the dynamic WSDL. This does not include the domain name with the hostname (for example, <code>http://myhost</code>).
ipaddress	Publishes the IP address associated with the machine in the <code>http:address</code> element of the dynamic WSDL (for example, <code>http://10.1.2.3</code>). This is the default behavior.

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 <code>http:address</code> element.
true	Publishes the hostname of the machine hosting the running server in the <code>http:address</code> element of the WSDL contract.

Note: Setting the service URL programmatically overrides this configuration variable. For more details, see [Developing Artix Applications with C++](#).

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.

Note: See also [policies:http:server_address_mode_policy:local_hostname](#), which specifies the host name that the server listens on.

policies:http:buffer:prealloc_shared

`policies:http:buffer:prealloc_shared` specifies whether the HTTP pre-allocation buffer is shared among threads. Defaults to `false`. This means that each thread pre-allocates its own buffer on the first invocation for that thread.

If this variable is set to `true`, the buffer is shared among threads:

```
policies:http:buffer:prealloc_shared = "true";
```

This means that the same buffer pre-allocation is shared among all threads. Therefore, your application must ensure that multiple invocations are not active at the same time.

See also [policies:http:buffer:prealloc_size](#).

policies:http:buffer:prealloc_size

`policies:http:buffer:prealloc_size` specifies the pre-allocated size of the HTTP buffer in bytes. The default value is `0`, which means there is no pre-allocation.

When this variable is set, Artix pre-allocates chunks of the specified buffer size to avoid repeated allocations and deallocations. Each thread (dispatcher or reply consumer) performs this pre-allocation on the first message. Then repeated invocations on the same thread reuse this buffer. For example, the following setting specifies a 2 MB buffer:

```
policies:http:buffer:prealloc_size = "2097152";
```

User applications should work out their worst case load in advance, and set this variable to an appropriate value. This allocation can be reused by each subsequent request/reply on the dispatcher/consumer thread. When the Artix bus is shut down, the buffer allocation is freed.

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. It is unspecified by default, 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:local_hostname

policies:http:server_address_mode_policy:local_hostname enables you to explicitly specify the host name that the server listens on when using the HTTP transport. This is unspecified by default.

For example, if you have a multi-homed server host 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:server_address_mode_policy:local_hostname =  
"207.45.52.34";
```

Note: See also [policies:at_http:server_address_mode_policy:local_hostname](#), which specifies the hostname published in dynamically generated Artix contracts.

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 this `.cfg` file setting.

policies:http:trace_requests:enabled

policies:http:trace_requests:enabled specifies whether to enable HTTP-specific trace logging. The default is `false`. To enable HTTP tracing, set this variable as follows:

```
policies:http:trace_requests:enabled="true";
```

This setting outputs `INFO` level messages that show full HTTP buffers (headers and body) as they go to and from the wire.

You should also set your log filter as follows to pick up the HTTP additional messages, and then resend the logs:

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

For example, you could enable HTTP trace logging to verify that basic authentication headers are written to the wire correctly.

Similarly, to enable HTTPS-specific trace logging, use the following setting:

```
policies:https:trace_requests:enabled="true";
```

policies:iiop:client_address_mode_policy:local_hostname

`policies:iiop:client_address_mode_policy:local_hostname` enables you to explicitly specify the host name that the client binds on. This is unspecified by default.

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

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. This is unspecified by default.

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";
```

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` specifies 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`:

```
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 when using SOAP as a transport. 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:

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

For example:

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

The following values are deprecated:

- | | |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------|
| <code>false</code> | Publishes the IP address of the running server in the <code>soap:address</code> element. This is the default behavior. |
| <code>true</code> | Publishes the hostname of the machine hosting the running server in the <code>soap:address</code> element of the WSDL contract. |

Note: Setting the service URL programmatically overrides this configuration variable. For more details, see [Developing Artix Applications with C++](#).

policies:bus:resolved_endpoint:max_retries

`policies:bus:resolved_endpoint:max_retries` applies to service proxies whose address details are obtained using `resolve_initial_references()`. This variable specifies the number of transparent rebinds for transient connection errors. Values can be in the range of zero to infinity. The default value is 0.

Resolved proxies are initialized using the `ClientProxyBase(QName ServiceName)` constructor. This tells the Artix bus the name of the initialized service, but does not supply its WSDL or address. The bus obtains the address by calling `resolve_initial_references()`.

Resolved proxies are fault tolerant and can rebind if errors occur. By default, they do not rebind, and throw an exception if the connection is lost. However, if you set `policies:bus:resolved_endpoint:max_retries` to a value greater than zero, the proxy tries to rebind using `resolve_initial_references()`. For example, if you use the `locator_client` plug-in to resolve initial references, the service proxy can perform dynamic failover.

QName Aliases

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

`{NamespaceURI}LocalPart`

For example: `{http://ws.ionapro.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.ionapro.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.ionapro.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.ionapro.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.ionapro.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";
```

Reference Compatibility

The `bus` namespace includes configuration variables that specify backward compatibility with proprietary Artix reference and endpoint reference formats. It includes the following:

- `bus:non_compliant_epr_format`
- `bus:reference_2.1_compat`

bus:non_compliant_epr_format

`bus:non_compliant_epr_format` specifies backward compatibility with the Artix 4.0 proprietary endpoint reference format. The endpoint references published by Artix 4.1 or higher are compliant with the W3C WS-Addressing specification.

The default value of this variable in `artix.cfg` is `false`, which means to use WS-A compliant endpoint references. To use the proprietary Artix 4.0 endpoint reference format, set this variable as follows:

```
bus:non_compliant_epr_format="true";
```

Artix 4.0 endpoint reference format

Artix 4.0 does not support the `wsaw:ServiceName` element and `EndpointName` attribute specified by the WS-Addressing WSDL binding. This defines a `WSDLBindingSchema` for embedding WSDL information in the endpoint reference (EPR) metadata.

The proprietary format of an Artix 4.0 EPR can cause interoperability issues because it serializes the WSDL service as a `wSDL:service` element in EPR metadata. Other vendors cannot deserialize the `wSDL:service` element when processing EPR metadata. Artix 4.0 also does not support deserializing a `ServiceName` element, if present, in the inbound EPR.

Artix 4.1 or higher endpoint reference format

Artix 4.1 or higher supports the `wsaw:ServiceName` element and `EndpointName` attribute. The on-the-wire format of an Artix 4.1 or higher EPR containing metadata is different from an Artix 4.0 EPR. Artix 4.1 or higher serializes WSDL metadata in the EPR metadata as a `wsaw:ServiceName` element, and deserializes the `wsaw:ServiceName` element, and its `EndpointName` attribute, if present in the inbound EPR.

Artix 4.1 or higher does not publish the optional `EndpointName` attribute if the WSDL service has only one port, but does if the service has multiple endpoints. The EPR format introduced in Artix 4.1 is slightly different from the Artix 4.0 format, but complies with W3C specifications and facilitates interoperability between vendors.

Migrating from Artix 4.0

The following applies when migrating from Artix 4.0:

Zero impact scenarios There is no impact if deployed Artix 4.0 applications still use deprecated Artix references, and do not use WS-Addressing EPRs. Perform one-step migration to Artix 4.1 or higher, both on the client and server sides.

Mixed deployments The format of the WS-Addressing EPR that Artix 4.0 clients receive from Artix services (for example, the locator), depends on the value of the `bus:non_compliant_epr_format` variable set on the Artix service side. Some Artix 4.0 applications must be reconfigured if they use WS-A EPRs and decide to migrate to Artix 4.1 or higher in phases. For example, upgrade to Artix 4.1 or higher on server side, and Artix 4.0 on client side.

Possible failing scenarios In some cases of mixed deployment, Artix 4.0 client applications can fail while deserializing the EPR coming on the wire. For example, clients of Artix 4.1 or higher transient servants and default servants. Normal servants and multi-port services will still work.

Solution to failing cases If Artix 4.0 clients get an `IT_Bus` exception while creating a proxy using the EPR, the `bus:non_compliant_epr_format` configuration value on the Artix 4.1 or higher server side must be set to `true` to get the Artix 4.0 (non-compliant) format. There is no need to change any source code. The trace logs on the server side contain an entry for the `bus:non_compliant_epr_format` configuration variable.

bus:reference_2.1_compat

`bus:reference_2.1_compat` specifies backward compatibility 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 3.0.1 reference format

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.

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.ionas.com/bus"

              xmlns:m2="http://www.ionas.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.

Artix Plug-ins

Artix is built on 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.

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.

This chapter describes the following subjects:

- [AmberPoint](#)
- [Bus](#)
- [CA WSDM Observer](#)
- [Client-Side High Availability](#)
- [Container](#)
- [Database Environment](#)
- [FTP](#)
- [JMS](#)
- [JMX](#)
- [Local Log Stream](#)
- [Log4J Log Stream](#)
- [Locator Service](#)
- [Locator Endpoint Manager](#)
- [Monitoring](#)
- [Peer Manager](#)
- [Performance Logging](#)
- [Remote Logging](#)
- [Remote Method Invocation](#)
- [Routing](#)
- [Service Lifecycle](#)
- [Session Manager](#)
- [Session Endpoint Manager](#)
- [Session Manager Simple Policy](#)
- [SOAP](#)
- [SOAP 1.2](#)
- [Transformer Service](#)

- [Tuxedo](#)
- [Web Services Addressing](#)
- [Web Services Chain Service](#)
- [Web Services Reliable Messaging](#)
- [WSDL Publishing Service](#)
- [XML File Log Stream](#)
- [Custom Plug-ins](#)

AmberPoint

The `plugins:ap_nano_agent` namespace configures integration with the AmberPoint SOA management system. It includes the following variables:

- [plugins:ap_nano_agent:hostname_address:local_hostname](#)
- [plugins:ap_nano_agent:hostname_address:publish_hostname](#)

plugins:ap_nano_agent:hostname_address:local_hostname

`plugins:ap_nano_agent:hostname_address:local_hostname` is an arbitrary string used as the client hostname instead of trying to resolve it using the underlying IP runtime. This is undefined by default.

plugins:ap_nano_agent:hostname_address:publish_hostname

`plugins:ap_nano_agent:hostname_address:publish_hostname` specifies the form in which the Artix AmberPoint Agent resolves the host address that an Artix service consumer (Artix proxy) runs on. This variable takes the following values:

<code>unqualified</code>	The host name in short form, without the domain name (<i>hostname</i>).
<code>ipaddress</code>	The host name in the form of an IP address (for example, <code>123.4.56.789</code>). This is the default.
<code>canonical</code>	The host name takes a fully qualified form (<i>hostname.domainname</i>).
<code>true</code>	same as <code>unqualified</code>
<code>false</code>	same as <code>ipaddress</code>

Bus

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 the chapter [“CORBA”](#).
- `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.cfg
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

The `plugins:ca_wsdm_observer` namespace configures integration with the CA WSDM management system. It 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

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. Defaults to `sequential`. Specifying `random` enables client applications to select a random server each time they connect. The following example applies globally:

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

The following example applies at the level of a service:

```
plugins:ha_conf:strategy:http://www.iona.com/test:SOAPHTTPService="random";
```

`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`. Defaults to `always`.

Specify `always` if you want your clients to be uniformly load-balanced across different servers. The following example applies globally:

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

Specify `subsequent` if you want your clients to favor a particular server for their initial connectivity. The following example applies globally:

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

The following example applies at the level of a service:

```
plugins:ha_conf:strategy:http://www.iona.com/test:SOAPHTTPService="random";
plugins:ha_conf:random:selection:http://www.iona.com/test:SOAPHTTPService="subsequent";
```

Container

The `plugins:container` namespace specifies settings for the Artix container service. It 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 is 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 files 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

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:checkpoint_period`
- `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:error_file`
- `plugins:artix:db:home`
- `plugins:artix:db:iioport`
- `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:replace_when_forwarding`
- `plugins:artix:db:replica_name`
- `plugins:artix:db:replicas`
- `plugins:artix:db:roundtrip_timeout`
- `plugins:artix:db:sync_retry_attempts`
- `plugins:artix:db:use_shutdown_hook`
- `plugins:artix:db:verbose_logging`
- `plugins:artix:db:verify_aggressive` and `plugins:artix:db:recovery_attempt`

`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:checkpoint_period

`plugins:artix:db:checkpoint_period` specifies how often in seconds the Artix DB plug-in wakes up and performs a Berkeley DB checkpoint on its environment, as prescribed by Sleepycat.

Defaults to 900 seconds (15 minutes), which should be meet all eventualities. Setting this to a value less than 60 seconds may have a negative impact on performance.

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:error_file

`plugins:artix:db:error_file` specifies the file that Berkeley DB error messages are sent to. For example:

```
plugins:artix:db:error_file="c:\logs\berkeleydb.log";
```

The file name can have any extension, so long as it is valid for its operating system. The default value is "".

If `plugins:artix:db:verbose_logging` is set to `true`, additional Berkeley DB messages about replication, deadlock, and recovery are also sent to this file.

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 availability 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 availability 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:iop:port](#).

plugins:artix:db:replace_when_forwarding

`plugins:artix:db:replace_when_forwarding` specifies whether the port name or service name in the WSDL file is used as the replaceable artifact when mastership moves around. Possible values are `port` or `service`.

The Artix demos use `port`, and this is the IONA-preferred option. However, if you wish to have multiple WSDL services representing your cluster instead of multiple WSDL ports on one service, you can specify `service` instead.

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:use_shutdown_hook`

`plugins:artix:db:use_shutdown_hook` enables the Artix DB plug-in to be used correctly within a container. Defaults to `true`. This setting should not be changed, unless directed by Artix Support.

`plugins:artix:db:verbose_logging`

`plugins:artix:db:verbose_logging` specifies whether more Berkeley DB messages about replication, deadlock and recovery are sent to the error file. Defaults to `false`. Setting this variable to `true` has effect only when `plugins:artix:db:error_file` is also set to `true`.

`plugins:artix:db:verify_aggressive` and `plugins:artix:db:recovery_attempt`

The Berkeley database verification method is added to scan for a corrupted database file before opening the same. If the database verification (`db_verify`) fails, an exception is thrown and the initialization fails. To start the process with a corrupted database and join the replicated services, the following two new configuration variables are added to control the `db_verify`.

1. `plugins:artix:db:verify_aggressive`
2. `plugins:artix:db:recovery_attempt`

```
plugins:artix:db:verify_aggressive="false"
```

Default setting is `true`, which catches all database corruption. It is thorough as it verifies all the key/data pairs in the database, but therefore more costly.

```
plugins:artix:db:recovery_attempt="true";
```

Default setting is `false`. When enabled, the locator will try to recover to the latest checkpoint (if it exists) after failed db verification. With every start attempted and a `DB_VERIFY_BAD` it will rollback one checkpoint.

FTP

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 class name must be listed on the endpoint's class path. The default setting is:

```
plugins:ftp:policy:client:filenameFactory="com.iona.jbus.  
transports.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 class name must be listed on the endpoint's class path. The default setting is:

```
plugins:ftp:policy:client:replyFileLifecycle="com.iona.jbus.  
transports.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:credentials:password` must have credentials that allow 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 class name must be listed on the endpoint's class path. The default setting is:

```
plugins:ftp:policy:server:filenameFactory="com.iona.jbus.transports.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 class name must be listed on the endpoint's class path. The default setting is:

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

JMS

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](#)

For information on configuring multi-threading with JMS, see [policy:messaging_transport:concurrency](#).

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	$\text{initial_iteration_delay} \times \text{backoff_ratio}^0$
2	$\text{initial_iteration_delay} \times \text{backoff_ratio}^1$
n	$\text{initial_iteration_delay} \times \text{backoff_ratio}^{(n-1)}$

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 2² milliseconds.
- ...
- The nth retry waits 1000 x 2⁽ⁿ⁻¹⁾ milliseconds.

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

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_management: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_management: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

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:precision_logging`
- `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 2016, this results in a log file named `my_log_2016_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 `false`. When this setting has been enabled, the following example logging message shows the thread ID in bold:

```
Wed, 23 Sep 2015 12:22:26.0000000 [homer600:6870:1269287216]  
(IT_BUS.CORE:0) I - Registering Bus plugin  
SOAPServicePluginFactory
```

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:precision_logging

`plugins:local_log_stream:precision_logging` specifies whether events are logged with time precision in nanoseconds, or at the granularity of seconds. The default value is `false` (to avoid changing the logging output of deployed systems).

To enable precision logging, use the following setting:

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

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";
```

Log4J Log Stream

The `plugins:log4j_log_stream` namespace configures integration with Apache's log4j logging tool. This namespace contains the following variable:

- `plugins:log4j_log_stream:use_stderr`

`plugins:log4j_log_stream:use_stderr`

`plugins:log4j_log_stream:use_stderr` specifies whether to redirect the log4j log stream to standard error. The default is `false`. To redirect to standard error, specify the following setting:

```
plugins:log4j_log_stream:use_stderr = "true";
```

You must ensure that the `local_log_stream` plug-in is present in your `orb_plugins` list.

Locator Service

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 65](#).

`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 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 an Artix `.cfg` 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.ionas.com/advanced/LocatorQuery}CORBASvc";  
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

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:exclude_endpoints`

`plugins:locator_endpoint:exclude_endpoints` specifies endpoints to be excluded 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.

Monitoring

The `monitoring_plugin` enables integration with third-party monitoring tools (for example, Aurea's Actional[®] Application Performance Monitoring). This plug-in is configured by the following variables:

- `plugins:monitoring_plugin:classname`
- `plugins:monitoring_plugin:enable_si_payload`
- `plugins:monitoring_plugin:know_report_tool`
- `plugins:monitoring_plugin:max_reported_payload_size`
- `plugins:monitoring_plugin:show_service_facade`

`plugins:monitoring_plugin:classname`

`plugins:monitoring_plugin:classname` specifies the monitoring plug-in factory class. When configuring the Artix monitoring plug-in, you must also specify the `java` plug-in, and add monitoring handlers to the interceptor chain. This is shown in the following example:

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

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

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

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

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

For more details on configuring binding lists and interceptors, see [“Binding Lists for Custom Interceptors” on page 9](#).

`plugins:monitoring_plugin:enable_si_payload`

`plugins:monitoring_plugin:enable_si_payload` specifies whether reporting of the message payload on the server side is enabled (for example, for a SOAP message over HTTP). If this option is set to `false`, only the payload size is reported. The default value is:

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

plugins:monitoring_plugin:know_report_tool

`plugins:monitoring_plugin:know_report_tool` specifies the name of the reporting tool (in this case, `actional`). `actional` is currently the only supported value. For example:

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

plugins:monitoring_plugin:max_reported_payload_size

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

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

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

plugins:monitoring_plugin:show_service_facade

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

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

Peer Manager

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:ping_on_failure](#)
- [plugins:peer_manager:timeout_delta](#)

plugins:peer_manager:ping_on_failure

`plugins:peer_manager:ping_on_failure` specifies whether the receiver of a ping failure performs a reverse ping to verify the validity of the failure. Defaults to `false`. To enable this feature, set this variable as follows:

```
plugins:peer_manager:ping_on_failure = "true";
```

The peer manager service on both sides ping each other as a health check (for example, locator endpoint manager and locator service). If this variable is set, the peer manager that sees the ping failure confirms the validity of the failure by performing a ping itself. If this reverse ping succeeds, the ping failure is spurious and can be ignored. However, if it does not succeed, this is a genuine ping failure, and the appropriate callback is notified.

For example, this feature is useful in circumstances where a hardware clock malfunctions and creates unnecessary ping failure-like conditions (re-registrations or removal of endpoints).

For details on how the locator service and endpoint manager interact with the peer manager, and how they react to failure, see the *Artix Locator Guide: C++*.

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 the value of this variable to 10000 ensures that only a real failure results in an endpoint being removed from the locator's list of endpoints:

```
plugins:peer_manager:timeout_delta = "10000";
```

Performance Logging

The bus response monitor and response time collector plug-ins configure 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, C++ Runtime* for full details of Artix performance logging.

The Artix performance logging plug-ins include the following variables:

- `plugins:bus_response_monitor:type`
- `plugins:it_response_time_collector:filename`.
- `plugins:it_response_time_collector:server-id`.

See also “Remote Logging” on page 67.

plugins:bus_response_monitor:type

`plugins:bus_response_monitor:type` specifies whether logging is output to a file or stored in memory. Specifying `file` outputs performance logging data to a file, while specifying `memory` places the data into memory so it can be retrieved using the Artix container service. When `file` is enabled, `memory` is also enabled. For example:

```
plugins:bus_response_monitor:type = file;
```

plugins:it_response_time_collector:filename

plugins:it_response_time_collector:filename specifies the location of the performance log file. For example:

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

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. This setting can also be used to explicitly set a client ID that is reported in your log messages.

Remote Logging

Artix provides remote performance logging to enable you to send logging data to a remote endpoint where the data can be persisted and subsequently consumed by an application that is native to the remote system. For example, depending on your architecture, it might not be desirable or feasible to deploy the required management tools on a particular platform. Remote logging enables you to solve this problem.

The remote logging plug-in (`remote_log_receiver`) is configured by the following variables:

- `plugins:remote_log_receiver:iiop:addr_list`
- `plugins:remote_log_receiver:ior_filename`
- `plugins:remote_log_receiver:log_filename`
- `plugins:remote_log_receiver:prerequisite_plugins`

See also [“Performance Logging” on page 66](#).

plugins:remote_log_receiver:iiop:addr_list

plugins:remote_log_receiver:iiop:addr_list specifies the host name or IP address of the host on which the remote logger is running, and the port that it uses to listen for logging requests. Values are specified as follows:

```
plugins:remote_log_receiver:iiop:addr_list = ["host:port"];
```

plugins:remote_log_receiver:ior_filename

`plugins:remote_log_receiver:ior_filename` specifies the file to which the remote logger daemon writes a stringified Interoperable Object Reference (IOR) when it starts up. This IOR may be subsequently made available to the source applications that are acting as clients of the remote logger. However, this is not required if the source applications use a corbaloc URL rather than an IOR to contact the remote logger. For example:

```
plugins:remote_log_receiver:ior_filename =  
    "/var/publish/logger_ref.txt";
```

plugins:remote_log_receiver:log_filename

`plugins:remote_log_receiver:log_filename` specifies the local file on the remote host to which all logs are directed. For example:

```
plugins:remote_log_receiver:log_filename =  
    "/var/logs/remote_perflogs.txt";
```

plugins:remote_log_receiver:prerequisite_plugins

`plugins:remote_log_receiver:prerequisite_plugins` specifies the IIOp plug-ins that the remote logger needs for communication with the source host(s). For example:

```
plugins:remote_log_receiver:prerequisite_plugins =  
    ["iiop_profile", "giop", "iiop"];
```

Remote Method Invocation

The Java Remote Method Invocation plug-in, `rmi`, is configured by the following variables:

- `plugins:rmi:registry_port`
- `plugins:rmi:start_registry`

plugins:rmi:registry_port

`plugins:rmi:registry_port` specifies the port used to contact an RMI registry. The Artix bus can optionally run an RMI registry as a convenience for testing. The default setting is as follows:

```
plugins:rmi:registry_port = "1099";
```

plugins:rmi:start_registry

`plugins:rmi:start_registry` specifies whether to start an RMI registry. The Artix bus can optionally run an RMI registry as a convenience for testing. The default setting is `false`. To start an RMI registry, use the following setting:

```
plugins:rmi:start_registry = "true";
```

Routing

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:wrapped`

`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 1000 most recently used are present as unproxified references. While the remaining 450 do not exist in the router, but can be created on-demand.

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:

```
plugins:routing:reference_cache_size="1000";
```

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.

`plugins:routing:wrapped`

`plugins:routing:wrapped` specifies whether a SOAP message uses a `doc-literal WRAPPED` style. This enables the router to properly unwrap all parts of the message. The default value is `false`. To enable this feature, specify the following:

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

Service Lifecycle

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 passes references to other services. When the router encounters a reference in a message, it proxifies it 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.

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 proxies 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 includes the following variables:

- `plugins:service_lifecycle:evict_static_services`
- `plugins:service_lifecycle:long_lived_services`
- `plugins:service_lifecycle:max_cache_size`

`plugins:service_lifecycle:evict_static_services`

`plugins:service_lifecycle:evict_static_services` specifies whether the service lifecycle plug-in only evicts transient services or considers all services for eviction. By default, only transient services are evicted. To evict both transient and static services, specify the following setting:

```
plugins:service_lifecycle:evict_static_services="true";
```

`plugins:service_lifecycle:long_lived_services`

`plugins:service_lifecycle:long_lived_services` specifies a list of services to exclude from the eviction list. This marks certain services as important, so that even if they are not used over a long period that, they are not evicted. For example:

```
plugins:service_lifecycle:long_lived_services =  
  ["http://demo.myco.com/bank:ATMService",  
   "http://demo.myco.com/bank:LoanService"];
```

You can specify a single service as follows:

```
plugins:service_lifecycle:long_lived_services =  
  "http://demo.myco.com/bank:LoanService";
```

`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 proxy 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

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 65](#).

Session Endpoint Manager

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 65](#).

Session Manager Simple Policy

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 timeout setting. Zero means the unlimited. Default is 600.

SOAP

The `soap` plug-in includes the following configuration settings:

- `plugins:soap:decl_namespaces_at_root`
- `plugins:soap:encoding`
- `plugins:soap:sequence_validation`
- `plugins:soap:validating`
- `plugins:soap:write_xsi_type`
- `plugins:soap:enforce_namespaces_at_root`
- `plugins:soap:qname_validation`

plugins:soap:decl_namespaces_at_root

`plugins:soap:decl_namespaces_at_root` specifies whether XML namespace prefixes are added only to the root of a SOAP message. Defaults to `false`.

To specify that XML namespaces are added only in the SOAP Envelope at the root of the message, use the following configuration setting:

```
plugins:soap:decl_namespaces_at_root="true";
```

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:sequence_validation

`plugins:soap:sequence_validation` specifies whether to perform basic sequence validation on WSDL files at runtime. This is performed by default. To disable sequence validation, use the following setting:

```
plugins:soap:sequence_validation = "false";
```

For example, you may need to use this setting if you see the following error message from a client trying to connect to the Artix locator service:

```
Expected element: node_id
```

The error is caused by clients connecting to the locator using a different WSDL interface than the one published by the locator.

Note: This configuration variable is deprecated. If you experience the `Expected element: node_id` error message, you should ensure that your locator client is using the correct WSDL.

plugins:soap:validating

`plugins:soap:validating` specifies whether XML schema validation is performed at runtime. This is not performed by default. To enable runtime schema validation, use the following setting:

```
plugins:soap:validating = "true";
```

Schema validation is only available in the SOAP binding for read operations, and is not supported for write operations.

Note: Basic sequence validation is performed by default (see [plugins:soap:sequence_validation](#)).

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 is very useful for debugging purposes.

plugins:soap:enforce_namespaces_at_root

`plugins:soap:enforce_namespaces_at_root` variable enforces namespace written at SOAP envelope root during SOAP message serialization. The default setting is `true`.

plugins:soap:qname_validation

`plugins:soap:qname_validation` is a bus plugin configuration variable that enables throwing exception when an Artix server receives a request from a third party client that has an incorrect namespace as part of a soap message body.

The default setting is `false`. You need to set to `true` in your scope initializing bus in order to activate the variable.

SOAP 1.2

The `soap12` plugin is configured by the following variable:

- `plugins:soap12:enable_mtom`

`plugins:soap12:enable_mtom`

`plugins:soap12:enable_mtom` specifies whether Artix service consumers use the SOAP Message Transmission Optimization Mechanism (MTOM) to transmit binary data in SOAP messages. They do not use MTOM by default.

MTOM is a SOAP 1.2 mechanism for optimizing binary data transmission in SOAP messages. Artix service consumers must be configured to use MTOM. Artix service providers, on the other hand, do not need to be configured to use MTOM. When a service provider receives a SOAP 1.2 request that uses MTOM, it will respond using MTOM.

To enable an Artix service consumer to use MTOM, use the following setting:

```
plugins:soap12:enable_mtom = "true";
```

Note: To use MTOM with Artix ESB, you must also add the correct schema types to a service's WSDL file. For more information on MTOM and the schema types involved, see the *Bindings and Transports, C++ Runtime* guide.

Transformer Service

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` plug-in) includes the following configuration variables:

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

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` specifies 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:endpoint_name:operation_map

`plugins:xslt:endpoint_name:operation_map` specifies a list of XSLT operations and scripts to be used in processing the received 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:operation_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.

plugins:xslt:endpoint_name:use_element_name

`plugins:xslt:endpoint_name:use_element_name` specifies whether to use the message part element name or message part name when performing transformations. The default value is `false`, which means to use the message part name.

Using the message part element name matches the behavior of Artix content-based routing. To use the message part element name, specify the following setting:

```
plugins:xslt:endpoint_name:use_element_name = "true";
```

The following WSDL file extract shows an example message part element name and part name:

```
<message name="client_request_message">
  <part element="tns:client_request_type" name="client_request"/>
</message>
```

The following XSL file extract shows the example part element name when this variable is set to `true`:

```
<xsl:template match="client_request_type">
  <xsl:value-of select="first_name"/>
  <xsl:text> </xsl:text>
  <xsl:value-of select="last_name"/>
</xsl:template>
```

If this variable is set to `false`, the part name is used instead (in this case, `client_request`).

plugins:xslt:servant_list

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

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

Tuxedo

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 `tmbboot`. The default is:

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

Web Services Addressing

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:generic_pool_size`
- `plugins:messaging_port:supports_wsa_mep`
- `plugins:messaging_port:supports_wsa_2005_mep`
- `plugins:messaging_port:wsm_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 two Artix 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/ReplyTo0  
001";
```

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

```
"http://localhost:2356/WSATestClient/BaseReplyTo/ReplyTo0  
002";
```

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:generic_pool_size

`plugins:messaging_port:generic_pool_size` specifies the upper limit on messaging port pools. The messaging port maintains a number of pools for `MessageReader`, `MessageWriter`, `SendMessageContext` and `ReceiveMessageContext`.

The default value is `-1`. This means that these pools are unbounded, which ensures backwards compatibility.

To specify that no pooling is performed, use the following setting:

```
plugins:messaging_port:base_replyto_url="0";
```

plugins:messaging_port:supports_wsa_mep

`plugins:messaging_port:supports_wsa_mep` specifies whether a WS-Addressing 2004 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. When you enable WS-ReliableMessaging, a WS-Addressing 2004 MEP is enabled automatically (see [“plugins:messaging_port:wsm_enabled” on page 82](#)).

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";
```

Note: Either WS-A 2004 or WS-A 2005 should be enabled. If both are enabled, Artix enables WS-A 2005, and ignores WS-A 2004, and logs a `MessagingPort` warning message.

plugins:messaging_port:supports_wsa_2005_mep

`plugins:messaging_port:supports_wsa_2005_mep` specifies whether a WS-Addressing 2005 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_2005_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_2005_mep:  
  http://www.iona.com/bus/tests:SOAPHTTPService:SOAPHTTPPort="  
  true";
```

Note: A WS-Addressing 2004 MEP must be used with WS-RM. You can not use a WS-Addressing 2005 MEP with WS-Reliable Messaging (WS-RM).

plugins:messaging_port:wsm_enabled

plugins:messaging_port:wsm_enabled specifies whether WS-ReliableMessaging 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. If you wish to make a two-way invocation, you must configure a WS-RM-enabled WSDL port with a non-anonymous reply-to endpoint.

Bus-specific configuration

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

```
plugins:messaging_port:wsm_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:wsm_enabled:http://www.iona.com/b  
us/tests:SOAPHTTPService:SOAPHTTPPort="true";
```

Note: To enable WS-RM in the Artix runtime, you must also add the wsm plug-in to your orb_plugins list.

Web Services Chain Service

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 *operation_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

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

- `plugins:wsm:acknowledgement_interval`
- `plugins:wsm:acknowledgement_uri`
- `plugins:wsm:base_retransmission_interval`
- `plugins:wsm:delivery_assurance_policy`
- `plugins:wsm:disable_exponential_backoff_retransmission_interval`
- `plugins:wsm:enable_per_thread_sequence_scope`
- `plugins:wsm:max_messages_per_sequence`
- `plugins:wsm:max_unacknowledged_messages_threshold`
- `plugins:wsm:thread_pool:high_water_mark`
- `plugins:wsm:thread_pool:initial_threads`
- `plugins:wsm:thread_pool:low_water_mark`
- `plugins:wsm:thread_pool:max_queue_size`
- `plugins:wsm:thread_pool:stack_size`
- `plugins:wsm:use_server_endpoint_for_wsm_acknowledgement`
- `plugins:wsm:use_wsa_replyto_endpoint_for_wsm_acknowledgement`

See also [Web Services Addressing](#).

`plugins:wsm:acknowledgement_interval`

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

Bus configuration

The following example shows how to set for a specific bus:

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

WSDL port configuration

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

```
plugins:wsm:acknowledgement_interval:http://www.ionas.com/  
bus/tests:SOAPHTTPService:SOAPHTTPPort = "2500";
```

`plugins:wsm:acknowledgement_uri`

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

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

Bus configuration

The following example shows how to configure for a specific bus:

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

WSDL port configuration

The following example shows how to configure for a specific WSDL port:

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

plugins:wsmr:base_retransmission_interval

`plugins:wsmr: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 configuration

The following example shows how to set for a specific bus:

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

WSDL port configuration

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

```
plugins:wsmr:base_retransmission_interval:http://www.iona  
.com/bus/tests:SOAPHTTPService:SOAPHTTPPort = "3000";
```

plugins:wsmr:delivery_assurance_policy

`plugins:wsmr:delivery_assurance_policy` specifies the message delivery assurance policy. The available options are:

`ExactlyOnceInOrder` The RM destination delivers the messages to the application destination exactly once, in increasing order of RM message ID. The calls to the application destination are serialized. This is the default value.

`ExactlyOnceConcurrent` The RM destination delivers the messages to the application destination exactly once. Instead of a serialized message delivery (as in `ExactlyOnceInOrder`), messages are delivered concurrently, so they may not be delivered in order.

However, for a message with ID *n* that is being delivered, all the messages in the range of 1 to *n* are received and acknowledged by the RM destination.

`ExactlyOnceReceivedOrder` The RM destination delivers the messages to the application destination exactly once, as soon as it is received from the underlying transport.

The RM destination makes no attempt to ensure that the messages are delivered in order of message ID, or that all the previous messages have been received/acknowledged. The benefit of this policy is that it avoids a context switch during dispatch in the RM layer, and messages are not stored in the in-memory undelivered messages map.

Bus configuration

The following example shows how to set for a specific bus:

```
plugins:wsm:delivery_assurance_policy =  
  "ExactlyOnceConcurrent";
```

WSDL port configuration

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

```
plugins:wsm:delivery_assurance_policy:http://www.iona.com/bus/  
tests:SOAPHTTPService:SOAPHTTPPort =  
  "ExactlyOnceConcurrent";
```

plugins:wsm:disable_exponential_backoff_retransmission_interva

`plugins:wsm: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 configuration

The following example shows how to set for a specific bus:

```
plugins:wsm:disable_exponential_backoff_retransmission_i  
nterval = "true";
```

WSDL port configuration

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

```
plugins:wsm:disable_exponential_backoff_retransmission_i  
nterval:http://www.iona.com/bus/tests:SOAPHTTPService:  
SOAPHTTPPort = "true";
```

plugins:wsm:enable_per_thread_sequence_scope

`plugins:wsm:enable_per_thread_sequence_scope` specifies whether to create a separate RM sequence session for each invoking thread. By default, an RM session is shared by all threads. Enabling this setting creates a different RM sequence session for each thread, and eliminates the possibility of indeterminate message ID allocation. All messages sent by a particular thread are allocated a message ID in increasing order. When the RM source endpoint is closed, it closes all the open RM sequence sessions. The default value is `false` (disabled).

Bus configuration

The following example shows how to set for a specific bus:

```
plugins:wsm:enable_per_thread_sequence_scope = "true";
```

WSDL port configuration

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

```
plugins:wsm:enable_per_thread_sequence_scope:http://www.ionac.com/bus/tests:SOAPHTTPService:SOAPHTTPPort = "true";
```

plugins:wsm:max_messages_per_sequence

`plugins:wsm: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 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 acknowledgments for the messages previously sent. The new message is then sent using the new sequence.

Bus configuration

The following example shows how to set for a specific bus

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

WSDL port configuration

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

```
plugins:wsm:max_messages_per_sequence:http://www.ionac.com/bus/tests:SOAPHTTPService:SOAPHTTPPort = "1";
```

plugins:wsm:max_retransmission_attempts

`plugins:wsm:max_retransmission_attempts` specifies the maximum number of retransmission attempts that the RM source session makes for an unacknowledged message. If the number of retransmission attempts reaches this threshold, RM source session sends a `wsm:SequenceTerminated` fault to the peer RM destination session, and closes the session. Any subsequent attempt to send message on this session results in an `IT_Bus::Exception` being thrown. The default value is `-1` (no limit on the number of retransmission attempts).

Bus configuration

The following example shows how to set for a specific bus:

```
plugins:wsm:max_retransmission_attempts = "8";
```

WSDL port configuration

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

```
plugins:wsm:max_retransmission_attempts:http://www.iona.com/bus/tests:SOAPHTTPService:SOAPHTTPPort = "8";
```

plugins:wsm:max_unacknowledged_messages_threshold

`plugins:wsm: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 `wsm: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 configuration

The following example shows how to set for a specific bus:

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

WSDL port configuration

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

```
plugins:wsm:max_unacknowledged_messages_threshold:http://www.iona.com/bus/tests:SOAPHTTPService:SOAPHTTPPort = "50";
```

plugins:wsm:thread_pool:high_water_mark

`plugins:wsm:thread_pool:high_water_mark` specifies the maximum number of threads allowed in the WS-RM thread pool. The default is:

```
plugins:wsm:thread_pool:high_water_mark="-1";
```

plugins:wsm:thread_pool:initial_threads

`plugins:wsm:thread_pool:initial_threads` specifies the number of initial threads in the WS-RM thread pool. The default is:

```
plugins:wsm:thread_pool:initial_threads="5";
```

plugins:wsm:thread_pool:low_water_mark

plugins:wsm:thread_pool:low_water_mark specifies the minimum number of threads allowed in the WS-RM thread pool. The default is:

```
plugins:wsm:thread_pool:low_water_mark="-1";
```

plugins:wsm:thread_pool:max_queue_size

plugins:wsm:thread_pool:max_queue_size specifies the maximum number of request items that can be queued on the WS-RM thread work queue. The default is:

```
plugins:wsm:thread_pool:max_queue_size="-1";
```

plugins:wsm:thread_pool:stack_size

plugins:wsm:thread_pool:stack_size specifies the stack size for each thread. The stack size is specified in bytes. The default is:

```
plugins:wsm:thread_pool:stack_size="OS-specificDefault  
ThreadStackSize";
```

plugins:wsm:use_server_endpoint_for_wsm_acknowledgement

plugins:wsm:use_server_endpoint_for_wsm_acknowledgement specifies that the server endpoint, which receives the application request, also receives acknowledgements for the application response. This option only applies when a proxy is used to make two-way invocations.

Bus configuration

The following example shows how to configure for a specific Artix bus:

```
plugins:wsm:use_server_endpoint_for_wsm_acknowledgement  
= "true";
```

WSDL port configuration

The following example shows how to configure for a specific WSDL port:

```
plugins:wsm:use_server_endpoint_for_wsm_acknowledgement  
:http://www.iona.com/bus/tests:SOAPHTTPService:SOAPHTT  
PPort = "true";
```

plugins:wsm:use_wsa_replyto_endpoint_for_wsm_acknowledgement

plugins:wsm:use_wsa_replyto_endpoint_for_wsm_acknowledgement specifies that a reply-to endpoint (`wsa:replyTo`), which receives the application response, also receives acknowledgments for application requests. This option only applies when a proxy is used to make two-way invocations.

Bus configuration

The following example shows how to configure for a specific Artix bus:

```
plugins:wsm:use_wsa_replyto_endpoint_for_wsm_acknowledgement = "true";
```

WSDL port configuration

The following example shows how to configure for a specific WSDL port:

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

WSDL Publishing Service

The WSDL publishing service, `wSDL_publish`, includes the following configuration variables:

- `plugins:wSDL_publish:hostname`
- `plugins:wSDL_publish:processor`
- `plugins:wSDL_publish:publish_port`

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.

See also [enable_secure_wSDL_publish](#).

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 primary hostname is used. The possible values are as follows:

<code>canonical</code>	Use the fully qualified hostname of the machine in the URL (for example <code>http://myhost.mydomain.com</code>).
<code>unqualified</code>	Use the unqualified local hostname of the machine in the URL. This does not include the domain name with the hostname (for example, <code>http://myhost</code>). This is the default.
<code>ipaddress</code>	Use the IP address associated with the machine in the URL (for example <code>http://10.1.2.3</code>).
<code>SecondaryHostName</code>	For multi-homed machines, use the specified literal string for a secondary hostname in the URL. You can specify a logical name or a virtual IP address (for example, <code>http://myhost.mydomain.com</code> or <code>http://10.1.2.3</code>). Any leading or trailing white spaces are stripped out.

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:

<code>artix</code>	Strip out server-side artifacts. This is the default setting.
<code>standard</code>	Strip out server side artifacts and Artix proprietary extensors.
<code>none</code>	Disable preprocessing.

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.

XML File Log Stream

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 2016, this results in a log file named `my_log_2016_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

When you write a custom plug-in for Artix, 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 `.cfg` 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 `.cfg` 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" ...];
  ...
}
```

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 4](#) shows configuring some prerequisite plug-ins for a custom plug-in called `my_filter`.

Example 4: *Custom Prerequisite Plug-in Configuration*

```
plugins:my_filter:prerequisite_plugins = ["my_plugin_1",
"my_plugin_2", "my_plugin_3", "my_plugin4"];
```

Artix Security

This chapter describes variables used by the Artix Security Framework. The Artix security infrastructure is highly configurable.

This chapter discusses the following topics:

- [Applying Constraints to Certificates](#)
- [bus:initial_contract](#)
- [bus:security](#)
- [initial_references](#)
- [password_retrieval_mechanism](#)
- [plugins:asp](#)
- [plugins:at_http](#)
- [plugins:atli2_tls](#)
- [plugins:csi](#)
- [plugins:csi](#)
- [plugins:gsp](#)
- [plugins:https](#)
- [plugins:iiop_tls](#)
- [plugins:java_server](#)
- [plugins:login_client](#)
- [plugins:login_service](#)
- [plugins:security](#)
- [plugins:security](#)
- [plugins:wSDL_publish](#)
- [plugins:wSDL_publish](#)
- [plugins:wss](#)
- [policies](#)
- [policies:asp](#)
- [policies:bindings](#)
- [policies:csi](#)
- [policies:external_token_issuer](#)
- [policies:https](#)
- [policies:iiop_tls](#)
- [policies:security_server](#)
- [policies:soap:security](#)
- [principal_sponsor](#)
- [principal_sponsor:csi](#)
- [principal_sponsor:http](#)
- [principal_sponsor:https](#)
- [principal_sponsor:iiop_tls](#)
- [principal_sponsor:wsse](#)

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=Earth",
     "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:login_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.

url:login_service

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 a 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).

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 WSS username and password credentials. When `true`, the client sends WSS 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 WSS username and password credentials on the client side:

- *From the Artix .cfg file*—you can set the WSS credentials in the Artix configuration using the related `user_name` and `user_password` configuration variables. For example:

```
# Artix .cfg 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 WSS 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 WSS 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@security02:5002,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. The toolkit currently used is based on OpenSSL.

For example, to specify that an application should use the System SSL toolkit, you would set configuration variables as follows:

```
initial_references:IT_TLS_Toolkit:plugin = "systemssl_toolkit";  
plugins:systemssl_toolkit:shlib_name = "ORXSSSL";
```

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"](#).

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 600 seconds.

Note: This variable does not affect CORBA credentials. For details of how to configure the CORBA cache, see ["plugins:gsp"](#).

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 WSS 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:https:client_secure_invocation_policy:requires =
["Confidentiality","Integrity", "DetectReplay",
"DetectMisordering", "EstablishTrustInTarget"];
    policies:https: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:https:target_secure_invocation_policy:requires =
["Confidentiality", "Integrity", "DetectReplay",
"DetectMisordering", "EstablishTrustInClient"];
```

```
policies:https: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 ART runtime) Specifies whether or not to use the JSSE/JCE architecture with the CORBA binding.

Currently, when you deploy a new domain, Artix explicitly sets the `use_jsse_tk` variable to `true`.

`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.ion.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 authorization data is not read from incoming connections. Default is `true`.

Note: In Artix versions 4.0 and earlier, if no SAML authorization data is received and this variable is `true`, Artix would raise an exception. In Artix versions 4.1 and later, if no SAML authorization data is retrieved, Artix re-authenticates the client credentials with the security service, irrespective of whether the `accept_asserted_authorization_info` variable is `true` or `false`.

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 authorization 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 96](#).

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:https

The `plugins:https` namespace contains the following variable:

- [ClassName](#)

ClassName

(Java ART runtime) 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 ART runtime) 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 ART runtime) 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

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

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 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.

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 *Orbix 6 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 receiving 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` specifies 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 the *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.ionajbus.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.ionacorba.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.ion.corba.art.artimpl.ORBImpl",
"org.omg.CORBA.ORBSingletonClass=com.ion.corba.art.artimpl.
ORBSingleton",
"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:login_client

The `plugins:login_client` namespace contains the following variables:

- [wsdl_url](#)

wsdl_url

(Deprecated) Use `bus:initial_contract:url:login_service` instead.

plugins:login_service

The `plugins:login_service` namespace contains the following variables:

- [wsdl_url](#)

wsdl_url

(Deprecated) Use `bus:initial_contract:url:login_service` instead.

plugins:security

The `plugins:security` namespace contains the following variables:

- [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 `.cfg` 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 .cfg 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:security_cluster

The `plugins:security_cluster` namespace contains the following variable:

- [iiop_tls:addr_list](#)

iiop_tls:addr_list

The `plugins:security_cluster:iiop_tls:addr_list` variable lists the addresses for all of the security services in the cluster. Each address in the list is preceded by a `+` sign, which indicates that the service embeds the address in its generated IORs.

This variable is used in combination with the `plugins:security:iiop_tls:host` and `plugins:security:iiop_tls:port` settings, which specify the address where the security service listens for incoming IIOP/TLS request messages.

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 configured 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 .cfg 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.

plugins:wss

The `plugins:wss` namespace defines variables that are needed to configure the Artix partial message protection feature. Partial message protection is a WS-Security feature that enables you to apply cryptographic operations at the SOAP 1.1 binding level, including encrypting and signing a message's SOAP body. The variables belonging to this namespace are as follows:

- `classname`
- `keyretrieval:keystore:file`
- `keyretrieval:keystore:provider`
- `keyretrieval:keystore:storepass`
- `keyretrieval:keystore:storetype`
- `protection_policy:location`

classname

Specifies the name of the Java class that implements the WSS plug-in. This variable must be set to the value `com.iona.jbus.security.wss.plugin.BusPlugInFactory`.

keyretrieval:keystore:file

Specifies the location of a Java keystore file. This must be a filename or file pathname, not a URL.

keyretrieval:keystore:provider

Specifies the name of the Java keystore provider (*optional*). Using the Java cryptographic extension (JCE) package from Sun, it is possible to provide a custom implementation of the Java keystore. If your Java keystore is based on a custom provider, use this variable to set the *provider name*.

Default is to use the default provider provided by the Java virtual machine.

keyretrieval:keystore:storepass

Specifies the password to access the Java keystore. This variable is used in conjunction with `plugins:wss:keyretrieval:keystore:file` to associate a Java keystore with the WSS plug-in.

For example:

```
# Artix .cfg file
plugins:wss:keyretrieval:keystore:file="Keystore.jks";
plugins:wss:keyretrieval:keystore:storepass="StorePassword";
plugins:wss:keyretrieval:keystore:provider="";
plugins:wss:keyretrieval:keystore:storetype="";
```

keyretrieval:keystore:storetype

Specifies the type of the Java keystore (*optional*). Using the Java cryptographic extension (JCE) package from Sun, it is possible to provide a custom implementation of the Java keystore. If your Java keystore is based on a custom provider, use this variable to set the keystore type.

Default is `jks`.

protection_policy:location

Specifies the location of a policy configuration file that governs the behavior of the partial message protection feature. The policy configuration file is an XML file that conforms to the `protection-policy.xsd` XML schema (located in `ArtixInstallDir/schemas`).

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 96](#). 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: C++* 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.

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 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 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 5](#) 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

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_authorization](#)
- [enable_issue_external_token](#)
- [enable_security](#)
- [enable_sso](#)
- [load_balancing_policy](#)
- [use_artix_proxies](#)
- [server_interception_point](#)

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` was `false`.

enable_issue_external_token

Sometimes, instead of presenting full credentials (including a password) to an Artix server, a client might only be able to provide a username. In spite of the fact that these credentials are incomplete, the application might require the Artix security service to issue a security token.

For example, this kind of scenario can arise, if your security architecture has a mechanism for verifying credentials that is *external* to the Artix security framework. At a certain point in your application, the user's identity might already have been authenticated, but only the username is available for presentation to the Artix security service.

To deal with this special case, you can configure the Artix authentication mechanism to issue security tokens *based on usernames only*. To enable this feature, configure your Artix application as follows:

1. *Artix server configuration*—in the configuration file of the Artix server that needs to obtain security tokens from the Artix security service, set the `policies:asp:enable_issue_external_token` variable to `true`.
2. *Artix security server configuration*—configure the `policies:external_token_issuer:client_certificate_constraints` variable with the appropriate set of certificate constraints. See "[policies:external_token_issuer](#)" on page 129 for details of how to do this.

The Artix server will now be able to obtain a security token from the Artix security service for any kind of credentials that contains a username—for example, HTTP Basic Authentication credentials, WSSE UsernameToken credentials, or CORBA Principal. The password field in these credentials (if any) is ignored.

It is crucial that you configure the certificate constraints in the security service, so that *only* the X.509 certificate from the relevant Artix server matches the constraints.

WARNING: You *must* be certain that you can trust the Artix server to verify user identities independently of the Artix security service. Otherwise, your application's security will be compromised.

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` was `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`.

server_interception_point

Controls the point at which the Artix security interceptor is called. By default, the interceptor is called at the `intercept_around_dispatch` phase. The following setting (which is the default) is suitable for most applications:

```
policies:asp:server_interception_point =  
    "intercept_around_dispatch";
```

However, in some advanced applications, you might want to interpret incoming information on the wire and be able to set related information on the appropriate Artix security contexts *before* the Artix security interceptor is called. For example, if you want to perform dynamic credential mapping, use the following setting:

```
policies:asp:server_interception_point =  
    "intercept_pre_dispatch";
```

Note: This advanced security setting can *not* be used in conjunction with the router when it is configured to use pass-through mode. The Artix router in pass-threw mode skips the `intercept_pre_dispatch` interception point. This means that the Artix security interceptor only works with the Artix router when the secure application is configured to use the default setting (`intercept_around_dispatch`).

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 WSS 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 association 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 association 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 association 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 association 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 association 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 and to facilitate interoperability with security architectures that perform authentication independently of Artix, 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 96](#).

If you want to configure an Artix server that is not on the mainframe to gain access to the external token issuer, see [“enable_issue_external_token” on page 123](#).

For example, by inserting the following setting into the security service’s configuration scope in the Artix `.cfg` 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 includes the following variables:

- `buffer:prealloc_shared`
- `buffer:prealloc_size`
- `client_secure_invocation_policy:requires`
- `client_secure_invocation_policy:supports`
- `mechanism_policy:accept_v2_hellos`
- `mechanism_policy:ciphersuites`
- `mechanism_policy:protocol_version`
- `target_secure_invocation_policy:requires`

- [target_secure_invocation_policy:supports](#)
- [trace_requests:enabled](#)
- [trusted_ca_list_policy](#)

buffer:prealloc_shared

`policies:https:buffer:prealloc_shared` specifies whether the HTTPS pre-allocation buffer is shared among threads. Defaults to `false`. This means that each thread pre-allocates its own buffer on the first invocation for that thread.

If this variable is set to `true`, the buffer is shared among threads:

```
policies:https:buffer:prealloc_shared = "true";
```

This means that the same buffer pre-allocation is shared among all threads. Therefore, your application must ensure that multiple invocations are not active at the same time.

See also [buffer:prealloc_size](#).

buffer:prealloc_size

`policies:https:buffer:prealloc_size` specifies the pre-allocated size of the HTTP buffer in bytes. The default value is `0`, which means there is no pre-allocation.

When this variable is set, Artix pre-allocates chunks of the specified buffer size to avoid repeated allocations and deallocations. Each thread (dispatcher or reply consumer) performs this pre-allocation on the first message. Then repeated invocations on the same thread reuse this buffer. For example, the following setting specifies a 2 MB buffer:

```
policies:https:buffer:prealloc_size = "2097152";
```

User applications should work out their worst case load in advance, and set this variable to an appropriate value. This allocation can be reused by each subsequent request/reply on the dispatcher/consumer thread. When the Artix bus is shut down, the buffer allocation is freed.

client_secure_invocation_policy:requires

This policy overrides

`policies:client_secure_invocation_policy:requires` for the `https` plugin.

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: C++** for more details about association options.

This policy cannot be downgraded programmatically by the application.

client_secure_invocation_policy:supports

This policy overrides `policies:client_secure_invocation_policy:supports` for the `https` plugin.

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.

mechanism_policy:accept_v2_hellos

This HTTPS-specific policy overrides 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:iioptls: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 6: *Mechanism Policy Cipher Suites*

Null Encryption, Integrity and Authentication Ciphers	Standard Ciphers
<code>RSA_WITH_NULL_MD5</code>	<code>RSA_EXPORT_WITH_RC4_40_MD5</code>
<code>RSA_WITH_NULL_SHA</code>	<code>RSA_WITH_RC4_128_MD5</code>
	<code>RSA_WITH_RC4_128_SHA</code>
	<code>RSA_EXPORT_WITH_DES40_CBC_SHA</code>
	<code>RSA_WITH_DES_CBC_SHA</code>
	<code>RSA_WITH_3DES_EDE_CBC_SHA</code>

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 overrides 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  
TLS_V1_2  
SSL_V3
```

The default setting is `SSL_V3` and `TLS_V1_2`.

For example:

```
policies:https:mechanism_policy:protocol_version = ["TLS_V1_2",  
"SSL_V3"];
```

target_secure_invocation_policy:requires

This policy overrides `policies:target_secure_invocation_policy:requires` for the https 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.

This policy cannot be downgraded programmatically by the application.

target_secure_invocation_policy:supports

This policy overrides `policies:target_secure_invocation_policy:supports` for the https 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.

trace_requests:enabled

Specifies whether to enable HTTPS-specific trace logging. The default is `false`. To enable HTTPS tracing, set this variable as follows:

```
policies:https:trace_requests:enabled="true";
```

This setting outputs `INFO` level messages that show full HTTP buffers (headers and body) as they go to and from the wire.

You must also set log filtering as follows to pick up the additional HTTPS messages, and then resend the logs:

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

For example, you could enable HTTPS trace logging to verify that authentication headers are written to the wire correctly.

Similarly, to enable HTTP-specific trace logging, use the following setting:

```
policies:http:trace_requests:enabled="true";
```

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 settings 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 overrides `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.

mechanism_policy:accept_v2_hellos

This IIOP/TLS-specific policy overrides 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 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 overrides `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 7: *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 IIOP/TLS-specific policy overrides 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  
TLS_V1_2  
SSL_V3  
SSL_V2V3 (Deprecated)
```

The default setting is `SSL_V3` and `TLS_V1_2`.

For example:

```
policies:iiop_tls:mechanism_policy:protocol_version = ["TLS_V1",  
"SSL_V3"];
```

The `SSL_V2V3` value is now *deprecated*. 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_2"];  
policies:iiop_tls:mechanism_policy:accept_v2_hellos = "true";
```

server_address_mode_policy:local_domain

(Java ART runtime) 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 ART runtime) 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 ART runtime) 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` specifies 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. Artix servers do not publish IIOP 1.0 profiles. The default value is 1.2.

session_caching_policy

This policy overrides `policies:session_caching_policy` for the `iiop_tls` plugin.

target_secure_invocation_policy:requires

This policy overrides `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: C++* 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 overrides `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 corresponds 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 corresponds to using the default size defined by the operating system.

trusted_ca_list_policy

This policy overrides 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 96.

For example, by inserting the following setting into the security service's configuration scope in the Artix `.cfg` 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.

policies:soap:security

The `policies:soap:security` namespace contains just a single configuration variable, as follows:

- [enforce_must_understand](#)

enforce_must_understand

Specifies whether the Artix runtime enforces the semantics required by the `mustUnderstand` attribute, which appears in the WS-Security SOAP header.

The semantics are as follows: when the `mustUnderstand` attribute is set to 1, the message receiver *must* process all of the security elements contained in the corresponding `wsse:Security` header element. If the receiving program is unable to process the `wsse:Security` element completely, the message should be rejected.

You can disable this behavior by setting the `policies:soap:security:enforce_must_understand` variable to `false`. Default is `true`.

The `mustUnderstand` attribute appears as follows in a SOAP 1.1 header:

```
<S11:Envelope>
  <S11:Header>
    ...
    <wsse:Security S11:actor="..." S11:mustUnderstand="...">
      ...
    </wsse:Security>
    ...
  </S11:Header>
  ...
</S11:Envelope>
```

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.

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:

<code>pkcs12_file</code>	The authentication method uses a PKCS#12 file.
<code>pkcs11</code>	(Java ART runtime). The authentication data is provided by a smart card.
<code>security_label</code>	(Windows 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`:

<code>filename</code>	A PKCS#12 file that contains a certificate chain and private key— <i>required</i> .
<code>password</code>	A password for the private key— <i>optional</i> . 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.
<code>password_file</code>	The name of a file containing the password for the private key— <i>optional</i> . Make sure that the password file is read/write protected on your file system.

For the `pkcs11` (smart card) authentication method, the following authentication data can be provided in `auth_method_data`:

<code>provider</code>	A name that identifies the underlying PKCS #11 toolkit used by Artix to communicate with the smart card.
<code>slot</code>	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 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.ionacorba.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`:

<code>username</code>	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 .
<code>password</code>	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.
<code>domain</code>	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 <code>domain</code> 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:

<code>GSSUPMech</code>	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:http

The `principal_sponsor:http` namespace provides configuration variables that enable you to specify the HTTP Basic Authentication username and password credentials.

Note: Once the HTTP principal sponsor is enabled, the HTTP header containing the username and password is *always* included in outgoing messages. For example, it is not possible to omit the HTTP Basic Authentication credentials while talking to security unaware services. It is possible, however, to program the application to set the username and password values equal to empty strings.

The principal sponsor is disabled by default.

For example, to configure a HTTP client to use the credentials `test_username` and `test_password`, configure the HTTP principal sponsor as follows:

```
principal_sponsor:http:use_principal_sponsor = "true";
principal_sponsor:http:auth_method_id = "USERNAME_PASSWORD";
principal_sponsor:http:auth_method_data =
    ["username=test_username", "password=test_password"];
```

The following variables are in this namespace:

- `use_principal_sponsor`
- `auth_method_id`
- `auth_method_data`

use_principal_sponsor

`use_principal_sponsor` is used to enable or disable the HTTP principal sponsor. Defaults to `false`. If set to `true`, the following `principal_sponsor:http` variables must be set:

- `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:

`USERNAME_PASSWORD` The authentication method reads the HTTP Basic Authentication username and password from the `auth_method_data` variable.

For example, you can select the `USERNAME_PASSWORD` authentication method as follows:

```
principal_sponsor:http:auth_method_id = "USERNAME_PASSWORD";
```

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 `USERNAME_PASSWORD` authentication method, the following authentication data can be provided in `auth_method_data`:

<code>username</code>	The HTTP Basic Authentication username— <i>required</i> .
<code>password</code>	The HTTP Basic Authentication password. 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.
<code>password_file</code>	The name of a file containing the HTTP Basic Authentication password.

The `username` field is required, and you can include either a `password` field or a `password_file` field to specify the password.

For example, to configure an application with the username, `test_username`, whose password is stored in the `wsse_password_file.txt` file, set the `auth_method_data` as follows:

```
principal_sponsor:http:auth_method_data =
  ["username=test_username",
   "password_file=wsse_password_file.txt"];
```

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 HTTPS principal sponsor is disabled by default.

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:https: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`:

<code>filename</code>	A PKCS#12 file that contains a certificate chain and private key— <i>required</i> .
<code>password</code>	A password for the private key. 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.
<code>password_file</code>	The name of a file containing the password for the private key. 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:https:auth_method_data =  
["filename=c:\users\bob\bob.p12", "password=bobpass"];
```

principal_sponsor:iiop_tls

The `principal_sponsor:iiop_tls` namespace provides configuration variables that enable you to specify the *own credentials* used with the IIOP/TLS transport.

The IIOP/TLS principal sponsor is disabled by default.

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:iioptls` 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:iioptls: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.
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.
The password file must be read and write protected to prevent tampering.

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:iioptls:auth_method_data =  
["filename=c:\users\bob\bob.p12", "password=bobpass"];
```

principal_sponsor:wsse

The `principal_sponsor:wsse` namespace provides configuration variables that enable you to specify the WSS username and password credentials sent in a SOAP header.

Note: Once the WSS principal sponsor is enabled, the SOAP header containing the WSS username and password is *always* included in outgoing messages. For example, it is not possible to omit the WSS username/password header while talking to security unaware services. It is possible, however, to program the application to set the username and password values equal to empty strings.

The principal sponsor is disabled by default.

For example, to configure a SOAP client to use the credentials `test_username` and `test_password`, configure the WSS principal sponsor as follows:

```
principal_sponsor:wsse:use_principal_sponsor = "true";
principal_sponsor:wsse:auth_method_id = "USERNAME_PASSWORD";
principal_sponsor:wsse:auth_method_data =
    ["username=test_username", "password=test_password"];
```

If you use a SOAP 1.2 binding, you must also include the following configuration in the client and in the server:

```
# Artix .cfg file
...
orb_plugins = ["xmlfile_log_stream", "artix_security", ...];

plugins:artix_security:shlib_name = "it_security_plugin";
binding:artix:server_request_interceptor_list =
    "principal_context+security";
binding:artix:client_request_interceptor_list =
    "security+principal_context";
```

The following variables are in this namespace:

- [use_principal_sponsor](#)
- [auth_method_id](#)
- [auth_method_data](#)

use_principal_sponsor

`use_principal_sponsor` is used to enable or disable the WSS principal sponsor. Defaults to `false`. If set to `true`, the following `principal_sponsor:wsse` variables must be set:

- `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:

`USERNAME_PASSWORD` The authentication method reads the WSS username and password from the `auth_method_data` variable.

For example, you can select the `USERNAME_PASSWORD` authentication method as follows:

```
principal_sponsor:wsse:auth_method_id = "USERNAME_PASSWORD";
```

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 `USERNAME_PASSWORD` authentication method, the following authentication data can be provided in `auth_method_data`:

`username` The WSS username—*required*.

`password` The WSS password.

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 WSS password.

The `username` field is required, and you can include either a `password` field or a `password_file` field to specify the password.

For example, to configure an application with the WSS username, `test_username`, whose password is stored in the `wsse_password_file.txt` file, set the `auth_method_data` as follows:

```
principal_sponsor:wsse:auth_method_data =  
["username=test_username",  
 "password_file=wsse_password_file.txt"];
```

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 apply when Artix is using the CORBA transport.

The following CORBA-based variables are discussed in this chapter:

- `plugins:codeset`
- `plugins:giop`
- `plugins:giop_snoop`
- `plugins:http` and `https`
- `plugins:iiop`
- `plugins:naming`
- `plugins:ots`
- `plugins:ots_lite`
- `plugins:ots_encina`
- `plugins:poa`
- `poa:FQPN`
- Core Policies
- CORBA Timeout Policies
- Artix Timeout Policies
- `policies:giop`
- `policies:giop:interop_policy`
- `policies:http`
- `policies:iiop`
- `policies:invocation_retry`

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 8: *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 9: *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 10: *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

Table 10: *Defaults for the wide native codesets*

Platform/Locale	Language	Setting
ISO-8859-1/Cp-1292/ US-ASCII locale	Java	UTF-16
Shift_JIS locale	Java	UTF-16
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 11: *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 interceptors 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>.MDDYYYY
```

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 and https

The variables in this namespace configure both the HTTP and HTTPS transports. 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 is `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. Defaults 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 connection 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_delay` 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. This is used to ensure that TCP buffers are cleared when a socket is closed. This variable specifies the number of seconds to linger, using a value of type `long`. The default is `-1`, which means that the `SO_LINGER` socket option is not set.

tcp_listener:reincarnate_attempts

`tcp_listener:reincarnate_attempts` specifies the number of times that a `Listener` recreate its listener socket after receiving a `SocketException`. This configuration variable 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 is `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. Defaults 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 connection 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_delay` 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. This is used to ensure that TCP buffers are cleared when a socket is closed. This variable specifies the number of seconds to linger, using a value of type `long`. The default is `-1`, which means that the `SO_LINGER` socket option is not set.

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 a 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 through 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 `CosTransactions::TransactionFactory` which is optimized for use in a single resource system. For details, see the *Orbix 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

`otid_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` variable. 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` variable 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:

<code>permit</code>	Maps to the <code>NonTxTargetPolicy</code> value <code>PERMIT</code> .
<code>prevent</code>	Maps to the <code>NonTxTargetPolicy</code> value <code>PREVENT</code> . (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 *Orbix CORBA Programmer's Guide*.

CORBA Timeout Policies

Artix supports standard CORBA timeout policies, to enable clients to abort invocations. Artix 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 canceled using a `GIOP CancelRequest` message and all received reply data is discarded.

For more information about standard CORBA timeout policies, see the *Orbix CORBA Programmer's Guide*.

Artix Timeout Policies

This section lists configuration variables for proprietary Artix-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 bidirectional 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 Artix and Orbix. 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 compliance 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 principal 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 *Orbix 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 Artix 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_transmission_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 Artix 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 incoming `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 host names 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 host names 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` specifies 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. Artix 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:rcv_buffer_size

`tcp_options_policy:rcv_buffer_size` specifies the size of the TCP receive buffer. This variable can only be set to 0, which corresponds 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 corresponds 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 172](#).

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 timeout policies, see the ***Orbix CORBA Programmer's Guide***.

Index

A

- active connection management
 - HTTP 157
 - IIOp 159
- agent_ior_file 167
- allow_registration_after_rollback_only 167
- ANSI C strftime() function 59, 92
- artix:endpoint 77
- artix:endpoint:endpoint_list 78, 82
- artix:endpoint:endpoint_name:wSDL_location 78
- artix:endpoint:endpoint_name:wSDL_port 78
- artix:interceptors:message_snoop:enabled 22, 23
- artix:interceptors:message_snoop:log_level 23
- asynchronous acknowledgement 84
- at_http 3

B

- backoff_ratio, reinvoking 181
- backup_restart_file 167
- Berkeley DB 47
- BiDirPolicy::ALLOW 175
- BiDirPolicy::BiDirAcceptPolicy 174
- BiDirPolicy::BiDirExportPolicy 175
- BiDirPolicy::BiDirOfferPolicy 175
- binding:artix:client_message_interceptor_list 9, 64
- binding:artix:client_request_interceptor_list 9, 64
- binding:artix:server_message_interceptor_list 10, 64
- binding:artix:server_request_interceptor_list 10, 64
- binding:client_binding_list 7, 8
- binding:server_binding_list 8
- binding policies
 - transparent retries 182
- bus:initial_contract:url 16
- bus:initial_contract:url:container 17
- bus:initial_contract:url:locator 17
- bus:initial_contract:url:login_service 18
- bus:initial_contract:url:peermanager 17
- bus:initial_contract:url:sessionendpointmanager 17
- bus:initial_contract:url:sessionmanager 17
- bus:initial_contract:url:uddi_inquire 17
- bus:initial_contract:url:uddi_publish 17
- bus:initial_contract_dir 18

- bus:initial_references:url:container 21
- bus:initial_references:url:locator 18
- bus:initial_references:url:login_service 21
- bus:initial_references:url:peermanager 19
- bus:initial_references:url:sessionendpointmanager 20
- bus:initial_references:url:sessionmanager 19
- bus:initial_references:url:uddi_inquire 20
- bus:initial_references:url:uddi_publish 20
- bus:non_compliant_epr_format 36
- bus:qname_alias:container 34
- bus:qname_alias:locator 34
- bus:qname_alias:login_service 35
- bus:qname_alias:peermanager 34
- bus:qname_alias:sessionendpointmanager 35
- bus:qname_alias:sessionmanager 35
- bus:qname_alias:uddi_inquire 35
- bus:qname_alias:uddi_publish 35
- bus:reference_2.1_compat 37
- bus.transactions().begin_transaction() 41
- BUSCONFIG_ 28
- bus_loader 4
- bus_response_monitor 4

C

- canonical 28, 33, 40, 90
- CertConstraintsPolicy 96
- CertConstraintsPolicy policy 96
- certificate_constraints_policy variable 96
- Certificates
 - constraints 96
- certificates
 - CertConstraintsPolicy policy 96
 - constraint language 96
- checkpoint 47
- ClientProxyBase() 33
- ClientTransport 26
- client_version_policy
 - IIOp 135, 178
- colocation 6, 11
- colocation interceptor 11
- concurrent_transaction_map_size 163
- configuration updates 42
- connection_attempts 136
- constraint language 96
- Constraints
 - for certificates 96
- container 22
- ContainerService.url 19
- coordination service 41

- corbaloc 68
- CORBA router by-pass 70
- create_transaction_mbeans 167
- custom plug-ins 93

D

- DB checkpoint 47
- default_buffer_size 177, 179
- default_ots_policy 163
- default_transaction_policy 163
- default_transaction_timeout 163
- delivery assurance policies 85
- direct_persistence 167
 - naming service 162
 - OTS Encina 167
- documentation
 - .pdf format ix
 - updates on the web ix
- duplicate masters 46
- Dynamic 72
- dynamic proxies 72

E

- EndpointName 36
- endpoint reference formats 35
- ERROR 13
- event_log:filters 13, 31, 132
- event_log:filters:bus:pre_filter 14
- event_log:filter_sensitive_info 14
- event_log:log_service_names:active 15
- event_log:log_service_names:services 15
- ExactlyOnceConcurrent 85
- ExactlyOnceInOrder 85
- ExactlyOnceReceivedOrder 86
- extra hop 65

F

- FATAL_ERROR 13
- filename 59, 91
- fixed 3
- fml 3
- FTP daemon 53
- FTP LIST command 53

G

- G2 3
- GIOP
 - interoperability policies 175
 - policies 175
- giop 3
- global_namespace_poa 167

H

- handler type 43
- hard_limit
 - HTTP 157, 158
 - IIOP 159, 160
- high_water_mark 25
- HTTP 26
- HTTP buffer 130
- HTTP plug-in configuration

- hard connection limit
 - client 158
 - server 157
- soft connection limit
 - client 158
 - server 157
- HTTP policies
 - buffer sizes
 - maximum 177
 - ports 178
- https 3

I

- ignore_message_not_consumed 176
- iiop 3
- IIOP plug-in configuration
 - hard connection limit
 - client 160
 - server 159
 - soft connection limit
 - client 160
 - server 160
- IIOP plugin configuration 159
- IIOP policies 129, 134, 178
 - buffer sizes 179
 - default 179
 - maximum 179
 - client version 135, 178
 - connection attempts 136
 - export hostnames 32, 138, 178, 180
 - export IP addresses 32, 138, 178, 180
 - GIOP version in profiles 139, 180
 - server hostname 138, 179
 - TCP options
 - delay connections 139, 180
 - receive buffer size 140, 181
- IIOP policy
 - ports 32, 138, 180
- iiop_profile 3
- INFO_ALL 13
- INFO_HIGH 13
- INFO_LOW 13
- INFO_MEDIUM 13
- initial_disk 167
- initial_disk_size 168
- initialization 45
- initial references
 - Encina transaction factory 169
 - OTS lite transaction factory 166
 - OTS transaction factory 165
- initial_threads 24
- interceptor
 - colocation 11
- interceptor chain 64
- interceptors 7
 - client request-level 7
- interoperability configuration 175
 - code set negotiation 176
 - GIOP 1.1 support 176
 - incompatible message format 176
 - LocateRequest messages 176

- Principal data 177
- Interoperable Object Reference 68
- interposition_style 164
- invocation policies 181
 - forwarding limit 181
 - initial retry delay 181
 - retry delay 181
 - retry maximum 182
- IOR 68
- ip:receive_buffer_size 158, 160
- ip:send_buffer_size 157, 160
- ipaddress 29, 33, 40, 90
- IT_Bus::Exception 87
- it_container_admin 19

J

- java 3
- Java Message Service 54
- Java Platform Debugging Architecture 22
- java plug-in 2
- Java plug-ins
 - loading 2
- java_plugins 2, 3, 64
- java_uddi_proxy 3
- JCE architecture
 - enabling 105
- jit_transactions 164
- jms
 - temporary queues 56
- JMS transport 26
- JMS transport plug-in 2
- JMX Remote 57
- JMXServiceURL 58
- JPDA 22
- jvm_options 22

L

- lb_default_initial_load 162
- lb_default_load_timeout 162
- local_hostname 32, 138, 179
- local_log_stream plugin configuration 59
- locator_client 4
- locator_endpoint 4, 65
- log4j 61
- log_check_interval 168
- logging
 - passwords 14
 - service-based 15
- logging configuration
 - set filters for subsystems 12
- logstream configuration
 - output stream 59
 - output to local file 59, 91
 - output to rolling file 60, 92
- log_threshold 168

M

- max_buffer_size 177, 179
- max_forwards
 - reinvoking 181
- max_queue_size 25

- max_rebinds 182
- max_resource_failures 168
- max_retries 182
- MBeans 57
- MEP 81
- Message Exchange Pattern 81
- message part element 79
- MessageReader 81
- message snoop 22
- MessageWriter 81
- MESSAGING_PORT_DRIVEN 27
- messaging port pools 81
- mq 3
- MQ transactions 3
- MTOM 77
- multi-homed 90
- multi-homed hosts
 - clients 30, 31, 178
 - servers 180
- multi-homed hosts, configure support
 - for 138
- MULTI_INSTANCE 27
- MULTI_THREADED 26, 27

N

- namespace
 - artix:endpoint 78
 - binding 7
 - event_log 12
 - plugins:artix:db 45
 - plugins:bus 40
 - plugins:bus_management 57
 - plugins:ca_wsdm_observer 42
 - plugins:chain 82
 - plugins:codeset 153
 - plugins:container 45
 - plugins:csi 105
 - plugins:event 155
 - plugins:file_security_domain 162
 - plugins:ftp 52
 - plugins:gsp 106
 - plugins:ha_conf 44
 - plugins:http 157
 - plugins:https 157
 - plugins:iiop 159
 - plugins:jms 54
 - plugins:local_log_stream 59
 - plugins:locator 61
 - plugins:locator_endpoint 63
 - plugins:messaging_port 80
 - plugins:ots_mgmt 170
 - plugins:peer_manager 66
 - plugins:poa 170
 - plugins:routing 69
 - plugins:service_lifecycle 72
 - plugins:session_endpoint_manager 74
 - plugins:session_manager_service 73
 - plugins:sm_simple_policy 74
 - plugins:soap 75
 - plugins:soap12 77
 - plugins:tuxedo 80

- plugins:wSDL_publish 91
- plugins:wSRM 84
- plugins:xmlfile_log_stream 91
- poa:fqpn 170
- policies 118, 172, 173, 174
- policies:csi 127
- policies:http 177
- policies:https 129
- policies:iiop 178
- policies:iiop_tls 133
- policies:shmiop 182
- principal_sponsor:csi 145
- principle_sponsor 142, 147, 148, 151
- namespace_poa 168
- naming service configuration 162
 - default initial load value 162
 - default load value timeout 162
 - NT service dependencies 162
- negotiate_transmission_codeset 176
- no_delay 139, 180
- non_tx_target_policy 172
- nterceptor_factory:InterceptorFactoryName:plugin 12
- nt_service_dependencies 162

O

- orb_name
 - OTS Encina 168
 - OTS Lite 165
- orb_plugins 1, 64
- otid_format_id
 - OTS Encina 168
 - OTS Lite 165
- ots 4
- OTS configuration 163
 - default timeout 163
 - hash table size 163
 - initial reference for factory 165
 - initial reference for transaction factory 165
 - interposition style 164
 - JIT transaction creation 164
 - optimize transaction propagation 164
 - OTSPolicy default value 163
 - roll back transactions 164
 - TransactionPolicy default 163
 - transaction timeout default 163
- OTS Encina 41
- OTS Encina configuration 166
 - backup restart file 167
 - direct persistence 167
 - initial log file 167
 - internal ORB usage 170
 - log file growth checks 168
 - log file size 168
 - log file threshold 168
 - logging configuration 169
 - log resource failures 168
 - management agent IOR 167
 - ORB name 168
 - OTS management object creation 167
- POA namespace 168
- raw disk usage 170
- registration after rollback 167
- restart file 168
- retry timeout 168
- transaction factory initial reference 169
- transaction factory name 169
- transaction ID 168
- transaction timeout 170
- OTS Lite 41
- ots_lite 4
- OTS Lite configuration 165
 - internal ORB 166
 - ORB name 165
 - transaction ID 165
 - transaction timeout 166
- ots_tx_provider 41
- ots_v11_policy 164

P

- part element 79
- passwords
 - logging 14
- performance logging 66
- ping failure 65
- plug-in 1
- plugins 57
 - at_http 3
 - bus_loader 4
 - bus_response_monitor 4
 - corba 4
 - fixed 3
 - fml 3
 - G2 3
 - giop 3
 - https 3
 - iiop 3
 - iiop_profile 3
 - java 3
 - java_plugins 3
 - locator_client 4
 - locator_endpoint 4
 - log4j_log_stream 61
 - mq 3
 - routing 4
 - service_lifecycle 5
 - service_locator 4
 - session_endpoint_manager 5
 - session_manager_service 5
 - sm_simple_policy 5
 - soap 3
 - tagged 3
 - tunnel 3
 - tuxedo 3
 - uddi_proxy 5
 - ws_chain 5
 - ws_coloc 6
 - WSDL_publish 6
 - ws_orb 4
 - wSRM 6
 - wSRM_db 6

xmlfile_log_stream 6
 xslt 6
 plugins:ap_nano_agent:hostname_addresses:local_hostname 40
 plugins:ap_nano_agent:hostname_addresses:publish_hostname 40
 plugins:artix:db
 home 48
 plugins:artix:db:allow_minority_master 46
 plugins:artix:db:checkpoint_period 47
 plugins:artix:db:db_open_retry_attempts 46, 47
 plugins:artix:db:download_files 47
 plugins:artix:db:election_timeout 47
 plugins:artix:db:env_name 47
 plugins:artix:db:error_file 48
 plugins:artix:db:iiop:port 48
 plugins:artix:db:inter_db_open_sleep_period 48
 plugins:artix:db:max_buffered_msgs 48
 plugins:artix:db:max_msg_buffer_size 49
 plugins:artix:db:max_ping_retries 49
 plugins:artix:db:ping_lifetime 49
 plugins:artix:db:ping_retry_interval 49
 plugins:artix:db:priority 49
 plugins:artix:db:replace_when_forwarding 50
 plugins:artix:db:replica_name 50
 plugins:artix:db:replicas 50
 plugins:artix:db:roundtrip_timeout 50
 plugins:artix:db:sync_retry_attempts 51
 plugins:artix:db:use_shutdown_hook 51
 plugins:artix:db:verbose_logging 51
 plugins:asp:security_level 102
 plugins:bus:default_tx_provider:plugin 41
 plugins:bus:register_client_context 40
 plugins:bus_management:connector:enabled 57
 plugins:bus_management:connector:port 57
 plugins:bus_management:connector:registry:required 58
 plugins:bus_management:connector:url:file 58
 plugins:bus_management:connector:url:publish 58
 plugins:bus_management:enabled 57
 plugins:bus_management:http_adaptor:enabled 58
 plugins:bus_management:http_adaptor:port 58
 plugins:bus_response_monitor:type 66
 plugins:ca_wsdm_observer:auto_register 42
 plugins:ca_wsdm_observer:config_poll_time 42, 45
 plugins:ca_wsdm_observer:handler_type 42
 plugins:ca_wsdm_observer:max_queue_size 43
 plugins:ca_wsdm_observer:min_queue_size 43
 plugins:ca_wsdm_observer:report_wait_time 43
 plugins:chain:endpoint_name:operation_name:service_chain 82
 plugins:chain:init_on_first_call 83
 plugins:chain:servant_list 83
 plugins:codeset:always_use_default 155
 plugins:codeset:char:ccs 154
 plugins:codeset:char:ncs 154
 plugins:codeset:wchar:ncs 154
 plugins:codesets:wchar:ccs 155
 plugins:container:deployfolder 45
 plugins:container:deployfolder:readonly 45
 plugins:csi:ClassName 105
 plugins:csi:shlib_name 106
 plugins:file_security_domain 162
 plugins:ftp:policy:client:filenameFactory 52
 plugins:ftp:policy:client:replyFileLifecycle 52
 plugins:ftp:policy:connection:connectMode 53
 plugins:ftp:policy:connection:connectTimeout 53
 plugins:ftp:policy:connection:receive:Timeout 53
 plugins:ftp:policy:connection:scanInterval 53
 plugins:ftp:policy:connection:useFilenameMaskOnScan 53
 plugins:ftp:policy:credentials:name 54
 plugins:ftp:policy:credentials:password 54
 plugins:ftp:policy:server:filenameFactory 54
 plugins:ftp:policy:server:requestFileLifecycle 54
 plugins:giop:message_server_binding_list 155
 plugins:giop_snoop:filename 156
 plugins:giop_snoop:rolling_file 156
 plugins:giop_snoop:verbosity 156
 plugins:gsp:authorization_realm 107
 plugins:gsp:ClassName 107
 plugins:ha_conf:random:selection 44
 plugins:ha_conf:strategy 44
 plugins:http:connection_max_unsent_data 157
 plugins:http:incoming_connections:hard_limit 157
 plugins:http:incoming_connections:soft_limit 157
 plugins:http:ip:reuse_addr 158
 plugins:http:outgoing_connections:soft_limit 158
 plugins:http:tcp_connection:keep_alive 158
 plugins:http:tcp_connection:linger_on_close 159

plugins:http:tcp_connection:no_delay 158
 plugins:http:tcp_listener:reincarnate_attempts 159
 plugins:iio:connection
 max_unsent_data 159
 plugins:iio:incoming_connections:hard_limit 159
 plugins:iio:incoming_connections:soft_limit 160
 plugins:iio:ip:receive_buffer_size 160
 plugins:iio:ip:reuse_addr 160
 plugins:iio:ip:send_buffer_size 160
 plugins:iio:outgoing_connections:hard_limit 160
 plugins:iio:outgoing_connections:soft_limit 160
 plugins:iio:pool:max_threads 160
 plugins:iio:pool:min_threads 160
 plugins:iio:tcp_connection:keep_alive 160
 plugins:iio:tcp_connection:linger_on_close 161
 plugins:iio:tcp_connection:no_delay 161
 plugins:iio:tcp_connection:no_deploy 161
 plugins:iio:tcp_connectionlinger_on_close 161
 plugins:iio:tcp_listener:reincarnate_attempts 111, 161
 plugins:iio:tcp_listener:reincarnation_retry_backoff_ratio 111, 161
 plugins:iio:tcp_listener:reincarnation_retry_delay 111, 161
 plugins:iio:tls:hfs_keyring_file_password 136
 plugins:iio:tls:tcp_listener:reincarnation_retry_backoff_ratio 111
 plugins:iio:tls:tcp_listener:reincarnation_retry_delay 111
 plugins:it_response_time_collector:filename 67
 plugins:it_response_time_collector:server-id 66, 67
 plugins:jms:policies:binding_establishment:backoff_ratio 55
 plugins:jms:policies:binding_establishment:initial_iteration_delay 55
 plugins:jms:policies:binding_establishment:max_binding_iterations 56
 plugins:jms:pooled_session_high_watermark 56
 plugins:jms:pooled_session_low_watermark 56
 plugins:local_log_stream:buffer_file 59
 plugins:local_log_stream:filename 59
 plugins:local_log_stream:filename_date_format 59
 plugins:local_log_stream:log_elements 60, 92
 plugins:local_log_stream:log_thread_id 60
 plugins:local_log_stream:milliseconds_to_log 60, 92
 plugins:local_log_stream:precision_logging 60
 plugins:local_log_stream:rolling_file 61, 93
 plugins:locator:peer_timeout 61
 plugins:locator:persist_data 61
 plugins:locator:selection_method 62
 plugins:locator:service_group 62
 plugins:locator:wSDL_port 62
 plugins:locator_endpoint:exclude_endpoints 63
 plugins:locator_endpoint:include_endpoints 63
 plugins:log4j_log_stream:use_stderr 61
 plugins:messaging_port:base_replyto_url 80
 plugins:messaging_port:generic_pool_size 81
 plugins:messaging_port:supports_wsa_mep 81
 plugins:messaging_port:wsmr_enabled 82
 plugins:monitoring_plugin:classname 64
 plugins:monitoring_plugin:enable_si_payload 64
 plugins:monitoring_plugin:know_report_tool 65
 plugins:monitoring_plugin:max_reported_payload_size 65
 plugins:monitoring_plugin:show_service_facade 65
 plugins:naming:destructive_methods_allowed 162
 plugins:naming:direct_persistence 162
 plugins:naming:iio:port 162
 plugins:notify_log 163
 plugins:ots_encina:iio:port 167
 plugins:peer_manager:ping_on_failure 65
 plugins:peer_manager:timeout_delta 66
 plugins:PluginName:prerequisite_plugins 94
 plugins:PluginName:shlib_name 93
 plugins:poa:ClassName 170
 plugins:poa:root_name 170
 plugins:remote_log_receiver:iio:addr_list 67
 plugins:remote_log_receiver:ior_filename 68
 plugins:remote_log_receiver:log_filename 68
 plugins:remote_log_receiver:prerequisite_plugins 68
 plugins:rmi:registry_port 68
 plugins:rmi:start_registry 68
 plugins:routing:proxy_cache_size 69
 plugins:routing:reference_cache_size 69
 plugins:routing:use_bypass 70
 plugins:routing:use_pass_through 71
 plugins:routing:wrapped 71
 plugins:routing:wSDL_url 70

plugins:service_lifecycle:evict_static_services 72
 plugins:service_lifecycle:long_lived_services 72
 plugins:service_lifecycle:max_cache_size 72
 plugins:session_endpoint_manager:default_group 74
 plugins:session_endpoint_manager:header_validation 74
 plugins:session_endpoint_manager:peer_timeout 74
 plugins:session_manager_service:peer_timeout 73
 plugins:sm_simple_policy:max_concurrent_sessions 74
 plugins:sm_simple_policy:max_session_timeout 75
 plugins:sm_simple_policy:min_session_timeout 74
 plugins:soap:encoding 75
 plugins:soap:sequence_validation 75
 plugins:soap:validating 76
 plugins:soap:write_xsi_type 76
 plugins:soap12:enable_mtom 77
 plugins:tuxedo:server 80
 plugins:wSDL_publish:host_name 90
 plugins:wSDL_publish:processor 91
 plugins:wSDL_publish:publish_port 91
 plugins:wSRM:acknowledgement_interval 84
 plugins:wSRM:acknowledgement_uri 84
 plugins:wSRM:base_retransmission_interval 85
 plugins:wSRM:delivery_assurance_policy 85, 86
 plugins:wSRM:disable_exponential_backoff_retransmission_interval 86
 plugins:wSRM:enable_per_thread_sequence_scope 87
 plugins:wSRM:max_messages_per_sequence 87
 plugins:wSRM:max_retransmission_attempts 88
 plugins:wSRM:max_unacknowledged_messages_threshold 88
 plugins:wSRM:thread_pool:high_water_mark 88
 plugins:wSRM:thread_pool:initial_threads 88
 plugins:wSRM:thread_pool:low_water_mark 89
 plugins:wSRM:thread_pool:max_queue_size 89
 plugins:wSRM:thread_pool:stack_size 89
 plugins:wSRM:use_wsa_replyto_endpoint_for_wSRM_acknowledgement 89
 plugins:xmlfile_log_stream:buffer_file 91
 plugins:xmlfile_log_stream:filename 92
 plugins:xmlfile_log_stream:filename_date_format 92
 plugins:xmlfile_log_stream:log_thread_id 92
 plugins:xslt:endpoint_name:operation_map 78
 plugins:xslt:endpoint_name:trace_filter 78
 plugins:xslt:endpoint_name:use_element_name 79
 plugins:xslt:servant_list 79
 POA
 plugin class name 170
 root name 170
 POA::create_POA() 175
 poa:fqpn:direct_persistent 171
 poa:fqpn:well_known_address 171
 policies:max_chain_length_policy 120
 policies
 CertConstraintsPolicy 96
 policies:allow_unauthenticated_clients_policy 119
 policies:at_http:client:proxy_server 28
 policies:at_http:server_address_mode_policy:local_hostname 29
 policies:at_http:server_address_mode_policy:publish_hostname 28, 29
 policies:bus:resolved_endpoint:max_retries 33
 policies:certificate_constraints_policy 120
 policies:csi:attribute_service:client_supports 127
 policies:csi:attribute_service:target_supports 128
 policies:csi:auth_over_transport:target_supports 128
 policies:csi:auth_over_transport:client_supports 128
 policies:csi:auth_over_transport:target_requires 128
 policies:giop:bidirectional_accept_policy 175
 policies:giop:bidirectional_export_policy 175
 policies:giop:bidirectional_gen3_accept_policy 175
 policies:giop:bidirectional_offer_policy 175
 policies:giop:interop:allow_value_types_in_1_1 176
 policies:giop:interop:ignore_message_not_consumed 176
 policies:giop:interop:negotiate_transmission_codeset 176
 policies:giop:interop:send_locate_request 176
 policies:giop:interop:send_principal 177
 policies:giop:interop_policy:enable_principal_service_context 176
 policies:http:buffer:prealloc_shared 30
 policies:http:buffer:prealloc_size 30
 policies:http:buffer_sizes_policy:max_buffer_size 177

policies:http:client_address_mode_policy:local_hostname 30
 policies:http:keep-alive:enabled 177
 policies:http:server_address_mode_policy:local_hostname 31
 policies:http:server_address_mode_policy:port_range 31, 178
 policies:http:trace_requests:enabled 31
 policies:https:buffer:prealloc_shared 130
 policies:https:buffer:prealloc_size 130
 policies:https:mechanism_policy:ciphersuites 131
 policies:https:mechanism_policy:protocol_version 132
 policies:https:trace_requests:enabled 31, 133
 policies:https:trusted_ca_list_policy 133
 policies:iio:buffer_sizes_policy:default_buffer_size 179
 policies:iio:buffer_sizes_policy:max_buffer_size 179
 policies:iio:client_address_mode_policy:local_hostname 32, 179
 policies:iio:client_address_mode_policy:port_range 179
 policies:iio:client_version_policy 178
 policies:iio:server_address_mode_policy:local_hostname 32, 179
 policies:iio:server_address_mode_policy:port_range 32, 180
 policies:iio:server_address_mode_policy:publish_hostname 32, 178, 180
 policies:iio:server_version_policy 180
 policies:iio:tcp_options:send_buffer_size 181
 policies:iio:tcp_options_policy:no_delay 180
 policies:iio:tcp_options_policy:recv_buffer_size 181
 policies:iio:tls:allow_unauthenticated_clients_policy 134
 policies:iio:tls:certificate_constraints_policy 135
 policies:iio:tls:client_secure_invocation_policy:requires 135
 policies:iio:tls:client_secure_invocation_policy:supports 135
 policies:iio:tls:client_version_policy 135
 policies:iio:tls:connection_attempts 136
 policies:iio:tls:connection_retry_delay 136
 policies:iio:tls:max_chain_length_policy 136
 policies:iio:tls:mechanism_policy:ciphersuites 137
 policies:iio:tls:mechanism_policy:protocol_version 137
 policies:iio:tls:server_address_mode_policy:local_hostname 138
 policies:iio:tls:server_address_mode_policy:port_range 138
 policies:iio:tls:server_address_mode_policy:publish_hostname 138
 policies:iio:tls:server_version_policy 139
 policies:iio:tls:session_caching_policy 139
 policies:iio:tls:target_secure_invocation_policy:requires 139
 policies:iio:tls:target_secure_invocation_policy:supports 139
 policies:iio:tls:tcp_options:send_buffer_size 140
 policies:iio:tls:tcp_options_policy:no_delay 139
 policies:iio:tls:tcp_options_policy:recv_buffer_size 140
 policies:iio:tls:trusted_ca_list_policy 140
 policies:invocation_retry:backoff_ratio 181
 policies:invocation_retry:initial_retry_delay 181
 policies:invocation_retry:max_forwards 181
 policies:invocation_retry:max_rebinds 182
 policies:invocation_retry:max_retries 182
 policies:mechanism_policy:ciphersuites 121
 policies:mechanism_policy:protocol_version 122
 policies:non_tx_target_policy 172
 policies:rebind_policy 172
 policies:relative_binding_exclusive_request_timeout 174
 policies:relative_binding_exclusive_roundtrip_timeout 174
 policies:relative_connection_creation_timeout 174
 policies:relative_request_timeout 173
 policies:relative_roundtrip_timeout 173
 policies:routing_policy_max 172
 policies:routing_policy_min 172
 policies:session_caching_policy 122
 policies:shmiop 182
 policies:soap:server_address_mode_policy:local_hostname 32
 policies:soap:server_address_mode_policy:local_hostname 32
 policies:soap:server_address_mode_policy:publish_hostname 33
 policies:sync_scope_policy 172
 policies:target_secure_invocation_policy:requires 122
 policies:target_secure_invocation_policy:supports 122
 policies:trusted_ca_list_policy 123
 policies:work_queue_policy 173
 policy:messaging_transport:client_concurrency 26

- policy:messaging_transport:concurrency 27
- policy:messaging_transport:max_threads 27
- policy:messaging_transport:min_threads 27
- pool:java_max_threads 160
- pool:max_threads 158, 160
- pool:min_threads 158, 160
- pooling 81
- prerequisite plug-ins 94
- principal_sponsor:csi:auth_method_data 146
- principal_sponsor:csi:use_principal_sponsor 145
- principal_sponsor Namespace Variables 142, 147, 148, 151
- principle_sponsor:auth_method_data 143, 148, 149, 152
- principle_sponsor:auth_method_id 143, 147, 149, 152
- principle_sponsor:callback_handler:Class Name 144
- principle_sponsor:login_attempts 144
- principle_sponsor:use_principle_sponsor 143, 147, 148, 151
- propagate_separate_tid_optimization 164
- proprietary endpoint reference 36
- proxies 72
- proxification 69
- proxy interposition 164
- publish_hostname 32, 138, 180

R

- read/write folder 45
- read-only folder 45
- rebind_policy 172
- ReceiveMessageContext 81
- recv_buffer_size 140, 181
- reference formats 35
- relative_binding_exclusive_request_timeout 174
- relative_binding_exclusive_roundtrip_timeout 174
- relative_connection_creation_timeout 174
- relative_request_timeout 173
- relative_roundtrip_timeout 173
- remote logging 67
- remote_log_receiver 67
- replicas, minimum number 46
- reply-to endpoint 80
- request_forwarder 4
- request-level interceptor 11
- resolve_initial_references() 33
- resource_retry_timeout 168
- restart_file 168
- RMI Connector 57
- rollback_only_on_system_ex 164
- rolling_file 60, 92
- router 72
- router proxification 69

- routing 4
- routing plug-in 69
- routing_policy_max 172
- routing_policy_min 172

S

- schema validation 76
- secondary hostname 90
- send_locate_request 176
- SendMessageContext 81
- send_principal 177
- server ID, configuring 67
- server_version_policy IIOp 139, 180
- service:owns_workqueue 26
- service group, groups of services 62
- service_lifecycle 5
- service_locator 4, 61, 65
- session_endpoint_manager 5, 65, 74
- session_manager_service 5, 65, 73
- share_variables_with_internal_orb 7
- Sleepycat 47
- sm_simple_policy 5, 74
- soap 3, 75
- soap12 77
- SocketException 159
- soft_limit HTTP 157, 158 IIOp 160
- SO_KEEPALIVE 158, 160
- SO_LINGER 159, 161
- standard interposition 164
- strftime() 59, 92
- superior_ping_timeout 165
- support_ots_v11 164
- sync_scope_policy 172

T

- tagged 3
- TCP_NODELAY 158, 161
- TCP policies delay connections 139, 180 receive buffer size 140, 181
- temporary queues 56
- thread_pool:high_water_mark 25
- thread_pool:initial_threads 24
- thread_pool:low_water_mark 25
- thread_pool:max_queue_size 25
- thread_pool:stack_size 26
- thread pool policies 24 initial number of threads 24 maximum threads 25 request queue limit 25
- timeout policies 173
- toolkit replaceability enabling JCE architecture 105
- trace_file 169
- trace_on 169
- transaction configuration 41
- transaction factory, initial reference 165
- transaction_factory_name

- OTS 165
- OTS Encina 169
- OTS Lite 166
- transaction_factory_ns_name 169
- TransactionPolicy, configure default value 163
- transactions
 - handle non-transactional objects 172
- transaction_timeout_period
 - OTS Encina 170
 - OTS Lite 166
- tunnel 3
- tuxedo 3

U

- uddi_proxy 5
- unqualified 29, 33, 40, 90
- use_internal_orb 166, 170
- use_jsse_tk configuration variable 105
- use_raw_disk 170

V

- validation 76

W

- WARNING 13
- work_queue_policy 173
- WS-Addressing 80
- WS-Addressing 2004 81
- WS-Addressing 2005 81
- WS-AtomicTransaction 41
- wsat_protocol 5
- wsat_tx_provider 41
- wsaw:ServiceName 36
- ws_chain 5, 82
- ws_coloc 6, 11
- WS-Coordination 41
- ws_coordination_service 5
- wsdl:service 36
- WSDLBindingSchema 36
- WSDLPort 26
- wsdl_publish 6, 90
- ws_orb 4
- WS-ReliableMessages 80, 84
- wstrm 6
 - SequenceTerminated 87
- wstrm:AckRequested 88
- wstrm:AcksTo 84
- wstrm_db 6
- WS-S 14

X

- xmlfile_log_stream 6, 91
- xslt 6, 77