Artix™

Configuration Reference

Version 2.1, September 2004

Making Software Work Together™

Updated: 23-Sep-2005

Contents

Preface

What is Covered in this Book

The *Artix Configuration Reference* provides a comprehensive reference for the configuration settings in Artix.

Who Should Read this Book

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

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

How to Use this Book

This book is organized as follows:

- Chapter 1 provides a brief overview of Artix configuration, how it is organized, and the syntax for specifying variable entries.
- Chapter 2 describes the Artix runtime configuration variables.
- Chapter 3 describes the Artix plug-in namespaces and variables.
- Chapter 4 describes the configuration namespaces and variables used to configure Artix security features.
- Chapter 5 describes the CORBA plug-in configuration namespaces and variables.

Online Help

While using the Artix Designer you can access contextual online help, providing:

- A description of your current Artix Designer screen.
- Detailed step-by-step instructions on how to perform tasks from this screen.
- A comprehensive index and glossary.
- A full search feature.

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

- Click the **Help** button on the Artix Designer panel, or
- Select **Contents** from the Help menu.

Finding Your Way Around the Artix Library

The Artix library contains several books that provide assistance for any of the tasks you are trying to perform. The remainder of the Artix library is listed here, with an short description of each book.

If you are new to Artix

You may be interested in reading *Learning About Artix*. This book describes the basic Artix concepts. It also walks you through an example of using Artix to solve a real world problem using code provided in the product.

To design Artix solutions

You should read *Designing Artix Solutions.* This book provides detailed information about using the Artix Designer GUI to create WSDL-based Artix contracts, Artix stub and skeleton code, and Artix deployment descriptors.

This book also provides detailed information about Artix command-line interface and the WSDL extensions used in Artix contracts. It also explains the mappings between data types and Artix bindings.

To develop applications using Artix stub and skeleton code

Depending on your development environment you should read one or more of the following:

- *Developing Artix Applications in C++*. This book discusses the technical aspects of programming applications using the Artix C++ API.
- *Developing Artix Applications in Java*. This book discusses the technical aspects of programming applications using the Artix Java API.

To configure and manage your Artix solution

You should read *Deploying and Managing Artix Solutions.* This describes how to configure and deploy Artix-enabled systems. It also discusses how to manage them when they are deployed.

In addition, if you are integrating Artix with either the IBM Tivoli or BMC Patrol Enterprise Management System, you should read:

- *IONA Tivoli Integration Guide*.
- *IONA BMC Patrol Integration Guide*.

To learn more about Artix security

You should read the *Artix Security Guide*. This outlines how to enable and configure Artix's security features. It also discusses how to integrate Artix solutions into a secure environment.

Have you got the latest version?

The latest updates to the Artix documentation can be found at http://www.iona.com/support/docs. Compare the version details provided there with the last updated date printed on the inside cover of the book you are using (at the bottom of the copyright notice).

Additional Resources for Help

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

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

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

Comments on IONA documentation can be sent to docs-support@iona.com .

Typographical Conventions

This book uses the following typographical conventions:

`Constant width`	Constant width (courier font) in normal text represents portions of code and literal names of items such as classes, functions, variables, and data structures. For example, text might refer to the `CORBA::Object` class. Constant width paragraphs represent code examples or information a system displays on the screen. For example: `#include <stdio.h>`
Italic	Italic words in normal text represent *emphasis* and *new terms*. Italic words or characters in code and commands represent variable values you must supply, such as arguments to commands or path names for your particular system. For example: `% cd /users/`*`your_name`* **Note:** Some command examples may use angle brackets to represent variable values you must supply. This is an older convention that is replaced with *italic* words or characters.

Keying Conventions

This book uses the following keying conventions:

No prompt	When a command's format is the same for multiple platforms, a prompt is not used.
%	A percent sign represents the UNIX command shell prompt for a command that does not require root privileges.
#	A number sign represents the UNIX command shell prompt for a command that requires root privileges.
>	The notation > represents the DOS or Windows command prompt.
. . . . . .	Horizontal or vertical ellipses in format and syntax descriptions indicate that material has been eliminated to simplify a discussion.
[]	Brackets enclose optional items in format and syntax descriptions.
{}	Braces enclose a list from which you must choose an item in format and syntax descriptions.
|	A vertical bar separates items in a list of choices enclosed in {} (braces) in format and syntax descriptions.

CHAPTER 1

Introduction

This chapter introduces the main concepts and components in the Artix runtime configuration (for example, configuration domains, scopes, variables, and data types). It also explains how to use Artix configuration files to manage your applications.

In this chapter

This chapter includes the following sections:

Artix Configuration Concepts

Overview

Artix is built upon IONA's Adaptive Runtime architecture (ART). Runtime behaviors are established through common and application-specific configuration settings that are applied during application startup. As a result, the same application code can be run, and can exhibit different capabilities, in different configuration environments. This section includes the following:

- Configuration domains.
- Configuration scopes.
- Specifying configuration scopes.
- Configuration namespaces.
- Configuration variables.

Configuration domains

An Artix *configuration domain* is a collection of configuration information in an Artix runtime environment. This information consists of configuration variables and their values. A default Artix configuration is provided when Artix is installed. The default Artix configuration domain file has the following location:

Windows	`%IT_PRODUCT_DIR%\artix\2.1\etc\domains\artix.cfg`
UNIX	`$IT_PRODUCT_DIR/artix/2.1/etc/domains/artix.cfg`

The contents of this file can be modified to affect aspects of Artix behavior (for example, logging or routing).

Configuration scopes

An Artix configuration domain is subdivided into *configuration scopes*. These are typically organized into a hierarchy of scopes, whose fully-qualified names map directly to ORB names. By organizing configuration variables into various scopes, you can provide different settings for individual services, or common settings for groups of services.

Local configuration scopes

Configuration scopes apply to a subset of services or to a specific service in an environment. For example, the Artix `demo` configuration scope includes example local configuration scopes for demo applications.

Application-specific configuration variables either override default values assigned to common configuration variables, or establish new configuration variables. Configuration scopes are localized through a name tag and delimited by a set of curly braces terminated with a semicolon, for example, `( scopeNameTag {…}; )`.

A configuration scope may include nested configuration scopes. Configuration variables set within nested configuration scopes take precedence over values set in enclosing configuration scopes.

In the `artix.cfg` file, there are several predefined configuration scopes. For example, the `demo` configuration scope includes nested configuration scopes for some of the demo programs included with the product.

Example 1: *Demo Configuration Scope*

```
demo
{
 fml_plugin
 {
     orb_plugins = ["local_log_stream", "iiop_profile",
            "giop", "iiop", "soap", "http", "G2", "tunnel",
            "mq", "ws_orb", "fml"];
 };
 telco
 {
    orb_plugins = ["local_log_stream", "iiop_profile",
                   "giop", "iiop", "G2", "tunnel"];
    plugins:tunnel:iiop:port = "55002";
    poa:MyTunnel:direct_persistent = "true";
    poa:MyTunnel:well_known_address = "plugins:tunnel";

    server
      {
        orb_plugins = ["local_log_stream", "iiop_profile",
                      "giop", "iiop", "ots", "soap", "http", "G2:,
                        "tunnel"];
        plugins:tunnel:poa_name = "MyTunnel";
     };
 };
```

Example 1: *Demo Configuration Scope*

```
 tibrv
 {
     orb_plugins = ["local_log_stream", "iiop_profile",
                    "giop", "iiop", "soap", "http", "tibrv"];

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

Note: The `orb_plugins` list is redefined within each configuration scope.

Specifying configuration scopes

To make an Artix process run under a particular configuration scope, you specify that scope using the `-ORBname` parameter. Configuration scope names are specified using the following format

`scope.subscope`

For example, the scope for the `telco` server demo shown in Example 1 is specified as `demo.telco.server`. During process initialization, Artix searches for a configuration scope with the same name as the `-ORBname` parameter.

There are two ways of supplying the `-ORBname` parameter to an Artix process:

- Pass the argument on the command line.
- Specify the `-ORBname` as the third parameter to `IT_Bus::init()`.

For example, to start an Artix process using the configuration specified in the `demo.tibrv` scope, you could start the process use the following syntax:

```
<processName> [application parameters] -ORBname demo.tibrv
```

Alternately, you could use the following code fragment to initialize the Artix bus:

```
IT_Bus::init (argc, argv, “demo.tibrv”);
```

If a corresponding scope is not located, the process starts under the highest level scope that matches the specified scope name. If there are no scopes that correspond to the `ORBname` parameter, the Artix process runs under the default global scope. For example, if the nested `tibrv` scope does not exist, the Artix process uses the configuration specified in the `demo` scope; if the `demo` scope does not exist, the process runs under the default global scope.

Configuration namespaces

Most configuration variables are organized within namespaces, which group related variables. Namespaces can be nested, and are delimited by colons (`:`). For example, configuration variables that control the behavior of a plug-in begin with `plugins:` followed by the name of the plug-in for which the variable is being set. For example, to specify the port on which the Artix standalone service starts, set the following variable:

```
plugins:artix_service:iiop:port
```

To set the location of the routing plug-in's contract, set the following variable:

```
plugins:routing:wsdl_url
```

Configuration variables

Configuration data is stored in variables that are defined within each namespace. In some instances, variables in different namespaces share the same variable names.

Variables can also be reset several times within successive layers of a configuration scope. Configuration variables set in narrower configuration scopes override variable settings in wider scopes. For example, a `company.operations.orb_plugins` variable would override a `company.orb_plugins` variable. Plug-ins specified at the `company` scope would apply to all processes in that scope, except those processes that belong specifically to the `company.operations` scope and its child scopes.

Configuration Data Types

Overview

Each Artix configuration variable has an associated data type that determines the variable's value.

Data types can be categorized as follows:

- Primitive types
- Constructed types

Primitive types

Artix supports the following three primitive types:

- `boolean`
- `double`
- `long`

Constructed types

Artix supports two constructed types: `string` and `ConfigList` (a sequence of strings).

- In an Artix configuration file, the `string` character set is ASCII.
- The `ConfigList` type is simply a sequence of `string` types. For example:

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

Artix Configuration Files

Overview

This section explains how to use Artix configuration files to manage applications in your environment. It includes the following:

- "Default configuration file".
- "Importing configuration settings".
- "Working with multiple installations".

Default configuration file

The Artix configuration domain file contains all the configuration settings for the domain. You can edit the settings in this file to modify different aspects of Artix behavior (for example, routing or levels of logging).

The default Artix configuration domain file is found in the following location:

Windows	`%IT_PRODUCT_DIR%\artix\2.1\etc\domains\artix.cfg`
UNIX	`$IT_PRODUCT_DIR/artix/2.1/etc/domains/artix.cfg`

Importing configuration settings

You can manually create new Artix configuration domain files to compartmentalize your applications. These new configuration domain files can import information from other configuration domains using an `include` statement in your configuration file. This provides a convenient way of compartmentalizing your application-specific configuration from the global ART configuration information that is contained in the default configuration domain file.

Example 2 shows an `include` statement that imports the default configuration file. The include statement is typically the first line the configuration file.

Example 2: *Configuration file include statement*

```
include "../../../../../etc/domains/artix.cfg";

my_app_config {
...
}
```

For complete working examples of Artix applications that use this import mechanism, see the configuration files provided with Artix demos. These demo applications are available from the following directory:

`<install-dir>\artix\2.1\demos`

Working with multiple installations

If you are using multiple installations or versions of Artix, you can use your configuration files to help manage your applications as follows:

1. Install each version of Artix into a different directory.
2. Install your applications into their own directory.
3. Copy the `artix.cfg` file from whichever Artix release you want to use into another directory (for example, an application directory).
4. In your application's local configuration file, include the `artix.cfg` file from your copy location.

This enables you to switch between Artix versions by copying the corresponding `artix.cfg` file into a common location. This avoids having to update the directory information in your configuration file whenever you want to switch between Artix versions.

CHAPTER 2

Artix Runtime Configuration

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

In this chapter

This chapter includes the following:

ORB Plug-ins

Overview

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

The default `orb_plugins` entry includes the following:

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

All other plug-ins implementing bindings and transports load transparently when the WSDL file is loaded into an application. These plugins do not need to be explicitly listed in `orb_plugins`. However, other service plug-ins (for example, for security, locator, session manager, routing, XSLT transformation, logging, etc.) must all be listed in the `orb_plugins` entry.

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:

- "Transport plug-ins".
- "Payload format plug-ins".
- "Service plug-ins".

Java plug-ins

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

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 3.

Example 3: *Including the Java Plug-in Loader*

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

In addition to including the Java plug-in loader in the `orb_plugin` list you need to include another configuration variable, `java_plugins`, that 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 the Java plug-ins should not be listed in `orb_plugins`.

For example if you are using a Java plug-in called `java_handler` in your application you would use the configuration similar to the fragment shown in Example 4 to load the plug-ins.

Example 4: *Loading a Java Plug-in*

```
orb_plugins=["xml_log_stream", "java"];
java_plugins=["java_handler"];
```

Transport plug-ins

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

Table 1: *Artix Transport Plug-ins*

Plug-in	Transport
`http`	Provides support for HTTP and HTTPS.
`iiop`	Provides support for CORBA IIOP.
`iiop_profile`	Provides support for CORBA IIOP profile.
`giop`	Provides support for CORBA GIOP.
`tunnel`	Provides support for the IIOP transport using non-CORBA payloads.
`tuxedo`	Provides support for Tuxedo interoperability.

Table 1: *Artix Transport Plug-ins*

Plug-in	Transport
`mq`	Provides support for WebSphere MQ interoperability.
`tibrv`	Provides support for TIBCO Rendezvous interoperability.
`java`	Provides support for Java Message Service (JMS) interoperability.

Payload format plug-ins

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

Table 2: *Artix Payload Format Plug-ins*

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

Service plug-ins

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

Table 3: *Artix Service Plug-ins*

Plug-in	Artix Feature
`routing`	Enables Artix routing.
`locator_endpoint`	Enables endpoints to use the Artix locator service.
`service_locator`	Enables the Artix locator. An Artix server acting as the locator service must load this plug-in.
`wsdl_publish`	Enables Artix endpoints to publish and use Artix object references.
`bus_response_monitor`	Enables performance logging. Monitors response times of Artix client/server requests.
`session_manager_service`	Enables the Artix session manager. An Artix server acting as the session manager must load this plug-in.
`session_endpoint_manager`	Enables the Artix session manager. Endpoints wishing to be managed by the session manager must load this plug-in.
`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.
`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.
`ws_chain`	Enables you to link together a series of services into a multi-part process.
`xmlfile_log_stream`	Enables you to view Artix logging output in a file.
`xslt`	Enables Artix to process XSLT scripts.

Policies

Overview

The `policies` namespace contains the following variables for controlling the publishing of server hostnames:

- http:server_address_mode_policy:publish_hostname
- soap:server_address_mode_policy:publish_hostname

If the policy corresponding to the transport is used by the server, the dynamically generated contract will be published with the original contents of the address element.

http:server_address_mode_policy:publish_hostname

`http:server_address_mode_policy:publish_hostname` specifies how the server's address is published in dynamically generated Artix contracts. When set this policy is set to `false`, the dynamically generated contract will publish the IP address of the running server in the `<http:address>` element describing the server's location. When this policy is set to `true`, the hostname of the machine hosting the running server is published in the `<http:address>` element describing the server's location.

soap:server_address_mode_policy:publish_hostname

`soap:server_address_mode_policy:publish_hostname` specifies how the server's address is published in dynamically generated Artix contracts. When set this policy is set to `false`, the dynamically generated contract will publish the IP address of the running server in the `<soap:address>` element describing the server's location. When this policy is set to `true`, the hostname of the machine hosting the running server is published in the `<soap:address>` element describing the server's location.

Binding Lists

Overview

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

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

The `binding` namespace includes the following variables:

- `client_binding_list`
- `server_binding_list`

client_binding_list

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

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

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

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

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

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

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

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

server_binding_list

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

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

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

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

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

Binding Lists for Custom Interceptors

Overview

The `binding:artix` namespace includes variables that configure Artix applications to use custom-based interceptors. Message handlers 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 "`tns:mercury+tns:hermes`", a message is passed into the message handler `mercury` as it leaves the binding. When `mercury` processes the message, it is passed into `hermes` for more processing. `hermes` then passes the message along to the application code.

All message handlers are specified as a qualified name (QName). This is a unique tag name in an XML document, consisting of a namespace URI and a local part (for example, "`tns:mercury`"). The namespace must match the namespace of the WSDL file that you are using. In addition, the interceptor chain must be a single string, and each interceptor name must be separated by a + delimiter (for example, "`tns:mercury+tns:hermes`").

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`

client_message_interceptor_list

`binding:artix:client_message_interceptor_list` is an ordered list of QNames that specifies the message-level handers for a Java or C++ client application. Entries take the following format:

```
binding:artix:client_message_interceptor_list =
 "tns:message_handler_1+tns:message_handler_2";
```

There is no default value.

client_request_interceptor_list

`binding:artix:client_request_interceptor_list` is an ordered list of QNames that specifies the request-level handlers for a Java or C++ client application. Entries take the following format:

```
binding:artix:client_request_interceptor_list =
 "tns:request_handler_1+tns:request_handler_2";
```

There is no default value.

server_message_interceptor_list

`binding:artix:server_message_interceptor_list` is an ordered list of QNames that specifies the message-level handlers for a Java or C++ server application. Entries take the following format:

```
binding:artix:server_message_interceptor_list =
 "tns:message_handler_1+tns:message_handler_2";
```

There is no default value.

server_request_interceptor_list

`binding:artix:server_request_interceptor_list` is an ordered list of QNames that specifies the request-level handlers for a Java or C++ server application. Entries take the following format:

```
binding:artix:server_request_interceptor_list =
 "tns:request_handler_1+tns:request_handler_2";
```

There is no default value.

Event Log

The event_log namespace control logging levels in Artix. It contains the event_log:filters variable.

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. For more details about using this variable, see *Deploying and Managing Artix Solutions*.

Thread Pool Control

Overview

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

```
thread_pool:variable_name
```

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

```
thread_pool:variable_name:service_uri:service_name
```

The high and low water mark settings specify the values for the thread pool on a per-service basis. However, the initial thread setting works on a per-port basis. This namespace includes following variables:

- `initial_threads`
- `low_water_mark`
- `high_water_mark`

initial_threads

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

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

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

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

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

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

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

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

---

low_water_mark

`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 example is a global setting:

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

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

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

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

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

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

high_water_mark

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

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

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

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

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

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

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

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

Custom Plug-in Configuration

Overview

When you write a custom plug-in for Artix, in either C++ or Java, you need to provide some configuration information to the Artix runtime so that Artix can locate the libraries and initial settings required to properly instantiate the plug-in. This information is provided in the Artix configuration file used by your application. Typically you will want to place the information in the global scope so that more than one of your applications can use the plug-in.

C++ plug-in configuration

When writing custom C++ plug-ins you build your plug-in as a shared library that the bus loads at runtime. In the Artix configuration file you need to provide the name of the shared library that loads the plug-in. This is done using the configuration variable `plugins:`*`plugin_name`*`:shlib_name`. The plug-in name provided must correspond to the plug-in name listed in the `orb_plugins` list.

Example 5 shows an example of configuring a custom plug-in called `my_filter` that is implemented by the shared library `my_filter.dll`.

Example 5: *Custom C++ Plug-in Configuration*

```
plugins:my_filter:shlib_name="my_filter"
...
my_app
{
  orb_plugins=["my_filter" ...];
  ...
}
```

Java plug-in configuration

Java plug-ins are loaded using the plug-in factory you implemented for the custom plugin. In the Artix configuration file you need to provide that name for the plug-in factory class. This is done using the configuration variable `plugins:`*`plugin_name`*`:classname`. The plug-in name provided must correspond to the plug-in name listed in the `orb_plugins` list.

Example 6 shows an example of configuring a custom plug-in called `my_java_filter` that has the factory class `myJavaFilterFactory`.

Example 6: *Custom Java Plug-in Configuration*

```
plugins:my_java_filter:shlib_name="myJavaFilterFactory"
...
my_app
{
  orb_plugins=[..., "java"];
  java_plugins=["my_java_filter"];
  ...
}
```

Plug-in dependencies

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. This information in provided in the configuration variable `plugins:`*`plugin_name`*`: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 will ensure that the required plug-ins are loaded when ever your plug-in is loaded.

CHAPTER 3

Artix Plug-in Configuration

Artix is built on IONA's Adaptive Runtime architecture (ART), which enables users to configure services as plugins to the core product. This chapter explains the configuration settings for Artix-specific plug-ins. For information on CORBA plug-ins, see Chapter 5.

Overview

Each Artix transport, payload format, and service has properties that are configurable as plug-ins to the Artix runtime. The variables used to configure plug-in behavior are specified in the configuration scopes of each Artix runtime instance, and follow the same order of precedence. A plug-in setting specified in the global configuration scope is overridden in favor of 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 `widget_form` scope, all Artix runtimes, except for those running in the `widget_form` scope, would use `true` for this value. Any Artix instance using the `widget_form` scope would use `false` for this value.

In this chapter

This chapter includes the following:

Locator Service

Overview

The locator service plugin, `service_locator`, has the following configuration variables:

- plugins:locator:service_url
- plugins:locator:peer_timeout

plugins:locator:service_url

`plugins:locator:service_url` specifies the location of the Artix contract defining the location service and configuring its address. A copy of this contract, `locator.wsdl`, is located in the `wsdl` folder of your Artix installation.

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 is `4000000` (4 seconds).

The locator uses a third-party peer manager to ping its endpoints. For more details, see "Peer Manager" on page 29.

Locator Service Endpoint

Overview

The locator service endpoint plug-in, `locator_endpoint`, has the following configuration variables:

- plugins:locator:wsdl_url
- plugins:locator:peer_timeout

plugins:locator:wsdl_url

`plugins:locator:wsdl_url` specifies the location of the Artix contract that defines the location service, and specifies the address locator endpoints use to communicate with the locator service. A copy of this contract, `locator.wsdl`, is located in the `wsdl` folder of your Artix installation.

plugins:locator:peer_timeout

`plugins:locator:peer_timeout` specifies the amount of time, in milliseconds, that the locator endpoint plug-in waits between keep-alive pings back to the locator. The default is `4000000` (4 seconds).

The locator service endpoint uses a third-party peer manager to ping back to the locator. For more details, see "Peer Manager" on page 29.

Peer Manager

Overview

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

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

The `peer_manager` includes the following configuration variables:

- plugins:peer_manager:wsdl_url
- plugins:peer_manager:timeout_delta

plugins:peer_manager:wsdl_url

`plugins:peer_manager:wsdl_url` specifies the location of the Artix contract defining the peer manager service. A copy of this contract, `peer_manager.wsdl`, is located in the `wsdl` folder of your Artix installation.

plugins:peer_manager:timeout_delta

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

Response Time Collector

Overview

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

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

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

plugins:it_response_time_collector:client-id

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

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

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

plugins:it_response_time_collector:filename

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

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

plugins:it_response_time_collector:log_properties

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

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

plugins:it_response_time_collector:period

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

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

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

plugins:it_response_time_collector:server-id

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

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

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

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

plugins:it_response_time_collector:syslog_appID

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

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

plugins:it_response_time_collector:system_logging_enabled

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

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

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

Routing Plug-in

Overview

The routing plug-in uses the following variables:

- plugins:routing:wsdl_url
- plugins:routing:use_pass_through

plugins:routing:wsdl_url

`plugins:routing:wsdl_url` specifies the URL to search for Artix contracts containing the routing rules for your application. 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";
```

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

plugins:routing:use_pass_through

`plugins:routing:use_pass_through` specifies if the routing plug-in uses the pass-through routing optimization. This optimization enables the router to copy the message buffer directly from the source endpoint to the destination endpoint (if both use the same binding). The default value is `true`.

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

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

Service Lifecycle

Overview

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

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

The Artix `plugins:service_lifecycle` namespace has the following variable:

`plugins:service_lifecycle:max_cache_size`

plugins:service_lifecycle:max_cache_size

`plugins:service_lifecycle:max_cache_size` specifies the maximum cache size of the service lifecycle. For example:

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

To enable service lifecycle, you must also add the `service_lifecycle` plugin to the `orb_plugins` list, for example:

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

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

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

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

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

Session Manager

Overview

The session manager, `session_manager_service`, has the following configuration variables:

- plugins:session_manager_service:service_url
- plugins:session_manager_service:peer_timeout

plugins:session_manager_service:service_url

`plugins:session_manager_service:service_url` specifies the location of the Artix contract defining the session manager. A copy of this contract, `session-manager.wsdl`, is located in the `wsdl` folder of your Artix installation.

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 is `4000000` (`4` seconds).

The session manager uses a third-party peer manager to ping its endpoints For more details, see "Peer Manager" on page 29.

Session Manager Endpoint

Overview

The session manager endpoint plug-in, `session_endpoint_manager`, has the following configuration variables:

- plugins:session_endpoint_manager:wsdl_url
- plugins:session_endpoint_manager:endpoint_manager_url
- plugins:session_endpoint_manager:default_group
- plugins:session_endpoint_manager:header_validation
- plugins:session_endpoint_manager:peer_timeout

plugins:session_endpoint_manager:wsdl_url

`plugins:session_endpoint_manager:wsdl_url` specifies the location of the contract defining the session management service that the endpoint manager is to contact.

plugins:session_endpoint_manager:endpoint_manager_url

`plugins:session_endpoint_manager:endpoint_manager_url` specifies the location of the contract defining the endpoint manager. The contract contains the contact information for the endpoint manager.

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 is `4000000` (4 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 29.

Session Manager Simple Policy

Overview

The session manager's simple policy plug-in, `sm_simple_policy`, has the following configuration variables:

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

plugins:sm_simple_policy:max_concurrent_sessions

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

plugins:sm_simple_policy:min_session_timeout

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

plugins:sm_simple_policy:max_session_timeout

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

SOAP Plug-in

Overview

The SOAP plug-in, soap, has the following configuration setting:

- plugins:soap:encoding

plugins:soap:encoding

`plugins:soap:encoding` specifies the character encoding used when the SOAP plugin 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).

Transformer Service

Overview

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

- artix:endpoint:endpoint_list
- artix:endpoint:endpoint_name:wsdl_location
- artix:endpoint:endpoint_name:service_namespace
- artix:endpoint:endpoint_name:service_name
- artix:endpoint:endpoint_name:port_name

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

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

artix:endpoint:endpoint_list

`artix:endpoint:endpoint_list` specifies a list of endpoint names that will be 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 plugin and the Artix transformer.

artix:endpoint:*endpoint_name*:wsdl_location

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

artix:endpoint:*endpoint_name*:service_namespace

`artix:endpoint:`*`endpoint_name`*`:service_namespace` specifies the XML namespace in which the interface for this endpoint is defined.

artix:endpoint:*endpoint_name*:service_name

`artix:endpoint:`*`endpoint_name`*`:service_name` specifies the name of the `<portType>` that defines this endpoint's logical interface.

artix:endpoint:*endpoint_name*:port_name

`artix:endpoint:endpoint_name:port_name` specifes the `<port>` that defines the physical representation of the endpoint

plugins:xslt:servant_list

`plugins:xslt:servant_list` specifies a list of endpoints that will be instaniated as servants by the transformer.

plugins:xslt:*endpoint_name*:operation_map

`plugins:xslt:`*`endpoint_name`*`:operation_map` specifies an ordered list of XSLT operations and scripts to be used in processing the recieved XML messages.

Tuxedo Plug-in

Overview

The Tuxedo plug-in has only one configuration variable:

- plugins:tuxedo:server

plugins:tuxedo:server

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

Web Service Chain Service

Overview

The Web service 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:servant_list
- plugins:chain:endpoint_name:client:operation_list
- plugins:chain:endpoint_name:operation_name:service_chain

plugins:chain:servant_list

`plugins:chain:servant_list` specifies a list of the endpoints in the Web service chain. Each name in the list must correspond to an endpoint specified in the `artix:endpoint:endpoint_list` set in the configuration scope.

plugins:chain:endpoint_name:client:operation_list

`plugins:chain:`*`endpoint_name`*`:operation_list` specifies the list of operations the Web service chain plug-in is implementing. The operations in the list must be defined in the Artix contract defining the endpoint specified by *`endpoint_name`*.

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

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

Service chains are specified using the syntax shown in Example 7.

Example 7: *Service Chain Specification 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.

WSDL Publishing Service

Overview

The WSDL publishing service, `wsdl_publishing`, has the following configuration variables:

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

plugins:wsdl_publish:publish_port

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

plugins:wsdl_publish:hostname

`plugins:wsdl_publish:hostname` specifies how the hostname will be published. By default, the local name of the machine will be published. The possible values are as follows:

`canonical`	Publishes the fully qualified hostname of the machine in the dynamic WSDL.
`unqualified`	Publishes the unqualified local hostname of the machine in the dynamic WSDL. This does not include domain name with the hostname.
`ipaddress`	Publishes the IP address associated with the machine in the dynamic WSDL.

XML File Log Stream

Overview

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

- "plugins:xmlfile_log_stream:filename".
- "plugins:xmlfile_log_stream:max_file_size".
- "plugins:xmlfile_log_stream:rolling_file".
- "plugins:xmlfile_log_stream:use_pid".

plugins:xmlfile_log_stream:filename

`plugins:xmlfile_log_stream:filename` specifies an optional filename for your log file, for example:

```
plugins:xmlfile_log_stream:filename = "artix_logfile.xml";
```

The default filename is `it_bus.log`.

plugins:xmlfile_log_stream:max_file_size

`plugins:xmlfile_log_stream:max_file_size` specifies an optional maximum size for your log file,for example:

```
plugins:xmlfile_log_stream:max_file_size = "100000";
```

The default maximum size is 2 MB.

plugins:xmlfile_log_stream:rolling_file

`plugins:xmlfile_log_stream:rolling_file` specifies that the logging plug-in uses a rolling file to prevent the local log from growing indefinitely. In this model, the log stream appends the current date to the configured filename. This produces a complete filename, for example:

```
/var/adm/art.log.02171999
```

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:

```
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";
```

CHAPTER 4

Artix Security

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

In this chapter

This chapter discusses the following topics:

Applying Constraints to Certificates

Certificate constraints policy

You can use the `CertConstraintsPolicy` to apply constraints to peer X.509 certificates by the default `CertificateValidatorPolicy`. These conditions are applied to the owner's distinguished name (DN) on the first certificate (peer certificate) of the received certificate chain. Distinguished names are made up of a number of distinct fields, the most common being Organization Unit (OU) and Common Name (CN).

Configuration variable

You can specify a list of constraints to be used by `CertConstraintsPolicy` through the `policies:iiop_tls:certificate_constraints_policy` or `policies:https:certificate_constraints_policy` configuration variables. For example:

```
policies:iiop_tls:certificate_constraints_policy =
   ["CN=Johnny*,OU=[unit1|IT_SSL],O=IONA,C=Ireland,ST=Dublin,L=Ea
   rth","CN=Paul*,OU=SSLTEAM,O=IONA,C=Ireland,ST=Dublin,L=Earth",
"CN=TheOmnipotentOne"];
```

Constraint language

These are the special characters and their meanings in the constraint list:

`*`	Matches any text. For example: `an*` matches `ant` and `anger`, but not `aunt`
`[ ]`	Grouping symbols.
`|`	Choice symbol. For example: `OU=[unit1|IT_SSL]` signifies that if the `OU` is `unit1` or `IT_SSL`, the certificate is acceptable.
`=, !=`	Signify equality and inequality respectively.

Example

This is an example list of constraints:

```
policies:iiop_tls:certificate_constraints_policy = [
   "OU=[unit1|IT_SSL],CN=Steve*,L=Dublin",
"OU=IT_ART*,OU!=IT_ARTtesters,CN=[Jan|Donal],ST=
Boston" ];
```

This constraint list specifies that a certificate is deemed acceptable if and only if it satisfies one or more of the constraint patterns:

```
If
     The OU is unit1 or IT_SSL
     And
     The CN begins with the text Steve
     And
     The location is Dublin
Then the certificate is acceptable
Else  (moving on to the second constraint)
If
     The OU begins with the text IT_ART but isn't IT_ARTtesters
     And
     The common name is either Donal or Jan
     And
     The State is Boston
Then the certificate is acceptable
Otherwise the certificate is unacceptable.
```

The language is like a boolean OR, trying the constraints defined in each line until the certificate satisfies one of the constraints. Only if the certificate fails all constraints is the certificate deemed invalid.

Note that this setting can be sensitive about white space used within it. For example, `"CN ="` might not be recognized, where `"CN="` is recognized.

Distinguished names

For more information on distinguished names, see the *Security Guide*.

initial_references

The `initial_references` namespace contains the following configuration variables:

- IT_TLS_Toolkit:plugin

IT_TLS_Toolkit:plugin

(Windows only.) This configuration variable enables you to specify the underlying SSL/TLS toolkit to be used by Artix. It is used in conjunction with the `plugins:baltimore_toolkit:shlib_name` and `plugins:schannel_toolkit:shlib_name` configuration variables to implement SSL/TLS toolkit replaceability.

The default is the Baltimore toolkit.

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

```
initial_references:IT_TLS_Toolkit:plugin = "schannel_toolkit";
plugins:schannel_toolkit:shlib_name = "it_tls_schannel";
```

plugins:asp

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

- authentication_cache_size
- authentication_cache_timeout
- authorization_realm
- default_password
- security_type
- security_level

authentication_cache_size

For SOAP bindings, 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

For SOAP bindings, 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.

A value of -1 (the default) means an infinite time-out. A value of `0` means disable the cache.

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 plugins:asp:security_type variable is set to either PRINCIPAL or CERT_SUBJECT, this 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.

When either the PRINCIPAL or CERT_SUBJECT security type is selected, 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.

security_type

Specifies the source of the user identity that is sent to the Artix security service for authentication. Because the Artix Security Framework supports several different security mechanisms for propagating user identities, it is necessary to specify which of the propagated identities is actually used for the authentication step. The following options are currently supported by the artix_security plug-in:

USERNAME_PASSWORD	Authenticate the username and password propagated as WSDL message attributes. For example, you can configure these values on the client side using the UserName and Password attributes in the <http-conf:client> tag in the WSDL contract.
CERT_SUBJECT	Authenticate the Common Name (CN) from the client certificate's subject DN.

`ENCODED_TOKEN`	*Reserved for future use.*
`KERBEROS_TOKEN`	Authenticate the Kerberos token. You must have the Kerberos adapter configured to use this option. For more information.
`PRINCIPAL`	Authenticate the CORBA principal. This is needed to support interoperability with legacy CORBA applications. This options can be used in combination with the `plugins:asp:default_password` setting.

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:atli2_tls

The `plugins:atli2_tls` namespace contains the following variable:

- use_jsse_tk

use_jsse_tk

(Java only) Specifies whether or not to use the JSSE/JCE architecture with the CORBA binding. If `true`, the CORBA binding uses the JSSE/JCE architecture to implement SSL/TLS security; if `false`, the CORBA binding uses the Baltimore SSL/TLS toolkit.

The default is `false`.

plugins:csi

The `policies:csi` namespace includes variables that specify settings for Common Secure Interoperability version 2 (CSIv2):

- `ClassName`
- `shlib_name`

ClassName

`ClassName` specifies the Java class that implements the `csi` plugin. The default setting is:

```
plugins:csi:ClassName = "com.iona.corba.security.csi.CSIPlugin";
```

This configuration setting makes it possible for the Artix core to load the plugin on demand. Internally, the Artix core uses a Java class loader to load and instantiate the `csi` class. Plugin loading can be initiated either by including the `csi` in the `orb_plugins` list, or by associating the plugin with an initial reference.

shlib_name

`shlib_name` identifies the shared library (or DLL in Windows) containing the `csi` plugin implementation.

```
plugins:csi:shlib_name = "it_csi_prot";
```

The `csi` plug-in becomes associated with the `it_csi_prot` shared library, where `it_csi_prot` is the base name of the library. The library base name, `it_csi_prot`, is expanded in a platform-dependent manner to obtain the full name of the library file.

plugins:gsp

The `plugins:gsp` namespace includes variables that specify settings for the Generic Security Plugin (GSP). This provides authorization by checking a user's roles against the permissions stored in an action-role mapping file. It includes the following:

- `accept_asserted_authorization_info`
- `assert_authorization_info`
- `authentication_cache_size`
- `authentication_cache_timeout`
- `authorization_realm`
- `ClassName`
- `enable_authorization`
- `enable_gssup_sso`
- `enable_x509_sso`
- `enforce_secure_comms_to_sso_server`
- `enable_security_service_cert_authentication`
- `sso_server_certificate_constraints`

accept_asserted_authorization_info

If `false`, SAML data is not read from incoming connections. Default is `true`.

assert_authorization_info

If `false`, SAML data is not sent on outgoing connections. Default is `true`.

authentication_cache_size

The maximum number of credentials stored in the authentication cache. If this size is exceeded the oldest credential in the cache is removed.

A value of -1 (the default) means unlimited size. A value of `0` means disable the cache.

authentication_cache_timeout

The time (in seconds) after which a credential is considered *stale*. Stale credentials are removed from the cache and the server must re-authenticate with the Artix security service on the next call from that user. The cache timeout should be configured to be smaller than the timeout set in the `is2.properties` file (by default, that setting is `is2.sso.session.timeout=600`).

A value of -1 (the default) means an infinite time-out. A value of `0` means disable the cache.

authorization_realm

`authorization_realm` specifies the iSF authorization realm to which a server belongs. The value of this variable determines which of a user's roles are considered when making an access control decision.

For example, consider a user that belongs to the `ejb-developer` and `corba-developer` roles within the `Engineering` realm, and to the ordinary role within the Sales realm. If you set `plugins:gsp:authorization_realm` to Sales for a particular server, only the ordinary role is considered when making access control decisions (using the `action-role` mapping file).

ClassName

`ClassName` specifies the Java class that implements the `gsp` plugin. This configuration setting makes it possible for the Artix core to load the plugin on demand. Internally, the Artix core uses a Java class loader to load and instantiate the `gsp` class. Plugin loading can be initiated either by including the `csi` in the `orb_plugins` list, or by associating the plugin with an initial reference.

enable_authorization

A boolean GSP policy that, when `true`, enables authorization using action-role mapping ACLs in server.

Default is `true`.

enable_gssup_sso

Enables SSO with a username and a password (that is, GSSUP) when set to `true`.

enable_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 53.

plugins:http

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

- `client:client_certificate`
- `client:client_certificate_chain`
- `client:client_private_key`
- `client:client_private_key_password`
- `client:trusted_root_certificates`
- `client:use_secure_sockets`
- `server:server_certificate`
- `server:server_certificate_chain`
- `server:server_private_key`
- `server:server_private_key_password`
- `server:trusted_root_certificates`
- `server:use_secure_sockets`

client:client_certificate

This variable specifies the full path to the PEM-encoded X.509 certificate issued by the certificate authority for the client. For example:

```
plugins:http:client:client_certificate =
    "c:\aspen\x509\certs\key.cert.pem"
```

This setting is ignored if `plugins:http:client:use_secure_sockets` is `false`.

client:client_certificate_chain

(Optional) This variable specifies the full path to the PEM-encoded X.509 certificate chain for the client. For example:

```
plugins:http:client:client_certificate_chain =
    "c:\aspen\x509\certs\key.cert.pem"
```

This setting is ignored if `plugins:http:client:use_secure_sockets` is `false`.

client:client_private_key

This variable specifies a PEM file containing the client certificate's encrypted private key. This private key enables the client to respond to a challenge from a server during an SSL/TLS handshake.

This setting is ignored if `plugins:http:client:use_secure_sockets` is `false`.

client:client_private_key_password

This variable specifies the password to decrypt the contents of the `client_private_key` file.

This setting is ignored if `plugins:http:client:use_secure_sockets` is `false`.

client:trusted_root_certificates

This variable specifies the path to a file containing a concatenated list of CA certificates in PEM format. The client uses this CA list during the TLS handshake to verify that the server's certificate has been signed by a trusted CA.

This setting is ignored if `plugins:http:client:use_secure_sockets` is `false`.

client:use_secure_sockets

This variable specifies whether the client wants to open a HTTPS connection (that is, HTTP running over SSL or TLS) or an insecure connection (that is, plain HTTP).

Valid values are `true`, for HTTPS, and `false`, for HTTP. The default is `false`.

server:server_certificate

This variable specifies the full path to the PEM-encoded X.509 certificate issued by the certificate authority for the server. For example:

```
plugins:http:server:server_certificate =
   "c:\aspen\x509\certs\key.cert.pem"
```

This setting is ignored if `plugins:http:server:use_secure_sockets` is `false`.

server:server_certificate_chain

(Optional) This variable specifies the full path to the PEM-encoded X.509 certificate chain for the server. For example:

```
plugins:http:server:server_certificate_chain =
   "c:\aspen\x509\certs\key.cert.pem"
```

This setting is ignored if `plugins:http:server:use_secure_sockets` is `false`.

server:server_private_key

This variable specifies a PEM file containing the server certificate's encrypted private key. This private key enables the server to respond to a challenge from a client during an SSL/TLS handshake.

This setting is ignored if `plugins:http:server:use_secure_sockets` is `false`.

server:server_private_key_password

This variable specifies the password to decrypt the contents of the `server_private_key` file.

This setting is ignored if `plugins:http:server:use_secure_sockets` is `false`.

server:trusted_root_certificates

This variable specifies the path to a file containing a concatenated list of CA certificates in PEM format. The server uses this CA list during the TLS handshake to verify that the client's certificate has been signed by a trusted CA.

This setting is ignored if `plugins:http:server:use_secure_sockets` is `false`.

server:use_secure_sockets

This variable specifies whether the server accepts HTTPS connection attempts (that is, HTTP running over SSL or TLS) or insecure connection attempts (that is, plain HTTP) from a client.

Valid values are `true`, for HTTPS, and `false`, for HTTP. The default is `false`.

plugins:iiop_tls

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

buffer_pool:recycle_segments

(Java only) When this variable is set, the `iiop_tls` plug-in reads this variable's value instead of the `plugins:iiop:buffer_pool:recycle_segments` variable's value.

buffer_pool:segment_preallocation

(Java only) When this variable is set, the `iiop_tls` plug-in reads this variable's value instead of the `plugins:iiop:buffer_pool:segment_preallocation` variable's value.

buffer_pools:max_incoming_buffers_in_pool

(C++ only) When this variable is set, the `iiop_tls` plug-in reads this variable's value instead of the `plugins:iiop:buffer_pools:max_incoming_buffers_in_pool` variable's value.

buffer_pools:max_outgoing_buffers_in_pool

(C++ only) When this variable is set, the `iiop_tls` plug-in reads this variable's value instead of the `plugins:iiop:buffer_pools:max_outgoing_buffers_in_pool` variable's value.

delay_credential_gathering_until_handshake

(Windows and Schannel only) This client configuration variable provides an alternative to using the `principal_sponsor` variables to specify an application's own certificate. When this variable is set to `true` and `principal_sponsor:use_principal_sponsor` is set to `false`, the client delays sending its certificate to a server. The client will wait until the server *explicitly* requests the client to send its credentials during the SSL/TLS handshake.

This configuration variable can be used in conjunction with the `plugins:schannel:prompt_with_credential_choice` configuration variable.

enable_iiop_1_0_client_support

This variable enables client-side interoperability of Artix SSL/TLS applications with legacy IIOP 1.0 SSL/TLS servers, which do not support IIOP 1.1.

The default value is `false`. When set to `true`, Artix SSL/TLS searches secure target IIOP 1.0 object references for legacy IIOP 1.0 SSL/TLS tagged component data, and attempts to connect on the specified port.

Note: This variable will not be necessary for most users.

incoming_connections:hard_limit

Specifies the maximum number of incoming (server-side) connections permitted to IIOP. IIOP does not accept new connections above this limit. Defaults to -1 (disabled).

When this variable is set, the `iiop_tls` plug-in reads this variable's value instead of the `plugins:iiop:incoming_connections:hard_limit` variable's value.

Please see the chapter on ACM in the *CORBA Programmer's Guide* for further details.

incoming_connections:soft_limit

Specifies the number of connections at which IIOP should begin closing incoming (server-side) connections. Defaults to -1 (disabled).

When this variable is set, the `iiop_tls` plug-in reads this variable's value instead of the `plugins:iiop:incoming_connections:soft_limit` variable's value.

Please see the chapter on ACM in the *CORBA Programmer's Guide* for further details.

outgoing_connections:hard_limit

When this variable is set, the `iiop_tls` plug-in reads this variable's value instead of the `plugins:iiop:outgoing_connections:hard_limit` variable's value.

outgoing_connections:soft_limit

When this variable is set, the `iiop_tls` plug-in reads this variable's value instead of the `plugins:iiop:outgoing_connections:soft_limit` variable's value.

tcp_listener:reincarnate_attempts

(C++/Windows only)

`plugins:iiop_tls:tcp_listener:reincarnate_attempts` specifies the number of attempts that are made to reincarnate a listener before giving up, logging a fatal error, and shutting down the ORB. Datatype is `long`. Defaults to `0` (no attempts).

Sometimes an network error may occur, which results in a listening socket being closed. On Windows, you can configure the listener to attempt a reincarnation. This enables new connections to be established.

tcp_listener:reincarnation_retry_backoff_ratio

(C++/Windows only)
`plugins:iiop_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

(C++/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:is2_authorization

The plugins:is2_authorization namespace contains the following variable:

- action_role_mapping

action_role_mapping

Specifies the action-role mapping file URL. For example:

```
plugins:is2_authorization:action_role_mapping =
   "file:///my/action/role/mapping";
```

plugins:kdm

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

- cert_constraints
- iiop_tls:port
- checksums_optional

cert_constraints

Specifies the list of certificate constraints for principals attempting to open a connection to the KDM server plug-in. See "Applying Constraints to Certificates" on page 53 for a description of the certificate constraint syntax.

To protect the sensitive data stored within it, the KDM applies restrictions on which entities are allowed talk to it. A security administrator should choose certificate constraints that restrict access to the following principals:

- The locator service (requires read-only access).
- The `kdm_adm` plug-in, which is normally loaded into the itadmin utility (requires read-write access).

All other principals should be blocked from access. For example, you might define certificate constraints similar to the following:

```
plugins:kdm:cert_constraints =
   ["C=US,ST=Massachusetts,O=ABigBank*,CN=Secure admin*",
   "C=US,ST=Boston,O=ABigBank*,CN=Orbix2000 Locator Service*"]
```

Your choice of certificate constraints will depend on the naming scheme for your subject names.

iiop_tls:port

Specifies the well known IP port on which the KDM server listens for incoming calls.

checksums_optional

When equal to `false`, the secure information associated with a server must include a checksum; when equal to `true`, the presence of a checksum is optional. Default is `false`.

plugins:kdm_adm

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

- cert_constraints

cert_constraints

Specifies the list of certificate constraints that are applied when the KDM administration plug-in authenticates the KDM server. See "Applying Constraints to Certificates" on page 53 for a description of the certificate constraint syntax.

The KDM administration plug-in requires protection against attack from applications that try to impersonate the KDM server. A security administrator should, therefore, choose certificate constraints that restrict access to trusted KDM servers only. For example, you might define certificate constraints similar to the following:

```
plugins:kdm_adm:cert_constraints =
   ["C=US,ST=Massachusetts,O=ABigBank*,CN=IT_KDM*"];
```

Your choice of certificate constraints will depend on the naming scheme for your subject names.

plugins:login_client

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

- wsdl_url

wsdl_url

Specifies the location of the login service WSDL to the `login_client` plug-in. The value of this variable can either be a relative pathname or an URL. The `login_client` requires access to the login service WSDL in order to obtain details of the physical contract (for example, host and IP port).

plugins:login_service

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

- wsdl_url

wsdl_url

Specifies the location of the login service WSDL to the `login_service` plug-in. The value of this variable can either be a relative pathname or an URL. The `login_service` requires access to the login service WSDL in order to obtain details of the physical contract (for example, host and IP port).

plugins:schannel

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

- prompt_with_credential_choice

prompt_with_credential_choice

(Windows and Schannel only) Setting both this variable and the `plugins:iiop_tls:delay_credential_gathering_until_handshake` variable to `true` on the client side allows the user to choose which credentials to use for the server connection. The choice of credentials offered to the user is based on the trusted CAs sent to the client in an SSL/TLS handshake message.

If `prompt_with_credential_choice` is set to `false`, Artix chooses the first certificate it finds in the certificate store that meets the applicable constraints.

The certificate prompt can be replaced by implementing an IDL interface and registering it with the ORB.

plugins:security

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

- share_credentials_across_orbs

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`.

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:ciphersuites`
- `mechanism_policy:protocol_version`
- `session_caching_policy`
- `session_caching`
- `target_secure_invocation_policy:requires`
- `target_secure_invocation_policy:supports`
- `trusted_ca_list_policy`

allow_unauthenticated_clients_policy

(Deprecated in favor of `policies:iiop_tls:allow_unauthenticated_clients_policy` and `policies:https:allow_unauthenticated_clients_policy`.)

A generic variable that sets this policy both for `iiop_tls` and `https`. The recommended alternative is to use the variables prefixed by `policies:iiop_tls` and `policies:https` instead, which take precedence over this generic variable.

certificate_constraints_policy

(Deprecated in favor of `policies:iiop_tls:certificate_constraints_policy` and `policies:https:certificate_constraints_policy`.)

A generic variable that sets this policy both for `iiop_tls` and `https`. The recommended alternative is to use the variables prefixed by `policies:iiop_tls` and `policies:https` instead, which take precedence over this generic variable.

client_secure_invocation_policy:requires

(Deprecated in favor of `policies:iiop_tls:client_secure_invocation_policy:requires` and `policies:https:client_secure_invocation_policy:requires`.)

A generic variable that sets this policy both for `iiop_tls` and `https`. The recommended alternative is to use the variables prefixed by `policies:iiop_tls` and `policies:https` instead, which take precedence over this generic variable.

client_secure_invocation_policy:supports

(Deprecated in favor of `policies:iiop_tls:client_secure_invocation_policy:supports` and `policies:https:client_secure_invocation_policy:supports`.)

A generic variable that sets this policy both for `iiop_tls` and `https`. The recommended alternative is to use the variables prefixed by `policies:iiop_tls` and `policies:https` instead, which take precedence over this generic variable.

max_chain_length_policy

(Deprecated in favor of `policies:iiop_tls:max_chain_length_policy` and `policies:https:max_chain_length_policy`.)

`max_chain_length_policy` specifies the maximum certificate chain length that an ORB will accept. The policy can also be set programmatically using the `IT_TLS_API::MaxChainLengthPolicy` CORBA policy. Default is `2`.

Note: The `max_chain_length_policy` is not currently supported on the OS/390 platform.

mechanism_policy:ciphersuites

(Deprecated in favor of `policies:iiop_tls:mechanism_policy:ciphersuites` and `policies:https:mechanism_policy:ciphersuites`.)

`mechanism_policy:ciphersuites` specifies a list of cipher suites for the default mechanism policy. One or more of the cipher suites shown in Table 4 can be specified in this list.

Table 4: *Mechanism Policy Cipher Suites*

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

mechanism_policy:protocol_version

(Deprecated in favor of `policies:iiop_tls:mechanism_policy:protocol_version` and `policies:https:mechanism_policy:protocol_version`.)

`mechanism_policy:protocol_version` specifies the protocol version used by a security capsule (ORB instance). It can be set to `SSL_V3` or `TLS_V1`. For example:

```
policies:mechanism_policy:protocol_version="TLS_V1"
```

session_caching_policy

(Java only) `session_caching_policy` specifies whether a Java 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.

session_caching

(C++ only) session_caching specifies whether a C++ 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

(Deprecated in favor of policies:iiop_tls:target_secure_invocation_policy:requires and policies:https:target_secure_invocation_policy:requires.)

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

(Deprecated in favor of policies:iiop_tls:target_secure_invocation_policy:supports and policies:https:target_secure_invocation_policy:supports.)

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

(Deprecated in favor of `policies:iiop_tls:trusted_ca_list_policy` and `policies:https:trusted_ca_list_policy`.)

`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_sso

enable_authorization

A boolean variable that specifies whether Artix should enable authorization using the Artix Security Framework. Default is `false`.

enable_sso

A boolean variable that specifies whether Artix enables single-sign on (SSO) on the server-side. Default is `false`.

policies:csi

The `policies:csi` namespace includes variables that specify settings for Common Secure Interoperability version 2 (CSIv2):

- `attribute_service:backward_trust:enabled`
- `attribute_service:client_supports`
- `attribute_service:target_supports`
- `auth_over_transport:authentication_service`
- `auth_over_transport:client_supports`
- `auth_over_transport:server_domain_name`
- `auth_over_transport:target_requires`
- `auth_over_transport:target_supports`

attribute_service:backward_trust:enabled

(Obsolete)

attribute_service:client_supports

`attribute_service:client_supports` is a client-side policy that specifies the association options supported by the CSIv2 attribute service (principal propagation). The only assocation option that can be specified is `IdentityAssertion`. This policy is normally specified in an intermediate server so that it propagates CSIv2 identity tokens to a target server. For example:

```
policies:csi:attribute_service:client_supports =
   ["IdentityAssertion"];
```

attribute_service:target_supports

`attribute_service:target_supports` is a server-side policy that specifies the association options supported by the CSIv2 attribute service (principal propagation). The only assocation option that can be specified is `IdentityAssertion`. For example:

```
policies:csi:attribute_service:target_supports =
   ["IdentityAssertion"];
```

auth_over_transport:authentication_service

(Java CSI plug-in only) The name of a Java class that implements the `IT_CSI::AuthenticateGSSUPCredentials` IDL interface. The authentication service is implemented as a callback object that plugs into the CSIv2 framework on the server side. By replacing this class with a custom implementation, you could potentially implement a new security technology domain for CSIv2.

By default, if no value for this variable is specified, the Java CSI plug-in uses a default authentication object that always returns `false` when the `authenticate()` operation is called.

auth_over_transport:client_supports

`auth_over_transport:client_supports` is a client-side policy that specifies the association options supported by CSIv2 authorization over transport. The only assocation option that can be specified is `EstablishTrustInClient`. For example:

```
policies:csi:auth_over_transport:client_supports =
   ["EstablishTrustInClient"];
```

auth_over_transport:server_domain_name

The iSF security domain (CSIv2 authentication domain) to which this server application belongs. The iSF security domains are administered within an overall security technology domain.

The value of the `server_domain_name` variable will be embedded in the IORs generated by the server. A CSIv2 client about to open a connection to this server would check that the domain name in its own CSIv2 credentials matches the domain name embedded in the IOR.

auth_over_transport:target_requires

`auth_over_transport:target_requires` is a server-side policy that specifies the association options required for CSIv2 authorization over transport. The only assocation option that can be specified is `EstablishTrustInClient`. For example:

```
policies:csi:auth_over_transport:target_requires =
   ["EstablishTrustInClient"];
```

auth_over_transport:target_supports

`auth_over_transport:target_supports` is a server-side policy that specifies the association options supported by CSIv2 authorization over transport. The only assocation option that can be specified is `EstablishTrustInClient`. For example:

```
policies:csi:auth_over_transport:target_supports =
   ["EstablishTrustInClient"];
```

policies:iiop_tls

The `policies:iiop_tls` namespace contains variables used to set IIOP-related policies for a secure environment. These setting affect the `iiop_tls` plugin. It contains the following variables:

- `allow_unauthenticated_clients_policy`
- `buffer_sizes_policy:default_buffer_size`
- `buffer_sizes_policy:max_buffer_size`
- `certificate_constraints_policy`
- `client_secure_invocation_policy:requires`
- `client_secure_invocation_policy:supports`
- `client_version_policy`
- `connection_attempts`
- `connection_retry_delay`
- `max_chain_length_policy`
- `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.

max_chain_length_policy

This policy overides `policies:max_chain_length_policy` for the `iiop_tls` plugin.

The maximum certificate chain length that an ORB will accept.

The policy can also be set programmatically using the `IT_TLS_API::MaxChainLengthPolicy` CORBA policy. Default is 2.

Note: The `max_chain_length_policy` is not currently supported on the OS/390 platform.

mechanism_policy:ciphersuites

This policy overides `policies:mechanism_policy:ciphersuites` for the `iiop_tls` plugin.

Specifies a list of cipher suites for the default mechanism policy. One or more of the following cipher suites can be specified in this list:

Table 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

mechanism_policy:protocol_version

This policy overides `policies:mechanism_policy:protocol_version` for the `iiop_tls` plugin.

Specifies the protocol version used by a security capsule (ORB instance). Can be set to one of the following values:

```
TLS_V1
SSL_V3
SSL_V2V3
```

The `SSL_V2V3` value is a special setting that facilitates interoperability with an Artix application deployed on the OS/390 platform. Artix security on the OS/390 platform is based on IBM's System/SSL toolkit, which implements SSL version 3, but does so by using SSL version 2 hellos as part of the handshake. This form of handshake causes interoperability problems, because applications on other platforms identify the handshake as an SSL version 2 handshake. The misidentification of the SSL protocol version can be avoided by setting the protocol version to be `SSL_V2V3` in the non-OS/390 application (this bug also affects some old versions of Microsoft Internet Explorer).

For example:

```
policies:mechanism_policy:protocol_version = "SSL_V2V3";
```

server_address_mode_policy:local_domain

(Java only) When this policy is set, the `iiop_tls` plug-in reads this policy's value instead of the `policies:iiop:server_address_mode_policy:local_domain` policy's value.

server_address_mode_policy:local_hostname

(Java only) When this policy is set, the `iiop_tls` plug-in reads this policy's value instead of the `policies:iiop:server_address_mode_policy:local_hostname` policy's value.

`server_address_mode_policy:local_hostname` specifies the hostname advertised by the locator daemon, and listened on by server-side IIOP.

Some machines have multiple hostnames or IP addresses (for example, those using multiple DNS aliases or multiple network cards). These machines are often termed *multi-homed hosts*. The `local_hostname` variable supports these type of machines by enabling you to explicitly specify the host that servers listen on and publish in their IORs.

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

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

By default, the `local_hostname` variable is unspecified. Servers use the default hostname configured for the machine with the Orbix configuration tool.

server_address_mode_policy:port_range

(Java only) When this policy is set, the `iiop_tls` plug-in reads this policy's value instead of the `policies:iiop:server_address_mode_policy:port_range` policy's value.

`server_address_mode_policy:port_range` specifies the range of ports that a server uses when there is no well-known addressing policy specified for the port.

server_address_mode_policy:publish_hostname

When this policy is set, the `iiop_tls` plug-in reads this policy's value instead of the `policies:iiop:server_address_mode_policy:publish_hostname` policy's value.

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

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

The following `itadmin` command is equivalent:

```
itadmin variable create -type bool -value true
   policies:iiop:server_address_mode_policy:publish_hostname
```

server_version_policy

When this policy is set, the `iiop_tls` plug-in reads this policy's value instead of the `policies:iiop:server_version_policy` policy's value.

`server_version_policy` specifies the GIOP version published in IIOP profiles. This variable takes a value of either `1.1` or `1.2`. Orbix servers do not publish IIOP 1.0 profiles. The default value is `1.2`.

session_caching_policy

This policy overides `policies:session_caching_policy`(Java) and `policies:session_caching`(C++) for the `iiop_tls` plugin.

target_secure_invocation_policy:requires

This policy overides `policies:target_secure_invocation_policy:requires` for the `iiop_tls` plugin.

Specifies the minimum level of security required by a server. The value of this variable is specified as a list of association options—see the *Artix Security Guide* for more details about association options.

In accordance with CORBA security, this policy cannot be downgraded programmatically by the application.

target_secure_invocation_policy:supports

This policy overides `policies:target_secure_invocation_policy:supports` for the `iiop_tls` plugin.

Specifies the maximum level of security supported by a server. The value of this variable is specified as a list of association options—see the *Artix Security Guide* for more details about association options.

This policy can be upgraded programmatically using either the `QOP` or the `EstablishTrust` policies.

tcp_options_policy:no_delay

When this policy is set, the `iiop_tls` plug-in reads this policy's value instead of the `policies:iiop:tcp_options_policy:no_delay` policy's value.

`tcp_options_policy:no_delay` specifies whether the `TCP_NODELAY` option should be set on connections. Defaults to `false`.

tcp_options_policy:recv_buffer_size

When this policy is set, the `iiop_tls` plug-in reads this policy's value instead of the `policies:iiop:tcp_options_policy:recv_buffer_size` policy's value.

`tcp_options_policy:recv_buffer_size` specifies the size of the TCP receive buffer. This variable can only be set to `0`, which coresponds to using the default size defined by the operating system.

tcp_options_policy:send_buffer_size

When this policy is set, the `iiop_tls` plug-in reads this policy's value instead of the `policies:iiop:tcp_options_policy:send_buffer_size` policy's value.

`tcp_options_policy:send_buffer_size` specifies the size of the TCP send buffer. This variable can only be set to `0`, which coresponds to using the default size defined by the operating system.

trusted_ca_list_policy

This policy overides the `policies:trusted_ca_list_policy` for the `iiop_tls` plugin.

Contains a list of filenames (or a single filename), each of which contains a concatenated list of CA certificates in PEM format. The aggregate of the CAs in all of the listed files is the set of trusted CAs.

For example, you might specify two files containing CA lists as follows:

```
policies:trusted_ca_list_policy =
   ["ASPInstallDir/asp/6.0/etc/tls/x509/ca/ca_list1.pem",
   "ASPInstallDir/asp/6.0/etc/tls/x509/ca/ca_list_extra.pem"];
```

The purpose of having more than one file containing a CA list is for administrative convenience. It enables you to group CAs into different lists and to select a particular set of CAs for a security domain by choosing the appropriate CA lists.

principal_sponsor

The `principal_sponsor` namespace stores configuration information to be used when obtaining credentials. the CORBA binding provides an implementation of a principal sponsor that creates credentials for applications automatically.

Use of the `PrincipalSponsor` is disabled by default and can only be enabled through configuration.

The `PrincipalSponsor` represents an entry point into the secure system. It must be activated and authenticate the user, before any application-specific logic executes. This allows unmodified, security-unaware applications to have `Credentials` established transparently, prior to making invocations.

In this section

The following variables are in this namespace:

- `use_principal_sponsor`
- `auth_method_id`
- `auth_method_data`
- `callback_handler:ClassName`
- `login_attempts`

use_principal_sponsor

`use_principal_sponsor` specifies whether an attempt is made to obtain credentials automatically. Defaults to `false`. If set to `true`, the following `principal_sponsor` variables must contain data in order for anything to actually happen.

auth_method_id

`auth_method_id` specifies the authentication method to be used. The following authentication methods are available:

`pkcs12_file`	The authentication method uses a PKCS#12 file.
`pkcs11`	Java only. The authentication data is provided by a smart card.
`security_label`	Windows and Schannel only. The authentication data is specified by supplying the common name (CN) from an application certificate's subject DN.

For example, you can select the `pkcs12_file` authentication method as follows:

```
principal_sponsor:auth_method_id = "pkcs12_file";
```

auth_method_data

`auth_method_data` is a string array containing information to be interpreted by the authentication method represented by the `auth_method_id`.

For the `pkcs12_file` authentication method, the following authentication data can be provided in `auth_method_data`:

`filename`	A PKCS#12 file that contains a certificate chain and private key—*required*.
`password`	A password for the private key—*optional*. It is bad practice to supply the password from configuration for deployed systems. If the password is not supplied, the user is prompted for it.
`password_file`	The name of a file containing the password for the private key—*optional*. This option is not recommended for deployed systems.

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

`provider`	A name that identifies the underlying PKCS #11 toolkit used by Orbix to communicate with the smart card. The toolkit currently used by Orbix has the provider name `dkck132.dll` (from Baltimore).
`slot`	The number of a particular slot on the smart card (for example, `0`) containing the user's credentials.
`pin`	A PIN to gain access to the smart card—*optional*. It is bad practice to supply the PIN from configuration for deployed systems. If the PIN is not supplied, the user is prompted for it.

For the `security_label` authentication method on Windows, the following authentication data can be provided in `auth_method_data`:

`label`	(Windows and Schannel only.) The common name (CN) from an application certificate's subject DN

For example, to configure an application on Windows to use a certificate, `bob.p12`, whose private key is encrypted with the `bobpass` password, set the `auth_method_data` as follows:

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

The following points apply to Java implementations:

- If the file specified by `filename=` is not found, it is searched for on the classpath.
- The file specified by `filename=` can be supplied with a URL instead of an absolute file location.
- The mechanism for prompting for the password if the password is supplied through `password=` can be replaced with a custom mechanism, as demonstrated by the `login` demo.

- There are two extra configuration variables available as part of the `principal_sponsor` namespace, namely `principal_sponsor:callback_handler` and `principal_sponsor:login_attempts`. These are described below.
- These Java-specific features are available subject to change in future releases; any changes that can arise probably come from customer feedback on this area.

callback_handler:ClassName

`callback_handler:ClassName` specifies the class name of an interface that implements the interface `com.iona.corba.tls.auth.CallbackHandler`. This variable is only used for Java clients.

login_attempts

`login_attempts` specifies how many times a user is prompted for authentication data (usually a password). It applies for both internal and custom `CallbackHandlers`; if a `CallbackHandler` is supplied, it is invoked upon up to `login_attempts` times as long as the `PrincipalAuthenticator` returns `SecAuthFailure`. This variable is only used by Java clients.

principal_sponsor:csi

The principal_sponsor:csi namespace stores configuration information to be used when obtaining CSI (Common Secure Interoperability) credentials. It includes the following:

- use_existing_credentials
- use_principal_sponsor
- auth_method_data
- auth_method_id

use_existing_credentials

A boolean value that specifies whether ORBs that share credentials can also share CSI credentials. If true, any CSI credentials loaded by one credential-sharing ORB can be used by other credential-sharing ORBs loaded after it; if false, CSI credentials are not shared.

This variable has no effect, unless the plugins:security:share_credentials_across_orbs variable is also true.

Default is false.

use_principal_sponsor

use_principal_sponsor is a boolean value that switches the CSI principal sponsor on or off.

If set to true, the CSI principal sponsor is enabled; if false, the CSI principal sponsor is disabled and the remaining principal_sponsor:csi variables are ignored. Defaults to false.

auth_method_data

auth_method_data is a string array containing information to be interpreted by the authentication method represented by the auth_method_id.

For the GSSUPMech authentication method, the following authentication data can be provided in auth_method_data:

username	The username for CSIv2 authorization. This is optional. Authentication of CSIv2 usernames and passwords is performed on the server side. The administration of usernames depends on the particular security mechanism that is plugged into the server side see auth_over_transport:authentication_service.
password	The password associated with username. This is optional. It is bad practice to supply the password from configuration for deployed systems. If the password is not supplied, the user is prompted for it.
domain	The CSIv2 authentication domain in which the username/password pair is authenticated. When the client is about to open a new connection, this domain name is compared with the domain name embedded in the relevant IOR (see policies:csi:auth_over_transport:server_domain_name). The domain names must match. **Note:** If domain is an empty string, it matches any target domain. That is, an empty domain string is equivalent to a wildcard.

If any of the preceding data are omitted, the user is prompted to enter authentication data when the application starts up.

For example, to log on to a CSIv2 application as the administrator user in the US-SantaClara domain:

```
principal_sponsor:csi:auth_method_data =
   ["username=administrator", "domain=US-SantaClara"];
```

When the application is started, the user is prompted for the administrator password.

Note: It is currently not possible to customize the login prompt associated with the CSIv2 principal sponsor. As an alternative, you could implement your own login GUI by programming and pass the user input directly to the principal authenticator.

auth_method_id

`auth_method_id` specifies a string that selects the authentication method to be used by the CSI application. The following authentication method is available:

`GSSUPMech`	The Generic Security Service Username/Password (GSSUP) mechanism.

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

```
principal_sponsor:csi:auth_method_id = "GSSUPMech";
```

CHAPTER 5

CORBA Plug-ins

Artix is built on IONA's Adaptive Runtime architecture (ART), which enables users to configure services as plugins to the core product.

Note: The variables described in this chapter only apply when Artix is using the CORBA transport.

Overview

A plugin is a class or code library that can be loaded into an Artix application at link-time or runtime. The `plugins` namespace contains child namespaces for plugins, such as `naming` and `iiop`. Each child namespace has information specific to each plugin. Child namespaces usually have a C++ `shlib_name` variable, indicating the class or library in which the plugin resides. The following examples show how the configuration specifies the library for the `iiop` plugin:

```
plugins:iiop:shlib_name = "it_iiop";
```

Plugins also have their own specific configuration variables. For example, the following variable sets the default timeout of a transaction in seconds:

```
plugins:ots:default_transaction_timeout
```

In this chapter

The following plugins are discussed in this chapter:

plugins:atli2_shm

The variables in this namespace control the behavior of the shared memory ATLI2 plugin. This namespace includes the following:

- max_buffer_wait_time
- shared_memory_segment_basename
- shared_memory_size
- shared_memory_segment

max_buffer_wait_time

max_buffer_wait_time specifies the maximum wait time on a shared memory buffer before raising a no resources exception. The default is 5 seconds.

shared_memory_segment_basename

shared_memory_segment_basename defines the prefix used when the shared memory transport creates internal files (for example, in /var/tmp/SAMD and /tmp on Solaris). The default is iona.

shared_memory_size

`shared_memory_size` specifies the size of the shared memory segment created (for example, in the call to `mmap` on Solaris). The default value is `8*1024*1024`.

This size should be larger than the largest data payload passed between a client and server. If the setting is too small, the shared memory transport will run out of memory, and will be unable to marshal the data. If there is danger of this occurring, add `GIOP+IIOP` to your `client_binding_list` setting. This enables the ORB to use the normal network transport if a large payload can not make it through shared memory.

shared_memory_segment

`shared_memory_segment` specifies the name of the already existing shared memory segment to use in place of creating a new segment. There is no default name. Artix creates a new segment by default.

plugins:codeset

The variables in this namespace specify the codesets used by the CORBA portion of Artix. This is useful when internationalizing your environment.

The following variables are contained in this namespace:

- `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 6: *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 7: *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 8: *Defaults for the wide native codesets*

Platform/Locale	Language	Setting
non-MVS, Latin-1 locale	C++	UCS-2, UCS-4
MVS	C++	UCS-2, UCS-4
ISO-8859-1/Cp-1292/US-ASCII locale	Java	UTF-16
Shift_JIS locale	Java	UTF-16

Table 8: *Defaults for the wide native codesets*

Platform/Locale	Language	Setting
EUC-JP locale	Java	UTF-16
other	Java	UTF-16

wchar:ccs

`wchar:ccs` specifies the list of conversion codesets supported for wide characters. The default setting is determined as follows:

Table 9: *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:egmiop

The variables in this namespace configure endpoint functionality for the MIOP transport. This namespace contains the following variables:

- `ip:send_buffer_size`
- `ip:receive_buffer_size`
- `pool:max_threads`
- `pool:max_threads`
- `pool:min_threads`
- `udp:packet_size`

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 buffer size is static.

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.

udp:packet_size

`udp:packet_size` specifies the maximum size for outgoing UDP packets. A larger UDP packet size increases the probability of IP packet fragmentation on the wire hence increasing the possibility of data loss. A smaller UDP packet size increases the overhead per packet and decreases throughput. Defaults to 120kb.

plugins:giop

This namespace contains the `plugins:giop:message_server_binding_list` configuration variable, which is one of the variables used to configure bidirectional GIOP. This feature allows callbacks to be made using a connection opened by the client, instead of requiring the server to open a new connection for the callback.

message_server_binding_list

`plugins:giop:message_server_binding_list` specifies a list message inceptors that are used for bidirectional GIOP. On the client-side, the `plugins:giop:message_server_binding_list` must be configured to indicate that an existing outgoing message interceptor chain may be re-used for an incoming server binding, similarly by including an entry for `BiDir_GIOP`, for example:

```
plugins:giop:message_server_binding_list=["BiDir_GIOP","GIOP" ];
```

Further information

For information on other variables used to set bidirectional GIOP, see "policies:giop" on page 148. 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`
- `shlib_name`
- `verbosity`

filename

`plugins:giop_snoop:filename` specifies a file for GIOP Snoop output. By default, output is directed to standard error (`stderr`). This variable has the following format:

```
plugins:giop_snoop:filename = "<some-file-path>";
```

A *`month/day/year`* time stamp is included in the output filename with the following general format:

```
<filename>.MMDDYYYY
```

rolling_file

`plugins:giop_snoop:rolling_file` prevents the GIOP Snoop output file from growing indefinitely. This setting specifies to open and then close the output file for each snoop message trace, instead of holding the output files open. This enables administrators to control the size and content of output files. This setting is enabled with:

```
plugins:giop_snoop:rolling_file = "true";
```

shlib_name

(C++ only) `plugins:giop_snoop:shlib_name` locates and loads the `giop_snoop` plug-in. This is configured by default as follows:

```
plugins:giop_snoop:shlib_name = "it_giop_snoop";
```

Note: In addition, for both client or server configuration, the `giop_snoop` plug-in must be included in your `orb_plugins` list.

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: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: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 is `true`. Setting this to `false` switches `SO_REUSEADDR` to `false`. This does not allow a process to listen on the same port. An exception indicating that the address is already in use will be thrown.

outgoing_connections:hard_limit

`plugins:iiop:outgoing_connections:hard_limit` sets the maximum number of outgoing (client-side) connections permitted to IIOP. IIOP does not allow new outgoing connections above this limit. Defaults to -1 (disabled).

outgoing_connections:soft_limit

`plugins:iiop:outgoing_connections:soft_limit` specifies the number of connections at which IIOP begins closing outgoing (client-side) connections. Defaults to -1 (disabled).

pool:max_threads

`plugins:iiop:pool:max_threads` specifies the maximum number of threads reserved from the `WorkQueue` to support tasks working on behalf of the ATLI transport. Defaults to 5.

pool:min_threads

`plugins:iiop:pool:min_threads` specifies the minimum number of threads reserved from the `WorkQueue` to support tasks working on behalf of the ATLI transport. Defualts to 1.

tcp_connection:keep_alive

`plugins:iiop:tcp_connection:keep_alive` specifies the setting of `SO_KEEPALIVE` on sockets used to maintain IIOP connections. If set to `TRUE`, the socket will send a *'keepalive probe'* to the remote host if the conneciton has been idle for a preset period of time. The remote system, if it is still running, will send an `ACK` response. Defaults to `TRUE`.

tcp_connection:no_delay

`plugins:iiop:tcp_connection:no_deplay` specifies if `TCP_NODELAY` is set on the sockets used to maintain IIOP connections. If set to false, small data packets are collected and sent as a group. The algorithm used allows for no more than a 0.2 msec delay between collected packets. Defaults to `TRUE`.

tcp_connection:linger_on_close

`plugins:iiop:tcp_connection:linger_on_close` specifies the setting of `SO_LINGER` on all tcp connections to ensure that tcp buffers get cleared once a socket is closed. Defaults to `TRUE`.

tcp_listener:reincarnate_attempts

(C++/Windows only)

`plugins:iiop:tcp_listener:reincarnate_attempts` specifies the number of attempts that are made to reincarnate a listener before giving up, logging a fatal error, and shutting down the ORB. Datatype is `long`. Defaults to `0` (no attempts).

Sometimes an network error may occur, which results in a listening socket being closed. On Windows, you can configure the listener to attempt a reincarnation. This enables new connections to be established.

tcp_listener:reincarnation_retry_backoff_ratio

(C++/Windows only)
`plugins:iiop:tcp_listener:reincarnation_retry_delay` specifies a delay between reincarnation attempts. Data type is `long`. Defaults to `0` (no delay).

tcp_listener:reincarnation_retry_delay

(C++/Windows only)
`plugins:iiop:tcp_listener:reincarnation_retry_backoff_ratio`specifies the degree to which delays between retries increase from one retry to the next. Datatype is `long`. Defaults to `1`.

plugins:local_log_stream

The variables in this namespace configure how Artix logs runtime information. By default, Artix is configured to log messages to standard error. You can change this behavior for an ORB by specifying a logstream plugin.

This namespace contains the following variables:

- `filename`
- `rolling_file`

For full information about Artix logging, see *Managing and Deploying Artix Solutions*.

filename

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

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

rolling_file

`rolling_file` is a boolean which specifies that the logging plugin is to use a rolling file to prevent the local log 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/art.log.02171999
```

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";
```

plugins:naming

The variables in this namespace configure the naming service plugin. The naming service allows you to associate abstract names with CORBA objects, enabling clients to locate your objects.

This namespace contains the following variables:

- destructive_methods_allowed
- direct_persistence
- iiop:port
- lb_default_initial_load
- lb_default_load_timeout
- nt_service_dependencies

destructive_methods_allowed

destructive_methods_allowed specifies if users can make destructive calls, such as destroy(), on naming service elements. The default value is true, meaning the destructive methods are allowed.

direct_persistence

direct_persistence specifies if the service runs using direct or indirect persistence. The default value is false, meaning indirect persistence.

iiop:port

iiop:port specifies the port that the service listens on when running using direct persistence.

lb_default_initial_load

`lb_default_initial_load` specifies the default initial load value for a member of an active object group. The load value is valid for a period of time specified by the timeout assigned to that member. Defaults to `0.0`. For more information, see the *Orbix Administrator's Guide*.

lb_default_load_timeout

`lb_default_load_timeout` specifies the default load timeout value for a member of an active object group. The default value of `-1` indicates no timeout. This means that the load value does not expire. For more information, see the *Orbix Administrator's Guide*.

nt_service_dependencies

`nt_service_dependencies` specifies the naming service's dependencies on other NT services. The dependencies are listed in the following format:

```
IT ORB-name domain-name
```

This variable only has meaning if the naming service is installed as an NT service.

plugins:ots

The variables in this namespace configure the object transaction service (OTS) generic plugin. The generic OTS plugin contains client and server side transaction interceptors and the implementation of `CosTransactions::Current`. For details of this plugin, refer to the *CORBA OTS Guide*.

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

- `default_ots_policy`
- `default_transaction_policy`
- `default_transaction_timeout`
- `interposition_style`
- `jit_transactions`
- `ots_v11_policy`
- `propagate_separate_tid_optimization`
- `rollback_only_on_system_ex`
- `support_ots_v11`
- `transaction_factory_name`

default_ots_policy

`default_ots_policy` specifies the default `OTSPolicy` value used when creating a POA. Set to one of the following values:

```
requires
forbids
adapts
```

If no value is specified, no `OTSPolicy` is set for new POAs.

default_transaction_policy

`default_transaction_policy` specifies the default `TransactionPolicy` value used when creating a POA.

Set to one of the following values:

- `requires` corresponds to a `TransactionPolicy` value of `Requires_shared`.
- `allows` corresponds to a `TransactionPolicy` value of `Allows_shared`.

If no value is specified, no `TransactionPolicy` is set for new POAs.

default_transaction_timeout

`default_transaction_timeout` specifies the default timeout, in seconds, of a transaction created using `CosTransactions::Current`. A value of zero or less specifies no timeout. Defaults to `30` seconds.

interposition_style

`interposition_style` specifies the style of interposition used when a transaction first visits a server. Set to one of the following values:

- `standard`: A new subordinator transaction is created locally and a resource is registered with the superior coordinator. This subordinate transaction is then made available through the `Current` object.
- `proxy`: (default) A locally constrained proxy for the imported transaction is created and made available though the `Current` object.

Proxy interposition is more efficient, but if you need to further propagate the transaction explicitly (using the `Control` object), standard interposition must be specified.

jit_transactions

`jit_transactions` is a boolean which determines whether to use just-in-time transaction creation. If set to `true`, transactions created using `Current::begin()` are not actually created until necessary. This can be used in conjunction with an `OTSPolicy` value of `SERVER_SIDE` to delay creation of a transaction until an invocation is received in a server. Defaults to `false`.

ots_v11_policy

`ots_v11_policy` specifies the effective `OTSPolicy` value applied to objects determined to support `CosTransactions::TransactionalObject`, if `support_ots_v11` is set to `true`.

Set to one of the following values:

- `adapts`
- `requires`

propagate_separate_tid_optimization

`propagate_separate_tid_optimization` specifies whether an optimization is applied to transaction propagation when using C++ applications. Must be set for both the sender and receiver to take affect. Defaults to `true`.

rollback_only_on_system_ex

`rollback_only_on_system_ex` specifies whether to mark a transaction for rollback if an invocation on a transactional object results in a system exception being raised. Defaults to `true`.

support_ots_v11

`support_ots_v11` specifies whether there is support for the OMG OTS `v1.1` `CosTransactions::TransactionalObject` interface. This option can be used in conjunction with `ots_v11_policy`. When this option is enabled, the OTS interceptors might need to use remote `_is_a()` calls to determine the type of an interface. Defaults to `false`.

transaction_factory_name

`transaction_factory_name` specifies the initial reference for the transaction factory. This option must match the corresponding entry in the configuration scope of your transaction service implementation. Defaults to `TransactionFactory`.

plugins:ots_lite

The variables in this namespace configure the Lite implementation of the object transaction service. The `ots_lite` plugin contains an implementation of `CosTransacitons::TransactionFactory` which is optimized for use in a single resource system. For details, see the *CORBA Programmer's Guide*.

This namespace contains the following variables:

- `orb_name`
- `otid_format_id`
- `superior_ping_timeout`
- `transaction_factory_name`
- `transaction_timeout_period`
- `use_internal_orb`

orb_name

`orb_name` specifies the ORB name used for the plugin's internal ORB when `use_internal_orb` is set to `true`. The ORB name determines where the ORB obtains its configuration information and is useful when the application ORB configuration needs to be different from that of the internal ORB. Defaults to the ORB name of the application ORB.

otid_format_id

`otid_format_id` specifies the value of the `formatID` field of a transaction's identifier (`CosTransactions::otid_t`). Defaults to `0x494f4e41`.

superior_ping_timeout

`superior_ping_timeout` specifies, in seconds, the timeout between queries of the transaction state, when standard interposition is being used to recreate a foreign transaction. The interposed resource periodically queries the recovery coordinator, to ensure that the transaction is still alive when the timeout of the superior transaction has expired. Defaults to `30`.

transaction_factory_name

`transaction_factory_name` specifies the initial reference for the transaction factory. This option must match the corresponding entry in the configuration scope of your generic OTS plugin to allow it to successfully resolve a transaction factory. Defaults to `TransactionFactory`.

transaction_timeout_period

`transaction_timeout_period` specifies the time, in milliseconds, of which all transaction timeouts are multiples. A low value increases accuracy of transaction timeouts, but increases overhead. This value is added to all transaction timeouts. To disable all timeouts, set to `0` or a negative value. Defaults to `1000`.

use_internal_orb

`use_internal_orb` specifies whether the ots_lite plugin creates an internal ORB for its own use. By default, `ots_lite` creates POAs in the application's ORB. This option is useful if you want to isolate the transaction service from your application ORB. Defaults to `false`.

plugins:ots_encina

The `plugins:ots_encina` namespace stores configuration variables for the Encina OTS plugin. The `ots_encina` plugin contains an implementation of IDL interface `CosTransactions::TransactionFactory` that supports the recoverable 2PC protocol. For details, see the *CORBA OTS Guide*.

This namespace contains the following variables:

- `agent_ior_file`
- `allow_registration_after_rollback_only`
- `backup_restart_file`
- `direct_persistence`
- `direct_persistence`
- `global_namespace_poa`
- `iiop:port`
- `initial_disk`
- `initial_disk_size`
- `log_threshold`
- `log_check_interval`
- `max_resource_failures`
- `namespace_poa`
- `orb_name`
- `otid_format_id`
- `resource_retry_timeout`
- `restart_file`
- `trace_comp`
- `trace_file`
- `trace_on`
- `transaction_factory_name`
- `transaction_factory_ns_name`
- `transaction_timeout_period`
- `use_internal_orb`
- `use_raw_disk`

agent_ior_file

`agent_ior_file` specifies the file path where the management agent object's IOR is written. Defaults to an empty string.

allow_registration_after_rollback_only

`allow_registration_after_rollback_only` (C++ only) specifies whether registration of resource objects is permitted after a transaction is marked for rollback.

- `true` specifies that resource objects can be registered after a transaction is marked for rollback.
- `false` (default) specifies that resource objects cannot be registered once a transaction is marked for rollback.

This has no effect on the outcome of the transaction.

backup_restart_file

`backup_restart_file` specifies the path for the backup restart file used by the Encina OTS to locate its transaction logs. If unspecified, the backup restart file is the name of the primary restart file—set with `restart_file`—with a `.bak` suffix. Defaults to an empty string.

direct_persistence

`direct_persistence` specifies whether the transaction factory object can use explicit addressing—for example, a fixed port. If set to `true`, the addressing information is picked up from `plugins:ots_encina`. For example, to use a fixed port, set `plugins_ots_encina:iiop:port`. Defaults to `false`.

global_namespace_poa

`global_namespace_poa` specifies the top-level transient POA used as a namespace for OTS implementations. Defaults to `iOTS`.

iiop:port

`iiop:port` specifies the port that the service listens on when using direct persistence.

initial_disk

`initial_disk` specifies the path for the initial file used by the Encina OTS for its transaction logs. Defaults to an empty string.

initial_disk_size

`initial_disk_size` specifies the size of the initial file used by the Encina OTS for its transaction logs. Defaults to `2`.

log_threshold

`log_threshold` specifies the percentage of transaction log space, which, when exceeded, results in a management event. Must be between `0` and `100`. Defaults to `90`.

log_check_interval

`log_check_interval` specifies the time, in seconds, between checks for transaction log growth. Defaults to `60`.

max_resource_failures

`max_resource_failures` specifies the maximum number of failed invocations on `CosTransaction::Resource` objects to record. Defaults to `5`.

namespace_poa

`namespace_poa` specifies the transient POA used as a namespace. This is useful when there are multiple instances of the plugin being used; each instance must use a different namespace POA to distinguish itself. Defaults to `Encina`.

orb_name

`orb_name` specifies the ORB name used for the plugin's internal ORB when `use_internal_orb` is set to `true`. The ORB name determines where the ORB obtains its configuration information, and is useful when the application ORB configuration needs to be different from that of the internal ORB. Defaults to the ORB name of the application ORB.

otid_format_id

`otis_format_id` specifies the value of the `formatID` field of a transaction's identifier (`CosTransactions::otid_t`). Defaults to `0x494f4e41`.

resource_retry_timeout

`resource_retry_timeout` specifies the time, in seconds, between retrying a failed invocation on a resource object. A negative value means the default is used. Defaults to `5`.

restart_file

`restart_file` specifies the path for the restart file used by the Encina OTS to locate its transaction logs. Defaults to an empty string.

trace_comp

`trace_`*`comp`* sets the Encina trace levels for the component *`comp`*, where *`comp`* is one of the following:

```
bde
log
restart
tran
tranLog_log
tranLog_tran
util
vol
```

Set this variable to a bracket-enclosed list that includes one or more of the following string values:

- `event`: interesting events.
- `entry`: entry to a function.
- `param`: parameters to a function.
- `internal_entry`: entry to internal functions.
- `internal_param`: parameters to internal functions.
- `global`.

Defaults to `[]`.

trace_file

`trace_file` specifies the file to which Encina level tracing is written when enabled via `trace_on`. If not set or set to an empty string, Encina level transactions are written to standard error. Defaults to an empty string.

trace_on

`trace_on` specifies whether Encina level tracing is enabled. If set to `true`, the information that is output is determined from the trace levels (see `trace_comp`). Defaults to `false`.

transaction_factory_name

`transaction_factory_name` specifies the initial reference for the transaction factory. This option must match the corresponding entry in the configuration scope of your generic OTS plugin to allow it to successfully resolve a transaction factory. Defaults to `TransactionFactory`.

transaction_factory_ns_name

`transaction_factory_ns_name` specifies the name used to publish the transaction factory reference in the naming service. Defaults to an empty string.

transaction_timeout_period

`transaction_timeout_period` specifies the time, in milliseconds, of which all transaction timeouts are multiples. A low value increases accuracy of transaction timeouts, but increases overhead. This value multiplied to all transaction timeouts. To disable all timeouts, set to `0` or a negative value. Defaults to `1000`.

use_internal_orb

`use_internal_orb` specifies whether the `ots_encina` plugin creates an internal ORB for its own use. By default the `ots_encina` plugin creates POA's in the application's ORB. This option is useful if you want to isolate the transaction service from your application ORB. Defaults to `false`.

use_raw_disk

use_raw_disk specifies whether the path specified by `initial_disk` is of a raw disk (`true`) or a file (`false`). If set to `false` and the file does not exist, the Encina OTS plugin tries to create the file with the size specified in `initial_disk_size`. Defaults to `false`.

plugins:poa

This namespace contains variables to configure the CORBA POA plugin. 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.

plugins:pss

For C++ applications, the `plugins:pss` namespace stores configuration variables for the persistent state service (PSS) plugin. PSS is a CORBA service for building CORBA servers that access persistent data.

The following variables are contained in this namespace:

- `disable_caching`

For more details of this service, refer to the *CORBA Programmer's Guide*.

disable_caching

`disable_caching` specifies whether caching is disabled. When set to `true`, PSS does not perform any caching. This is useful for testing, and causes core dumps in code that does not manage PSS objects correctly. Defaults to `false`.

plugins:pss_db:envs:*env-name*

For C++ applications, the `plugins:pss_db:envs:`*`env-name`* namespace contains variables for the persistent state service (PSS) database plugin, where *`env-name`* represents the environment name. For example, `it_locator` represents persistent storage for the locator daemon. For details on this service, refer to the *CORBA Programmer's Guide*.

The following variables are contained in this namespace:

- `checkpoint_period`
- `checkpoint_archives_old_logs`
- `checkpoint_deletes_old_logs`
- `checkpoint_min_size`
- `concurrent_users`
- `create_dirs`
- `data_dir`
- `db_home`
- `deadlock_detector_aborts`
- `init_txn`
- `lg_bsize`
- `lg_max`
- `lk_max`
- `log_dir`
- `old_log_dir`
- `private`
- `pull_period`
- `push_all_updates`
- `push_update_period`
- `replication_model`
- `recover_fatal`
- `run_deadlock_detector`
- `tmp_dir`
- `tx_max`

- verb_all
- verb_chkpoint
- verb_deadlock
- verb_recovery
- verb_waitsfor

---

checkpoint_period

checkpoint_period is used in TX mode only, and specifies, in minutes, the transaction-log checkpoint period. Defaults to 15.

---

checkpoint_archives_old_logs

checkpoint_archives_old_logs specifies whether PSS archives old log files in the old_logs directory. To archive old log files, set this variable to true. Defaults to false.

---

checkpoint_deletes_old_logs

checkpoint_deletes_old_logs is used in TX mode only, and specifies whether PSS deletes old log files after each checkpoint. When false, PSS moves old log files to the old_logs directory. Defaults to true.

---

checkpoint_min_size

checkpoint_min_size is used in TX mode only, and specifies the minimum checkpoint size. If less than the checkpoint_min_size of data is written to the log since the last checkpoint, do not checkpoint. Defaults to 0.

---

concurrent_users

concurrent_users specifies the number of threads expected to use this environment at the same time. Defaults to 20.

create_dirs

`create_dirs` specifies whether the `db_home`, `log` and `tmp` directories are to be created, if they do not exist. Defaults to `false`.

data_dir

`data_dirs` specifies the directory where the data files are stored; relative paths are relative to `db_home`. The directory must be on a local file system. Defaults to `data`.

db_home

`db_home` specifies the home directory of the Berkeley DB database. For example, in Orbix, `plugins:pss_db:envs:it_locator:db_home` specifies the home directory for the locator daemon.

deadlock_detector_aborts

`deadlock_detector_aborts` specifies when the deadlock detector aborts, when the value of `run_deadlock_detector` is set to `true`. Set this variable to:

- `default`
- `youngest`
- `oldest`
- `random`

init_txn

`init_txn` specifies whether to use transactions to access this database. Defaults to `false`.

log_dir

`log_dir` specifies the directory where the log files are stored; relative paths are relative to `db_home`. The directory must be on a local file system. For maximum performance and reliability, place data files and log files on separate disks, managed by different disk controllers. Defaults to `logs`.

lg_bsize

`lg_bsize` specifies the size of the in-memory log buffer for the *env-name* PSS database environment in bytes. By default, or if the value is set to 0, a size of 32 K is used. The size of the log file must be at least four times the size of the the in-memory log buffer (see `lg_max`).

Log information is stored in-memory until the storage space fills up or until a transaction commit forces the information to be flushed to stable storage. In the presence of long-running transactions or transactions producing large amounts of data, larger buffer sizes can increase throughput.

lg_max

`lg_max` specifies the maximum size of a transaction log file. This configuration setting is measured in bytes.

By default, or if `lg_max` is set to `0`, a size of `10` MB is used. The size of the log file must be at least four times the size of the in-memory log buffer (see `lg_bsize`).

lk_max

`lq_max` specifies the maximum number of locks available to the Berkeley DB The default is `1000`.

For example, you may need to increase this value if you have loaded a large number of IDL interfaces into the interface repository, and then try to destroy the contents of the IFR, using `itadmin ifr destroy_contents`.

```
iona_services {
   ...
   ifr {
      ...
      plugins:pss_db:envs:ifr_store:lk_max = "10000";
    };
  };
```

This example setting prevents the IFR from crashing with the following entry in the IFR log file:

```
ERROR: DB del failed; env is ifr_store, db is
   IRObjectPSHomeImpl:1.0, errno is 12 (Not enough space)
```

old_log_dir

`old_log_dir` is used in TX mode only, and specifies the directory where the old logs are moved, when `checkpoint_deletes_old_logs` is `false`. Defaults to `old_logs`.

private

`private` specifies whether only one process is permitted to use this environment. Set to `false` when you want to obtain statistics on your database with `db_stat`. Defaults to `true`.

pull_period

`pull_period` specifies the interval, in minutes, between pull attempts by a replica. Defaults to 10.

push_all_updates

`push_all_updates` specifies if a master server will push all updates to its registered push replicas as the changes occur. Defaults to `false`.

push_update_period

`push_update_period` specifies the interval, in seconds, between a master pushing updates to its registered replicas.

replication_model

`replication_model` specifies how a replica receives updates from the master server. The two models are `push` and `pull`. Defaults to `pull`.

recover_fatal

`recover_fatal` specifies whether to performa fatal recovery instead of a normal recovery. Defaults to `false`.

run_deadlock_detector

`run_deadlock_detector` is used in TX mode only, and specifies whether the deadlock detector checks if there is a deadlock, each time a lock conflict occurs. Defaults to `true`.

tmp_dir

`tmp_dir` specifies the directory for temporary files. The directory must be on a local file system. Defaults to `tmp`.

tx_max

`tx_max` is used in TX mode only, and specifies the maximum number of concurrent transactions. Defaults to `20`.

verb_all

`verb_all` specifies whether to send verbose diagnostics about any event to the event log. Defaults to `false`.

verb_chkpoint

`verb_checkpoint` specifies whether verbose diagnostics about checkpointing are sent to the event log. Defaults to `false`.

verb_deadlock

`verb_deadlock` specifies whether to send verbose diagnostics about deadlock detection to the event log. Defaults to `false`.

verb_recovery

`verb_recovery` specifies whether to send verbose diagnostics about recovery to the event log. Defaults to `false`.

verb_waitsfor

`verb_waitsfor` specifies whether to send verbose diagnostics about lock waits to the event log. Defaults to `false`.

plugins:pss_db:envs:*env-name*:dbs:*storage-home-type-id*

Variables in `plugins:pss_db:envs:`*`env-name`*`:dbs:`*`storage-home-type-id`* act on the specified storage home—for example, `BankDemoStore/Bank:1.0`.

The following variables are contained in this namespace:

- `file_name`
- `create_file`
- `truncate_file`
- `file_mode`
- `btree`
- `rdonly`
- `bt_minkey`
- `cachesize_bytes`
- `cachesize_gbytes`
- `h_factor`
- `h_nelem`
- `pagesize`

file_name

`file_name` specifies a database file that can be shared by several storage home families.

If not specified, the storage home family is stored in its own database file. The name of this file is *`storage-home-type-id`*, with the following characters replaced with an underscore (_): forward slash and backslash (`/ \`), colon (`:`), and period (`.`). If specified, the string value must not contain any of the same characters.

create_file

`create_file` specifies whether to create the file for this storage home family, if it does not already exist. Defaults to `true`.

truncate_file

`truncate_file` specifies whether to truncate this storage home family's file. Defaults to `false`.

file_mode

`file_mode` specifies the file mode on UNIX platforms. Defaults to `0`.

btree

`btree` specifies whether a binary tree or a hash map is used. Defaults to `true`.

rdonly

`rdonly` specifies whether this storage home is family read-only. Defaults to `false`.

bt_minkey

`bt_minkey` specifies the minimum number of keys per binary tree page.

cachesize_bytes

`cachesize_bytes` specifies the database cache size in bytes. Defaults to `0`.

cachesize_gbytes

`cachesize_gbytes` specifies the database cache size in gigabytes. Defaults to `0`.

h_factor

`h_factor` specifies the hash table density.

h_nelem

`h_nelem` specifies the maximum number of elements in the hash table.

pagesize

`pagesize` specifies the database page size. Defaults to `0`.

plugins:shmiop

The variables in this namespace configure the behavior of the shared memory plugin. It contains the following variables:

- `incoming_connections:hard_limit`
- `incoming_connections:soft_limit`
- `outgoing_connections:hard_limit`
- `outgoing_connections:soft_limit`

incoming_connections:hard_limit

`incoming_connections:hard_limit` specifies the maximum number of incoming (server-side) connections permitted to SHMIOP. SHMIOP does not accept new connections above this limit. Defaults to `-1` (disabled).

incoming_connections:soft_limit

`incoming_connections:soft_limit` specifies the number of connections at which SHMIOP begins closing incoming (server-side) connections. Defaults to `-1` (disabled).

outgoing_connections:hard_limit

`outgoing_connections:hard_limit` specifies the maximum number of outgoing (client-side) connections permitted to the SHMIOP. SHMIOP 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 SHMIOP begins closing outgoing (client-side) connections. Defaults to `-1` (disabled).

Index

N

O

P

U

V

W

X