



Security Guide

Version 2.0, March 2004

Orbix, IONA Enterprise Integrator, Enterprise Integrator, Orbix E2A Application Server, Orbix E2A XMLBus, XMLBus, are trademarks or registered trademarks of IONA Technologies PLC and/or its subsidiaries.

IONA, IONA Technologies, the IONA logo, Making Software Work Together, IONA e-Business Platform, and Total Business Integration are trademarks or registered trademarks of IONA Technologies PLC and/or its subsidiaries.

Java and J2EE are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

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

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

COPYRIGHT NOTICE

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

Copyright © 2003–2004 IONA Technologies PLC. All rights reserved.

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

Updated: 05-Apr-2004

M 3 1 9 9

Contents

List of Tables	ix
List of Figures	xi
Preface	xiii
What is Covered in this Book	xiii
Who Should Read this Book	xiii
Organization of this Book	xiii
Related Documentation	xiii
Online Help	xiv
Suggested Path for Further Reading	xiv
Additional Resources for Information	xv
Typographical Conventions	xv
Keying Conventions	xvi
Chapter 1 Getting Started with Artix Security	1
Security for SOAP Bindings	2
Secure Hello World Example	3
HTTPS Connection	6
IIOP/TLS Connection	11
Security Layer	18
Chapter 2 Introduction to the Artix Security Framework	25
Artix Security Architecture	26
Types of Security Credential	27
Protocol Layers	29
Security Layer	31
Using Multiple Bindings	32
Caching of Credentials	33
Chapter 3 Security for HTTP-Compatible Bindings	35
Overview of HTTP Security	36

Securing HTTP Communications with SSL/TLS	39
HTTP Basic Authentication	45
X.509 Certificate-Based Authentication with HTTPS	48
Chapter 4 Security for SOAP Bindings	53
Overview of SOAP Security	54
Chapter 5 Security for CORBA Bindings	57
Overview of CORBA Security	58
Securing IIOP Communications with SSL/TLS	60
Securing Two-Tier CORBA Systems with CSI	66
Securing Three-Tier CORBA Systems with CSI	72
X.509 Certificate-Based Authentication for CORBA Bindings	78
Chapter 6 Single Sign-On	85
SSO and the Login Service	86
Username/Password-Based SSO for SOAP Bindings	89
SSO Sample Configuration for SOAP Bindings	96
Chapter 7 Configuring the iS2 server	99
Configuring the File Adapter	100
Configuring the LDAP Adapter	102
Configuring the SiteMinder Adapter	108
Configuring the Kerberos Adapter	110
Additional Security Configuration	113
Configuring Single Sign-On Properties	114
Configuring the Log4J Logging	116
Chapter 8 Managing Users, Roles and Domains	119
Introduction to Domains and Realms	120
Artix security domains	121
Artix Authorization Realms	123
Managing a File Security Domain	128
Managing an LDAP Security Domain	131
Managing a SiteMinder Security Domain	132
Chapter 9 Managing Access Control Lists	133

Overview of Artix ACL Files	134
Artix Action-Role Mapping ACL	135
Generating ACL Files	138
Chapter 10 Managing Certificates	141
What are X.509 Certificates?	142
Certification Authorities	144
Commercial Certification Authorities	145
Private Certification Authorities	146
Certificate Chaining	147
PKCS#12 Files	149
Creating Your Own Certificates	151
Set Up Your Own CA	152
Use the CA to Create Signed Certificates	155
Deploying Certificates	158
Overview of Certificate Deployment	159
Deploying Trusted Certificate Authority Certificates	160
Deploying Application Certificates	165
Chapter 11 Configuring HTTPS and IIOP/TLS Authentication	169
Requiring Authentication	170
Target-Only Authentication	171
Mutual Authentication	175
Specifying Trusted CA Certificates	178
Specifying an Application's Own Certificate	179
Providing a Certificate Pass Phrase	180
Certificate Pass Phrase for HTTPS	181
Certificate Pass Phrase for IIOP/TLS	183
Advanced IIOP/TLS Configuration Options	185
Setting a Maximum Certificate Chain Length	186
Applying Constraints to Certificates	187
Chapter 12 Configuring IIOP/TLS Secure Associations	189
Overview of Secure Associations	190
Setting IIOP/TLS Association Options	192
Secure Invocation Policies	193
Association Options	194
Choosing Client Behavior	196

Choosing Target Behavior	198
Hints for Setting Association Options	200
Specifying IIOP/TLS Cipher Suites	204
Supported Cipher Suites	205
Setting the Mechanism Policy	208
Constraints Imposed on Cipher Suites	210
Caching IIOP/TLS Sessions	213
Chapter 13 Principal Propagation	215
Introduction to Principal Propagation	216
Configuring	217
Programming	220
Interoperating with .NET	223
Explicitly Declaring the Principal Header	224
Modifying the SOAP Header	226
Chapter 14 Programming Authentication	229
Propagating a Username/Password Token	230
Propagating a Kerberos Token	232
Chapter 15 Setting Security Properties in Artix Contracts	235
Appendix A Security Configuration	239
initial_references Namespace	240
plugins Namespace	241
policies Namespace	252
principal_sponsor Namespace	261
principal_sponsor:csi Namespace	264
Appendix B iS2 Configuration	267
Properties File Syntax	268
iS2 Properties File	269
log4j Properties File	281
Appendix C ASN.1 and Distinguished Names	283
ASN.1	284
Distinguished Names	285

Appendix D Action-Role Mapping DTD	289
Appendix E OpenSSL Utilities	293
Using OpenSSL Utilities	294
The x509 Utility	295
The req Utility	297
The rsa Utility	299
The ca Utility	301
The OpenSSL Configuration File	303
[req] Variables	304
[ca] Variables	305
[policy] Variables	306
Example openssl.cnf File	307
Appendix F License Issues	309
OpenSSL License	310
Index	313

CONTENTS

List of Tables

Table 1: LDAP Properties in the com.ionas.adapter.LDAP.param Scope	106
Table 2: Description of Different Types of Association Option	200
Table 3: Setting EstablishTrustInTarget and EstablishTrustInClient Association Options	201
Table 4: Setting Quality of Protection Association Options	202
Table 5: Setting the NoProtection Association Option	203
Table 6: Cipher Suite Definitions	206
Table 7: Association Options Supported by Cipher Suites	211
Table 8: Contract Security Attributes	236
Table 9: Mechanism Policy Cipher Suites	256
Table 10: Commonly Used Attribute Types	286

LIST OF TABLES

List of Figures

Figure 1: Overview of the Secure HelloWorld Example	3
Figure 2: A HTTPS Connection in the HelloWorld Example	6
Figure 3: An IIOP/TLS Connection in the HelloWorld Example	11
Figure 4: The Security Layer in the HelloWorld Example	18
Figure 5: Protocol Layers in a HTTP-Compatible Binding	29
Figure 6: Protocol Layers in a SOAP Binding	30
Figure 7: Protocol Layers in a CORBA Binding	30
Figure 8: Example of an Application with Multiple Bindings	32
Figure 9: HTTP-Compatible Binding Security Layers	36
Figure 10: Overview of Certificate-Based Authentication with HTTPS	49
Figure 11: Overview of Security for SOAP Bindings	54
Figure 12: A Secure CORBA Application within the Artix Security Framework	58
Figure 13: Two-Tier CORBA System Using CSI Credentials	66
Figure 14: Three-Tier CORBA System Using CSv2	72
Figure 15: Overview of Certificate-Based Authentication	79
Figure 16: Client Requesting an SSO Token from the Login Service	87
Figure 17: Overview of Username/Password Authentication without SSO	89
Figure 18: Overview of Username/Password Authentication with SSO	90
Figure 19: Architecture of an Artix security domain	121
Figure 20: Server View of Artix authorization realms	124
Figure 21: Role View of Artix authorization realms	125
Figure 22: Assignment of Realms and Roles to Users Janet and John	126
Figure 23: A Certificate Chain of Depth 2	147
Figure 24: A Certificate Chain of Depth 3	148
Figure 25: Elements in a PKCS#12 File	149
Figure 26: Target Authentication Only	171

LIST OF FIGURES

Figure 27: Mutual Authentication	175
Figure 28: Configuration of a Secure Association	191
Figure 29: Constraining the List of Cipher Suites	210

Preface

What is Covered in this Book

This book describes how to develop and configure secure Artix solutions.

Who Should Read this Book

This book is aimed at C++ developers who are developing Artix client and server applications. The C++ API described in this book can be used with any Artix binding or transport (CORBA, SOAP and so on). It is assumed that the reader has a good knowledge of C++ and an elementary understanding of WSDL and XML concepts.

Organization of this Book

This book is divided as follows:

Part I <PART-TITLE>

<PART-DESCRIPTION>

Related Documentation

The Artix library includes the following books:

- *Artix Tutorial*
- *Getting Started with Artix Encompass*
- *Getting Started with Artix Relay*
- *Getting Started with Artix Java*
- *Designing Artix Solutions with Artix Designer*

- *Designing Artix Solutions from the Command Line*
- *Deploying and Managing Artix Solutions*
- *Developing Artix Applications in C++*
- *Developing Artix Applications in Java*
- *Artix Security Guide*

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

Online Help

Artix includes comprehensive online help, providing:

- Detailed step-by-step instructions on how to perform important tasks.
- A description of each screen.
- A comprehensive index and glossary.
- A full search feature.
- Context-sensitive help.

The **Help** menu in Artix Designer provides access to this online help.

Suggested Path for Further Reading

If you are new to Artix, we suggest you read the documentation in the following order:

1. *Getting Started with Artix Encompass*

The Getting Started book describes the basic concepts behind Artix. It also provides details on installing the system and a detailed walk through for developing a C++ Web Service.

2. *Artix Tutorial*

The Tutorial guides you through programming Artix applications against all of the supported transports.

3. *Deploying and Managing Artix Solutions*

The deployment guide describes deploying Artix enabled systems. It provides detailed examples for a number of typical use cases.

4. *Designing Artix Solutions with Artix Designer*

The Artix Designer book describes how to use the Artix GUI to describe your services in an Artix contract.

5. *Developing Artix Applications in C++/Java*

The development guide discusses the technical aspects of programming applications using the Artix API.

6. *Designing Artix Solutions from the Command Line*

This book provides detailed information about the WSDL extentions used in Artix contracts and explains the mappings between data types and Artix bindings.

Additional Resources for Information

The [IONA knowledge base](http://www.ionaproducts.com/support/knowledge_base/index.xml) (http://www.ionaproducts.com/support/knowledge_base/index.xml) contains helpful articles, written by IONA experts, about Artix and other products. You can access the knowledge base at the following location:

The [IONA update center](http://www.ionaproducts.com/support/updates/index.xml) (<http://www.ionaproducts.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, contact IONA at support@ionaproducts.com. Comments on IONA documentation can be sent to docs-support@ionaproducts.com.

Typographical Conventions

This book uses the following typographical conventions:

Constant width	<p>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 <code>CORBA::Object</code> class.</p> <p>Constant width paragraphs represent code examples or information a system displays on the screen. For example:</p> <pre>#include <stdio.h></pre>
----------------	---

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.

Getting Started with Artix Security

This chapter introduces features of Artix security by explaining the architecture and configuration of the secure HelloWorld demonstration in some detail.

In this chapter

This chapter discusses the following topics:

Security for SOAP Bindings
--

page 2

Security for SOAP Bindings

Overview

This section provides a brief overview of how the Artix Security Framework provides security for SOAP bindings between Artix applications. The Artix security framework is a comprehensive security framework that supports authentication and authorization using data stored in a central security service (the iS2 server). This discussion is illustrated by reference to the secure HelloWorld demonstration.

In this section

This section contains the following subsections:

Secure Hello World Example	page 3
HTTPS Connection	page 6
IIOP/TLS Connection	page 11
Security Layer	page 18

Secure Hello World Example

Overview

This section provides an overview of the secure HelloWorld demonstration, which introduces several features of the Artix Security Framework. In particular, this demonstration shows you how to configure a typical Artix client and server that communicate with each other using a SOAP binding over a HTTPS transport. [Figure 1](#) shows all the parts of the secure HelloWorld system, including the various configuration files.

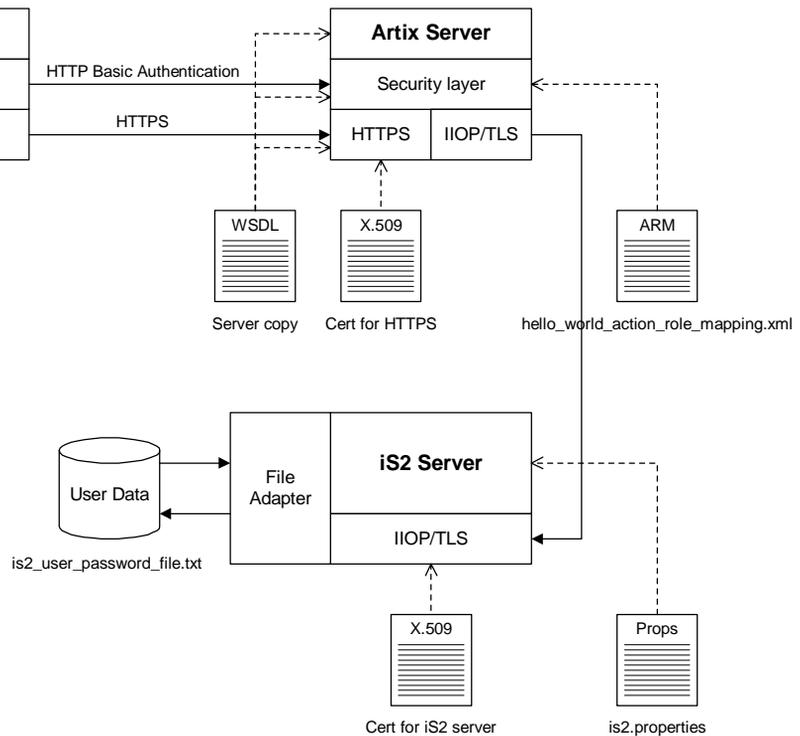


Figure 1: Overview of the Secure HelloWorld Example

Location

The secure HelloWorld demonstration is located in the following directory:
ArtixInstallDir/artix/Version/demos/security/full_security

Main elements of the example

The main elements of the secure HelloWorld example shown in [Figure 1](#) are, as follows:

- [HelloWorld client](#).
 - [HelloWorld server](#).
 - [iS2 server](#).
 - [File adapter](#).
-

HelloWorld client

The HelloWorld client communicates with the HelloWorld server using SOAP over HTTPS, thus providing confidentiality for transmitted data. In addition, the HelloWorld client is configured to use HTTP BASIC authentication to transmit a username and a password to the server.

HelloWorld server

The HelloWorld server employs two different kinds of secure transport, depending on which part of the system it is talking to:

- HTTPS—to receive SOAP invocations securely from the HelloWorld client.
 - IIOP/TLS—to communicate securely with the iS2 server, which contains the central store of user data.
-

iS2 server

The iS2 server manages a central repository of security-related user data. The iS2 server can be accessed remotely by Artix servers and offers the service of authenticating users and retrieving authorization data.

File adapter

The iS2 server supports a number of adapters that can be used to integrate with third-party security products (for example, an LDAP adapter and a SiteMinder adapter are available). This example uses the *iSF file adapter*, which is a simple adapter provided for demonstration purposes.

WARNING: The file adapter is provided for demonstration purposes only. IONA does not support the use of the file adapter in a production environment.

Security layers

To facilitate the discussion of the HelloWorld security infrastructure, it is helpful to analyze the security features into the following layers:

- [HTTPS layer](#).
 - [IIOP/TLS layer](#).
 - [Security layer](#).
-

HTTPS layer

The HTTPS layer provides a secure transport layer for SOAP bindings. In Artix, the HTTPS transport is configured by editing the WSDL contract (both the client copy and the server copy).

For more details, see [“HTTPS Connection” on page 6](#).

IIOP/TLS layer

The IIOP/TLS layer consists of the OMG’s Internet Inter-ORB Protocol (IIOP) combined with the SSL/TLS protocol. The IIOP/TLS transport can be used either with CORBA bindings or with the Artix Tunnel plug-in. In Artix, the IIOP/TLS is configured by editing the `artix.cfg` (or `artix-secure.cfg`) file.

For more details, see [“IIOP/TLS Connection” on page 11](#).

Security layer

The security layer provides support for a simple username/password authentication mechanism, a principal authentication mechanism and support for authorization. A security administrator can edit an *action-role mapping file* to restrict user access to particular WSDL port types and operations.

For more details, see [“Security Layer” on page 18](#).

HTTPS Connection

Overview

Figure 2 shows an overview of the HelloWorld example, focusing on the elements relevant to the HTTPS connection. HTTPS is used on the SOAP binding between the Artix client and the Artix server.

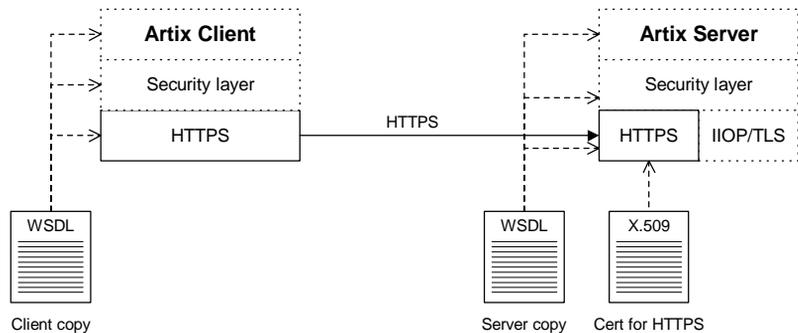


Figure 2: A HTTPS Connection in the HelloWorld Example

OpenSSL toolkit

HTTPS transport security is provided by the OpenSSL toolkit, which is a publicly available implementation of the SSL protocol.

The OpenSSL libraries (`libeay.dll` and `ssleay.dll` on Windows) are provided with Artix. The version of the OpenSSL libraries provided with Artix are, however, subject to certain restrictions as follows:

- IDEA is not supported.
- Certain encryption suites are not supported.

HTTPS cipher suites

The OpenSSL libraries provided with Artix support the following cipher suites, which can be used by the HTTPS protocol:

- Null encryption, integrity-only ciphers:

NULL-MD5

NULL-SHA

- Standard ciphers:

RC4-SHA

RC4-MD5

DES-CBC3-SHA

DES-CBC-SHA

EXP-DES-CBC-SHA

EXP-RC2-CBC-MD5

EXP-RC4-MD5

EDH-RSA-DES-CBC-SHA

EDH-DSS-DES-CBC-SHA

EXP-EDH-RSA-DES-CBC

EXP-EDH-DSS-DES-CBC-SHA

EDH-RSA-DES-CBC3-SHA

EDH-DSS-DES-CBC3-SHA

Target-only authentication

The HelloWorld example is configured to use *target-only authentication* on the HTTPS connection. That is, during the TLS handshake, the server authenticates itself to the client (using an X.509 certificate), but the client does not authenticate itself to the server. Hence, there is no X.509 certificate associated with the client.

Client HTTPS configuration

[Example 1](#) shows how to configure the client side of a HTTPS connection in Artix, in the case of target-only authentication.

Example 1: WSDL Contract with Client HTTPS Configuration

```

<definitions name="HelloWorldService"
  targetNamespace="http://xmlbus.com/HelloWorld"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:http-conf="http://schemas.iona.com/transport/http/configuration" ... >
  ...
  <service name="HelloWorldService">
    <port binding="tns:HelloWorldPortBinding"
      name="HelloWorldPort">
      <soap:address location="https://localhost:55012"/>
      <http-conf:client

```

1
2

Example 1: WSDL Contract with Client HTTPS Configuration

```

3         UseSecureSockets="true"
4   TrustedRootCertificates="..\certificates/openssl/x509/ca/cacert.
      pem"
          UserName="user_test"
          Password="user_password"
      />
    </port>
  </service>
</definitions>

```

The preceding WSDL contract can be described as follows:

1. The fact that this is a secure connection is signalled here by using `https:` instead of `http:` in the location URL attribute.
2. The `<http-conf:client>` tag contains all the attributes for configuring the client side of the HTTPS connection.
3. If the `UseSecureSockets` attribute is `true`, the client will try to open a secure connection to the server.

Note: If `UseSecureSockets` is `false` and the `<soap:address>` location URL begins with `https:`, however, the client will nevertheless attempt to open a secure connection.

4. The file specified by the `TrustedRootCertificates` contains 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.

Server HTTPS configuration

Example 2 shows how to configure the server side of a HTTPS connection in Artix, in the case of target-only authentication.

Example 2: WSDL Contract with Server HTTPS Configuration

```

<definitions name="HelloWorldService"
  targetNamespace="http://xmlbus.com/HelloWorld"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:http-conf="http://schemas.iona.com/transport/http/configu
  ration"
1   xmlns:bus-security="http://schemas.iona.com/bus/security"
  ... >

```

Example 2: WSDL Contract with Server HTTPS Configuration

```

...
<service name="HelloWorldService">
  <port binding="tns:HelloWorldPortBinding"
name="HelloWorldPort">
2     <soap:address location="https://localhost:55012"/>
3     <http-conf:server
4         UseSecureSockets="true"
5
6     ServerCertificate="../certificates/openssl/x509/certs/key.cer
t.pem"
7
8     ServerPrivateKey="../certificates/openssl/x509/certs/privkey.
pem"
        ServerPrivateKeyPassword="testaspen"
    TrustedRootCertificates="../certificates/openssl/x509/ca/cace
rt.pem"
    />
  </port>
</service>
</definitions>

```

The preceding WSDL contract can be described as follows:

1. The `bus-security` namespace prefix must be defined here, because this prefix is used to identify the artix security interceptor in the server's domain configuration (see ["Server domain configuration and access control" on page 21](#)).
2. The fact that this is a secure connection is signalled by using `https:` instead of `http:` in the location URL attribute.
3. The `<http-conf:server` tag contains all the attributes for configuring the server side of the HTTPS connection.
4. If the `UseSecureSockets` attribute is `true`, the server will open a port to listen for secure connections.

Note: If `UseSecureSockets` is `false` and the `<soap:address>` location URL begins with `https:`, however, the server will listen for secure connections.

5. The `ServerCertificate` attribute specifies the server's own certificate in PEM format. For more background details about X.509 certificates, see ["Managing Certificates" on page 141](#).
6. The `ServerPrivateKey` attribute specifies a PEM file containing the server certificate's encrypted private key.
7. The `ServerPrivateKeyPassword` attribute specifies the password to decrypt the server certificate's private key.

Note: The presence of the private key password in the WSDL contract file implies that this file must be read and write-protected to prevent unauthorized users from obtaining the password.

For the same reason, it is also advisable to remove the `<http-conf:server>` tag from the copy of the WSDL contract that is distributed to clients.

8. The file specified by the `TrustedRootCertificates` contains a concatenated list of CA certificates in PEM format. This attribute value is not used in the case of target-only authentication.

IIOP/TLS Connection

Overview

Figure 3 shows an overview of the HelloWorld example, focusing on the elements relevant to the IIOP/TLS connection between the Artix server and the iS2 server. In general, the iS2 server is accessible only through the IIOP/TLS transport.

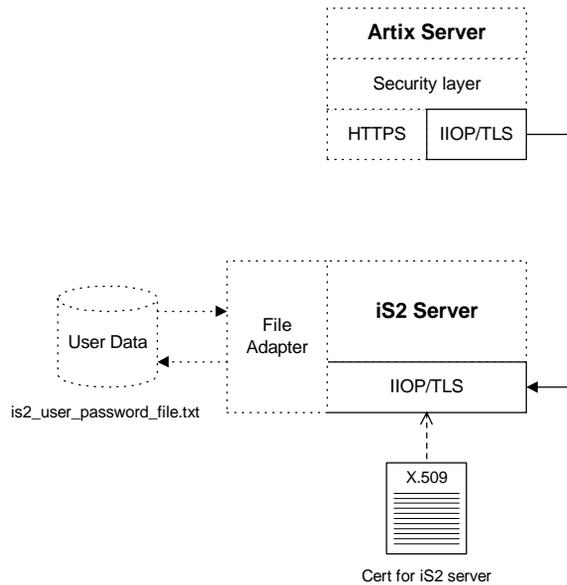


Figure 3: An IIOP/TLS Connection in the HelloWorld Example

Baltimore toolkit

IIOP/TLS transport security is provided by the Baltimore toolkit, which is a commercial implementation of the SSL/TLS protocol.

The Baltimore toolkit supports a wide range of cipher suites—see “Supported Cipher Suites” on page 205.

Target-only authentication

The HelloWorld example is configured to use *target-only authentication* on the IIOP/TLS connection between the Artix server and the iS2 server. That is, during the TLS handshake, the iS2 server authenticates itself to the Artix server (using an X.509 certificate), but the Artix server does not authenticate itself to the iS2 server. Hence, in this example there is no X.509 certificate associated with the IIOP/TLS transport in the Artix server.

WARNING: For a real deployment, you *must* modify the configuration of the iS2 server so that it requires *mutual* authentication. Otherwise, your system will be insecure.

Artix server IIOP/TLS configuration

The Artix server's IIOP/TLS transport is configured by the settings in the *ArtixInstallDir/artix/2.0/etc/domains/artix-secure.cfg* file. [Example 3](#) shows an extract from the *artix-secure.cfg* file, highlighting some of the settings that are important for the HelloWorld Artix server.

Example 3: *Extract from the Artix Server IIOP/TLS Configuration*

```
# artix-secure.cfg File
secure_artix
{
  ...
  1 policies:trusted_ca_list_policy =
    "C:\artix\artix\1.2\demos\secure_hello_world\http_soap\certif
    icates\tls\x509\trusted_ca_lists\ca_list1.pem";
  ...
  2 initial_references:IT_SecurityService:reference =
    "corbaloc:iiops:1.2@localhost:55020,it_iiops:1.2@localhost:55
    020/IT_SecurityService";
  ...
  demos
  {
    hello_world
    {
      # IIOP/TLS Settings
      3 orb_plugins = ["xmlfile_log_stream", "iiop_profile",
        "giop", "iiop_tls", "soap", "http", "tunnel", "mq", "ws_orb",
        "fixed"];
        binding:client_binding_list = ["OTS+POA_Coloc",
        "POA_Coloc", "OTS+GIOP+IIOP", "GIOP+IIOP", "GIOP+IIOP_TLS"];
      4 principal_sponsor:use_principal_sponsor = "false";
    }
  }
}
```

Example 3: *Extract from the Artix Server IOP/TLS Configuration*

```

5 policies:iop_tls:client_secure_invocation_policy:requires =
  ["Integrity", "Confidentiality", "DetectReplay",
  "DetectMisordering", "EstablishTrustInTarget"];

  policies:iop_tls:client_secure_invocation_policy:supports =
  ["Integrity", "Confidentiality", "DetectReplay",
  "DetectMisordering", "EstablishTrustInTarget"];

6     # Security Layer Settings
      ...
    };
  };
};

```

The preceding extract from the `artix.cfg` file can be explained as follows:

1. The `policies:trusted_ca_list_policy` variable specifies a file containing a concatenated list of CA certificates. These CA certificates are used to check the acceptability of any certificates received by the Artix server over the IOP/TLS transport. If a received certificate has not been digitally signed by one of the CA certificates in the list, it will be rejected by the Artix server.
For more details, see [“Specifying Trusted CA Certificates” on page 178](#).
2. This `IT_SecurityService` initial reference gives the location of the iS2 server. When login security is enabled, the Artix server uses this information to open an IOP/TLS connection to the iS2 server. In this example, the iS2 server is presumed to be running on `localhost` and listening on the `55020` IP port.

Note: If you want to change the location of the iS2 server, you should replace both instances of `localhost:55020` on this line. It would also be necessary to change the listening details on the iS2 server (see [“iS2 server IOP/TLS configuration” on page 15](#)).

3. The ORB `plugins` list specifies which of the Artix plug-ins should be loaded into the Artix server. Of particular relevance is the fact that the `iiop_tls` plug-in is included in the list (thus enabling IIOP/TLS connections), whereas the `iiop` plug-in is excluded (thus disabling plain IIOP connections).
4. The `principal_sponsor` settings can be used to attach a certificate to the Artix server, which would be used to identify the server to its peers during an IIOP/TLS handshake. In this example, however, the principal sponsor is disabled (that is, `principal_sponsor:use_principal_sponsor="false"`).

Note: In a realistic deployment, you should enable the principal sponsor and attach a certificate to the Artix server so that the Artix server can identify itself to the iS2 server.

5. The client secure invocation policies specify what sort of secure IIOP/TLS connections the Artix server can open when it acts in a client role. In particular, these client invocation policies impose conditions on the IIOP/TLS connection to the iS2 server.

For more details about the client secure invocation policy, see [“Setting IIOP/TLS Association Options” on page 192](#).

Note: In a realistic deployment, you should add the `EstablishTrustInClient` association option to the list of supported client invocation policies. This is needed for mutual authentication.

6. Independently of the IIOP/TLS settings, you also configure the security layer using settings in the `artix-secure.cfg` file. These settings are described in [“Security Layer” on page 18](#).

iS2 server IIOP/TLS configuration

Example 4 shows an extract from the `artix-secure.cfg` file, highlighting the IIOP/TLS settings that are important for the iS2 server.

Example 4: *Extract from the iS2 server IIOP/TLS Configuration*

```
# artix-secure.cfg File
secure_artix
{
    ...
1   policies:trusted_ca_list_policy =
    "C:\artix\artix\1.2\demos\secure_hello_world\http_soap\certif
    icates\tls\x509\trusted_ca_lists\ca_list1.pem";
    ...
    initial_references:IT_SecurityService:reference =
    "corbaloc:iiops:1.2@localhost:55020,it_iiops:1.2@localhost:55
    020/IT_SecurityService";
    ...
    security
    {
        # IIOP/TLS Settings
        ...
2   principal_sponsor:use_principal_sponsor = "true";
        principal_sponsor:auth_method_id = "pkcs12_file";
        principal_sponsor:auth_method_data =
3   ["filename=C:\artix\artix\1.2\demos\secure_hello_world\http_s
        oap\certificates\tls\x509\certs\services\administrator.pl2",
        "password_file=C:\artix\artix\1.2\demos\secure_hello_world\ht
        tp_soap\certificates\tls\x509\certs\services\administrator.pw
        f"];
        ...
4   policies:target_secure_invocation_policy:requires =
        ["NoProtection"];
        policies:target_secure_invocation_policy:supports =
        ["NoProtection", "Confidentiality", "EstablishTrustInTarget",
        "EstablishTrustInClient", "DetectMisordering",
        "DetectReplay", "Integrity"];
        ...
        policies:client_secure_invocation_policy:requires =
        ["NoProtection"];
        policies:client_secure_invocation_policy:supports =
        ["NoProtection", "Confidentiality", "EstablishTrustInTarget",
        "EstablishTrustInClient", "DetectMisordering",
        "DetectReplay", "Integrity"];
        policies:allow_unauthenticated_clients_policy = "true";
    }
}
```

Example 4: *Extract from the iS2 server IIOP/TLS Configuration*

```

5     orb_plugins = ["local_log_stream", "iiop_profile", "giop",
"iiop_tls"];
    ...
6     plugins:security:iiop_tls:port = "55020";
    plugins:security:iiop_tls:host = "localhost";
    ...
    };
    ...
};

```

The preceding extract from the `artix.cfg` file can be explained as follows:

1. The `policies:trusted_ca_list_policy` variable specifies a file containing a concatenated list of CA certificates. These CA certificates are used to check the acceptability of any certificates received by the iS2 server over the IIOP/TLS transport. If a received certificate has not been digitally signed by one of the CA certificates in the list, it will be rejected by the iS2 server.
2. The `principal_sponsor` settings are used to attach an X.509 certificate to the iS2 server. The certificate is used to identify the iS2 server to its peers during an IIOP/TLS handshake.

In this example, the iS2 server's certificate is stored in a PKCS#12 file, `administrator.p12`, and the certificate's private key password is stored in another file, `administrator.pwf`.

For more details about configuring the IIOP/TLS principal sponsor, see [“principal_sponsor Namespace” on page 260](#) and [“Providing a Certificate Pass Phrase” on page 180](#).

Note: The certificate format used by the IIOP/TLS transport (PKCS#12) differs from the format used by the HTTPS transport (PEM).

3. The target secure invocation policies specify what sort of secure IIOp/TLS connections the iS2 server can accept when it acts in a server role. For more details about the target secure invocation policy, see [“Setting IIOp/TLS Association Options” on page 192](#).

WARNING: The target secure invocation policies shown here are too weak for a realistic deployment of the iS2 server. In particular, you should at least remove support for `NoProtection` and require `EstablishTrustInClient`. For example, see [“Mutual Authentication” on page 175](#).

4. The client secure invocation policies specify what sort of secure IIOp/TLS connections the iS2 server can open when it acts in a client role.
5. The ORB `plugins` list specifies which plug-ins should be loaded into the iS2 server. Of particular relevance is the fact that the `iiop_tls` plug-in is included in the list (thus enabling IIOp/TLS connections), whereas the `iiop` plug-in is excluded (thus disabling plain IIOp connections).
6. If you want to relocate the iS2 server, you must modify the `plugins:security:iiop_tls:host` and `plugins:security:iiop_tls:port` settings to specify, respectively, the host where the server is running and the IP port on which the server listens for secure IIOp/TLS connections.

Security Layer

Overview

Figure 4 shows an overview of the HelloWorld example, focusing on the elements relevant to the security layer. The security layer, in general, takes care of those aspects of security that arise *after* the initial SSL/TLS handshake has occurred and the secure connection has been set up.

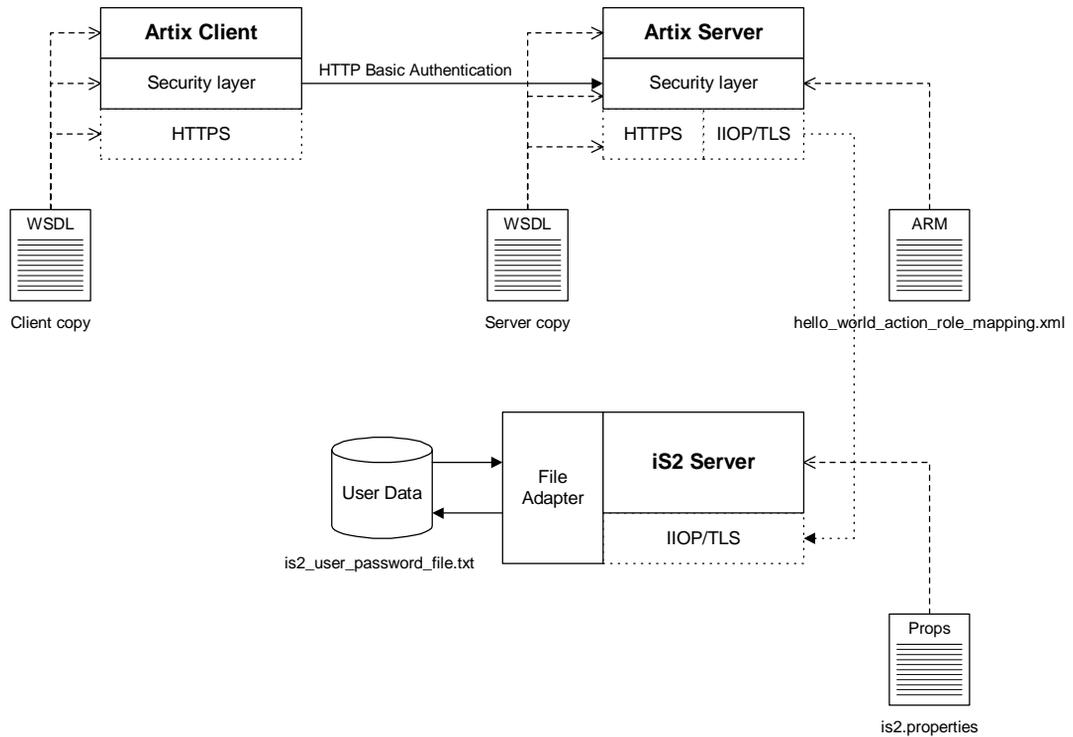


Figure 4: *The Security Layer in the HelloWorld Example*

The security layer normally uses a simple username/password combination for authentication, because clients usually do not have a certificate with which to identify themselves. The username and password are sent along with every operation, enabling the Artix server to check every invocation and make fine-grained access decisions.

HTTP BASIC login

The mechanism that the Artix client uses to transmit a username and password over a SOAP binding is *HTTP BASIC login*. This is a standard login mechanism commonly used by Web browsers and Web services. On its own, HTTP BASIC login would be relatively insecure, because the username and password would be transmitted in plaintext. When combined with the HTTPS protocol, however, the username and password are transmitted securely over an encrypted connection, thus preventing eavesdropping.

The following extract from the client copy of the WSDL contract shows how the `UserName` and `Password` attributes in the `<http-conf:client>` tag set the HTTP BASIC login parameters for the Artix SOAP client.

```
<definitions name="HelloWorldService"
  targetNamespace="http://xmlbus.com/HelloWorld"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:http-conf="http://schemas.iona.com/transport/http/configuration" ... >
  ...
  <service name="HelloWorldService">
    <port binding="tns:HelloWorldPortBinding"
      name="HelloWorldPort">
      <soap:address location="https://localhost:55012"/>
      <http-conf:client
        ...
        UserName="user_test"
        Password="user_password"
      />
    </port>
  </service>
</definitions>
```

Authentication through the iSF file adapter

On the server side, the Artix server delegates authentication to the iS2 server, which acts as a central repository for user data. The iS2 server is configured by the `is2.properties` file, whose location is specified in the `artix-secure.cfg` file as follows:

```
# artix-secure.cfg File
secure_artix {
  ...
  security {
    plugins:java_server:system_properties =
    ["org.omg.CORBA.ORBClass=com.ionacorba.art.artimpl.ORBImpl",
    "org.omg.CORBA.ORBSingletonClass=com.ionacorba.art.artimpl.ORBSingleton",
    "is2.properties=C:\artix\artix\1.2\demos\secure_hello_world\http_soap\bin\is2.properties.FILE",
    "java.endorsed.dirs=C:\artix\artix\1.2\lib\endorsed"];
    ...
  };
  ...
};
```

In this example, the `is2.properties` file specifies that the iS2 server should use a file adapter. The file adapter is configured as follows:

```
# is2.properties File
...
#####
##
## File Adapter Properties
##
#####
com.iona.isp.adapter.file.class=com.iona.security.is2adapter.file.FileAuthAdapter
com.iona.isp.adapter.file.params=filename
com.iona.isp.adapter.file.param.filename=../config/is2_user_password_file.txt
```

The `com.iona.isp.adapter.file.param.filename` property is used to specify the location of a file, `is2_user_password_file.txt`, which contains the user data for the iSF file adapter. [Example 5](#) shows the contents of the user data file for the secure HelloWorld demonstration.

Example 5: *User Data from the `is2_user_password_file.txt` File*

```
<?xml version="1.0" encoding="utf-8" ?>

<ns:securityInfo xmlns:ns="urn:www-xmlbus-com:simple-security">
  <users>
    <users>
      <user name="user_test" password="user_password">
        <realm name="IONAGlobalRealm">
          <role name="IONAUserRole" />
          <role name="PaulOnlyRole" />
        </realm>
      </user>
    </users>
  </ns:securityInfo>
```

In order for the login step to succeed, an Artix client must supply one of the usernames and passwords that appear in this file. The realm and role data, which also appear, are used for authorization and access control.

For more details about the iSF file adapter, see [“Managing a File Security Domain” on page 128](#).

WARNING: The file adapter is provided for demonstration purposes only. IONA does not support the use of the file adapter in a production environment.

Server domain configuration and access control

On the server side, authentication and authorization must be enabled by the appropriate settings in the `artix-secure.cfg` file. [Example 6](#) explains the security layer settings that appear in the `artix-secure.cfg` file.

Example 6: *Security Layer Settings from the `artix-secure.cfg` File*

```
# artix-secure.cfg File
secure_artix
{
  ...
  demos
  {
```

Example 6: Security Layer Settings from the `artix-secure.cfg` File

```

hello_world
{
  # IIOP/TLS Settings
  ...

  # Security Layer Settings
  plugins:artix_security:shlib_name="it_security_plugin";
1  binding:artix:server_request_interceptor_list=
  "bus-security:security";
2  orb_plugins = ["xmlfile_log_stream", "iiop_profile",
  "giop", "iiop_tls", "soap", "http", "artix_security"];

3  policies:asp:enable_authorization = "true";
4  plugins:is2_authorization:action_role_mapping =
  "file://C:\artix\artix\1.2\demos\secure_hello_world\http_soap
  /config/helloworld_action_role_mapping.xml";
5  plugins:asp:authorization_realm = "IONAGlobalRealm";
6  plugins:asp:security_type = "USERNAME_PASSWORD";
  };
};
};

```

The security layer settings from the `artix-secure.cfg` file can be explained as follows:

1. The Artix server request interceptor list must include the `bus-security:security` interceptor, which provides part of the functionality for the Artix security layer.

Note: The `bus-security` namespace prefix must be defined in the application WSDL contract—see [“Server HTTPS configuration” on page 8](#).

2. The server’s `orb_plugins` list must include the `artix_security` plug-in.
3. The `policies:asp:enable_authorization` variable is set to `true` to enable authorization.
4. This setting specifies the location of an *action-role mapping file* that provides fine-grained access control to operations and port types.

5. The Artix authorization realm determines which of the user's roles will be considered during an access control decision. Artix authorization realms provide a way of grouping user roles together. The `IONAGlobalRealm` (the default) includes all user roles.
6. The `plugins:asp:security_type` variable specifies which kind of user data is used for the purposes of authentication and authorization on the server side (in this case, `USERNAME_PASSWORD` indicates that HTTP Basic Login is supported). This configuration setting is necessary, because the Artix security framework supports different mechanisms for propagating user identities and some of these mechanisms can be activated simultaneously.

[Example 7](#) shows the contents of the action-role mapping file for the HelloWorld demonstration.

Example 7: *Action-Role Mapping file for the HelloWorld Demonstration*

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE secure-system SYSTEM "actionrolemapping.dtd">
<secure-system>
  <action-role-mapping>

    <server-name>secure_artix.demos.hello_world</server-name>

    <interface>

      <name>http://xmlbus.com/HelloWorld:HelloWorldPortType</name>
      <action-role>
        <action-name>sayHi</action-name>
        <role-name>IONAUserRole</role-name>
      </action-role>
      <action-role>
        <action-name>greetMe</action-name>
        <role-name>IONAUserRole</role-name>
      </action-role>
    </interface>

  </action-role-mapping>
</secure-system>
```

For a detailed discussion of how to define access control using action-role mapping files, see [“Managing Users, Roles and Domains”](#) on page 119.

Introduction to the Artix Security Framework

This chapter describes the overall architecture of the Artix Security Framework.

In this chapter

This chapter discusses the following topics:

Artix Security Architecture	page 26
Caching of Credentials	page 33

Artix Security Architecture

Overview

The Artix security architecture embraces a variety of protocols and security technologies. This section provides a brief overview of the security features supported by the different kinds of Artix bindings.

In this section

This section contains the following subsections:

Types of Security Credential	page 27
Protocol Layers	page 29
Security Layer	page 31
Using Multiple Bindings	page 32

Types of Security Credential

Overview

The following types of security credentials are supported by the Artix security framework:

- [WSSE username token](#).
 - [WSSE Kerberos token](#).
 - [CORBA Principal](#).
 - [HTTP Basic Authentication](#).
 - [X.509 certificate](#).
 - [CSI authorization over transport](#).
 - [CSI identity assertion](#).
 - [SSO token](#).
-

WSSE username token

The Web service security extension (WSSE) UsernameToken is a username/password combination that can be sent in a SOAP header. The specification of WSSE UsernameToken is contained in the [WSS UsernameToken Profile 1.0](#) document from [OASIS](#) (www.oasis-open.org).

This type of credential is available for the SOAP binding in combination with any kind of Artix transport.

WSSE Kerberos token

The WSSE Kerberos specification is used to send a Kerberos security token in a SOAP header. If you use Kerberos, you must also configure the iS2 server to use the Kerberos adapter.

This type of credential is available for the SOAP binding in combination with any kind of Artix transport.

CORBA Principal

The CORBA Principal is a legacy feature originally defined in the early versions of the CORBA GIOP specification. The CORBA Principal is effectively just a username (no username can be propagated).

This type of credential is available only for the CORBA binding and for SOAP over HTTP.

HTTP Basic Authentication

HTTP Basic Authentication is used to propagate username/password credentials in a HTTP header.

This type of credential is available to any HTTP-compatible binding.

X.509 certificate

Two different kinds of X.509 certificate-based authentication are provided, depending on the type of Artix binding, as follows:

- *HTTP-compatible binding*—in this case, the common name (CN) is extracted from the X.509 certificate's subject DN. A combination of the common name and a default password is then sent to the iS2 server to be authenticated.
- *CORBA binding*—in this case, authentication is based on the entire X.509 certificate, which is sent to the iS2 server to be authenticated.

This type of credential is available to any transport that uses SSL/TLS.

CSI authorization over transport

The OMG's Common Secure Interoperability (CSI) specification defines an *authorization over transport* mechanism, which passes username/password data inside a GIOP service context. This kind of authentication is available only for the CORBA binding.

This type of credential is available only for the CORBA binding.

CSI identity assertion

The OMG's Common Secure Interoperability (CSI) specification also defines an *identity assertion* mechanism, which passes username data (no password) inside a GIOP service context. The basic idea behind CSI identity assertion is that the request message comes from a secure peer that can be trusted to assert the identity of a user. This kind of authentication is available only for the CORBA binding.

This type of credential is available only for the CORBA binding.

SSO token

An SSO token is propagated in the context of a system that uses *single sign-on*. For details of the Artix single sign-on feature, see [“Single Sign-On” on page 85](#).

Protocol Layers

Overview

Within the Artix security architecture, each binding type consists of a stack of protocol layers, where a protocol layer is typically implemented as a distinct Artix plug-in. This subsection describes the protocol layers for the following binding types:

- [HTTP-compatible binding](#).
 - [SOAP binding](#).
 - [CORBA binding](#).
-

HTTP-compatible binding

HTTP-compatible means any Artix binding that can be layered on top of the HTTP protocol. [Figure 5](#) shows the protocol layers and the kinds of authentication available to a HTTP-compatible binding.

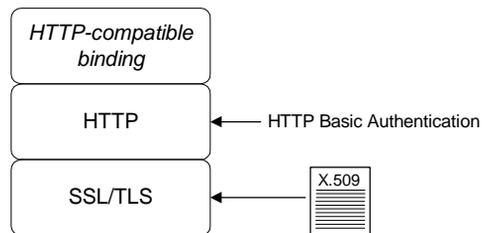


Figure 5: *Protocol Layers in a HTTP-Compatible Binding*

SOAP binding

The SOAP binding is a specific example of a HTTP-compatible binding. The SOAP binding is special, because it defines several additional credentials that can be propagated only in a SOAP header. Figure 6 shows the protocol layers and the kinds of authentication available to the SOAP binding over HTTP.

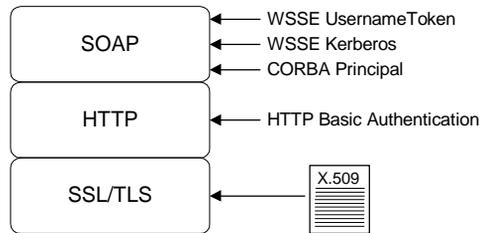


Figure 6: Protocol Layers in a SOAP Binding

CORBA binding

For the CORBA binding, there are only two protocol layers (CORBA binding and IIOP/TLS). This is because CORBA is compatible with only one kind of message format (that is, GIOP). Figure 7 shows the protocol layers and the kinds of authentication available to the CORBA binding.

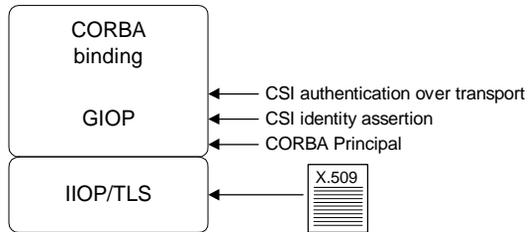


Figure 7: Protocol Layers in a CORBA Binding

Security Layer

Overview

The *security layer* is responsible for implementing a variety of different security features with the exception, however, of propagating security credentials, which is the responsibility of the protocol layers. The security layer is at least partially responsible for implementing the following security features:

- [Authentication](#).
- [Authorization](#).
- [Single sign-on](#).

Authentication

On the server side, the security layer selects one of the client credentials (a server can receive more than one kind of credentials from a client) and calls the central iS2 server to authenticate the credentials. If the authentication call succeeds, the security layer proceeds to make an authorization check; otherwise, an exception is thrown back to the client.

Authorization

The security layer makes an authorization check by matching a user's roles and realms against the ACL entries in an *action-role mapping file*. If the user does not have permission to invoke the current action (that is, WSDL operation), an exception is thrown back to the client.

Single sign-on

Single sign-on is an optional feature that increases security by reducing the number of times that a user's credentials are sent across the network. The security layer works in tandem with the login service to provide the single sign-on feature.

Artix security plug-in

The Artix security plug-in provides the security layer for all Artix bindings except CORBA. The ASP security layer is loaded, if `artix_security` is listed in the `orb_plugins` list in the Artix domain configuration, `artix.cfg`.

GSP security plug-in

The GSP security plug-in provides the security layer for the CORBA binding only. The GSP security layer is loaded, if `gsp` is listed in the `orb_plugins` list in the Artix domain configuration, `artix.cfg`.

Using Multiple Bindings

Overview

Figure 8 shows an example of an advanced application that uses multiple secure bindings.

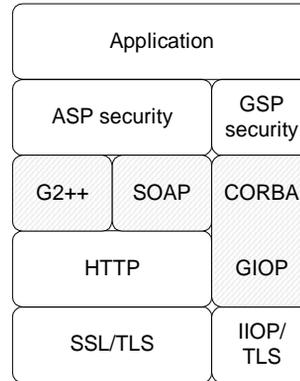


Figure 8: Example of an Application with Multiple Bindings

This type of application might be used as a bridge, for example, to link a CORBA domain to a SOAP domain. Alternatively, the application might be a server designed as part of a migration strategy, where the server can support requests in multiple formats, such as G2++, SOAP, or CORBA.

Example bindings

The following bindings are used in the application shown in Figure 8:

- G2++—consisting of the following layers: ASP security, G2++ binding, HTTP, SSL/TLS.
- SOAP—consisting of the following layers: ASP security, SOAP binding, HTTP, SSL/TLS.
- CORBA—consisting of the following layers: GSP security, CORBA binding, GIOP, IIOP/TLS.

Caching of Credentials

Overview

To improve the performance of servers within the Artix Security Framework, both the GSP plug-in (CORBA binding only) and the artix security plug-in (all other bindings) implement caching of credentials (that is, the authentication and authorization data received from the iS2 server).

The credentials cache reduces a server's response time by reducing the number of remote calls to the iS2 server. On the first call from a given user, the server calls the iS2 server and caches the received credentials. On subsequent calls from the same user, the cached credentials are used, thereby avoiding a remote call to iS2 server.

Cache time-out

The cache can be configured to time-out credentials, forcing the server to call the iS2 server again after using cached credentials for a certain period.

Cache size

The cache can also be configured to limit the number of stored credentials.

GSP configuration variables

The following variables configure the credentials cache for CORBA bindings:

`plugins:gsp: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.

`plugins:gsp: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 iS2 server 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.

ASP configuration variables

The following variables configure the credentials cache for all non-CORBA bindings:

`plugins:asp: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.

`plugins:asp: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 iS2 server 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.

Security for HTTP-Compatible Bindings

This chapter describes the security features supported by the Artix HTTP plug-in. These security features are available to any Artix binding that can be layered on top of the HTTP transport.

In this chapter

This chapter discusses the following topics:

Overview of HTTP Security	page 36
Securing HTTP Communications with SSL/TLS	page 39
HTTP Basic Authentication	page 45
X.509 Certificate-Based Authentication with HTTPS	page 48

Overview of HTTP Security

Overview

Figure 9 gives an overview of HTTP security within the Artix security framework, showing the various security layers (ASP, binding layer, HTTP, and SSL/TLS) and the different authentication types associated with the security layers. Because many different binding types (for example, SOAP or G2++) can be layered on top of HTTP, Figure 9 does not specify a particular binding layer. Any HTTP-compatible binding could be substituted into this architecture.

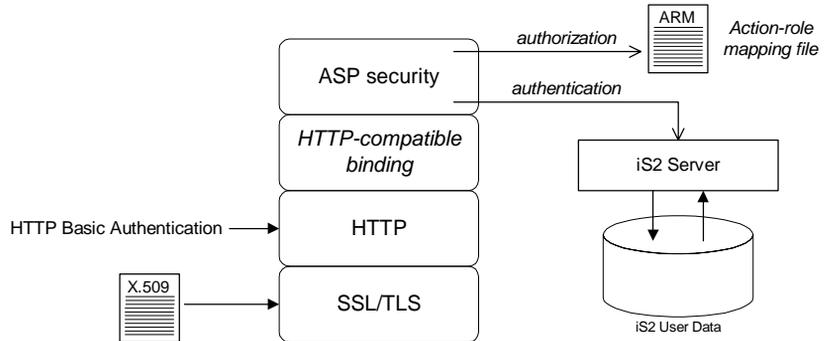


Figure 9: HTTP-Compatible Binding Security Layers

Security layers

As shown in Figure 9, a HTTP-compatible binding has the following security layers:

- SSL/TLS layer.
- HTTP layer.
- HTTP-compatible binding layer.
- ASP security layer.

SSL/TLS layer

The SSL/TLS layer provides guarantees of confidentiality, message integrity, and authentication (using X.509 certificates). The TLS functionality is integrated into the HTTP plug-in and can be switched on or off by a WSDL configuration setting.

The HTTP plug-in's TLS layer is configured by editing an application's WSDL contract.

HTTP layer

The HTTP layer offers the option of sending username/password data in the HTTP message header (that is, *HTTP Basic Authentication*).

HTTP Basic Authentication is configured by editing an application's WSDL contract.

HTTP-compatible binding layer

The HTTP-compatible binding layer could provide additional security features (for example, propagation of security credentials), depending on the type of binding. The following binding types are HTTP-compatible:

- SOAP binding.
 - XML format binding.
 - G2++ binding.
 - Fixed record length binding.
 - Tagged data binding.
 - MIME binding.
-

ASP security layer

The ASP security layer is implemented by the Artix security plug-in, which provides authentication and authorization checks for all binding types, except the CORBA binding, as follows:

- *Authentication*—by selecting one of the available client credentials and calling out to the iS2 server to check the credentials.
- *Authorization*—by reading an action-role mapping (ARM) file and checking whether a user's roles allow it to perform a particular action.

Authentication options

The following authentication options are common to all HTTP-compatible bindings:

- [HTTP Basic Authentication](#).
 - [X.509 certificate-based authentication](#).
-

HTTP Basic Authentication

HTTP Basic Authentication works by sending a username and password embedded in the HTTP message header. This style of authentication is commonly used by clients running in a Web browser.

For details of HTTP Basic Authentication, see [“HTTP Basic Authentication” on page 45](#).

X.509 certificate-based authentication

X.509 certificate-based authentication is an authentication step that is performed *in addition to* the checks performed at the socket layer during the SSL/TLS security handshake.

For details of X.509 certificate-based authentication, see [“X.509 Certificate-Based Authentication with HTTPS” on page 48](#).

Securing HTTP Communications with SSL/TLS

Overview

This subsection describes how to configure the HTTP transport to use SSL/TLS security, a combination usually referred to as HTTPS. In Artix, HTTPS security is implemented by the HTTP plug-in and configured by settings in the WSDL contract.

The following topics are discussed in this subsection:

- [Generating X.509 certificates.](#)
- [HTTPS client with no certificate.](#)
- [HTTPS client with certificate.](#)
- [HTTPS server configuration.](#)

Generating X.509 certificates

A basic prerequisite for using SSL/TLS security is to have a collection of X.509 certificates available to identify your server applications and, optionally, your client applications. You can generate X.509 certificates in one of the following ways:

- Use a commercial third-party tool to generate and manage your X.509 certificates.
- Use the free `openssl` utility provided with Artix—see [“Creating Your Own Certificates” on page 151](#) for details of how to use it.

HTTPS client with no certificate

[Example 8](#) shows how to configure the client side of a HTTPS connection in Artix, in the case of target-only authentication (no client certificate).

Example 8: *WSDL Contract for HTTPS Client with No Certificate*

```
<definitions name="HelloWorldService"
  targetNamespace="http://xmlbus.com/HelloWorld"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:http-conf="http://schemas.iona.com/transport/http/configuration" ... >
  ...
  <service name="HelloWorldService">
    <port binding="tns:HelloWorldPortBinding"
```

Example 8: WSDL Contract for HTTPS Client with No Certificate

```
1         name="HelloWorldPort">
2         <soap:address location="https://localhost:55012"/>
3         <http-conf:client
4         UseSecureSockets="true"
TrustedRootCertificates="../certificates/openssl/x509/ca/cacert.
pem"
        />
        </port>
    </service>
</definitions>
```

The preceding WSDL contract can be described as follows:

1. The fact that this is a secure connection is signalled here by using `https:` instead of `http:` in the location URL attribute.
2. The `<http-conf:client>` tag contains all the attributes for configuring the client side of the HTTPS connection.
3. If the `UseSecureSockets` attribute is `true`, the client will try to open a secure connection to the server.

Note: If `UseSecureSockets` is `false` and the `<soap:address>` location URL begins with `https:`, however, the client will nevertheless attempt to open a secure connection.

4. The file specified by the `TrustedRootCertificates` contains 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.

Alternatively, instead of setting security attributes in the `<http-conf:client>` tag, you can add security settings to the relevant scope of your Artix domain configuration file, `artix.cfg`, as shown in [Example 9](#) (you still have to set the `<soap:address>` in WSDL).

Example 9: *Alternative Configuration for HTTPS Client with no Certificate*

```
# Artix Domain Configuration
...
SecureClientScope {
    ...
    plugins:http:client:use_secure_sockets = "true";
    plugins:http:client:trusted_root_certificates="../certificates/openssl/x509/ca/cacert.pem"
};
```

HTTPS client with certificate

[Example 10](#) shows how to configure the client side of a HTTPS connection in Artix, in the case of mutual authentication.

Example 10: *WSDL Contract for HTTPS Client with Certificate*

```
<definitions name="HelloWorldService"
    targetNamespace="http://xmlbus.com/HelloWorld"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:http-conf="http://schemas.iona.com/transport/http/configuration" ... >
    ...
    <service name="HelloWorldService">
        <port binding="tns:HelloWorldPortBinding"
            name="HelloWorldPort">
            <soap:address location="https://localhost:55012"/>
            <http-conf:client
                UseSecureSockets="true"
                TrustedRootCertificates="../certificates/openssl/x509/ca/cacert.pem"
                1 ClientCertificate="../certificates/openssl/x509/certs/client_certificate.pem"
                2 ClientPrivateKey="../certificates/openssl/x509/certs/client_private_key.pem"
                3 ClientPrivateKeyPassword="ClientPrivKeyPass"
            />
        </port>
    </service>
</definitions>
```

The preceding WSDL contract can be described as follows:

1. The `ClientCertificate` attribute specifies the client's own certificate in PEM format.
2. The `ClientPrivateKey` attribute 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.
3. The `ClientPrivateKeyPassword` attribute specifies the password to decrypt the contents of the `ClientPrivateKey` file.

Note: The presence of the private key password in the WSDL contract file implies that this file must be read and write-protected to prevent unauthorized users from obtaining the password.

Alternatively, instead of setting security attributes in the `<http-conf:client>` tag, you can add security settings to the relevant scope of your Artix domain configuration file, `artix.cfg`, as shown in [Example 11](#) (you still have to set the `<soap:address>` in WSDL).

Example 11: *Alternative Configuration for HTTPS Client with Certificate*

```
# Artix Domain Configuration
...
SecureClientScope {
    ...
    plugins:http:client:use_secure_sockets = "true";
    plugins:http:client:trusted_root_certificates="../certificates/openssl/x509/ca/cacert.pem"
    plugins:http:client:client_certificate="../certificates/openssl/x509/certs/client_cert.pem"
    plugins:http:client:client_private_key="../certificates/openssl/x509/certs/client_privkey.pem"
    plugins:http:client:client_private_key_password="ClientPrivKeyPas
S"
};
```

HTTPS server configuration

Example 12 shows how to configure the server side of a HTTPS connection in Artix.

Example 12: WSDL Contract with Server HTTPS Configuration

```

<definitions name="HelloWorldService"
  targetNamespace="http://xmlbus.com/HelloWorld"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:http-conf="http://schemas.iona.com/transport/http/configuration" ... >
  ...
  <service name="HelloWorldService">
    <port binding="tns:HelloWorldPortBinding"
      name="HelloWorldPort">
1      <soap:address location="https://localhost:55012"/>
2      <http-conf:server
3        UseSecureSockets="true"
4      ServerCertificate="../certificates/openssl/x509/certs/server_certificate.pem"
5      ServerPrivateKey="../certificates/openssl/x509/certs/server_private_key.pem"
6          ServerPrivateKeyPassword="ServerPrivKeyPass"
7      TrustedRootCertificates="../certificates/openssl/x509/ca/cacert.pem"
        />
    </port>
  </service>
</definitions>

```

The preceding WSDL contract can be described as follows:

1. The fact that this is a secure connection is signalled by using `https:` instead of `http:` in the location URL attribute.
2. The `<http-conf:server>` tag contains all the attributes for configuring the server side of the HTTPS connection.
3. If the `UseSecureSockets` attribute is `true`, the server will open a port to listen for secure connections.

Note: If `UseSecureSockets` is `false` and the `<soap:address>` location URL begins with `https:`, however, the server will listen for secure connections.

4. The `ServerCertificate` attribute specifies the server's own certificate in PEM format. For more background details about X.509 certificates, see ["Managing Certificates" on page 141](#).
5. The `ServerPrivateKey` attribute specifies a PEM file containing the server certificate's encrypted private key.
6. The `ServerPrivateKeyPassword` attribute specifies the password to decrypt the server certificate's private key.

Note: The presence of the private key password in the WSDL contract file implies that this file must be read and write-protected to prevent unauthorized users from obtaining the password.

For the same reason, it is also advisable to remove the `<http-conf:server>` tag from the copy of the WSDL contract that is distributed to clients.

7. The file specified by the `TrustedRootCertificates` contains a concatenated list of CA certificates in PEM format. This attribute value is not used in the case of target-only authentication.

Alternatively, instead of setting server security attributes in the `<http-conf:server>` tag, you can add security settings to the relevant scope of your Artix domain configuration file, `artix.cfg`, as shown in [Example 11](#) (you still have to set the `<soap:address>` in WSDL).

Example 13: *Alternative Configuration for HTTPS Server*

```
# Artix Domain Configuration
...
SecureServerScope {
    ...
    plugins:http:server:use_secure_sockets = "true";
    plugins:http:server:trusted_root_certificates="./certificates/openssl/x509/ca/cacert.pem"
    plugins:http:server:server_certificate="./certificates/openssl/x509/certs/server_cert.pem"
    plugins:http:server:server_private_key="./certificates/openssl/x509/certs/server_privkey.pem"
    plugins:http:server:server_private_key_password="ServerPrivKeyPass"
}
};
```

HTTP Basic Authentication

Overview

This section describes how to configure an Artix client and server to use HTTP Basic Authentication. With HTTP Basic Authentication, username/password credentials are sent in a HTTP header.

For more details, see the [W3 specification](http://www.w3.org/Protocols/HTTP/1.0/spec.html) (<http://www.w3.org/Protocols/HTTP/1.0/spec.html>) for HTTP/1.0.

HTTP Basic Authentication client configuration

Example 14 shows how to configure a client WSDL contract to use HTTP Basic Authentication.

Example 14: WSDL Contract with Client HTTP Basic Authentication

```

<definitions name="HelloWorldService"
  targetNamespace="http://xmlbus.com/HelloWorld"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:http-conf="http://schemas.iona.com/transport/http/configu
1   ration"
  xmlns:bus-security="http://schemas.iona.com/bus/security"
  ... >
  ...
  <service name="HelloWorldService">
    <port binding="tns:HelloWorldPortBinding"
2     name="HelloWorldPort">
      <soap:address location="https://localhost:55012"/>
      <http-conf:client
        UseSecureSockets="true"
        TrustedRootCertificates="../certificates/openssl/x509/ca/cacert.
3         pem"
4         Username="user_test"
         Password="user_password"
      />
    </port>
  </service>
</definitions>

```

The preceding WSDL contract can be described as follows:

1. The `bus-security` namespace prefix must be defined here, because this prefix is used to identify the artix security interceptor in the server's domain configuration (see [“HTTP Basic Authentication server configuration” on page 46](#)).
2. In this example, HTTP Basic Authentication is combined with SSL/TLS security. This ensures that the username and password are transmitted across an encrypted connection, protecting them from snooping.
3. The `UserName` attribute sets the user name for the HTTP Basic Authentication credentials.
4. The `Password` attribute sets the password for the HTTP Basic Authentication credentials.

HTTP Basic Authentication server configuration

There is no need to make any modifications to the WSDL contract for servers that support HTTP Basic Authentication.

However, it is necessary to make modifications to the domain configuration file, `artix.cfg` (in the `ArtixInstallDir/artix/Version/etc/domains` directory), as shown in [Example 15](#).

Example 15: Artix Configuration for Server HTTP Basic Authentication

```
# Artix Configuration File
security_artix {
    ...
    demos
    {
        hello_world
        {
            plugins:artix_security:shlib_name="it_security_plugin";
1         binding:artix:server_request_interceptor_list=
            "bus-security:security";
            binding:client_binding_list = ["OTS+POA_Coloc",
2         "POA_Coloc", "OTS+GIOP+IIOP", "GIOP+IIOP", "GIOP+IIOP_TLS"];
            orb_plugins = ["xmlfile_log_stream", "iiop_profile",
3         "giop", "iiop_tls", "soap", "http", "tunnel", "mq", "ws_orb",
            "fixed", "artix_security"];
            plugins:is2_authorization:action_role_mapping =
4         "file://ArtixInstallDir/artix/2.0/demos/secure_hello_world/http_
5         soap/config/helloworld_action_role_mapping.xml";
            policies:asp:enable_authorization = "true";
            plugins:asp:security_type = "USERNAME_PASSWORD";
```

Example 15: *Artix Configuration for Server HTTP Basic Authentication*

```

6      plugins:asp:security_level = "MESSAGE_LEVEL";
      plugins:asp:authentication_cache_size = "5";
      plugins:asp:authentication_cache_timeout = "10";
      };
      ...
    };
  };
};

```

The preceding extract from the domain configuration can be explained as follows:

1. The Artix server request interceptor list must include the `bus-security:security` interceptor, which provides part of the functionality for the Artix security layer.

Note: The `bus-security` namespace prefix must be defined in the application WSDL contract—see [“HTTP Basic Authentication client configuration” on page 45](#).

2. The `orb_plugins` list should include the `artix_security` plug-in, which is responsible for enabling authentication and authorization.
3. The action-role mapping file is used to apply access control rules to the authenticated user. The file determines which actions (that is, WSDL operations) can be invoked by an authenticated user, on the basis of the roles assigned to that user.
See [“Managing Access Control Lists” on page 133](#) for more details.
4. The `policies:asp:enable_authorization` variable must be set to `true` to enable authorization.
5. The next two configuration variables, `plugins:asp:security_type` and `plugins:asp:security_level`, are used together to specify the type of credentials authenticated on the server side. The particular combination of settings shown here, `USERNAME_PASSWORD` and `MESSAGE_LEVEL`, selects the username/password credentials from the HTTP Basic Authentication header.
6. The next pair of configuration variables configure the `asp` caching mechanism. For more details, see [“ASP configuration variables” on page 34](#).

X.509 Certificate-Based Authentication with HTTPS

Overview

This section describes how to enable X.509 certificate authentication for HTTP-compatible bindings, based on a simple two-tier client/server scenario. In this scenario, the iS2 server authenticates the client's certificate and retrieves roles and realms based on the identity of the certificate subject. When certificate-based authentication is enabled, the X.509 certificate is effectively authenticated twice, as follows:

- *SSL/TLS-level authentication*—this authentication step occurs during the SSL/TLS handshake and is governed by the HTTPS configuration settings in the WSDL contract.
- *ASP security-level authentication and authorization*—this authentication step occurs after the SSL/TLS handshake and is performed by the iS2 server working in tandem with the `artix_security` plug-in.

Certificate-based authentication scenario

Figure 10 shows an example of a two-tier system, where authentication of the client's X.509 certificate is integrated with the i2 server.

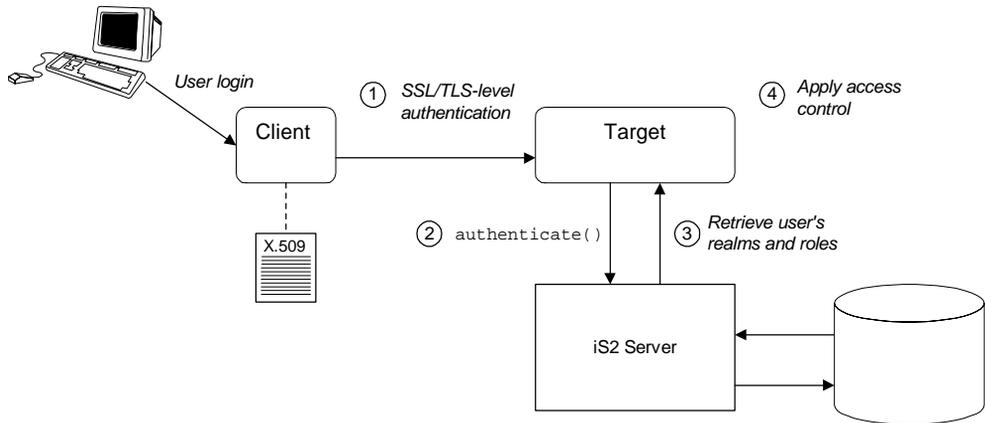


Figure 10: Overview of Certificate-Based Authentication with HTTPS

Scenario description

The scenario shown in Figure 10 can be described as follows:

Stage	Description
1	<p>When the client opens a connection to the server, the client sends its X.509 certificate as part of the SSL/TLS handshake (HTTPS). The server then performs SSL/TLS-level authentication, checking the certificate as follows:</p> <ul style="list-style-type: none"> The certificate is checked against the server's <i>trusted CA list</i> to ensure that it is signed by a trusted certification authority.

Stage	Description
2	<p>The server performs security layer authentication by calling <code>authenticate()</code> on the iS2 server, passing username and password arguments as follows:</p> <ul style="list-style-type: none"> • <i>Username</i>—obtained by extracting the common name (CN) from the client certificate’s subject DN. • <i>Password</i>—obtained from the value of the <code>plugins:asp:default_password</code> configuration variable in the server’s <code>artix.cfg</code> domain configuration. <p>WARNING: This step is <i>not</i> a true authentication step, because the password is cached on the server side. Effectively, this authentication is performed with a dummy password.</p>
3	<p>If authentication is successful, the iS2 server returns the user’s realms and roles.</p>
4	<p>The ASP security layer controls access to the target’s WSDL operations by consulting an <i>action-role mapping file</i> to determine what the user is allowed to do.</p>

HTTPS prerequisites

In general, a basic prerequisite for using X.509 certificate-based authentication is that both client and server are configured to use HTTPS. See [“Securing HTTP Communications with SSL/TLS” on page 39](#).

Certificate-based authentication client configuration

To enable certificate-based authentication on the client side, it is sufficient for the client to be configured to use HTTPS with its own certificate. For example, see [“HTTPS client with certificate” on page 41](#).

Certificate-based authentication server configuration

A prerequisite for using certificate-based authentication on the server side is that the server's WSDL contract is configured to use HTTPS. For example, see ["HTTPS server configuration" on page 43](#).

Additionally, on the server side it is also necessary to configure the ASP security layer by editing the `artix.cfg` domain configuration file (in the `ArtixInstallDir/artix/Version/etc/domains` directory), as shown in EX.

Example 16: Artix Configuration for X.509 Certificate-Based Authentication

```
# Artix Configuration File
security_artix {
    ...
    demos
    {
        hello_world
        {
            plugins:artix_security:shlib_name="it_security_plugin";
            binding:artix:server_request_interceptor_list=
1         "bus-security:security";
            binding:client_binding_list = ["OTS+POA_Coloc",
2         "POA_Coloc", "OTS+GIOP+IIOP", "GIOP+IIOP", "GIOP+IIOP_TLS"];
            orb_plugins = ["xmlfile_log_stream", "iiop_profile",
3         "giop", "iiop_tls", "soap", "http", "tunnel", "mq", "ws_orb",
4         "fixed", "artix_security"];
            plugins:is2_authorization:action_role_mapping =
5         "file://ArtixInstallDir/artix/2.0/demos/secure_hello_world/http_
6         soap/config/helloworld_action_role_mapping.xml";
7         policies:asp:enable_authorization = "true";
            plugins:asp:security_type = "CERT_SUBJECT";
            plugins:asp:security_level = "MESSAGE_LEVEL";
            plugins:asp:default_password = "CertPassword";
            plugins:asp:authentication_cache_size = "5";
            plugins:asp:authentication_cache_timeout = "10";
        }
    }
    ...
};
```

The preceding extract from the domain configuration can be explained as follows:

1. The Artix server request interceptor list must include the `bus-security:security` interceptor, which provides part of the functionality for the Artix security layer.

Note: The `bus-security` namespace prefix must be defined in the application WSDL contract—see [“HTTP Basic Authentication client configuration” on page 45](#).

2. The `orb_plugins` list should include the `artix_security` plug-in, which is responsible for enabling authentication and authorization.
3. The action-role mapping file is used to apply access control rules to the authenticated user. The file determines which actions (that is, WSDL operations) can be invoked by an authenticated user, on the basis of the roles assigned to that user.
See [“Managing Access Control Lists” on page 133](#) for more details.
4. `policies:asp:enable_authorization` variable must be set to `true` to enable authorization.
5. The next two configuration variables, `plugins:asp:security_type` and `plugins:asp:security_level`, are used together to specify the type of credentials authenticated on the server side. The particular combination of settings shown here, `CERT_SUBJECT` and `MESSAGE_LEVEL`, selects X.509 certificate-based authentication.
In this case, the username is taken to be the common name (CN) from the client certificate’s subject DN (for an explanation of X.509 certificate terminology, see [“ASN.1 and Distinguished Names” on page 283](#)).
6. When certificate-based authentication is used with HTTPS, a default password, *CertPassword*, must be supplied on the server side. This password is then used in the `authenticate()` call to the iS2 server.
7. The next pair of configuration variables configure the `asp` caching mechanism. For more details, see [“ASP configuration variables” on page 34](#).

Security for SOAP Bindings

This chapter describes the security features that are specific to the SOAP binding—for example, such as security credentials that can be propagated in a SOAP header.

In this chapter

This chapter discusses the following topic:

Overview of SOAP Security

page 54

Overview of SOAP Security

Overview

Figure 11 gives an overview of security for a SOAP binding within the Artix security framework. SOAP security consists of four different layers (SSL/TLS, HTTP, SOAP, and ASP) and support is provided for several different types of authentication. Figure 11 shows you how the different authentication types are associated with the different security layers.

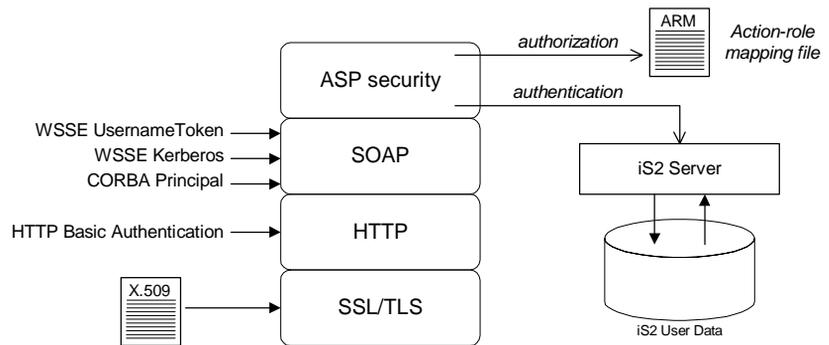


Figure 11: Overview of Security for SOAP Bindings

Security layers

As shown in Figure 11, the SOAP binding includes the following security layers:

- SSL/TLS layer.
- HTTP layer.
- SOAP layer.
- ASP security layer.

SSL/TLS layer

The SSL/TLS layer provides the SOAP binding with message encryption, message integrity and authentication using X.509 certificates. The implementation of SSL/TLS that underlies HTTPS is based on the [OpenSSL](http://www.openssl.org) (www.openssl.org) security toolkit.

To enable SSL/TLS for HTTP, you must edit the WSDL contract—see [“Securing HTTP Communications with SSL/TLS” on page 39](#).

HTTP layer

The HTTP layer provides a means of sending username/password credentials in a HTTP header (HTTP Basic Authentication). The HTTP layer relies on SSL/TLS to prevent password snooping.

SOAP layer

The SOAP layer can send various credentials (WSSE UsernameToken, WSSE Kerberos and CORBA Principal) embedded in a SOAP message header. The SOAP layer relies on SSL/TLS to prevent password snooping.

ASP security layer

The ASP security layer implements a variety of security features for non-CORBA bindings. The main features of the ASP security layer are, as follows:

- *Authentication*—the ASP security layer calls the iS2 server (which maintains a database of user data) to authenticate a user’s credentials. If authentication is successful, the iS2 server returns a list of the user’s roles and realms.
 - *Authorization*—the ASP security layer matches the user’s roles and realms against an action-role mapping file to determine whether the user has permission to invoke the relevant WSDL operation.
-

Authentication options

As shown in [Figure 11 on page 54](#), the SOAP binding supports the following authentication options:

- [WSSE UsernameToken](#).
- [WSSE Kerberos](#).
- [CORBA Principal](#).
- [HTTP Basic Authentication](#).
- [X.509 certificate-based authentication](#).

WSSE UsernameToken

The Web service security extension (WSSE) UsernameToken is a username/password combination that can be sent in a SOAP header. The specification of WSSE UsernameToken is contained in the [WSS UsernameToken Profile 1.0](#) document from [OASIS](#) (www.oasis-open.org). Currently, the WSSE UsernameToken can be set *only* by programming. See [“Propagating a Username/Password Token” on page 230](#).

WSSE Kerberos

The WSSE Kerberos specification is used to send a Kerberos security token in a SOAP header. If you use Kerberos, you must also configure the iS2 server to use the Kerberos adapter—see [“Configuring the Kerberos Adapter” on page 110](#).

Currently, the WSSE Kerberos token can be set *only* by programming. See [“Propagating a Kerberos Token” on page 232](#).

CORBA Principal

The CORBA Principal is a legacy feature originally defined in the early versions of the CORBA GIOP specification. To facilitate interoperability with early CORBA implementations, the Artix SOAP binding is also able to propagate CORBA Principals. This feature is available only for SOAP over HTTP and a SOAP header is used to propagate the CORBA Principal.

For details, see [“Principal Propagation” on page 215](#).

HTTP Basic Authentication

HTTP Basic Authentication is used to propagate username/password credentials in a HTTP header. This kind of authentication is available to any HTTP-compatible binding.

For details, see [“HTTP Basic Authentication” on page 45](#).

X.509 certificate-based authentication

X.509 certificate-based authentication obtains credentials by extracting the common name (CN) from a client certificate’s subject DN. This kind of authentication is available to any HTTP-compatible binding.

For details, see [“X.509 Certificate-Based Authentication with HTTPS” on page 48](#).

Security for CORBA Bindings

Using IONA's modular ART technology, you make a CORBA binding secure by configuring it to load the relevant security plug-ins. This section describes how to load and configure security plug-ins to reach the appropriate level of security for applications with a CORBA binding.

In this chapter

This chapter discusses the following topics:

Overview of CORBA Security	page 58
Securing IIOP Communications with SSL/TLS	page 60
Securing Two-Tier CORBA Systems with CSI	page 66
Securing Three-Tier CORBA Systems with CSI	page 72
X.509 Certificate-Based Authentication for CORBA Bindings	page 78

Overview of CORBA Security

Overview

There are three layers of security available for CORBA bindings: IIOP over SSL/TLS (IIOP/TLS), which provides secure communication between client and server; CSI, which provides a mechanism for propagating username/password credentials; and the GSP plug-in, which is concerned with higher-level security features such as authentication and authorization.

The following combinations are recommended:

- IIOP/TLS only—for a pure SSL/TLS security solution.
- IIOP/TLS, CSI, and GSP layers—for a highly scalable security solution, based on username/password client authentication.

CORBA applications and the Artix security framework

Figure 12 shows the main features of a secure CORBA application in the context of the Artix security framework.

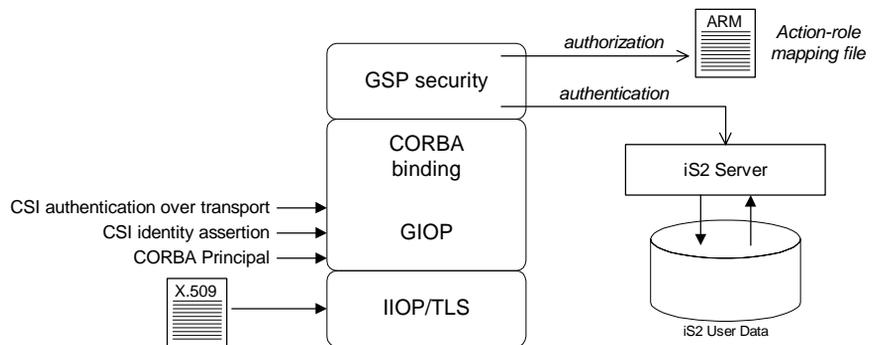


Figure 12: A Secure CORBA Application within the Artix Security Framework

Security plug-ins

Within the Artix security framework, a CORBA application becomes fully secure by loading the following plug-ins:

- [IIOP/TLS plug-in](#)
- [CSlv2 plug-in](#)
- [GSP plug-in](#)

IIOP/TLS plug-in

The IIOP/TLS plug-in, `iiop_tls`, enables a CORBA application to transmit and receive IIOP requests over a secure SSL/TLS connection. This plug-in can be enabled independently of the other two plug-ins.

See [“Securing IIOP Communications with SSL/TLS” on page 60](#) for details on how to enable IIOP/TLS in a CORBA application.

CSlv2 plug-in

The CSlv2 plug-in, `csi`, provides a client authentication mechanism for CORBA applications. The authentication mechanism is based on a username and a password. When the CSlv2 plug-in is configured for use with the Artix security framework, the username and password are forwarded to a central iS2 server to be authenticated. This plug-in is needed to support the Artix security framework.

Note: The IIOP/TLS plug-in also provides a client authentication mechanism (based on SSL/TLS and X.509 certificates). The SSL/TLS and CSlv2 authentication mechanisms are independent of each other and can be used simultaneously.

GSP plug-in

The GSP plug-in, `gsp`, provides authorization by checking a user’s roles against the permissions stored in an action-role mapping file. This plug-in is needed to support the Artix security framework.

Securing IIOP Communications with SSL/TLS

Overview

This section describes how to configure a CORBA binding to use SSL/TLS security. In this section, it is assumed that your initial configuration comes from a secure location domain (that is, the `artix.cfg` domain configuration file has been modified to include `artix-secure.cfg`).

WARNING: The default certificates used in the CORBA configuration samples are for demonstration purposes only and are completely insecure. You must generate your own custom certificates for use in your own CORBA applications.

Sample client configuration

For example, consider a secure SSL/TLS client whose configuration is modelled on the `demos.tls.secure_client_with_no_cert` configuration. [Example 17](#) shows how to configure such a sample client.

Example 17: Sample SSL/TLS Client Configuration

```
# Artix Configuration File
...
# General configuration at root scope.
...
my_secure_apps {
  # Common SSL/TLS configuration settings.
1  orb_plugins = ["local_log_stream", "iiop_profile", "giop",
  "iiop_tls"];
2
  binding:client_binding_list = ["GIOP+EGMIOP",
  "OTS+TLS_Coloc+POA_Coloc", "TLS_Coloc+POA_Coloc",
  "OTS+POA_Coloc", "POA_Coloc", "GIOP+SHMIOP",
  "CSI+OTS+GIOP+IIOP_TLS", "OTS+GIOP+IIOP_TLS",
  "CSI+GIOP+IIOP_TLS", "GIOP+IIOP_TLS", "CSI+OTS+GIOP+IIOP",
  "OTS+GIOP+IIOP", "CSI+GIOP+IIOP", "GIOP+IIOP"];
3
  policies:trusted_ca_list_policy =
  "ArtixInstallDir\artix\Version\demos\secure_hello_world\http_soap\certificates\tls\x509\trusted_ca_lists\ca_list1.pem";
4
  policies:mechanism_policy:protocol_version = "SSL_V3";
```

Example 17: Sample SSL/TLS Client Configuration

```

policies:mechanism_policy:ciphersuites =
["RSA_WITH_RC4_128_SHA", "RSA_WITH_RC4_128_MD5"];
5
event_log:filters = ["IT_ATLI_TLS=*", "IT_IIOp=*",
"IT_IIOp_TLS=*", "IT_TLS=*"];
...
my_client {
6
    # Specific SSL/TLS client configuration settings
    principal_sponsor:use_principal_sponsor = "false";
7
    policies:client_secure_invocation_policy:requires =
["Confidentiality", "EstablishTrustInTarget"];
    policies:client_secure_invocation_policy:supports =
["Confidentiality", "Integrity", "DetectReplay",
"DetectMisordering", "EstablishTrustInTarget"];
    };
};
...

```

The preceding client configuration can be described as follows:

1. Make sure that the `orb_plugins` variable in this configuration scope includes the `iiop_tls` plug-in.

Note: For fully secure applications, you should *exclude* the `iiop` plug-in (insecure IIOp) from the ORB plug-ins list. This renders the application incapable of making insecure IIOp connections.

For semi-secure applications, however, you should *include* the `iiop` plug-in before the `iiop_tls` plug-in in the ORB plug-ins list.

If you plan to use the full Artix Security Framework, you should include the `gsp` plug-in in the ORB plug-ins list as well—see [“Securing Two-Tier CORBA Systems with CSI” on page 66](#).

2. Make sure that the `binding:client_binding_list` variable includes bindings with the `IIOp_TLS` interceptor. You can use the value of the `binding:client_binding_list` shown here.
3. An SSL/TLS application needs a list of trusted CA certificates, which it uses to determine whether or not to trust certificates received from other SSL/TLS applications. You must, therefore, edit the

`policies:trusted_ca_list_policy` variable to point at a list of trusted certificate authority (CA) certificates. See [“Specifying Trusted CA Certificates”](#) on page 178.

Note: If using Schannel as the underlying SSL/TLS toolkit (Windows only), the `policies:trusted_ca_list_policy` variable is ignored. Within Schannel, the trusted root CA certificates are obtained from the Windows certificate store.

4. The SSL/TLS mechanism policy specifies the default security protocol version and the available cipher suites—see [“Specifying IIOPTLS Cipher Suites”](#) on page 204.
5. This line enables console logging for security-related events, which is useful for debugging and testing. Because there is a performance penalty associated with this option, you might want to comment out or delete this line in a production system.
6. The SSL/TLS principal sponsor is a mechanism that can be used to specify an application’s own X.509 certificate. Because this client configuration does not use a certificate, the principal sponsor is disabled by setting `principal_sponsor:use_principal_sponsor` to `false`.
7. The following two lines set the *required* options and the *supported* options for the client secure invocation policy. In this example, the policy is set as follows:
 - ◆ Required options—the options shown here ensure that the client can open only secure SSL/TLS connections.
 - ◆ Supported options—the options shown include all of the association options, except for the `EstablishTrustInClient` option. The client cannot support `EstablishTrustInClient`, because it has no X.509 certificate.

Sample server configuration

Generally speaking, it is rarely necessary to configure such a thing as a *pure server* (that is, a server that never makes any requests of its own). Most real servers are applications that act in both a server role and a client role. Hence, the sample server described here is a hybrid of the following two demonstration configurations:

- demos.tls.secure_server_request_client_auth
- demos.tls.secure_client_with_cert

Example 18 shows how to configure such a sample server.

Example 18: Sample SSL/TLS Server Configuration

```

# Artix Configuration File
...
# General configuration at root scope.
...
my_secure_apps {
1   # Common SSL/TLS configuration settings.
    ...
    my_server {
2       # Specific SSL/TLS server configuration settings
        policies:target_secure_invocation_policy:requires =
["Confidentiality"];
        policies:target_secure_invocation_policy:supports =
["EstablishTrustInClient", "Confidentiality", "Integrity",
"DetectReplay", "DetectMisordering",
"EstablishTrustInTarget"];
3
        principal_sponsor:use_principal_sponsor = "true";
4        principal_sponsor:auth_method_id = "pkcs12_file";
5        principal_sponsor:auth_method_data =
["filename=CertsDir\server_cert.p12"];

        # Specific SSL/TLS client configuration settings
6        policies:client_secure_invocation_policy:requires =
["Confidentiality", "EstablishTrustInTarget"];
        policies:client_secure_invocation_policy:supports =
["Confidentiality", "Integrity", "DetectReplay",
"DetectMisordering", "EstablishTrustInClient",
"EstablishTrustInTarget"];
    };
};
...

```

The preceding server configuration can be described as follows:

1. You can use the same common SSL/TLS settings here as described in the preceding [“Sample client configuration” on page 60](#)
2. The following two lines set the *required* options and the *supported* options for the target secure invocation policy. In this example, the policy is set as follows:

- ◆ Required options—the options shown here ensure that the server accepts only secure SSL/TLS connection attempts.
 - ◆ Supported options—all of the target association options are supported.
3. A server must always be associated with an X.509 certificate. Hence, this line enables the SSL/TLS principal sponsor, which specifies a certificate for the application.
 4. This line specifies that the X.509 certificate is contained in a PKCS#12 file. For alternative methods, see [“Specifying an Application’s Own Certificate” on page 179](#).

Note: If using Schannel as the underlying SSL/TLS toolkit (Windows only), the `principal_sponsor:auth_method_id` value must be `security_label` instead of `pkcs12_file`.

5. Replace the X.509 certificate, by editing the `filename` option in the `principal_sponsor:auth_method_data` configuration variable to point at a custom X.509 certificate. The `filename` value should be initialized with the location of a certificate file in PKCS#12 format—see [“Specifying an Application’s Own Certificate” on page 179](#) for more details.

Note: If using Schannel as the underlying SSL/TLS toolkit (Windows only), you would set the `label` option instead of the `filename` option in the `principal_sponsor:auth_method_data` configuration variable. The `label` specifies the common name (CN) from the application certificate’s subject DN.

For details of how to specify the certificate’s pass phrase, see [“Certificate Pass Phrase for IIOP/TLS” on page 183](#).

6. The following two lines set the *required* options and the *supported* options for the client secure invocation policy. In this example, the policy is set as follows:
 - ◆ Required options—the options shown here ensure that the application can open only secure SSL/TLS connections to other servers.

- ◆ Supported options—all of the client association options are supported. In particular, the `EstablishTrustInClient` option is supported when the application is in a client role, because the application has an X.509 certificate.

Mixed security configurations

Most realistic secure server configurations are mixed in the sense that they include both server settings (for the server role), and client settings (for the client role). When combining server and client security settings for an application, you must ensure that the settings are consistent with each other.

For example, consider the case where the server settings are secure and the client settings are insecure. To configure this case, set up the server role as described in [“Sample server configuration” on page 62](#). Then configure the client role by adding (or modifying) the following lines to the `my_secure_apps.my_server` configuration scope:

```
orb_plugins = ["local_log_stream", "iiop_profile", "giop",
              "iiop", "iiop_tls"];
policies:client_secure_invocation_policy:requires =
  ["NoProtection"];
policies:client_secure_invocation_policy:supports =
  ["NoProtection"];
```

The first line sets the ORB plug-ins list to make sure that the `iiop` plug-in (enabling insecure IIOp) is included. The `NoProtection` association option, which appears in the required and supported client secure invocation policy, effectively disables security for the client role.

Customizing SSL/TLS security policies

You can, optionally, customize the SSL/TLS security policies in various ways. For details, see the following references:

- [“Configuring IIOp/TLS Secure Associations” on page 189](#).
- [“Configuring HTTPS and IIOp/TLS Authentication” on page 169](#).

Securing Two-Tier CORBA Systems with CSI

Overview

This section describes how to secure a two-tier CORBA system using the OMG's Common Secure Interoperability specification version 2.0 (CSIv2). The client supplies username/password authentication data which is transmitted as CSI credentials and then authenticated on the server side. The following configurations are described in detail:

- [Client configuration.](#)
- [Target configuration.](#)

Two-tier CORBA system

Figure 13 shows a basic two-tier CORBA system using CSI credentials, featuring a client and a target server.

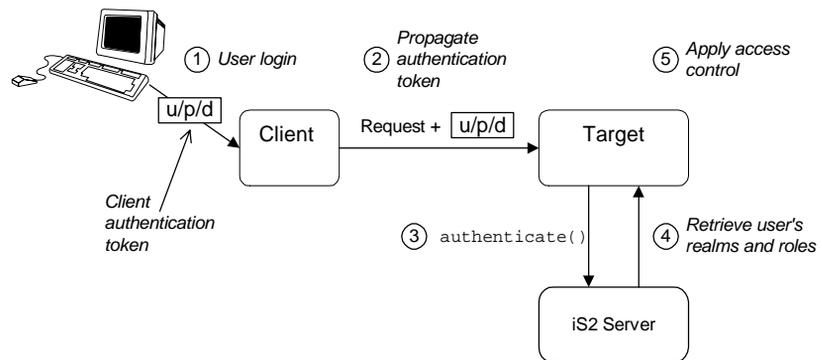


Figure 13: Two-Tier CORBA System Using CSI Credentials

Scenario description

The scenario shown in [Figure 13](#) can be described as follows:

Stage	Description
1	The user enters a username, password, and domain name on the client side (user login). Note: The domain name must match the value of the <code>policies:csi:auth_over_transport:server_domain_name</code> configuration variable set on the server side.
2	When the client makes a remote invocation on the server, the CSI username/password/domain authentication data is transmitted to the target along with the invocation request.
3	The server authenticates the received username and password by calling out to the external iS2 server.
4	If authentication is successful, the iS2 server returns the user's realms and roles.
5	The GSP security layer controls access to the target's IDL interfaces by consulting an <i>action-role mapping file</i> to determine what the user is allowed to do.

Client configuration

The CORBA client from [Example 13 on page 66](#) can be configured as shown in [Example 19](#).

Example 19: Configuration of a CORBA client Using CSI Credentials

```

# Artix Configuration File
...
# General configuration at root scope.
...
1 my_secure_apps {
    # Common SSL/TLS configuration settings.
    ...
    # Common Artix security framework configuration settings.
2 orb_plugins = ["local_log_stream", "iiop_profile", "giop",
    "iiop_tls", "ots", "gsp"];

```

Example 19: Configuration of a CORBA client Using CSI Credentials

```

3 binding:client_binding_list = ["GIOP+EGMIOP",
    "OTS+TLS_Coloc+POA_Coloc", "TLS_Coloc+POA_Coloc",
    "OTS+POA_Coloc", "POA_Coloc", "GIOP+SHMIOP",
    "CSI+OTS+GIOP+IIOP_TLS", "OTS+GIOP+IIOP_TLS",
    "CSI+GIOP+IIOP_TLS", "GIOP+IIOP_TLS", "CSI+OTS+GIOP+IIOP",
    "OTS+GIOP+IIOP", "CSI+GIOP+IIOP", "GIOP+IIOP"];
4 binding:server_binding_list = ["CSI+GSP+OTS", "CSI+GSP",
    "CSI+OTS", "CSI"];
    ...
5 my_client {
    # Specific SSL/TLS configuration settings.
    ...
    # Specific Artix security framework settings.
6 policies:csi:auth_over_transport:client_supports =
    ["EstablishTrustInClient"];
7
    principal_sponsor:csi:use_principal_sponsor = "true";
    principal_sponsor:csi:auth_method_id = "GSSUPMech";
    principal_sponsor:csi:auth_method_data = [];
    };
};
...

```

The preceding client configuration can be explained as follows:

1. The SSL/TLS configuration variables common to all of your applications can be placed here—see [“Securing IIOP Communications with SSL/TLS” on page 60](#) for details of the SSL/TLS configuration.
2. Make sure that the `orb_plugins` variable in this configuration scope includes both the `iiop_tls` and the `gsp` plug-ins in the order shown.
3. Make sure that the `binding:client_binding_list` variable includes bindings with the `CSI` interceptor. You can use the value of the `binding:client_binding_list` shown here.
4. Make sure that the `binding:server_binding_list` variable includes bindings with both the `CSI` and `GSP` interceptors. You can use the value of the `binding:server_binding_list` shown here.
5. The SSL/TLS configuration variables specific to the CORBA client can be placed here—see [“Securing IIOP Communications with SSL/TLS” on page 60](#).

6. This configuration setting specifies that the client supports sending username/password authentication data to a server.
7. The next three lines specify that the client uses the CSI principal sponsor to obtain the user's authentication data. With the configuration as shown, the user would be prompted to enter the username and password when the client application starts up.

Target configuration

The CORBA target server from [Figure 13 on page 66](#) can be configured as shown in [Example 20](#).

Example 20: *Configuration of a Second-Tier Target Server in the Artix security framework*

```

# Artix Configuration File
...
# General configuration at root scope.
...
my_secure_apps {
    # Common SSL/TLS configuration settings.
    ...
    # Common Artix security framework configuration settings.
    orb_plugins = [ ..., "iiop_tls", "gsp", ... ];
    binding:client_binding_list = [ ... ];
    binding:server_binding_list = [ ... ];
    ...
    my_two_tier_target {
1      # Specific SSL/TLS configuration settings.
        ...
        # Specific Artix security framework settings.
2      policies:csi:auth_over_transport:target_supports =
["EstablishTrustInClient"];
3      policies:csi:auth_over_transport:target_requires =
["EstablishTrustInClient"];
4      policies:csi:auth_over_transport:server_domain_name =
"CSIDomainName";

5      plugins:gsp:authorization_realm = "AuthzRealm";
6      plugins:is2_authorization:action_role_mapping =
"ActionRoleURL";

```

Example 20: Configuration of a Second-Tier Target Server in the Artix security framework

```

7      # Artix security framework client configuration settings.
      policies:csi:auth_over_transport:client_supports =
        [ "EstablishTrustInClient" ];

      principal_sponsor:csi:use_principal_sponsor = "true";
      principal_sponsor:csi:auth_method_id = "GSSUPMech";
      principal_sponsor:csi:auth_method_data = [];
    };
  };
};

```

The preceding target server configuration can be explained as follows:

1. The SSL/TLS configuration variables specific to the CORBA target server can be placed here—see [“Securing IIOP Communications with SSL/TLS” on page 60](#).
2. This configuration setting specifies that the target server *supports* receiving username/password authentication data from the client.
3. This configuration setting specifies that the target server *requires* the client to send username/password authentication data.
4. The `server_domain_name` configuration variable sets the server’s CSIV2 authentication domain name, *CSIDomainName*. The domain name embedded in a received CSIV2 credential must match the value of the `server_domain_name` variable on the server side.
5. This configuration setting specifies the Artix authorization realm, *AuthzRealm*, to which this server belongs. For more details about Artix authorization realms, see [“Artix Authorization Realms” on page 123](#).
6. The `action_role_mapping` configuration variable specifies the location of an action-role mapping that controls access to the IDL interfaces implemented by the server. The file location is specified in an URL format, for example:

```

file:///security_admin/action_role_mapping.xml (UNIX) or
file:///c:/security_admin/action_role_mapping.xml (Windows).

```

For more details about the action-role mapping file, see [“Artix Action-Role Mapping ACL” on page 135](#).

7. You should also set secure client configuration variables in the server configuration scope, because a secure server application usually behaves as a secure client of the core CORBA services. For example, almost all CORBA servers need to contact both the locator service and the CORBA naming service.
-

Related administration tasks

After securing your CORBA applications with the Artix security framework, you might need to perform related administration tasks, for example:

- See [“Managing Users, Roles and Domains” on page 119](#).
- See [“Artix Action-Role Mapping ACL” on page 135](#).

Securing Three-Tier CORBA Systems with CSI

Overview

This section describes how to secure a three-tier CORBA system using CSIv2. In this scenario there is a client, an intermediate server, and a target server. The intermediate server is configured to propagate the client identity when it invokes on the target server in the third tier. The following configurations are described in detail:

- [Intermediate configuration.](#)
- [Target configuration.](#)

Three-tier CORBA system

Figure 14 shows a basic three-tier CORBA system using CSIv2, featuring a client, an intermediate server and a target server.

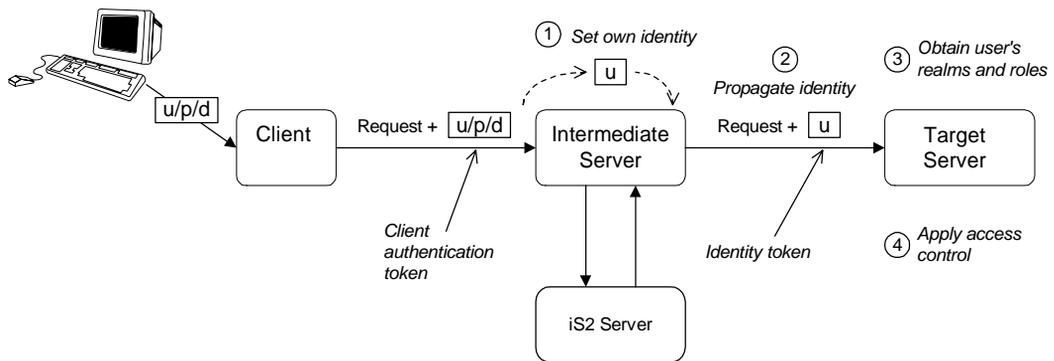


Figure 14: Three-Tier CORBA System Using CSIv2

Scenario description

The second stage of the scenario shown in [Figure 14](#) (intermediate server invokes an operation on the target server) can be described as follows:

Stage	Description
1	The intermediate server sets its own identity by extracting the user identity from the received username/password CSI credentials. Hence, the intermediate server assumes the same identity as the client.
2	When the intermediate server makes a remote invocation on the target server, CSI identity assertion is used to transmit the user identity data to the target.
3	The target server then obtains the user's realms and roles.
4	The GSP security layer controls access to the target's IDL interfaces by consulting an <i>action-role mapping file</i> to determine what the user is allowed to do.

Client configuration

The client configuration for the three-tier scenario is identical to that of the two-tier scenario, as shown in ["Client configuration" on page 67](#).

Intermediate configuration

The CORBA intermediate server from [Figure 14 on page 72](#) can be configured as shown in [Example 21](#).

Example 21: *Configuration of a Second-Tier Intermediate Server in the Artix security framework*

```
# Artix Configuration File
...
# General configuration at root scope.
...
my_secure_apps {
    # Common SSL/TLS configuration settings.
    ...
    # Common Artix security framework configuration settings.
    orb_plugins = [ ..., "iiop_tls", "gsp", ... ];
    binding:client_binding_list = [ ... ];
    binding:server_binding_list = [ ... ];
    ...
}
```

Example 21: Configuration of a Second-Tier Intermediate Server in the Artix security framework

```

1   my_three_tier_intermediate {
      # Specific SSL/TLS configuration settings.
      ...
      # Specific Artix security framework settings.
2   policies:csi:attribute_service:client_supports =
      ["IdentityAssertion"];

3   policies:csi:auth_over_transport:target_supports =
      ["EstablishTrustInClient"];
4   policies:csi:auth_over_transport:target_requires =
      ["EstablishTrustInClient"];
5   policies:csi:auth_over_transport:server_domain_name =
      "CSIDomainName";

6   plugins:gsp:authorization_realm = "AuthzRealm";
7   plugins:is2_authorization:action_role_mapping =
      "ActionRoleURL";

8   # Artix security framework client configuration settings.
      policies:csi:auth_over_transport:client_supports =
      ["EstablishTrustInClient"];

      principal_sponsor:csi:use_principal_sponsor = "true";
      principal_sponsor:csi:auth_method_id = "GSSUPMech";
      principal_sponsor:csi:auth_method_data = [];
    };
};

```

The preceding intermediate server configuration can be explained as follows:

1. The SSL/TLS configuration variables specific to the CORBA intermediate server can be placed here—see [“Securing IIOP Communications with SSL/TLS” on page 60](#).
2. This configuration setting specifies that the intermediate server is capable of propagating the identity it receives from a client. In other words, the server is able to assume the identity of the client when invoking operations on third-tier servers.
3. This configuration setting specifies that the intermediate server *supports* receiving username/password authentication data from the client.

4. This configuration setting specifies that the intermediate server *requires* the client to send username/password authentication data.
5. The `server_domain_name` configuration variable sets the server's CSIV2 authentication domain name, *CSIDomainName*. The domain name embedded in a received CSIV2 credential must match the value of the `server_domain_name` variable on the server side.
6. This configuration setting specifies the Artix authorization realm, *AuthzRealm*, to which this server belongs. For more details about Artix authorization realms, see [“Artix Authorization Realms” on page 123](#).
7. This configuration setting specifies the location of an action-role mapping that controls access to the IDL interfaces implemented by the server. The file location is specified in an URL format, for example:
`file:///security_admin/action_role_mapping.xml` (UNIX) or
`file:///c:/security_admin/action_role_mapping.xml` (Windows).
 For more details about the action-role mapping file, see [“Artix Action-Role Mapping ACL” on page 135](#).
8. You should also set Artix security framework client configuration variables in the intermediate server configuration scope, because a secure server application usually behaves as a secure client of the core CORBA services. For example, almost all CORBA servers need to contact both the locator service and the CORBA naming service.

Target configuration

The CORBA target server from [Figure 14 on page 72](#) can be configured as shown in [Example 22](#).

Example 22: Configuration of a Third-Tier Target Server Using CSI

```
# Artix Configuration File
...
# General configuration at root scope.
...
my_secure_apps {
    # Common SSL/TLS configuration settings.
    ...
    # Common Artix security framework configuration settings.
    orb_plugins = [ ..., "iop_tls", "gsp", ... ];
    binding:client_binding_list = [ ... ];
    binding:server_binding_list = [ ... ];
    ...
}
```

Example 22: Configuration of a Third-Tier Target Server Using CSI

```

my_three_tier_target {
    # Specific SSL/TLS configuration settings.
1     ...
2     policies:iiop_tls:target_secure_invocation_policy:requires
= ["Confidentiality", "DetectMisordering", "DetectReplay",
"Integrity", "EstablishTrustInClient"];
3     policies:iiop_tls:certificate_constraints_policy =
["ConstraintString1", "ConstraintString2", ...];

    # Specific Artix security framework settings.
4     policies:csi:attribute_service:target_supports =
["IdentityAssertion"];

    plugins:gsp:authorization_realm = "AuthzRealm";
5     plugins:is2_authorization:action_role_mapping =
6     "ActionRoleURL";

7     # Artix security framework client configuration settings.
    policies:csi:auth_over_transport:client_supports =
["EstablishTrustInClient"];

    principal_sponsor:csi:use_principal_sponsor = "true";
    principal_sponsor:csi:auth_method_id = "GSSUPMech";
    principal_sponsor:csi:auth_method_data = [];
};
};

```

The preceding target server configuration can be explained as follows:

1. The SSL/TLS configuration variables specific to the CORBA target server can be placed here—see [“Securing IIOP Communications with SSL/TLS” on page 60](#).
2. It is recommended that the target server require its *clients* to authenticate themselves using an X.509 certificate. For example, the intermediate server (acting as a client of the target) would then be required to send an X.509 certificate to the target during the SSL/TLS handshake.

You can specify this option by including the `EstablishTrustInClient` association option in the target secure invocation policy, as shown here (thereby overriding the policy value set in the outer configuration scope).

3. In addition to the preceding step, it is also advisable to restrict access to the target server by setting a certificate constraints policy, which allows access only to those clients whose X.509 certificates match one of the specified constraints—see [“Applying Constraints to Certificates” on page 187](#).

Note: The motivation for limiting access to the target server is that clients of the target server obtain a special type of privilege: propagated identities are granted access to the target server without the target server performing authentication on the propagated identities. Hence, the target server trusts the intermediate server to do the authentication on its behalf.

4. This configuration setting specifies that the target server supports receiving propagated user identities from the client.
5. This configuration setting specifies the Artix authorization realm, *AuthzRealm*, to which this server belongs. For more details about Artix authorization realms, see [“Artix Authorization Realms” on page 123](#).
6. This configuration setting specifies the location of an action-role mapping that controls access to the IDL interfaces implemented by the server. The file location is specified in an URL format, for example:
`file:///security_admin/action_role_mapping.xml`
 For more details about the action-role mapping file, see [“Artix Action-Role Mapping ACL” on page 135](#).
7. You should also set secure client configuration variables in the target server configuration scope, because a secure server application usually behaves as a secure client of the core CORBA services. For example, almost all CORBA servers need to contact both the locator service and the CORBA naming service.

Related administration tasks

After securing your CORBA applications with the Artix security framework, you might need to perform related administration tasks, for example:

- See [“Managing Users, Roles and Domains” on page 119](#).
- See [“Artix Action-Role Mapping ACL” on page 135](#).

X.509 Certificate-Based Authentication for CORBA Bindings

Overview

This section describes how to enable X.509 certificate authentication for CORBA bindings, based on a simple two-tier client/server scenario. In this scenario, the iS2 server authenticates the client's certificate and retrieves roles and realms based on the identity of the certificate subject. When certificate-based authentication is enabled, the X.509 certificate is effectively authenticated twice, as follows:

- *SSL/TLS-level authentication*—this authentication step occurs during the SSL/TLS handshake and is governed by Artix configuration settings and programmable SSL/TLS policies.
- *GSP security-level authentication and authorization*—this authentication step occurs after the SSL/TLS handshake and is performed by the iS2 server working in tandem with the `gsp` plug-in.

Certificate-based authentication scenario

Figure 15 shows an example of a two-tier system, where authentication of the client's X.509 certificate is integrated with the iS2 server.

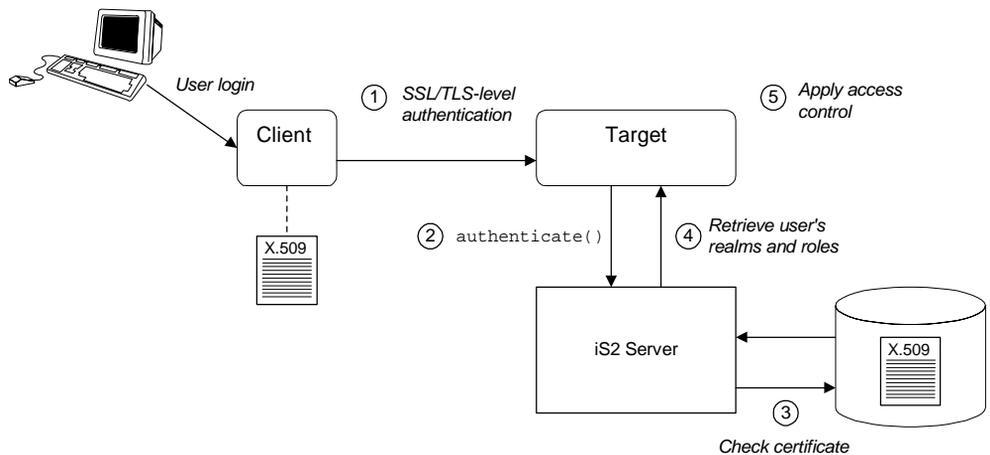


Figure 15: Overview of Certificate-Based Authentication

Scenario description

The scenario shown in Figure 15 can be described as follows:

Stage	Description
1	<p>When the client opens a connection to the server, the client sends its X.509 certificate as part of the SSL/TLS handshake. The server then performs SSL/TLS-level authentication, checking the certificate as follows:</p> <ul style="list-style-type: none"> • The certificate is checked against the server's <i>trusted CA list</i> to ensure that it is signed by a trusted certification authority. • If a certificate constraints policy is set, the certificate is checked to make sure it satisfies the specified constraints. • If a certificate validator policy is set (by programming), the certificate is also checked by this policy.

Stage	Description
2	The server then performs security layer authentication by calling <code>authenticate()</code> on the iS2 server, passing the client's X.509 certificate as the argument.
3	The iS2 server authenticates the client's X.509 certificate by checking it against a cached copy of the certificate. The type of checking performed depends on the particular <i>third-party enterprise security service</i> that is plugged into the iS2 server.
4	If authentication is successful, the iS2 server returns the user's realms and roles.
5	The security layer controls access to the target's IDL interfaces by consulting an <i>action-role mapping file</i> to determine what the user is allowed to do.

Client configuration

[Example 23](#) shows a sample client configuration that you can use for the security-level certificate-based authentication scenario ([Figure 15 on page 79](#)).

Example 23: Client Configuration for Security-Level Certificate-Based Authentication

```
# Artix Configuration File
corba_cert_auth
{
    orb_plugins = ["local_log_stream", "iiop_profile", "giop",
"iiop_tls", "gsp"];

    event_log:filters = ["IT_GSP=*", "IT_CSI=*", "IT_TLS=*",
"IT_IIOP_TLS=*", "IT_ATLI2_TLS=*"];

    binding:client_binding_list = ["GIOP+EGMIOP",
"OTS+POA_Coloc", "POA_Coloc", "OTS+TLS_Coloc+POA_Coloc",
"TLS_Coloc+POA_Coloc", "GIOP+SHMIOP", "CSI+OTS+GIOP+IIOP",
"CSI+GIOP+IIOP", "CSI+OTS+GIOP+IIOP_TLS",
"CSI+GIOP+IIOP_TLS", "GIOP+IIOP", "GIOP+IIOP_TLS"];

    client_x509
    {
```

Example 23: *Client Configuration for Security-Level Certificate-Based Authentication*

```

policies:iiop_tls:client_secure_invocation_policy:supports =
["Integrity", "Confidentiality", "DetectReplay",
 "DetectMisordering", "EstablishTrustInTarget",
 "EstablishTrustInClient"];

policies:iiop_tls:client_secure_invocation_policy:requires =
["Integrity", "Confidentiality", "DetectReplay",
 "DetectMisordering"];

    principal_sponsor:use_principal_sponsor = "true";
    principal_sponsor:auth_method_id = "pkcs12_file";
    principal_sponsor:auth_method_data =
["filename=W:\art\etc\tls\x509\certs\demos\bob.p12",
 "password=bobpass"];
};
};

```

The preceding client configuration is a typical SSL/TLS configuration. The only noteworthy feature is that the client must have an associated X.509 certificate. Hence, the `principal_sponsor` settings are initialized with the location of an X.509 certificate (provided in the form of a PKCS#12 file). For a discussion of these client SSL/TLS settings, see [“Sample client configuration” on page 60](#) and [“Deploying Application Certificates” on page 165](#).

Target configuration

[Example 24](#) shows a sample server configuration that you can use for the security-level certificate-based authentication scenario ([Figure 15 on page 79](#)).

Example 24: *Server Configuration for Security-Level Certificate-Based Authentication*

```

# Artix Configuration File
corba_cert_auth
{
    orb_plugins = ["local_log_stream", "iiop_profile", "giop",
 "iiop_tls", "gsp"];

    event_log:filters = ["IT_GSP=", "IT_CSI=", "IT_TLS=",
 "IT_IIOP_TLS=", "IT_ATLI2_TLS="];
}

```

Example 24: Server Configuration for Security-Level Certificate-Based Authentication

```

binding:client_binding_list = ["GIOP+EGMIOP",
"OTS+POA_Coloc", "POA_Coloc", "OTS+TLS_Coloc+POA_Coloc",
"TLS_Coloc+POA_Coloc", "GIOP+SHMIOP", "CSI+OTS+GIOP+IIOP",
"CSI+GIOP+IIOP", "CSI+OTS+GIOP+IIOP_TLS",
"CSI+GIOP+IIOP_TLS", "GIOP+IIOP", "GIOP+IIOP_TLS"];

server
{
    policies:csi:auth_over_transport:authentication_service
= "com.iona.corba.security.csi.AuthenticationService";

    principal_sponsor:use_principal_sponsor = "true";
    principal_sponsor:auth_method_id = "pkcs12_file";
    principal_sponsor:auth_method_data =
1 ["filename=CertDir\target_cert.p12",
"password=TargetCertPass"];

    binding:server_binding_list = ["CSI+GSP", "CSI",
"GSP"];

    initial_references:IS2Authorization:plugin =
"it_is2_authorization";

    plugins:it_is2_authorization:ClassName =
"com.iona.corba.security.authorization.IS2AuthorizationPlugIn
";

2    plugins:is2_authorization:action_role_mapping =
"file:///PathToARMFile";

    auth_x509
3 {

    plugins:gsp:enable_security_service_cert_authentication =
"true";

    policies:iiop_tls:target_secure_invocation_policy:supports =
["Integrity", "Confidentiality", "DetectReplay",
"DetectMisordering", "EstablishTrustInTarget",
"EstablishTrustInClient"];

```

Example 24: Server Configuration for Security-Level Certificate-Based Authentication

4

```

policies:iiop_tls:target_secure_invocation_policy:requires =
["Integrity", "Confidentiality", "DetectReplay",
 "DetectMisordering", "EstablishTrustInClient"];
    };
};
};

```

The preceding server configuration can be explained as follows:

1. As is normal for an SSL/TLS server, you must provide the server with its own certificate, `target_cert.p12`. The simplest way to do this is to specify the location of a PKCS#12 file using the principal sponsor.
2. This configuration setting specifies the location of an action-role mapping file, which controls access to the server's interfaces and operations. See [“Artix Action-Role Mapping ACL” on page 135](#) for more details.
3. The `plugins:gsp:enable_security_service_cert_authentication` variable is the key to enabling security-level certificate-based authentication. By setting this variable to `true`, you cause the server to perform certificate authentication in the GSP security layer.
4. The IIOP/TLS target secure invocation policy must require `EstablishTrustInClient`. Evidently, if the client does not provide a certificate during the SSL/TLS handshake, there will be no certificate available to perform the security layer authentication.

Related administration tasks

When using X.509 certificate-based authentication for CORBA bindings, it is necessary to add the appropriate user data to your *enterprise security system* (which is integrated with the iS2 server through an iSF adapter), as follows:

- File adapter (do not use in deployed systems)—see [“Certificate-based authentication for the file adapter” on page 129](#).
- LDAP adapter—see [“Certificate-based authentication for the LDAP adapter” on page 131](#).

Single Sign-On

Single sign-on (SSO) is an Artix security framework feature which is used to minimize the exposure of usernames and passwords to snooping. After initially signing on, a client communicates with other applications by passing an SSO token in place of the original username and password.

Note: The SSO feature is unavailable in some editions of Artix. Please check the conditions of your Artix license to see whether your installation supports SSO.

In this chapter

This chapter discusses the following topics:

SSO and the Login Service	page 86
Username/Password-Based SSO for SOAP Bindings	page 89
SSO Sample Configuration for SOAP Bindings	page 96

SSO and the Login Service

Overview

There are two different implementations of the login service, depending on the type of bindings you use in your application:

- [SOAP binding](#).
 - [CORBA Binding](#).
-

SOAP binding

For SOAP bindings, SSO is implemented by the following elements of the Artix security framework:

- *Artix login service*—a central service that authenticates username/password combinations and returns SSO tokens. Clients connect to this service using the HTTP/S protocol.
 - *login_client plug-in*—the `login_client` plug-in, which is loaded by SOAP clients, is responsible for contacting the Artix login service to obtain an SSO token.
 - *artix_security plug-in*—on the server side, the `artix_security` plug-in is responsible for parsing the received SSO credentials and authenticating the SSO token with the iS2 server.
-

CORBA Binding

For CORBA bindings, SSO is implemented by the following elements of the Artix security framework:

- *CORBA login service*—a central service that authenticates username/password combinations and generates SSO tokens. Clients connect to this service using the IIOP/TLS protocol.
- *GSP plug-in*—the generic security plug-in is responsible for the following:
 - ◆ On the client side—contacts the CORBA login service to obtain an SSO token.
 - ◆ On the server side—sends a received SSO token to be authenticated by the iS2 server.

Advantages of SSO

SSO greatly increases the security of an Artix security framework system, offering the following advantages:

- Password visibility is restricted to the login service.
- Clients use SSO tokens to communicate with servers.
- Clients can be configured to use SSO with no code changes.
- SSO tokens are configured to expire after a specified length of time.
- When an SSO token expires, the Artix client automatically requests a new token from the login service. No additional user code is required.

Login service

Figure 16 shows an overview of a login service. The client Bus automatically requests an SSO token by sending a username and a password to the login service. If the username and password are successfully authenticated, the login service returns an SSO token.

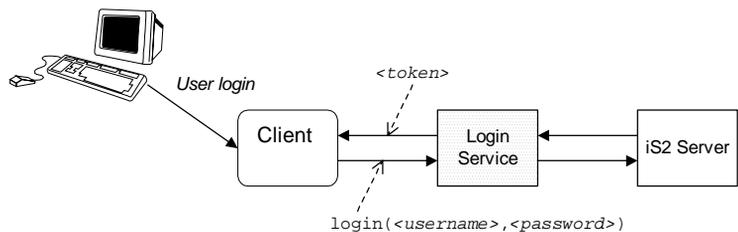


Figure 16: Client Requesting an SSO Token from the Login Service

SSO token

The SSO token is a compact key that the iS2 server uses to access a user's session details, which are stored in a cache.

SSO token expiry

The iS2 server is configured to impose the following kinds of timeout on an SSO token:

- *SSO session timeout*—this timeout places an absolute limit on the lifetime of an SSO token. When the timeout is exceeded, the token expires.
- *SSO session idle timeout*—this timeout places a limit on the amount of time that elapses between authentication requests involving the SSO token. If the central iS2 server receives no authentication requests in this time, the token expires.

For more details, see [“Configuring Single Sign-On Properties” on page 114](#).

Automatic token refresh

In theory, the expiry of SSO tokens could prove a nuisance to client applications, because servers will raise a security exception whenever an SSO token expires. In practice, however, when SSO is enabled, the relevant plug-in (`login_service` for SOAP and `gsp` for CORBA) catches the exception on the client side and contacts the login service again to refresh the SSO token automatically. The plug-in then automatically retries the failed operation invocation.

Username/Password-Based SSO for SOAP Bindings

Overview

When using SOAP bindings, usernames and passwords can be transmitted using one of the following mechanisms:

- WSSE UsernameToken.
- HTTP Basic Authentication.
- CORBA Principal (username only).

This section describes how to configure a client so that it transmits an SSO token in place of a username and a password.

Username/password authentication without SSO

Figure 17 gives an overview of ordinary username/password-based authentication without SSO. In this case, the username, `<username>`, and password, `<password>`, are passed directly to the target server, which then contacts the iS2 server to authenticate the username/password combination.

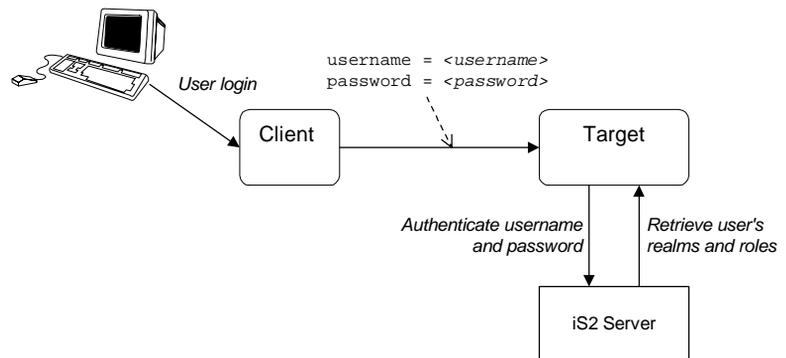


Figure 17: Overview of Username/Password Authentication without SSO

Username/password authentication with SSO

Figure 18 gives an overview of username/password-based authentication when SSO is enabled.

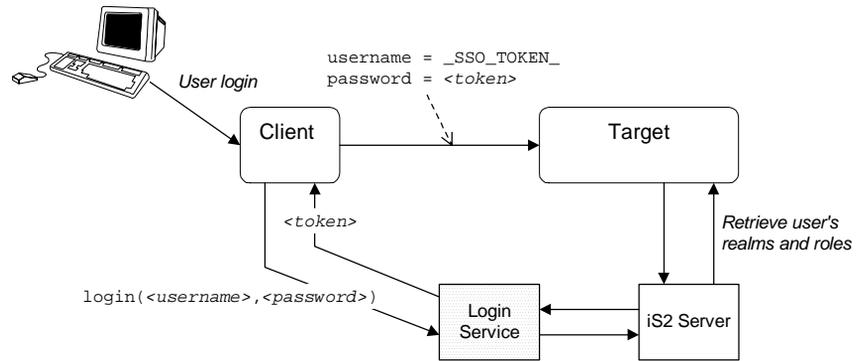


Figure 18: Overview of Username/Password Authentication with SSO

Prior to contacting the target server for the first time, the client Bus sends the username, `<username>`, and password, `<password>`, to the login server, getting an SSO token, `<token>`, in return. The client Bus then includes a IONA-proprietary SOAP header (extension of WSSE BinarySecurityToken) in the next request to the target server, sending the special string, `_SSO_TOKEN_`, instead of a username and the SSO token, `<token>`, instead of a password. The target server's Bus contacts the iS2 server to authenticate the username/password combination.

Client configuration

Example 25 shows a typical domain configuration for an SSO SOAP client that employs username/password authentication.

Example 25: SOAP Client Configuration for Username/Password-Based SSO

```

# artix.cfg Domain Configuration
...
1 plugins:login_client:wSDL_url="../../wSDL/login_service.wSDL";
  plugins:login_client:shlib_name = "it_login_client";
...

```

Example 25: SOAP Client Configuration for Username/Password-Based SSO

```

2 sso_soap_client {
3   orb_plugins = ["xmlfile_log_stream", "iiop_profile", "giop",
  "iiop", "soap", "http", "login_client"];
  binding:artix:client_request_interceptor_list=
  "login_client:login_client";
  ...
};

```

The preceding Artix configuration can be described as follows:

1. The `plugins:login_client:wSDL_url` variable specifies the location of the Artix login service WSDL contract. You must edit this setting, if you store this contract at a different location.
2. The `orb_plugins` list must include the `login_client` plug-in.
3. The Artix client request interceptor list must include the `login_client:login_client` interceptor. The format of an entry in the interceptor list is:

`<namespace-prefix> : <interceptor-name>`

which is the format used in the default interceptor list, for example:

`login_client:login_client`

The preceding format requires that the `login_client` namespace prefix is defined in your application's WSDL contract.

Target configuration

[Example 26](#) shows a typical domain configuration for an SSO SOAP target server that accepts connections from clients that authenticate themselves using username/password authentication.

Example 26: SOAP Target Configuration for Username/Password-Based SSO

```

# artix.cfg Domain Configuration
...
sso_soap_target {
1  plugins:artix_security:shlib_name = "it_security_plugin";
  binding:artix:server_request_interceptor_list=
  "bus-security:security";
  binding:client_binding_list = ["OTS+POA_Coloc", "POA_Coloc",
  "OTS+GIOP+IIOP", "GIOP+IIOP", "GIOP+IIOP-TLS"];
}

```

Example 26: SOAP Target Configuration for Username/Password-Based SSO

```

2 orb_plugins = ["xmlfile_log_stream", "iiop_profile", "giop",
  "iiop_tls", "soap", "http", "artix_security"];
3
4 policies:asp:enable_sso = "true";
  policies:asp:enable_authorization = "true";
  plugins:asp:authentication_cache_size = "5";
  plugins:asp:authentication_cache_timeout = "10";
  plugins:is2_authorization:action_role_mapping =
  "file://C:\artix_20\artix\2.0\demos\security\single_signon/et
  c/helloworld_action_role_mapping.xml";
5 plugins:asp:security_level = "REQUEST_LEVEL";
  };

```

The preceding Artix configuration can be described as follows:

1. The `bus-security:security` interceptor must appear in the Artix server interceptor list to enable the `artix_security` plug-in functionality.

Note: The `bus-security` namespace prefix must be defined in the application's WSDL contract—see [“Application WSDL configuration” on page 94](#).

2. The `orb_plugins` list must include the `artix_security` plug-in.
3. The `policies:asp:enable_sso` variable must be set to `true` to enable SSO on the target server.
4. You can enable SSO with or without authentication. In this example, the authentication feature is enabled.
5. The security level is set to `REQUEST_LEVEL`, implying that the username and password are extracted from the SOAP header. There is no need to set the security type when SSO is enabled (hence, `plugins:asp:security_type` is omitted from this configuration).

Artix login service configuration

Example 27 shows the domain configuration for a standalone Artix login service. The clients of this login service authenticate themselves to the login service using WSSE UsernameToken credentials.

Example 27: Artix Login Service Domain Configuration

```

# artix.cfg Domain Configuration
...
sso_login_service {
  1   plugins:artix_security:shlib_name = "it_security_plugin";
      binding:artix:server_request_interceptor_list=
        "bus-security:security";
  2   binding:client_binding_list = ["OTS+POA_Coloc", "POA_Coloc",
        "OTS+GIOP+IIOP", "GIOP+IIOP", "GIOP+IIOP_TLS"];
      orb_plugins = ["xmlfile_log_stream", "iiop_profile", "giop",
  3   "iiop_tls", "soap", "http", "artix_security",
        "login_service"];
  4   plugins:login_service:wSDL_url="../../wSDL/login_service.wSDL";
      plugins:login_service:shlib_name = "it_login_service";
      plugins:asp:security_type = "USERNAME_PASSWORD";
      plugins:asp:security_level = "REQUEST_LEVEL";
};

```

The preceding Artix configuration can be described as follows:

1. The `bus-security:security` interceptor must appear in the Artix server interceptor list to enable the `artix_security` plug-in functionality.
2. The `orb_plugins` list must include the `artix_security` plug-in and the `login_service` plug-in.
3. The `plugins:login_service:wSDL_url` variable specifies the location of the Artix login service WSDL contract. You must edit this setting, if you store this contract at a different location.
4. The security type and security level settings selected here (`USERNAME_PASSWORD` and `REQUEST_LEVEL` respectively) imply that the login service reads the WSSE UsernameToken credentials from the incoming client request messages.

You can change these settings to use different client credentials (for example, `USERNAME_PASSWORD` and `MESSAGE_LEVEL` for HTTP Basic

Authentication), but you must be careful to ensure that this matches the kind of credentials sent by the client.

Application WSDL configuration

If you are using SSO security, you must modify your application's WSDL contract by adding namespace definitions for `bus-security` and `login_client`, as shown in [Example 28](#).

Example 28: Additions to Application WSDL Required for SSO

```
# Application WSDL Contract
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="HelloWorld"
  targetNamespace="http://www.iona.com/full_security"
  xmlns="http://schemas.xmlsoap.org/wsdl/"

  xmlns:http-conf="http://schemas.iona.com/transport/http/conf
  igation"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.iona.com/full_security"
  xmlns:wSDL="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:bus-security="http://schemas.iona.com/bus/security"
xmlns:login_client="http://schemas.iona.com/security/login_client"
  >
  ...
</wsdl:definitions>
```

The additional namespace definitions are used in the Artix domain configuration as follows:

- `bus-security` is used as a prefix to identify the `bus-security:security` interceptor in a server configuration.
- `login_client` is used as a prefix to identify the `login_client:login_client` interceptor in a client configuration.

Login service WSDL configuration

[Example 29](#) shows an extract from the login service WSDL contract (in the directory, `artix/Version/demos/security/single_signon/wsdl`) showing details of the WSDL port settings.

Example 29: Extract from the Login Service WSDL Configuration

```
# Login Service WSDL Contract
<definitions ... >
  ...
  <service name="LoginService">
    <port binding="tns:LoginServiceBinding"
          name="LoginServicePort">
      <soap:address
        location="http://localhost:49675"/>
      <bus-security:security
        enableSSO="false"
        enableAuthorization="false"
        authenticationCacheSize="1"
        authenticationCacheTimeout="1" />
    </port>
  </service>
</definitions>
```

Note the following points about the WSDL port settings:

- The login service listens on a fixed host and port, `http://localhost:4975`. You will probably need to edit this setting before deploying the login service in a real system. However, you should *not* choose dynamic IP port allocation (for example, using `http://localhost:0`), because the clients would not be able to discover the value of the dynamically allocated port.
- You should not change the values of the attributes in the `<bus-security:security>` tag. The values shown in [Example 29](#) are essential for the correct functioning of the Artix login service.

WARNING: [Example 29](#) shows a login service configuration with insecure communications (HTTP). It is strongly recommended that you modify this configuration to use TLS security (HTTPS).

Related administration tasks

For details of how to configure SSO token timeouts, see [“Configuring Single Sign-On Properties” on page 114](#).

SSO Sample Configuration for SOAP Bindings

Overview

This section provides SSO sample configurations for the SOAP binding including configurations for a client, a server, and a standalone Artix login service.

Client SSO configuration

The `secure_artix.single_signon.client` configuration scope from [Example 30](#) can be used to configure a SOAP SSO client. This client configuration has the following characteristics:

- The SSO client loads the `login_client` plug-in, which is responsible for contacting the HTTP login server to obtain an SSO token.
- The client's SOAP and HTTP security settings are stored separately in the client's copy of the WSDL contract.

WARNING: It is strongly recommended that you configure the client's WSDL contract to use TLS security (HTTPS).

Server SSO configuration

The `secure_artix.single_signon.server` configuration scope from [Example 30](#) can be used to configure a SOAP SSO server. This server configuration has the following characteristics:

- The SSO server loads the `artix_security` plug-in, which provides the implementation of SSO on the server side.
- You can enable authorization while using SSO credentials (set `policies:asp:enable_authorization` to `true`).

WARNING: It is strongly recommended that you configure the server's WSDL contract to use TLS security (HTTPS).

Artix login service configuration

The `secure_artix.single_signon.sso_service` configuration scope from [Example 30](#) gives an example of a standalone Artix login service.

WARNING: It is strongly recommended that you configure the login server's WSDL contract to use TLS security (HTTPS).

SSO configuration example

Example 30 shows sample configurations for a SOAP SSO client and a SOAP SSO server.

Example 30: SOAP SSO Client and Server Configuration Examples

```
secure_artix {
    ...
    single_signon
    {
        initial_references:IT_SecurityService:reference =
        "corbaloc:iiops:1.2@localhost:55349,it_iiops:1.2@localhost:55
        349/IT_SecurityService";

        security_service
        {
            ...
        };

        client
        {
            plugins:login_client:wSDL_url="../../wSDL/login_service.wSDL";
            plugins:login_client:shlib_name = "it_login_client";
            binding:artix:client_request_interceptor_list=
            "login_client:login_client";
            orb_plugins = ["xmlfile_log_stream", "soap", "http",
            "login_client"];
        };

        server
        {
            plugins:artix_security:shlib_name =
            "it_security_plugin";
            binding:artix:server_request_interceptor_list=
            "bus-security:security";
            binding:client_binding_list = ["OTS+POA_Coloc",
            "POA_Coloc", "OTS+GIOP+IIOP", "GIOP+IIOP", "GIOP+IIOP_TLS"];
            orb_plugins = ["xmlfile_log_stream", "iiop_profile",
            "giop", "iiop_tls", "soap", "http", "artix_security"];

            policies:asp:enable_sso = "true";
            policies:asp:enable_authorization = "true";
            plugins:asp:authentication_cache_size = "5";
            plugins:asp:authentication_cache_timeout = "10";
            plugins:is2_authorization:action_role_mapping =
            "file://C:\artix_20\artix\2.0\demos\security\single_signon\et
            c\helloworld_action_role_mapping.xml";
        };
    };
}
```

Example 30: SOAP SSO Client and Server Configuration Examples

```

        plugins:asp:security_level = "REQUEST_LEVEL";
    };

    sso_service
    {
        plugins:artix_security:shlib_name =
        "it_security_plugin";
        binding:artix:server_request_interceptor_list=
        "bus-security:security";
        binding:client_binding_list = ["OTS+POA_Coloc",
        "POA_Coloc", "OTS+GIOP+IIOP", "GIOP+IIOP", "GIOP+IIOP_TLS"];
        orb_plugins = ["xmlfile_log_stream", "iiop_profile",
        "giop", "iiop_tls", "soap", "http", "artix_security",
        "login_service"];

        plugins:login_service:wSDL_url="../../wSDL/login_service.wSDL";
        plugins:login_service:shlib_name = "it_login_service";
        plugins:asp:security_type = "USERNAME_PASSWORD";
        plugins:asp:security_level = "REQUEST_LEVEL";
    };
};
...
};

```

Configuring the iS2 server

This chapter describes how to configure the properties of the iS2 server and, in particular, how to configure a variety of adapters that can integrate the iS2 server with third-party enterprise security back-ends (for example, LDAP and SiteMinder).

In this chapter

This chapter discusses the following topics:

Configuring the File Adapter	page 100
Configuring the LDAP Adapter	page 102
Configuring the SiteMinder Adapter	page 108
Configuring the Kerberos Adapter	page 110
Additional Security Configuration	page 113

Configuring the File Adapter

Overview

The iSF file adapter enables you to store information about users, roles, and realms in a flat file, a *security information file*. The file adapter is easy to set up and configure, but is appropriate for demonstration purposes only. This section describes how to set up and configure the iSF file adapter.

WARNING: The file adapter is provided for demonstration purposes only. IONA does not support the use of the file adapter in a production environment.

File locations

The following files configure the iSF file adapter:

- `is2.properties` file—the default location of the iS2 properties file is as follows:

```
ArtixInstallDir/artix/2.0/bin/is2.properties
```

See [“iS2 Properties File” on page 268](#) for details of how to customize the default iS2 properties file location.

- Security information file—this file’s location is specified by the `com.iona.isp.adapter.file.param.filename` property in the `is2.properties` file.

File adapter properties

[Example 31](#) shows the properties to set for a file adapter.

Example 31: Sample File Adapter Properties

```
1 com.iona.isp.adapters=file

#####
##
## Demo File Adapter Properties
##
#####
2 com.iona.isp.adapter.file.class=com.iona.security.is2adapter.file.FileAuthAdapter
```

Example 31: Sample File Adapter Properties

```

3 com.iona.isp.adapter.file.param.filename=ArtixInstallDir/artix/2.0/
  bin/is2_user_password_role_file.txt

#####
## General iS2 server Properties
#####
4 # ... Generic properties not shown here ...

```

The necessary properties for a file adapter are described as follows:

1. Set `com.iona.isp.adapters=file` to instruct the Is2 server to load the file adapter.
2. The `com.iona.isp.adapter.file.class` property specifies the class that implements the iSF file adapter.
3. The `com.iona.isp.adapter.file.param.filename` property specifies the location of the security information file, which contains information about users and roles.
4. *(Optionally)* You might also want to edit the general Is2 server properties.

See [“Additional Security Configuration” on page 113](#) for details.

Configuring the LDAP Adapter

Overview

The IONA security platform integrates with the Lightweight Directory Access Protocol (LDAP) enterprise security infrastructure by using an LDAP adapter. The LDAP adapter is configured in an `is2.properties` file. This section discusses the following topics:

- [Prerequisites](#)
 - [File location](#).
 - [Minimal LDAP configuration](#).
 - [Basic LDAP properties](#).
 - [LDAP.param properties](#).
 - [LDAP server replicas](#).
 - [Logging on to an LDAP server](#).
-

Prerequisites

Before configuring the LDAP adapter, you must have an LDAP security system installed and running on your system. LDAP is *not* a standard part of Artix, but you can use the Is2 server's LDAP adapter with any LDAP v.3 compatible system.

File location

The following file configures the LDAP adapter:

- `is2.properties` file—the default location of the iS2 properties file is as follows:

```
ArtixInstallDir/artix/2.0/is2.properties
```

See [“iS2 Properties File” on page 268](#) for details of how to customize the default iS2 properties file location.

Minimal LDAP configuration

[Example 32](#) shows the minimum set of iS2 properties that can be used to configure an LDAP adapter.

Example 32: A Sample LDAP Adapter Configuration File

```

1  com.iona.isp.adapters=LDAP
   #####
   ##
   ## LDAP Adapter Properties
   ##
   #####
2  com.iona.isp.adapter.LDAP.class=com.iona.security.is2adapter.ldap.LdapAdapter

3  com.iona.isp.adapter.LDAP.param.host.1=10.81.1.400
   com.iona.isp.adapter.LDAP.param.port.1=389

4  com.iona.isp.adapter.LDAP.param.UserNameAttr=uid
   com.iona.isp.adapter.LDAP.param.UserBaseDN=dc=iona,dc=com
   com.iona.isp.adapter.LDAP.param.UserObjectClass=organizationalPerson
   com.iona.isp.adapter.LDAP.param.UserSearchScope=SUB

5  com.iona.isp.adapter.LDAP.param.UserRoleDNAttr=nsroledn
   com.iona.isp.adapter.LDAP.param.RoleNameAttr=cn

6  com.iona.isp.adapter.LDAP.param.GroupNameAttr=cn
   com.iona.isp.adapter.LDAP.param.GroupObjectClass=groupofuniqueNames
   com.iona.isp.adapter.LDAP.param.GroupSearchScope=SUB
   com.iona.isp.adapter.LDAP.param.GroupBaseDN=dc=iona,dc=com
   com.iona.isp.adapter.LDAP.param.MemberDNAttr=uniqueMember

7  com.iona.isp.adapter.LDAP.param.version=3

```

The necessary properties for an LDAP adapter are described as follows:

1. Set `com.iona.isp.adapters=LDAP` to instruct the IONA Security Platform to load the LDAP adapter.
2. The `com.iona.isp.adapter.file.class` property specifies the class that implements the LDAP adapter.

3. For each LDAP server replica, you must specify the host and port where the LDAP server can be contacted. In this example, the host and port parameters for the primary LDAP server, `host.1` and `port.1`, are specified.
4. These properties specify how the LDAP adapter finds a user name within the LDAP directory schema. The properties are interpreted as follows:

<code>UserNameAttr</code>	The attribute type whose corresponding value uniquely identifies the user.
<code>UserBaseDN</code>	The base DN of the tree in the LDAP directory that stores user object class instances.
<code>UserObjectClass</code>	The attribute type for the object class that stores users.
<code>UserSearchScope</code>	The user search scope specifies the search depth relative to the user base DN in the LDAP directory tree. Possible values are: <code>BASE</code> , <code>ONE</code> , or <code>SUB</code> .

See [“iS2 Properties File” on page 268](#) for more details.

5. The following properties specify how the adapter extracts a user’s role from the LDAP directory schema:

<code>UserRoleDNAttr</code>	The attribute type that stores a user’s role DN.
<code>RoleNameAttr</code>	The attribute type that the LDAP server uses to store the role name.

6. These properties specify how the LDAP adapter finds a group name within the LDAP directory schema. The properties are interpreted as follows:

<code>GroupNameAttr</code>	The attribute type whose corresponding attribute value gives the name of the user group.
<code>GroupBaseDN</code>	The base DN of the tree in the LDAP directory that stores user groups.
<code>GroupObjectClass</code>	The object class that applies to user group entries in the LDAP directory structure.

GroupSearchScope	The group search scope specifies the search depth relative to the group base DN in the LDAP directory tree. Possible values are: BASE, ONE, or SUB.
MemberDNAttr	The attribute type that is used to retrieve LDAP group members.

See [“iS2 Properties File” on page 268](#) for more details.

7. The LDAP version number can be either 2 or 3, corresponding to LDAP v.2 or LDAP v.3 respectively.

Basic LDAP properties

The following properties must always be set as part of the LDAP adapter configuration:

```
com.iona.isp.adapters=LDAP
com.iona.isp.adapter.LDAP.class=com.iona.security.is2adapter.ldap.LdapAdapter
```

In addition to these basic properties, you must also set a number of LDAP parameters, which are prefixed by `com.iona.isp.adapter.LDAP.param`.

LDAP.param properties

Table 1 shows all of the LDAP adapter properties from the `com.iona.isp.adapter.LDAP.param` scope. Required properties are shown in bold:

Table 1: *LDAP Properties in the com.iona.isp.adapter.LDAP.param Scope*

LDAP Server Properties	LDAP User/Role Configuration Properties
host. <Index> port. <Index> SSLEnabled. <Index> SSLCACertDir. <Index> SSLClientCertFile. <Index> SSLClientCertPassword. <Index> PrincipalUserDN. <Index> PrincipalUserPassword. <Index>	UserNameAttr UserBaseDN UserObjectClass UserSearchScope UserSearchFilter UserRoleDNAttr RoleNameAttr UserCertAttrName
LDAP Group/Member Configuration Properties	Other LDAP Properties
GroupNameAttr GroupObjectClass GroupSearchScope GroupBaseDN MemberDNAttr MemberFilter	MaxConnectionPoolSize version UseGroupAsRole RetrieveAuthInfo CacheSize CacheTimeToLive

LDAP server replicas

The LDAP adapter is capable of failing over to one or more backup replicas of the LDAP server. Hence, properties such as `host.<Index>` and `port.<Index>` include a replica index as part of the parameter name.

For example, `host.1` and `port.1` refer to the host and port of the primary LDAP server, while `host.2` and `port.2` would refer to the host and port of an LDAP backup server.

Logging on to an LDAP server

The following properties can be used to configure login parameters for the *<Index>* LDAP server replica:

`PrincipalUserDN.` *<Index>*
`PrincipalUserPassword.` *<Index>*

The properties need only be set if the LDAP server is configured to require username/password authentication.

Secure connection to an LDAP server

The following properties can be used to configure SSL/TLS security for the connection between the Is2 server and the *<Index>* LDAP server replica:

`SSLEnabled.` *<Index>*
`SSLCA CertDir.` *<Index>*
`SSLClientCertFile.` *<Index>*
`SSLClientCertPassword.` *<Index>*

The properties need only be set if the LDAP server requires SSL/TLS mutual authentication.

iS2 properties reference

For more details about the Is2 server properties, see [“iS2 Configuration” on page 265](#).

Configuring the SiteMinder Adapter

Overview

The SiteMinder adapter enables you to integrate the Is2 server with SiteMinder, which is an enterprise security product from Netegrity. By configuring the SiteMinder adapter, you ensure that any authentication requests within the Artix Security Framework are delegated to SiteMinder. This section describes how to set up and configure the SiteMinder adapter.

Prerequisites

Ensure that the SiteMinder product is installed and configured on your system. SiteMinder is *not* a standard part of Artix, but is available from Netegrity at <http://www.netegrity.com>.

File location

The following file configures the SiteMinder adapter:

- `is2.properties` file—the default location of the iS2 properties file is as follows:

```
ArtixInstallDir/artix/2.0/bin/is2.properties
```

See “[iS2 Properties File](#)” on page 268 for details of how to customize the default iS2 properties file location.

SiteMinder adapter properties

[Example 33](#) shows the properties to set for the SiteMinder adapter.

Example 33: SiteMinder Adapter Properties

```

1  com.iona.isp.adapters=SiteMinder
   #####
   ##
   ## SiteMinder Adapter Properties
   ##
   #####
2  com.iona.isp.adapter.SiteMinder.class=com.iona.security.is2adapt
   er.smapapter.SiteMinderAgent
3  com.iona.isp.adapter.SiteMinder.param.ServerAddress=localhost
   com.iona.isp.adapter.SiteMinder.param.ServerAuthnPort=400
   com.iona.isp.adapter.SiteMinder.param.AgentSecret=secret
   com.iona.isp.adapter.SiteMinder.param.AgentName=web

```

Example 33: SiteMinder Adapter Properties

```
#####
## General Is2 Server Properties
#####
4 # ... Generic properties not shown here ...
```

The necessary properties for a SiteMinder adapter are described as follows:

1. Set `com.ionas.isp.adapters=SiteMinder` to instruct the Is2 server to load the SiteMinder adapter.
2. The `com.ionas.isp.adapter.SiteMinder.class` property specifies the class that implements the SiteMinder adapter.
3. A SiteMinder adapter requires the following parameters:

<code>ServerAddress</code>	Host address where SiteMinder is running.
<code>ServerAuthnPort</code>	SiteMinder's IP port number.
<code>AgentName</code>	SiteMinder agent's name.
<code>AgentSecret</code>	SiteMinder agent's password.

4. (Optionally) You might also want to edit the general Is2 server properties.
See ["Additional Security Configuration" on page 113](#) for details.

Configuring the Kerberos Adapter

Overview

The Kerberos adapter enables you to use the Kerberos Authentication Service. By configuring the Kerberos adapter, you ensure that any authentication requests within the Artix Security Framework are delegated to Kerberos. This section describes how to set up and configure the Kerberos adapter.

File location

The following file configures the Kerberos adapter:

- `is2.properties` file—the default location of the iS2 properties file is as follows:

```
ArtixInstallDir/artix/2.0/bin/is2.properties
```

See [“iS2 Properties File” on page 268](#) for details of how to customize the default iS2 properties file location.

Kerberos adapter properties

[Example 34](#) shows the properties to set for the Kerberos adapter.

Example 34: Kerberos Adapter Properties

```

1  com.iona.isp.adapters=kbr5
   #####
   ##
   ## Kerberos Adapter Properties
   ##
   #####
2  com.iona.isp.adapter.kbr5.class=com.iona.security.is2adapter.kbr
   5.IS2KerberosAdapter
3  com.iona.isp.adapter.krb5.param.java.security.krb5.realm=MYREALM
   .COMPANY.COM
   com.iona.isp.adapter.krb5.param.java.security.krb5.kdc=10.65.3.7
   4
   com.iona.isp.adapter.krb5.param.java.security.auth.login.config=
   jaas.conf
   com.iona.isp.adapter.krb5.param.javax.security.auth.useSubjectCr
   edsOnly=false

```

Example 34: Kerberos Adapter Properties

```
#####
## General Is2 Server Properties
#####
4 # ... Generic properties not shown here ...
```

The necessary properties for a Kerberos adapter are described as follows:

1. Set `com.iona.isp.adapters=kbr5` to instruct the Is2 server to load the Kerberos adapter.
2. The `com.iona.isp.adapter.kbr5.class` property specifies the class that implements the Kerberos adapter.
3. A Kerberos adapter requires the following parameters:

<code>java.security.kbr5.realm</code>	The Kerberos Realm Name.
<code>java.security.kbr5kdc</code>	The server name or IP address of the Kerberos KDC server.
<code>java.security.auth.login.config</code>	The configuration file for the JAAS Login Module.
<code>javax.security.auth.useSubjectCredsOnly</code>	A required JAAS Login Module property. Always set to <code>false</code> .

4. (Optionally) You might also want to edit the general Is2 server properties.
See [“Additional Security Configuration” on page 113](#) for details.

Retrieving the user’s group information

Once the Kerberos token has been authenticated, the Kerberos adapter can be configured to retrieve the user’s group information and save it for future authorization purposes.

[Example 35](#) shows a sample iS2 configuration for the Kerberos adapter that retrieve the user’s group information.

Example 35: Kerberos Configuraiton to Retrieve User Group Information

```
1 com.iona.isp.adapter.krb5.param.RetrieveAuthInfo=true
```

Example 35: Kerberos Configuration to Retrieve User Group Information

```

2 com.iona.isp.adapter.krb5.param.host.1=$ACTIVE_DIRECTORY_SERVER_
  NAME$
com.iona.isp.adapter.krb5.param.port.1=389
com.iona.isp.adapter.krb5.param.SSLEnabled.1=no
com.iona.isp.adapter.krb5.param.SSLCACertDir.1=d:/certs/test
com.iona.isp.adapter.krb5.param.SSLClientCertFile.1=d:/certs/ver
  isign.pl2
com.iona.isp.adapter.krb5.param.SSLClientCertPassword.1=netfish
com.iona.isp.adapter.krb5.param.PrincipalUserDN.1=cn=administ
  rat
  or,cn=users,dc=boston,dc=amer,dc=iona,dc=com
com.iona.isp.adapter.krb5.param.PrincipalUserPassword.1=orbix
com.iona.isp.adapter.krb5.param.ConnectTimeout.1=15

3 com.iona.isp.adapter.krb5.param.UserNameAttr=CN
com.iona.isp.adapter.krb5.param.UserBaseDN=dc=boston,dc=amer,dc=
  iona,dc=com
com.iona.isp.adapter.krb5.param.version=3
com.iona.isp.adapter.krb5.param.UserObjectClass=Person
com.iona.isp.adapter.krb5.param.GroupObjectClass=group
com.iona.isp.adapter.krb5.param.GroupSearchScope=SUB
com.iona.isp.adapter.krb5.param.GroupBaseDN=dc=boston,dc=amer,dc
  =iona,dc=com
com.iona.isp.adapter.krb5.param.GroupNameAttr=CN
com.iona.isp.adapter.krb5.param.MemberDNAttr=memberOf
com.iona.isp.adapter.krb5.param.MaxConnectionPoolSize=1
com.iona.isp.adapter.krb5.param.MinConnectionPoolSize=1

```

The properties to configure the Kerberos adapter to retrieve a user's group information are explained as follows:

1. RetrieveAuthInfo=true activates this feature.
2. Set the connection information needed to open an LDAP connection to the Active Directory Server.

Note: If SSL needs to be enabled set

```
com.iona.isp.adapter.krb5.param.SSLEnabled.1=yes.
```

3. Tell the adapter how to construct a filter to search the Active Directory Server.

Additional Security Configuration

Overview

This section describes how to configure optional features of the Is2 server, such as single sign-on and the authorization manager. These features can be combined with any iSF adapter type.

In this section

This section contains the following subsections:

Configuring Single Sign-On Properties	page 114
Configuring the Log4J Logging	page 116

Configuring Single Sign-On Properties

Overview

The IONA security framework provides an optional *single sign-on* (SSO) feature. If you want to use SSO with your applications, you must configure the iS2 server as described in this section. SSO offers the following advantages:

- User credentials can easily be propagated between applications in the form of an SSO token.
 - Performance is optimized, because the authentication step only needs to be performed once within a distributed system.
 - Because the user's session is tracked centrally by the iS2 server, it is possible to impose timeouts on the user sessions and these timeouts are effective throughout the distributed system.
-

SSO tokens

The iS2 server generates an SSO token in response to an authentication operation. The SSO token is a compact key that the iS2 server uses to access a user's session details, which are stored in a cache.

SSO properties

[Example 36](#) shows the iS2 properties needed for SSO:

Example 36: Single Sign-On Properties

```
# iS2 Properties File
...
#####
## Single Sign On Session Info
#####
1 is2.sso.enabled=yes
2 is2.sso.session.timeout=6000
3 is2.sso.session.idle.timeout=300
4 is2.sso.cache.size=10000
```

The SSO properties are described as follows:

1. Setting this property to `yes` enables single sign-on.
2. The SSO session timeout sets the lifespan of SSO tokens, in units of seconds. Once the specified time interval elapses, the token expires.

3. The SSO session idle timeout sets the maximum length of time for which an SSO session can remain idle, in units of seconds. If the iS2 server registers no activity against a particular session for this amount of time, the session and its token expire.
 4. The size of the SSO cache, in units of number of sessions.
-

Related administration tasks

For details of how to configure CORBA applications to use SSO, see [“Single Sign-On for CORBA Applications” on page 126](#).

Configuring the Log4J Logging

Overview

log4j is a third-party toolkit from the Jakarta project, <http://jakarta.apache.org/log4j>, that provides a flexible and efficient system for capturing logging messages from an application. Because the Is2 server's logging is based on log4j, it is possible to configure the output of iSF logging using a standard log4j properties file.

log4j documentation

For complete log4j documentation, see the following Web page:
<http://jakarta.apache.org/log4j/docs/documentation.html>

Enabling log4j logging

To enable log4j logging, you can specify the location of the log4j properties file in either of the following ways:

- [In the CLASSPATH.](#)
 - [In the is2.properties file.](#)
-

In the CLASSPATH

You can specify the location of the log4j properties file by adding the file to your CLASSPATH. For example, you could add an `/is2_config/log4j.properties` file to your CLASSPATH as follows:

Windows

```
set CLASSPATH=C:\is2_config\log4j.properties;%CLASSPATH%
```

UNIX (Bourne shell)

```
export CLASSPATH=/is2_config/log4j.properties:$CLASSPATH;
```

In the is2.properties file

You can specify the location of the log4j properties file in the `is2.properties` file as follows:

```
# is2 Properties File, for Server ID=1
...
#####
## log4j Logging
#####
log4j.configuration=C:/is2_config/log4j.properties
...

```

Configuring the log4j properties file

The following example shows how to configure the log4j properties to perform basic logging. In this example, the lowest level of logging is switched on (`DEBUG`) and the output is sent to the console screen.

```
# log4j Properties File
log4j.rootCategory=DEBUG, A1

# A1 is set to be a ConsoleAppender.
log4j.appender.A1=org.apache.log4j.ConsoleAppender

# A1 uses PatternLayout.
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%-4r [%t] %-5p %c %x
- %m%n
```


Managing Users, Roles and Domains

The Is2 server provides a variety of adapters that enable you to integrate the Artix Security Framework with third-party enterprise security products. This allows you to manage users and roles using a third-party enterprise security product.

In this chapter

This chapter discusses the following topics:

Introduction to Domains and Realms	page 120
Managing a File Security Domain	page 128
Managing an LDAP Security Domain	page 131
Managing a SiteMinder Security Domain	page 132

Introduction to Domains and Realms

Overview

This section introduces the concepts of an Artix security domain and an Artix authorization realm, which are fundamental to the administration of the Artix Security Framework. Within an Artix security domain, you can create user accounts and within an Artix authorization realm you can assign roles to users.

In this section

This section contains the following subsections:

Artix security domains	page 121
Artix Authorization Realms	page 123

Artix security domains

Overview

This subsection introduces the concept of an Artix security domain.

Domain architecture

Figure 19 shows the architecture of an Artix security domain. The Artix security domain is identified with an enterprise security service that plugs into the Is2 server through an iSF adapter. User data needed for authentication, such as username and password, are stored within the enterprise security service. The Is2 server provides a central access point to enable authentication within the Artix security domain.

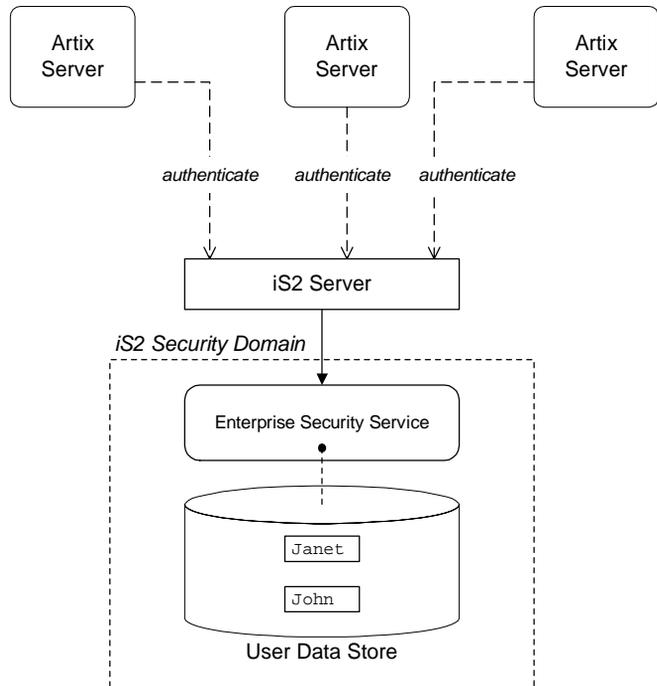


Figure 19: Architecture of an Artix security domain

Artix security domain

An *Artix security domain* is a particular security system, or namespace within a security system, designated to authenticate a user.

Here are some specific examples of Artix security domains:

- LDAP security domain—authentication provided by an LDAP security backend, accessed through the Is2 server.
- SiteMinder security domain—authentication provided by a SiteMinder security backend, accessed through the Is2 server.

Creating an Artix security domain

Effectively, you create an Artix security domain by configuring the Is2 server to link to an enterprise security service through an iSF adapter (such as a SiteMinder adapter or an LDAP adapter). The enterprise security service is the implementation of the Artix security domain.

Creating a user account

User account data is stored in a third-party enterprise security service. Hence, you should use the standard tools from the third-party enterprise security product to create a user account.

For a simple example, see [“Managing a File Security Domain” on page 128](#).

Artix Authorization Realms

Overview

This subsection introduces the concept of an Artix authorization realm and role-based access control, explaining how users, roles, realms, and servers are interrelated.

Artix authorization realm

An *Artix authorization realm* is a collection of secured resources that share a common interpretation of role names. An authenticated user can have different roles in different realms. When using a resource in realm \mathbb{R} , only the user's roles in realm \mathbb{R} are applied to authorization decisions.

Role-based access control

The Artix Security Framework supports a *role-based access control* (RBAC) authorization scheme. Under RBAC, authorization is a two step process, as follows:

1. User-to-role mapping—every user is associated with a set of roles in each realm (for example, `guest`, `administrator`, and so on, in a realm, `Engineering`). A user can belong to many different realms, having a different set of roles in each realm.

The user-to-role assignments are managed centrally by the Is2 server, which returns the set of realms and roles assigned to a user when required.

2. Role-to-permission mapping (or action-role mapping)—in the RBAC model, permissions are granted to *roles*, rather than directly to users. The role-to-permission mapping is performed locally by a server, using data stored in local access control list (ACL) files. For example, Artix servers in the iSF use an XML action-role mapping file to control access to WSDL port types and operations.

Servers and realms

From a server's perspective, an Artix authorization realm is a way of grouping servers with similar authorization requirements. [Figure 20](#) shows two Artix authorization realms, `Engineering` and `Finance`, each containing a collection of server applications.

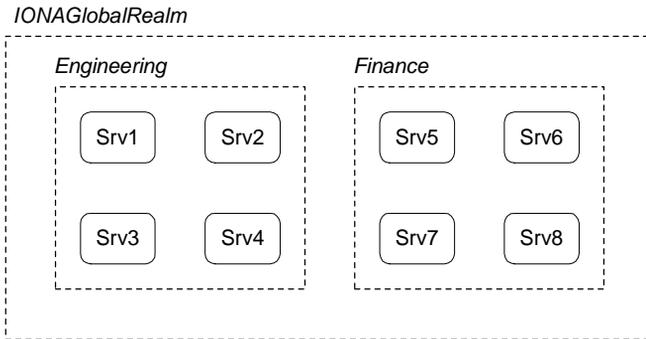


Figure 20: Server View of Artix authorization realms

Adding a server to a realm

To add an Artix server to a realm, add or modify the `plugins:asp:authorization_realm` configuration variable within the server's configuration scope (in the `artix.cfg` file).

For example, if your server's configuration is defined in the `my_server_scope` scope, you can set the Artix authorization realm to `Engineering` as follows:

```
# Artix configuration file
...
my_server_scope {
    plugins:asp:authorization_realm = "Engineering";
    ...
};
```

Roles and realms

From the perspective of role-based authorization, an Artix authorization realm acts as a namespace for roles. For example, [Figure 21](#) shows two Artix authorization realms, `Engineering` and `Finance`, each associated with a set of roles.

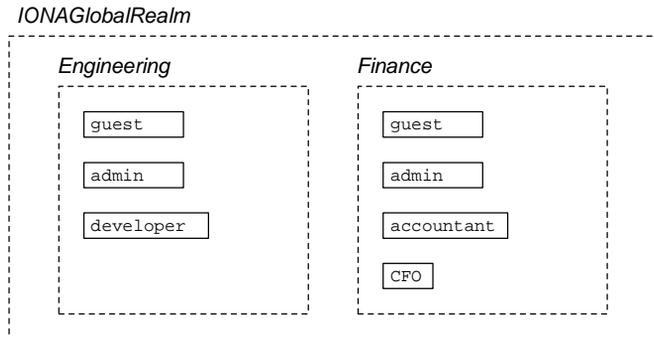


Figure 21: *Role View of Artix authorization realms*

Creating realms and roles

Realms and roles are usually administered from within the enterprise security system that is plugged into the Is2 server through an adapter. Not every enterprise security system supports realms and roles, however.

For example, in the case of a security file connected to a file adapter (a demonstration adapter provided by IONA), a realm or role is implicitly created whenever it is listed amongst a user's realms or roles.

Assigning realms and roles to users

The assignment of realms and roles to users is administered from within the enterprise security system that is plugged into the Is2 server. For example, [Figure 22](#) shows how two users, Janet and John, are assigned roles within the Engineering and Finance realms.

- Janet works in the engineering department as a developer, but occasionally logs on to the Finance realm with guest permissions.
- John works as an accountant in finance, but also has guest permissions with the Engineering realm.

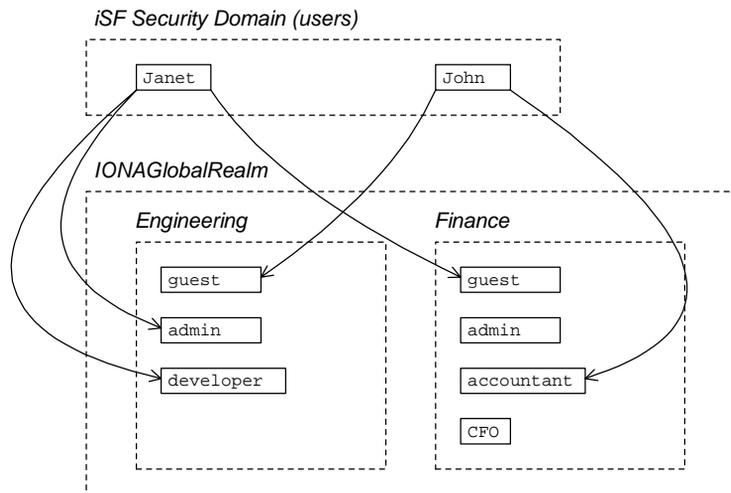


Figure 22: Assignment of Realms and Roles to Users Janet and John

Special realms and roles

The following special realms and roles are supported by the Artix Security Framework:

- `IONAGlobalRealm` realm—a special realm that encompasses every Artix authorization realm. Roles defined within the `IONAGlobalRealm` are valid within every Artix authorization realm.
- `UnauthenticatedUserRole`—a special role that can be used to specify actions accessible to an unauthenticated user (in an action-role mapping file). An unauthenticated user is a remote user without credentials (that is, where the client is not configured to send GSSUP credentials).

Actions mapped to the `UnauthenticatedUserRole` role are also accessible to authenticated users.

The `UnauthenticatedUserRole` can be used *only* in action-role mapping files.

Managing a File Security Domain

Overview

The file security domain is active if the Is2 server has been configured to use the iS2 file adapter (see [“Configuring the File Adapter” on page 100](#)). The main purpose of the iS2 file adapter is to provide a lightweight security domain for demonstration purposes. A realistic deployed system, however, would use one of the other adapters (LDAP, SiteMinder, or custom) instead.

WARNING: The file adapter is provided for demonstration purposes only. IONA does not support the use of the file adapter in a production environment.

Location of file

The location of the security information file is specified by the `com.iona.isp.adapter.file.param.filename` property in the Is2 server's `is2.properties` file.

Example

[Example 37](#) is an extract from a sample security information file that shows you how to define users, realms, and roles in a file security domain.

Example 37: Sample Security Information File for an iS2 File Domain

```
<?xml version="1.0" encoding="utf-8" ?>
1 <ns:securityInfo xmlns:ns="urn:www-xmlbus-com:simple-security">
2   <users>
3     <user name="IONAAdmin" password="admin"
4       description="Default IONA admin user">
5       <realm name="IONA" description="All IONA applications"/>
      </user>
      <user name="admin" password="admin" description="Old admin
user; will not have the same default privileges as
IONAAdmin.">
        <realm name="Corporate">
          <role name="Administrator"/>
        </realm>
      </user>
      <user name="alice" password="dost1234">
        <realm name="Financials"
          description="Financial Department">
```

Example 37: Sample Security Information File for an iS2 File Domain

```

    <role name="Manager" description="Department Manager" />
    <role name="Clerk"/>
  </realm>
</user>
<user name="bob" password="dost1234">
  <realm name="Financials">
    <role name="Clerk"/>
  </realm>
</user>
</users>
</ns:securityInfo>

```

1. The `<ns:securityInfo>` tag can contain a nested `<users>` tag.
2. The `<users>` tag contains a sequence of `<user>` tags.
3. Each `<user>` tag defines a single user. The `<user>` tag's name and password attributes specify the user's username and password. Within the scope of the `<user>` tag, you can list the realms and roles with which the user is associated.
4. When a `<realm>` tag appears within the scope of a `<user>` tag, it implicitly defines a realm and specifies that the user belongs to this realm. A `<realm>` must have a `name` and can optionally have a `description` attribute.
5. A realm can optionally be associated with one or more roles by including `<role>` elements within the `<realm>` scope.

Certificate-based authentication for the file adapter

When performing certificate-based authentication for the CORBA binding, the file adapter compares the certificate to be authenticated with a cached copy of the user's certificate.

Note: This configuration step is *not* required for non-CORBA bindings. Currently, the ASP security layer does not send the client's X.509 to the iS2 server.

To configure the file adapter to support X.509 certificate-based authentication for the CORBA binding, perform the following steps:

1. Cache a copy of each user's certificate, *CertFile.pem*, in a location that is accessible to the file adapter.
2. Make the following type of entry for each user with a certificate:

Example 38: *File Adapter Entry for Certificate-Based Authentication*

```
...
<user name="CNfromSubjectDN" certificate="CertFile.pem"
  description="User certificate">
  <realm name="RealmName">
    ...
  </realm>
</user>
```

The user's name, *CNfromSubjectDN*, is derived from the certificate by taking the Common Name (CN) from the subject DN of the X.509 certificate (for DN terminology, see [“ASN.1 and Distinguished Names” on page 529](#)). The `certificate` attribute specifies the location of this user's X.509 certificate, *CertFile.pem*.

Managing an LDAP Security Domain

Overview

The Lightweight Directory Access Protocol (LDAP) can serve as the basis of a database that stores users, groups, and roles. There are many implementations of LDAP and any of them can be integrated with the Is2 server by configuring the Is2 server's LDAP adapter.

Please consult documentation from your third-party LDAP implementation for detailed instructions on how to administer users and roles within LDAP.

Configuring the LDAP adapter

A prerequisite for using LDAP within the Artix Security Framework is that the iS2 server be configured to use the LDAP adapter.

See [“Configuring the LDAP Adapter” on page 102](#).

Certificate-based authentication for the LDAP adapter

When performing certificate-based authentication for CORBA bindings, the LDAP adapter compares the certificate to be authenticated with a cached copy of the user's certificate.

Note: This configuration step is *not* required for non-CORBA bindings. Currently, the ASP security layer does not send the client's X.509 to the iS2 server.

To configure the LDAP adapter to support X.509 certificate-based authentication, perform the following steps:

1. Cache a copy of each user's certificate, *CertFile.pem*, in a location that is accessible to the LDAP adapter.
2. The user's name, *CNfromSubjectDN*, is derived from the certificate by taking the Common Name (CN) from the subject DN of the X.509 certificate (for DN terminology, see [“ASN.1 and Distinguished Names” on page 529](#)).
3. Make (or modify) an entry in your LDAP database with the username, *CNfromSubjectDN*, and specify the location of the cached certificate.

Managing a SiteMinder Security Domain

Overview

SiteMinder is an enterprise security product from Netegrity, which allows you to manage user data stored in a central database. The iS2 server can communicate with the SiteMinder agent, using it to perform authentication.

Please consult the Netegrity SiteMinder documentation for detailed instructions on how to administer users and roles within the SiteMinder product.

Configuring the SiteMinder adapter

A prerequisite for using SiteMinder within the Artix Security Framework is that the iS2 server be configured to use the SiteMinder adapter.

See [“Configuring the SiteMinder Adapter” on page 108](#).

References

For more information on Netegrity SiteMinder, see the Netegrity Web site: <http://www.netegrity.com/>

Managing Access Control Lists

The Artix Security Framework defines access control lists (ACLs) for mapping roles to resources.

In this chapter

This chapter discusses the following topics:

Overview of Artix ACL Files	page 134
Artix Action-Role Mapping ACL	page 135
Generating ACL Files	page 138

Overview of Artix ACL Files

Action-role mapping file

The action-role mapping file is an XML file that specifies which user roles have permission to perform specific actions on the server (that is, invoking specific WSDL operations).

Artix Action-Role Mapping ACL

Overview

This subsection explains how to configure the action-role mapping ACL file for Artix applications. Using an action-role mapping file, you can specify that access to WSDL operations is restricted to specific roles.

File location

In your `artix.cfg` configuration file (located in the *ArtixInstallDir/artix/2.0/etc/domains* directory), the `plugins:is2_authorization:action_role_mapping` configuration variable specifies the location URL of the action-role mapping file, `action_role_mapping.xml`, for an Artix server. For example:

```
# artix.cfg Configuration File
...
my_server_scope {
    plugins:is2_authorization:action_role_mapping =
        "file:///security_admin/action_role_mapping.xml";
};
```

Example WSDL

For example, consider how to set the operation permissions for the WSDL port type shown in [Example 39](#).

Example 39: Sample WSDL for the ACL Example

```
<definitions name="HelloWorldService"
  targetNamespace="http://xmlbus.com/HelloWorld" ... >
  ...
  <portType name="HelloWorldPortType">
    <operation name="greetMe">
      <input message="tns:greetMe" name="greetMe" />
      <output message="tns:greetMeResponse"
        name="greetMeResponse" />
    </operation>
    <operation name="sayHi">
      <input message="tns:sayHi" name="sayHi" />
      <output message="tns:sayHiResponse"
        name="sayHiResponse" />
    </operation>
  </portType>
  ...
</definitions>
```

Example action-role mapping

[Example 40](#) shows how you might configure an action-role mapping file for the `HelloWorldPortType` port type given in the preceding [Example 39](#) on [page 136](#).

Example 40: Artix Action-Role Mapping Example

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE secure-system SYSTEM "actionrolemapping.dtd">
<secure-system>
  <action-role-mapping>
    <server-name>secure_artix.demos.hello_world</server-name>
    <interface>
      <name>http://xmlbus.com/HelloWorld:HelloWorldPortType</name>
      <action-role>
        <action-name>sayHi</action-name>
        <role-name>IONAUserRole</role-name>
      </action-role>
      <action-role>
        <action-name>greetMe</action-name>
        <role-name>IONAUserRole</role-name>
```

Example 40: *Artix Action-Role Mapping Example*

```

</action-role>
</interface>
</action-role-mapping>
</secure-system>

```

The preceding action-role mapping example can be explained as follows:

1. The `<action-role-mapping>` tag contains all of the permissions that apply to a particular server application.
2. The `<server-name>` tag specifies the ORB name that is used by the server in question. The value of this tag must match the ORB name exactly. The ORB name is usually passed to an Artix server as the value of the `-ORBname` command-line parameter.

Note: The ORB name also determines which configuration scopes are read by the server.

3. The `<interface>` tag contains all of the access permissions for one particular WSDL port type.
4. The `<name>` tag identifies a WSDL port type in the format *NamespaceURI:PortTypeName*. That is, the *PortTypeName* comes from a tag, `<portType name="PortTypeName">`, defined in the *NamespaceURI* namespace.

For example, in [Example 39 on page 136](#) the `<definitions>` tag specifies the *NamespaceURI* as `http://xmlbus.com>HelloWorld` and the *PortTypeName* is `HelloWorldPortType`. Hence, the port type name is identified as:

```
<name>http://xmlbus.com>HelloWorld:HelloWorldPortType</name>
```

5. The `sayHi` action name corresponds to the `sayHi` WSDL operation name in the `HelloWorldPortType` port type (from the `<operation name="sayHi">` tag).

Action-role mapping DTD

The syntax of the action-role mapping file is defined by the action-role mapping DTD. See [“Action-Role Mapping DTD” on page 289](#) for details.

Generating ACL Files

Overview

Artix provides a command-line tool, `wSDLtoacl`, that enables you to generate the prototype of an ACL file directly from a WSDL contract. You can use the `wSDLtoacl` utility to assign a default role to all of the operations in WSDL contract. Alternatively, if you require more fine-grained control over the role assignments, you can define a *role-properties file*, which assigns roles to individual operations.

WSDL-to-ACL utility

The `wSDLtoacl` command-line utility has the following syntax:

```
wSDLtoacl { -s server-name } WSDL-URL
[-i interface-name] [-r default-role-name]
[-d output-directory] [-o output-file]
[-props role-props-file] [-v] [-?]
```

Required arguments:

- `-s server-name` The server's configuration scope from the Artix domain configuration file (the same value as specified to the `-ORBname` argument when the Artix server is started from the command line).
For example, the `basic/hello_world_soap_http` demonstration uses the `demos.hello_world_soap_http` server name.
- `WSDL-URL` URL location of the WSDL file from which an ACL is generated.

Optional arguments:

- `-i interface-name` Generates output for a specific WSDL port type, *interface-name*. If this option is omitted, output is generated for all of the port types in the WSDL file.
- `-r default-role-name` Specify the role name that will be assigned to all operations by default. Default is `IONAUserRole`.
The default role-name is not used for operations listed in a role-properties file (see `-props`).

<code>-d output-directory</code>	Specify an output directory for the generated ACL file.
<code>-o output-file</code>	Specify the name of the generated ACL file. Default is <code>WSDLFileRoot-acl.xml</code> , where <code>WSDLFileRoot</code> is the root name of the WSDL file.
<code>-props role-props-file</code>	Specifies a file containing a list of <i>role-properties</i> , where a role-property associates an operation name with a list of roles. Each line of the role-properties file has the following format: <i>OperationName = Role1, Role2, ...</i>
<code>-v</code>	Display version information for the utility.
<code>-?</code>	Display usage summary for the <code>wsdltoacl</code> utility.

Example of generating an ACL file

As example of how to generate an ACL file from WSDL, consider the `hello_world.wsdl` WSDL file for the `basic/hello_world_soap_http` demonstration, which is located in the following directory:

```
ArtixInstallDir/artix/Version/demos/basic/hello_world_soap_http/etc
```

The HelloWorld WSDL contract defines a single port type, `Greeter`, and two operations: `greetMe` and `sayHi`. The server name (that is, configuration scope) used by the HelloWorld server is `demos.hello_world_soap_http`.

Sample role-properties file

For the HelloWorld WSDL contract, you can define a role-properties file, `role_properties.txt`, that assigns the `FooUser` role to the `greetMe` operation and the `FooUser` and `BarUser` roles to the `sayHi` operation, as follows:

```
greetMe = FooUser
sayHi = FooUser, BarUser
```

Sample generation command

To generate an ACL file from the HelloWorld WSDL contract, using the `role_properties.txt` role-properties file, enter the following at a command-line prompt:

```
wsdltoacl -s demos.hello_world_soap_http hello_world.wsdl -props
role_properties.txt
```

Sample ACL output

The preceding `wsdltoacl` command generates an ACL file, `hello_world-acl.xml`, whose contents are shown in [Example 41](#).

Example 41: ACL File Generated from HelloWorld WSDL Contract

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE secure-system SYSTEM "actionrolemapping.dtd">
<secure-system>
  <action-role-mapping>
    <server-name>demos.hello_world_soap_http</server-name>
    <interface>
      <name>http://www.iona.com/hello_world_soap_http:Greeter</name>
      <action-role>
        <action-name>greetMe</action-name>
        <role-name>FooUser</role-name>
      </action-role>
      <action-role>
        <action-name>sayHi</action-name>
        <role-name>FooUser</role-name>
        <role-name>BarUser</role-name>
      </action-role>
    </interface>
  </action-role-mapping>
</secure-system>
```

Managing Certificates

TLS authentication uses X.509 certificates—a common, secure and reliable method of authenticating your application objects. This chapter explains how you can create X.509 certificates that identify your Artix applications.

In this chapter

This chapter contains the following sections:

What are X.509 Certificates?	page 142
Certification Authorities	page 144
Certificate Chaining	page 147
PKCS#12 Files	page 149
Creating Your Own Certificates	page 151
Deploying Certificates	page 158

What are X.509 Certificates?

Role of certificates

An X.509 certificate binds a name to a public key value. The role of the certificate is to associate a public key with the identity contained in the X.509 certificate.

Integrity of the public key

Authentication of a secure application depends on the integrity of the public key value in the application's certificate. If an impostor replaced the public key with its own public key, it could impersonate the true application and gain access to secure data.

To prevent this form of attack, all certificates must be signed by a *certification authority (CA)*. A CA is a trusted node that confirms the integrity of the public key value in a certificate.

Digital signatures

A CA signs a certificate by adding its *digital signature* to the certificate. A digital signature is a message encoded with the CA's private key. The CA's public key is made available to applications by distributing a certificate for the CA. Applications verify that certificates are validly signed by decoding the CA's digital signature with the CA's public key.

WARNING: Most of the demonstration certificates supplied with Artix are signed by the CA `cacert.pem`. This CA is completely insecure because anyone can access its private key. To secure your system, you must create new certificates signed by a trusted CA. This chapter describes the set of certificates required by an Artix application and shows you how to replace the default certificates.

The contents of an X.509 certificate

An X.509 certificate contains information about the certificate subject and the certificate issuer (the CA that issued the certificate). A certificate is encoded in Abstract Syntax Notation One (ASN.1), a standard syntax for describing messages that can be sent or received on a network.

The role of a certificate is to associate an identity with a public key value. In more detail, a certificate includes:

- X.509 version information.
- A *serial number* that uniquely identifies the certificate.
- A *subject DN* that identifies the certificate owner.
- The *public key* associated with the subject.
- An *issuer DN* that identifies the CA that issued the certificate.
- The digital signature of the issuer.
- Information about the algorithm used to sign the certificate.
- Some optional X.509 v.3 extensions. For example, an extension exists that distinguishes between CA certificates and end-entity certificates.

Distinguished names

A distinguished name (DN) is a general purpose X.500 identifier that is often used in the context of security.

See [“ASN.1 and Distinguished Names” on page 283](#) for more details about DNs.

Certification Authorities

Choice of CAs

A CA must be trusted to keep its private key secure. When setting up an Artix system, it is important to choose a suitable CA, make the CA certificate available to all applications, and then use the CA to sign certificates for your applications.

There are two types of CA you can use:

- A *commercial CA* is a company that signs certificates for many systems.
- A *private CA* is a trusted node that you set up and use to sign certificates for your system only.

In this section

This section contains the following subsections:

Commercial Certification Authorities	page 145
Private Certification Authorities	page 146

Commercial Certification Authorities

Signing certificates

There are several commercial CAs available. The mechanism for signing a certificate using a commercial CA depends on which CA you choose.

Advantages of commercial CAs

An advantage of commercial CAs is that they are often trusted by a large number of people. If your applications are designed to be available to systems external to your organization, use a commercial CA to sign your certificates. If your applications are for use within an internal network, a private CA might be appropriate.

Criteria for choosing a CA

Before choosing a CA, you should consider the following criteria:

- What are the certificate-signing policies of the commercial CAs?
- Are your applications designed to be available on an internal network only?
- What are the potential costs of setting up a private CA?

Private Certification Authorities

Choosing a CA software package

If you wish to take responsibility for signing certificates for your system, set up a private CA. To set up a private CA, you require access to a software package that provides utilities for creating and signing certificates. Several packages of this type are available.

OpenSSL software package

One software package that allows you to set up a private CA is OpenSSL, <http://www.openssl.org>. OpenSSL is derived from SSLeay, an implementation of SSL developed by Eric Young (eay@cryptsoft.com). Complete license information can be found in [“License Issues” on page 309](#). The OpenSSL package includes basic command line utilities for generating and signing certificates and these utilities are available with every installation of Artix. Complete documentation for the OpenSSL command line utilities is available from <http://www.openssl.org/docs>.

Setting up a private CA using OpenSSL

For instructions on how to set up a private CA, see [“Creating Your Own Certificates” on page 151](#).

Choosing a host for a private certification authority

Choosing a host is an important step in setting up a private CA. The level of security associated with the CA host determines the level of trust associated with certificates signed by the CA.

If you are setting up a CA for use in the development and testing of Artix applications, use any host that the application developers can access. However, when you create the CA certificate and private key, do not make the CA private key available on hosts where security-critical applications run.

Security precautions

If you are setting up a CA to sign certificates for applications that you are going to deploy, make the CA host as secure as possible. For example, take the following precautions to secure your CA:

- Do not connect the CA to a network.
- Restrict all access to the CA to a limited set of trusted users.
- Protect the CA from radio-frequency surveillance using an RF-shield.

Certificate Chaining

Certificate chain

A *certificate chain* is a sequence of certificates, where each certificate in the chain is signed by the subsequent certificate.

Self-signed certificate

The last certificate in the chain is normally a *self-signed certificate*—a certificate that signs itself.

Example

Figure 23 shows an example of a simple certificate chain.

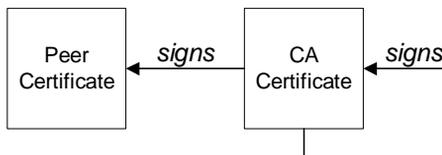


Figure 23: A Certificate Chain of Depth 2

Chain of trust

The purpose of certificate chain is to establish a chain of trust from a peer certificate to a trusted CA certificate. The CA vouches for the identity in the peer certificate by signing it. If the CA is one that you trust (indicated by the presence of a copy of the CA certificate in your root certificate directory), this implies you can trust the signed peer certificate as well.

Certificates signed by multiple CAs

A CA certificate can be signed by another CA. For example, an application certificate may be signed by the CA for the finance department of IONA Technologies, which in turn is signed by a self-signed commercial CA. [Figure 24](#) shows what this certificate chain looks like.

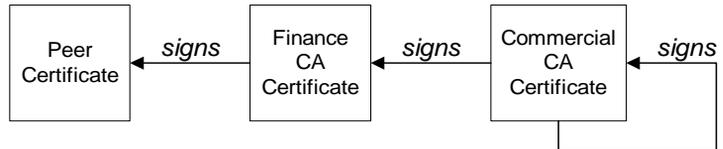


Figure 24: A Certificate Chain of Depth 3

Trusted CAs

An application can accept a signed certificate if the CA certificate for any CA in the signing chain is available in the certificate file in the local root certificate directory.

See [“Deploying Trusted Certificate Authority Certificates” on page 160](#).

Maximum chain length policy

You can limit the length of certificate chains accepted by your CORBA applications, with the maximum chain length policy. You can set a value for the maximum length of a certificate chain with the `policies:iiop_tls:max_chain_length_policy` configuration variable for IIOP/TLS.

PKCS#12 Files

Overview

Figure 25 shows the typical elements in a PKCS#12 file.

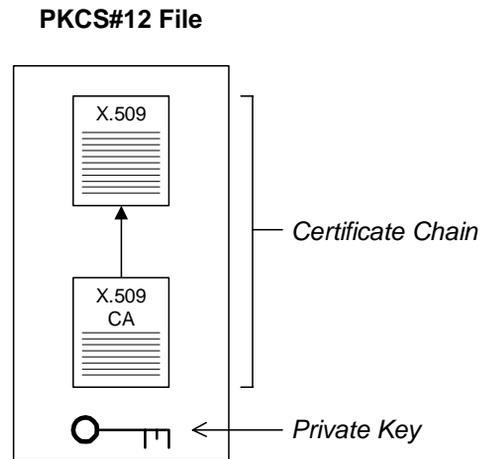


Figure 25: Elements in a PKCS#12 File

Contents of a PKCS#12 file

A PKCS#12 file contains the following:

- An X.509 peer certificate (first in a chain).
- All the CA certificates in the certificate chain.
- A private key.

The file is encrypted with a pass phrase.

PKCS#12 is an industry-standard format and is used by browsers such as Netscape and Internet Explorer.

Note: The same pass phrase is used both for the encryption of the private key within the PKCS#12 file and for the encryption of the PKCS#12 file overall. This condition (same pass phrase) is not officially part of the PKCS#12 standard, but it is enforced by most Web browsers and by Artix.

Creating a PKCS#12 file

To create a PKCS#12 file, see [“Use the CA to Create Signed Certificates” on page 155](#).

Viewing a PKCS#12 file

To view a PKCS#12 file, *CertName*.p12:

```
openssl pkcs12 -in CertName.p12
```

Importing and exporting PKCS#12 files

The generated PKCS#12 files can be imported into browsers such as IE or Netscape. Exported PKCS#12 files from these browsers can be used in Artix.

Note: Use OpenSSL v0.9.2 or later; Internet Explorer 5.0 or later; Netscape 4.7 or later.

Creating Your Own Certificates

Overview

This section describes the steps involved in setting up a CA and signing certificates.

OpenSSL utilities

The steps described in this section are based on the OpenSSL command-line utilities from the OpenSSL project, <http://www.openssl.org>—see “[OpenSSL Utilities](#)” on page 293. Further documentation of the OpenSSL command-line utilities can be obtained from <http://www.openssl.org/docs>.

Sample CA directory structure

For the purposes of illustration, the CA database is assumed to have the following directory structure:

```
X509CA/ca
X509CA/certs
X509CA/newcerts
X509CA/crl
```

Where *X509CA* is the parent directory of the CA database.

In this section

This section contains the following subsections:

Set Up Your Own CA	page 152
Use the CA to Create Signed Certificates	page 155

Set Up Your Own CA

Substeps to perform

This section describes how to set up your own private CA. Before setting up a CA for a real deployment, read the additional notes in [“Choosing a host for a private certification authority” on page 146](#).

To set up your own CA, perform the following substeps:

- [Step 1—Add the bin directory to your PATH](#)
 - [Step 2—Create the CA directory hierarchy](#)
 - [Step 3—Copy and edit the openssl.cnf file](#)
 - [Step 4—Initialize the CA database](#)
 - [Step 5—Create a self-signed CA certificate and private key](#)
-

Step 1—Add the bin directory to your PATH

On the secure CA host, add the OpenSSL `bin` directory to your path:

Windows

```
> set PATH=OpenSSLDir\bin;%PATH%
```

UNIX

```
% PATH=OpenSSLDir/bin:$PATH; export PATH
```

This step makes the `openssl` utility available from the command line.

Step 2—Create the CA directory hierarchy

Create a new directory, `X509CA`, to hold the new CA. This directory will be used to hold all of the files associated with the CA. Under the `X509CA` directory, create the following hierarchy of directories:

```
X509CA/ca  
X509CA/certs  
X509CA/newcerts  
X509CA/cr1
```

Step 3—Copy and edit the openssl.cnf file

Copy the sample `openssl.cnf` from your OpenSSL installation to the `X509CA` directory.

Edit the `openssl.cnf` to reflect the directory structure of the `X509CA` directory and to identify the files used by the new CA.

Edit the [CA_default] section of the openssl.cnf file to make it look like the following:

```
#####
[ CA_default ]

dir           = X509CA           # Where CA files are kept
certs        = $dir/certs       # Where issued certs are kept
crl_dir      = $dir/crl         # Where the issued crl are kept
database     = $dir/index.txt   # Database index file
new_certs_dir = $dir/newcerts   # Default place for new certs

certificate  = $dir/ca/new_ca.pem # The CA certificate
serial       = $dir/serial       # The current serial number
crl          = $dir/crl.pem      # The current CRL
private_key  = $dir/ca/new_ca_pk.pem # The private key
RANDFILE    = $dir/ca/.rand     # Private random number file

x509_extensions = usr_cert     # The extensions to add to the cert
...
```

You might like to edit other details of the OpenSSL configuration at this point—for more details, see [“The OpenSSL Configuration File” on page 303](#).

Step 4—Initialize the CA database

In the X509CA directory, initialize two files, serial and index.txt.

Windows

```
> echo 01 > serial
```

To create an empty file, index.txt, in Windows start a Windows Notepad at the command line in the X509CA directory, as follows:

```
> notepad index.txt
```

In response to the dialog box with the text, Cannot find the text.txt file. Do you want to create a new file?, click Yes, and close Notepad.

UNIX

```
% echo "01" > serial
% touch index.txt
```

These files are used by the CA to maintain its database of certificate files.

Note: The index.txt file must initially be completely empty, not even containing white space.

Step 5—Create a self-signed CA certificate and private key

Create a new self-signed CA certificate and private key:

```
openssl req -x509 -new -config
X509CA/openssl.cnf -days 365 -out X509CA/ca/new_ca.pem
-keyout X509CA/ca/new_ca_pk.pem
```

The command prompts you for a pass phrase for the CA private key and details of the CA distinguished name:

```
Using configuration from X509CA/openssl.cnf
Generating a 512 bit RSA private key
...+++++
.+++++
writing new private key to 'new_ca_pk.pem'
Enter PEM pass phrase:
Verifying password - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be
incorporated into your certificate request.
What you are about to enter is what is called a Distinguished
Name or a DN. There are quite a few fields but you can leave
some blank. For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) []:IE
State or Province Name (full name) []:Co. Dublin
Locality Name (eg, city) []:Dublin
Organization Name (eg, company) []:IONA Technologies PLC
Organizational Unit Name (eg, section) []:Finance
Common Name (eg, YOUR name) []:Gordon Brown
Email Address []:gbrown@iona.com
```

Note: The security of the CA depends on the security of the private key file and private key pass phrase used in this step.

You should ensure that the file names and location of the CA certificate and private key, `new_ca.pem` and `new_ca_pk.pem`, are the same as the values specified in `openssl.cnf` (see the preceding step).

You are now ready to sign certificates with your CA.

Use the CA to Create Signed Certificates

Substeps to perform

If you have set up a private CA, as described in “Set Up Your Own CA” on [page 152](#), you are now ready to create and sign your own certificates.

To create and sign a certificate in PKCS#12 format, *CertName.p12*, perform the following substeps:

- [Step 1—Add the bin directory to your PATH](#)
- [Step 2—Create a certificate signing request](#)
- [Step 3—Sign the CSR](#)
- [Step 4—Concatenate the files](#)
- [Step 5—Create a PKCS#12 file](#)
- [Step 6—Repeat steps as required](#)

Step 1—Add the bin directory to your PATH

If you have not already done so, add the OpenSSL `bin` directory to your path:

Windows

```
> set PATH=OpenSSLDir\bin;%PATH%
```

UNIX

```
% PATH=OpenSSLDir/bin:$PATH; export PATH
```

This step makes the `openssl` utility available from the command line.

Step 2—Create a certificate signing request

Create a new certificate signing request (CSR) for the *CertName.p12* certificate:

```
openssl req -new -config X509CA/openssl.cnf  
-days 365 -out X509CA/certs/CertName_csr.pem -keyout  
X509CA/certs/CertName_pk.pem
```

This command prompts you for a pass phrase for the certificate’s private key and information about the certificate’s distinguished name.

Some of the entries in the CSR distinguished name must match the values in the CA certificate (specified in the CA Policy section of the `openssl.cnf` file). The default `openssl.cnf` file requires the following entries to match:

- Country Name
- State or Province Name
- Organization Name

The Common Name must be distinct for every certificate generated by OpenSSL.

```
Using configuration from X509CA/openssl.cnf
Generating a 512 bit RSA private key
.++++
.++++
writing new private key to 'X509CA/certs/CertName_pk.pem'
Enter PEM pass phrase:
Verifying password - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be
incorporated into your certificate request.
What you are about to enter is what is called a Distinguished
Name or a DN. There are quite a few fields but you can leave
some blank. For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) []:IE
State or Province Name (full name) []:Co. Dublin
Locality Name (eg, city) []:Dublin
Organization Name (eg, company) []:IONA Technologies PLC
Organizational Unit Name (eg, section) []:Systems
Common Name (eg, YOUR name) []:Artix
Email Address []:info@iona.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:password
An optional company name []:IONA
```

Step 3—Sign the CSR

Sign the CSR using your CA:

```
openssl ca -config X509CA/openssl.cnf -days 365 -in
X509CA/certs/CertName_csr.pem -out
X509CA/certs/CertName.pem
```

This command requires the pass phrase for the private key associated with the `new_ca.pem` CA certificate:

```
Using configuration from X509CA/openssl.cnf
Enter PEM pass phrase:
Check that the request matches the signature
Signature ok
The Subjects Distinguished Name is as follows
countryName          :PRINTABLE:'IE'
stateOrProvinceName  :PRINTABLE:'Co. Dublin'
localityName         :PRINTABLE:'Dublin'
```

```

organizationName      :PRINTABLE:'IONA Technologies PLC'
organizationalUnitName:PRINTABLE:'Systems'
commonName            :PRINTABLE:'Bank Server Certificate'
emailAddress          :IA5STRING:'info@iona.com'
Certificate is to be certified until May 24 13:06:57 2000 GMT (365
    days)
Sign the certificate? [y/n]:y
1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated

To sign the certificate successfully, you must enter the CA private key pass
phrase—see “Set Up Your Own CA” on page 152.

```

Step 4—Concatenate the files

Concatenate the CA certificate file, *CertName* certificate file, and *CertName_pk.pem* private key file as follows:

Windows

```

copy X509CA\ca\new_ca.pem +
    X509CA\certs\CertName.pem +
    X509CA\certs\CertName_pk.pem
    X509CA\certs\CertName_list.pem

```

UNIX

```

cat X509CA/ca/new_ca.pem
    X509CA/certs/CertName.pem
    X509CA/certs/CertName_pk.pem >
    X509CA/certs/CertName_list.pem

```

Step 5—Create a PKCS#12 file

Create a PKCS#12 file from the *CertName_list.pem* file as follows:

```

openssl pkcs12 -export -in X509CA/certs/CertName_list.pem -out
    X509CA/certs/CertName.p12 -name "New cert"

```

Step 6—Repeat steps as required

Repeat steps 2 to 5, creating a complete set of certificates for your system. A minimum set of Artix certificates must include a set of certificates for the secure Artix services.

Deploying Certificates

Overview

This section provides an overview of deploying X.509 certificates in a typical secure Artix system, with detailed instructions on how to deploy certificates for different parts of the Artix system.

In this section

This section contains the following subsections:

Overview of Certificate Deployment	page 159
Deploying Trusted Certificate Authority Certificates	page 160
Deploying Application Certificates	page 165

Overview of Certificate Deployment

Overview

Because the HTTPS and IIOP/TLS transports use different security mechanisms, it is necessary to deploy certificates for each of these transports independently, as follows:

- [Certificate deployment for HTTPS](#).
 - [Certificate deployment for IIOP/TLS](#).
-

Certificate deployment for HTTPS

Certificates used by the HTTPS transport must be in Privacy Enhanced Mail (PEM) format. To specify certificates for the HTTPS transport, you must edit your application's WSDL contract.

Certificate deployment for IIOP/TLS

Certificates used by the IIOP/TLS transport must be in PKCS#12 format. To specify certificates for the IIOP/TLS transport, you must edit the Artix configuration file, *ArtixInstallDir/artix/2.0/etc/domains/artix.cfg*.

Sample deployment directory structure

For the purposes of illustration, the examples in this section deploy certificates into the following sample directory structure:

```
X509Deploy/trusted_ca_lists
```

```
X509Deploy/certs
```

Where *X509Deploy* is the parent directory for the deployed certificates.

Deploying Trusted Certificate Authority Certificates

Overview

This section how to deploy trusted root CA certificates for Artix applications. In the current version of Artix, the procedure for deploying trusted CA certificates depends on the type of transport, as follows:

- [Deploying for the HTTPS transport.](#)
 - [Deploying for the IIOP/TLS transport.](#)
-

Deploying for the HTTPS transport

To deploy one or more trusted root CAs for the HTTPS transport in Artix, perform the following steps:

1. Assemble the collection of trusted CA certificates that you want to deploy. The trusted CA certificates could be obtained from public CAs or private CAs (for details of how to generate your own CA certificates, see [“Set Up Your Own CA” on page 152](#)). The trusted CA certificates should be in PEM format. All you need are the certificates themselves—the private keys and passwords are not required.
2. Concatenate the CA certificates into a single CA list file. A CA list file can be created using a simple file concatenation operation. For example, if you have two CA certificate files, `ca_cert01.pem` and `ca_cert02.pem`, you could combine them into a single CA list file, `ca_list01.pem`, with the following command:

Windows

```
copy X509CA\ca\ca_cert01.pem +
    X509CA\ca\ca_cert02.pem
    X509Deploy\trusted_ca_lists\ca_list01.pem
```

UNIX

```
cat X509CA/ca/ca_cert01.pem X509CA/ca/ca_cert02.pem >>
    X509Deploy/trusted_ca_lists/ca_list01.pem
```

3. Edit the WSDL contract to specify the location of the CA list file. The details of this step depend on whether you are deploying a trusted CA list on the client side or on the server side:

Client side

Edit the client's copy of the WSDL contract by adding (or modifying) the `TrustedRootCertificates` attribute in the `<http-conf:client>`

tag. For example, to specify `X509CA/ca/ca_list01.pem` as the client's trusted CA certificate, modify the client's WSDL contract as follows:

```
<definitions
xmlns:http="http://schemas.ionas.com/transport/http"
xmlns:http-conf="http://schemas.ionas.com/transport/http/co
nfiguration" ... >
...
<service name="...">
  <port binding="...">
    <http-conf:client ...
      TrustedRootCertificates="X509CA/ca/ca_list01.pem"
    ... />
    ...
  </port>
</service>
```

Alternatively, set the

`plugins:http:client:trusted_root_certificates` variable in the Artix configuration file, as follows:

```
# Artix Configuration File
...
SecureClientScope {
plugins:http:client:trusted_root_certificates="X509CA/ca/c
a_list01.pem"
  ...
};
```

Server side

Edit the server's copy of the WSDL contract by adding (or modifying) the `TrustedRootCertificates` attribute in the `<http-conf:server>` tag. For example, to specify `X509CA/ca/ca_list01.pem` as the

server's trusted CA certificate, modify the server's WSDL contract as follows:

```
<definitions
xmlns:http="http://schemas.iona.com/transport/http"
xmlns:http-conf="http://schemas.iona.com/transport/http/co
nfiguration" ... >
...
<service name="...">
  <port binding="...">
    ...
    <http-conf:server ...
      TrustedRootCertificates="X509CA/ca/ca_list01.pem"
    ... />
  </port>
</service>
```

Alternatively, set the

`plugins:http:server:trusted_root_certificates` variable in the Artix configuration file, as follows:

```
# Artix Configuration File
...
SecureServerScope {
plugins:http:server:trusted_root_certificates="X509CA/ca/c
a_list01.pem"
  ...
};
```

Deploying for the IIOP/TLS transport

To deploy one or more trusted root CAs for the IIOP/TLS transport, perform the following steps (the procedure for client and server applications is the same):

1. Assemble the collection of trusted CA certificates that you want to deploy. The trusted CA certificates could be obtained from public CAs or private CAs (for details of how to generate your own CA certificates, see [“Set Up Your Own CA” on page 152](#)). The trusted CA certificates should be in PEM format. All you need are the certificates themselves—the private keys and passwords are not required.
2. Organize the CA certificates into a collection of CA list files. For example, you might create three CA list files as follows:

```
X509Deploy/trusted_ca_lists/ca_list01.pem
```

```
X509Deploy/trusted_ca_lists/ca_list02.pem
X509Deploy/trusted_ca_lists/ca_list03.pem
```

Each CA list file consists of a concatenated list of CA certificates. A CA list file can be created using a simple file concatenation operation. For example, if you have two CA certificate files, `ca_cert01.pem` and `ca_cert02.pem`, you could combine them into a single CA list file, `ca_list01.pem`, with the following command:

Windows

```
copy X509CA\ca\ca_cert01.pem +
    X509CA\ca\ca_cert02.pem
    X509Deploy\trusted_ca_lists\ca_list01.pem
```

UNIX

```
cat X509CA/ca/ca_cert01.pem X509CA/ca/ca_cert02.pem >>
    X509Deploy/trusted_ca_lists/ca_list01.pem
```

The CA certificates are organized as lists as a convenient way of grouping related CA certificates together.

3. Edit the `artix.cfg` file to specify which of the CA list files is used by your application. The `artix.cfg` file is located in the following directory:

```
ArtixInstallDir/artix/2.0/etc/domains
```

To specify the CA list files, edit the value of the

`policies:iiop_tls:trusted_ca_list_policy` variable in your application's configuration scope in the `artix.cfg` file.

For example, if your application picks up its configuration from the `SecureAppScope` configuration scope and you want to include the CA certificates from the `ca_list01.pem` and `ca_list02.pem` files, edit the `artix.cfg` file as follows:

```
# Artix configuration file.
...
SecureAppScope {
    ...
    policies:iiop_tls:trusted_ca_list_policy =
["X509Deploy/trusted_ca_lists/ca_list01.pem",
 "X509Deploy/trusted_ca_lists/ca_list02.pem"];
    ...
}
```

The directory containing the trusted CA certificate lists (for example, *X509Deploy/trusted_ca_lists/*) should be a secure directory.

Note: If an application supports authentication of a peer, that is a client supports `EstablishTrustInTarget`, then a file containing trusted CA certificates must be provided. If not, a `NO_RESOURCES` exception is raised.

Deploying Application Certificates

Overview

This section describes how to deploy an Artix application's own certificate. In the current version of Artix, the procedure for deploying application certificates depends on the type of transport, as follows:

- [Deploying for the HTTPS transport.](#)
- [Deploying for the IIOP/TLS transport](#)

Certificate formats

The format used for application certificates depends on the type of transport, as follows:

- *HTTPS transport*—uses the PEM format. This format consists of a certificate file, *CertName.pem*, containing an encrypted X.509 certificate chain, and a private key file, *CertPrivKey.pem*, containing an encrypted private key. Both PEM files are encrypted by the same password (the *private key password*).
- *IIOP/TLS transport*—uses the PKCS#12 format. This format consists of a single encrypted file, *CertName.p12*, that contains an X.509 certificate chain and a private key.

Note: Because Artix uses an IIOP/TLS connection to communicate with the iS2 security server, Artix applications that use HTTPS generally require you to configure *both* HTTPS and IIOP/TLS.

Deploying for the HTTPS transport

To deploy an Artix application's own certificate, *CertName.pem*, with private key, *CertPrivKey.pem*, for the HTTPS transport, perform the following steps:

1. Copy the application certificate, *CertName.pem*, and private key file, *CertPrivKey.pem*, to the certificates directory—for example, *X509Deploy/certs/applications*—on the deployment host.
The certificates directory should be a secure directory that is accessible only to administrators and other privileged users.
2. Edit the WSDL contract to specify the location of the application certificate file and private key file. The details of this step depend on whether you are deploying an application certificate on the client side or the server side:

Client side

Edit the client's copy of the WSDL contract by adding (or modifying) the following highlighted attributes in the `<http-conf:client>` tag:

```
<definitions
xmlns:http="http://schemas.iona.com/transport/http"
xmlns:http-conf="http://schemas.iona.com/transport/http/configuration" ... >
...
<service name="...">
  <port binding="...">
    <soap:address ...>
      <http-conf:client UseSecureSockets="true"
        ClientCertificate="X509Deploy/certs/applications/CertName.pem"
        ClientPrivateKey="X509Deploy/certs/applications/CertPrivKey.pem"
        ClientPrivateKeyPassword="MyKeyPassword"
        TrustedRootCertificates="RootCertPath"
        ... />
    </port>
  </service>
```

Alternatively, set the `plugins:http:client:*` variables in the Artix configuration file, as follows:

```
# Artix Configuration File
...
SecureClientScope {
  plugins:http:client:use_secure_sockets = "true";
  plugins:http:client:trusted_root_certificates="RootCertPath"
  plugins:http:client:client_certificate="X509Deploy/certs/applications/CertName.pem"
  plugins:http:client:client_private_key="X509Deploy/certs/applications/CertPrivKey.pem"
  plugins:http:client:client_private_key_password="MyKeyPassword"
  ...
};
```

Server side

Edit the server's copy of the WSDL contract by adding (or modifying) the following highlighted attributes in the `<http-conf:server>` tag:

```
<definitions
xmlns:http="http://schemas.ionac.com/transport/http"
xmlns:http-conf="http://schemas.ionac.com/transport/http/configuration" ... >
...
<service name="...">
  <port binding="...">
    <soap:address ...>
      <http-conf:server UseSecureSockets="true"
ServerCertificate="X509Deploy/certs/applications/CertName.pem"
ServerPrivateKey="X509Deploy/certs/applications/CertPrivKey.pem"
ServerPrivateKeyPassword="MyKeyPassword"
TrustedRootCertificates="RootCertPath"
... />
    </port>
  </service>
```

Alternatively, set the `plugins:http:server:*` variables in the Artix configuration file, as follows:

```
# Artix Configuration File
...
SecureServerScope {
  ...
  plugins:http:server:use_secure_sockets = "true";
  plugins:http:server:trusted_root_certificates="RootCertPath"
  plugins:http:server:server_certificate="X509Deploy/certs/applications/CertName.pem"
  plugins:http:server:server_private_key="X509Deploy/certs/applications/CertPrivKey.pem"
  plugins:http:server:server_private_key_password="MyKeyPassword"
};
```

3. Protect the private key passwords.

Because the private key passwords in the WSDL contracts appear in plaintext form, you must ensure that the WSDL contract files themselves are not readable/writable by every user. Use the operating system to restrict read/write access to trusted users only.

Additionally, to avoid revealing the server's security configuration to clients, you should remove the `<http-conf:server>` tag from the client copy of the WSDL contract.

Deploying for the IOP/TLS transport

To deploy an Artix application's own certificate, *CertName.p12*, for the IOP/TLS transport, perform the following steps:

1. Copy the application certificate, *CertName.p12*, to the certificates directory—for example, *X509Deploy/certs/applications*—on the deployment host.

The certificates directory should be a secure directory that is accessible only to administrators and other privileged users.

2. Edit the *artix.cfg* configuration file (usually *ArtixInstallDir/artix/2.0/etc/domains/artix.cfg*). Given that your application picks up its configuration from the *SecureAppScope* scope, change the principal sponsor configuration to specify the *CertName.p12* certificate, as follows:

```
# Artix configuration file
...
SecureAppScope {
    ...
    principal_sponsor:use_principal_sponsor = "true";
    principal_sponsor:auth_method_id = "pkcs12_file";
    principal_sponsor:auth_method_data =
        ["filename=X509Deploy/certs/applications/CertName.p12"];
};
```

3. By default, the application will prompt the user for the certificate pass phrase as it starts up. To choose another option for providing the pass phrase, see [“Providing a Certificate Pass Phrase” on page 180](#).

Configuring HTTPS and IIOP/TLS Authentication

This chapter describes how to configure HTTPS and IIOP/TLS authentication requirements for Artix applications.

In this chapter

This chapter discusses the following topics:

Requiring Authentication	page 170
Specifying Trusted CA Certificates	page 178
Specifying an Application's Own Certificate	page 179
Providing a Certificate Pass Phrase	page 180
Advanced IIOP/TLS Configuration Options	page 185

Requiring Authentication

Overview

This section discusses how to specify the kind of authentication required, whether mutual or target-only.

In this section

There are two possible arrangements for a TLS secure association:

Target-Only Authentication	page 171
Mutual Authentication	page 175

Target-Only Authentication

Overview

When an application is configured for target-only authentication, the target authenticates itself to the client but the client is not authentic to the target object—see [Figure 26](#).

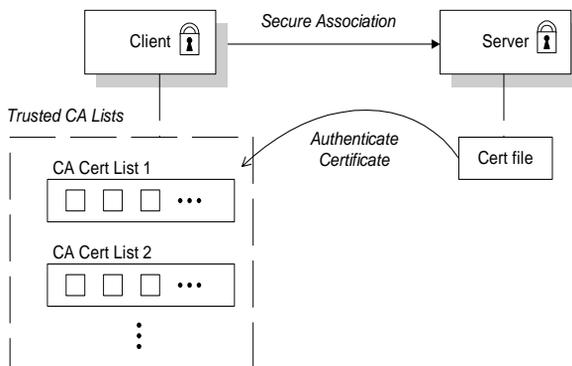


Figure 26: *Target Authentication Only*

Security handshake

Prior to running the application, the client and server should be set up as follows:

- A certificate chain is associated with the server—the certificate chain is provided in the form of a PEM file (for HTTPS) or a PKCS#12 file (for IIOP/TLS). See [“Specifying an Application’s Own Certificate” on page 179](#).
- One or more lists of trusted certification authorities (CA) are made available to the client—see [“Deploying Trusted Certificate Authority Certificates” on page 160](#).

During the security handshake, the server sends its certificate chain to the client—see [Figure 26](#). The client then searches its trusted CA lists to find a CA certificate that matches one of the CA certificates in the server’s certificate chain.

HTTPS example

You configure target-only authentication for the HTTPS transport by omitting a certificate on the client side. That is, the `ClientCertificate` attribute is not set in the `<http-conf:client>` tag. For example, you could configure the client side and the server side as follows:

Client side

Edit the client's copy of the WSDL contract by adding (or modifying) the following highlighted attributes in the `<http-conf:client>` tag:

```
<definitions
xmlns:http="http://schemas.ionas.com/transport/http"
xmlns:http-conf="http://schemas.ionas.com/transport/http/configuration" ... >
...
<service name="...">
  <port binding="...">
    <soap:address ...>
      <http-conf:client UseSecureSockets="true"
                        TrustedRootCertificates="RootCertPath"
                        ... />
    </port>
  </service>
```

Alternatively, instead of the `<http-conf:client>` security attributes, you can set the following variables in the Artix configuration file, `artix.cfg`:

```
# Artix Configuration File
...
SecureClientScope {
  plugins:http:client:use_secure_sockets = "true";
  plugins:http:client:trusted_root_certificates="RootCertPath"
  ...
};
```

Server side

Edit the server's copy of the WSDL contract by adding (or modifying) the following highlighted attributes in the `<http-conf:server>` tag:

```
<definitions
xmlns:http="http://schemas.iona.com/transports/http"
xmlns:http-conf="http://schemas.iona.com/transports/http/configuration" ... >
...
<service name="...">
  <port binding="...">
    <soap:address ...>
      <http-conf:server UseSecureSockets="true"
ServerCertificate="X509Deploy/certs/applications/CertName.pem"
ServerPrivateKey="X509Deploy/certs/applications/CertPrivKey.pem"
ServerPrivateKeyPassword="MyKeyPassword"
TrustedRootCertificates="RootCertPath"
... />
    </port>
  </service>
```

Alternatively, instead of the `<http-conf:server>` security attributes, you can set the following variables in the Artix configuration file, `artix.cfg`:

```
# Artix Configuration File
...
SecureServerScope {
  ...
  plugins:http:server:use_secure_sockets = "true";
  plugins:http:server:trusted_root_certificates="RootCertPath"
  plugins:http:server:server_certificate="X509Deploy/certs/applications/CertName.pem"
  plugins:http:server:server_private_key="X509Deploy/certs/applications/CertPrivKey.pem"
  plugins:http:server:server_private_key_password="MyKeyPassword"
};
```

IIOP/TLS example

The following extract from an `artix.cfg` configuration file shows the target-only configuration of an Artix client application, `bank_client`, and an Artix server application, `bank_server`, where the transport type is IIOP/TLS.

```
# Artix Configuration File
...
policies:iiop_tls:mechanism_policy:protocol_version = "SSL_V3";
policies:iiop_tls:mechanism_policy:ciphersuites =
    ["RSA_WITH_RC4_128_SHA", "RSA_WITH_RC4_128_MD5"];

bank_server {
    policies:iiop_tls:target_secure_invocation_policy:requires =
        ["Confidentiality"];
    policies:iiop_tls:target_secure_invocation_policy:supports =
        ["Confidentiality", "Integrity", "DetectReplay",
        "DetectMisordering", "EstablishTrustInTarget"];
    ...
};

bank_client {
    ...
    policies:iiop_tls:client_secure_invocation_policy:requires =
        ["Confidentiality", "EstablishTrustInTarget"];
    policies:iiop_tls:client_secure_invocation_policy:supports =
        ["Confidentiality", "Integrity", "DetectReplay",
        "DetectMisordering", "EstablishTrustInTarget"];
};
```

Mutual Authentication

Overview

When an application is configured for mutual authentication, the target authenticates itself to the client and the client authenticates itself to the target. This scenario is illustrated in Figure 27. In this case, the server and the client each require an X.509 certificate for the security handshake.

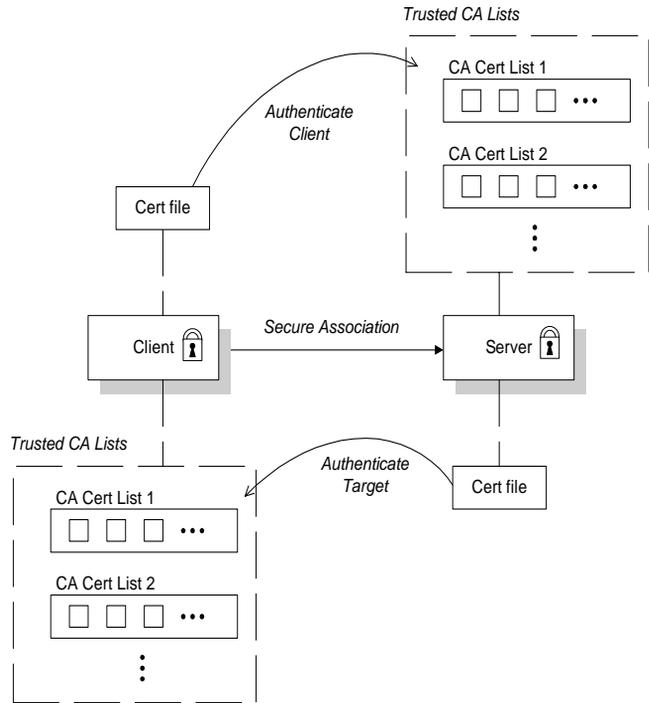


Figure 27: Mutual Authentication

Security handshake

Prior to running the application, the client and server should be set up as follows:

- Both client and server have an associated certificate chain (PEM file or PKCS#12 file)—see [“Specifying an Application’s Own Certificate” on page 179](#).
- Both client and server are configured with lists of trusted certification authorities (CA)—see [“Deploying Trusted Certificate Authority Certificates” on page 160](#).

During the security handshake, the server sends its certificate chain to the client, and the client sends its certificate chain to the server—see [Figure 26](#).

HTTPS example

To configure mutual authentication for the HTTPS transport, you should deploy an application certificate both on the client side and on the server side. For a detailed example, see the following reference:

- [“Deploying for the HTTPS transport” on page 165](#).

IIOP/TLS example

The following sample extract from an `artix.cfg` configuration file shows the configuration for mutual authentication of a client application, `secure_client_with_cert`, and a server application, `secure_server_enforce_client_auth`, where the transport type is IIOP/TLS.

```
# Artix Configuration File
...
policies:iioptls:mechanism_policy:protocol_version = "SSL_V3";
policies:iioptls:mechanism_policy:ciphersuites =
  ["RSA_WITH_RC4_128_SHA", "RSA_WITH_RC4_128_MD5"];

secure_server_enforce_client_auth
{
  policies:iioptls:target_secure_invocation_policy:requires =
    ["EstablishTrustInClient", "Confidentiality"];
  policies:iioptls:target_secure_invocation_policy:supports =
    ["EstablishTrustInClient", "Confidentiality", "Integrity",
     "DetectReplay", "DetectMisordering",
     "EstablishTrustInTarget"];
  ...
};
```

```
secure_client_with_cert
{
  policies:iiop_tls:client_secure_invocation_policy:requires =
    ["Confidentiality", "EstablishTrustInTarget"];
  policies:iiop_tls:client_secure_invocation_policy:supports =
    ["Confidentiality", "Integrity", "DetectReplay",
     "DetectMisordering", "EstablishTrustInClient",
     "EstablishTrustInTarget"];
    ...
};
```

Specifying Trusted CA Certificates

Overview

When an application receives an X.509 certificate during an SSL/TLS handshake, the application decides whether or not to trust the received certificate by checking whether the issuer CA is one of a pre-defined set of trusted CA certificates. If the received X.509 certificate is validly signed by one of the application's trusted CA certificates, the certificate is deemed trustworthy; otherwise, it is rejected.

Which applications need to specify trusted CA certificates?

Any application that is likely to receive an X.509 certificate as part of an HTTPS or IIOP/TLS handshake must specify a list of trusted CA certificates. For example, this includes the following types of application:

- All IIOP/TLS or HTTPS clients.
 - Any IIOP/TLS or HTTPS servers that support mutual authentication.
-

How to deploy trusted CA certificates

For more details about how to deploy trusted CA certificates, see the following references:

- [“Deploying for the HTTPS transport” on page 160.](#)
- [“Deploying for the IIOP/TLS transport” on page 162.](#)

Specifying an Application's Own Certificate

Overview

To enable an Artix application to identify itself, it must be associated with an X.509 certificate. The X.509 certificate is needed during an SSL/TLS handshake, where it is used to authenticate the application to its peers. The method you use to specify the certificate depends on the type of application:

- *Security unaware*—configuration only,

This section discusses how to specify a certificate by configuration only.

How to deploy an application certificate

For details about how to deploy an application's own certificate, see the following reference:

- [“Deploying Application Certificates” on page 165.](#)

Providing a Certificate Pass Phrase

Overview

If an application is configured to have an X.509 certificate, it is necessary to provide a pass phrase as the application starts up. There are various ways of providing the certificate pass phrase, depending on the particular type of transport used.

In this section

This section contains the following subsections:

Certificate Pass Phrase for HTTPS	page 181
Certificate Pass Phrase for IIOP/TLS	page 183

Certificate Pass Phrase for HTTPS

Overview

For the HTTPS transport, there is just one option for specifying a certificate's pass phrase, as follows:

- [Directly in the WSDL contract.](#)

Directly in the WSDL contract

For the HTTPS protocol, the same pass phrase is used to encrypt both the certificate and the private key. You can specify the certificate pass phrase by editing the WSDL contract as follows:

Client side

Edit the client's copy of the WSDL contract by adding (or modifying) the `ClientPrivateKeyPassword` attribute in the `<http-conf:client>` tag:

```
<definitions
xmlns:http="http://schemas.ionas.com/transports/http"
xmlns:http-conf="http://schemas.ionas.com/transports/http/configuration" ... >
...
<service name="...">
  <port binding="...">
    <soap:address ...>
      <http-conf:client ...
        ClientPrivateKeyPassword="MyKeyPassword"
        TrustedRootCertificates="RootCertPath"
        ... />
    </port>
  </service>
```

Alternatively, you can set the `plugins:http:client:client_private_key_password` variable in the Artix configuration file, `artix.cfg`, as follows:

```
# Artix Configuration File
...
SecureClientScope {
  plugins:http:client:client_private_key_password = "MyKeyPassword";
  ...
};
```

Server side

Edit the server's copy of the WSDL contract by adding (or modifying) the `ServerPrivateKeyPassword` attribute in the `<http-conf:server>` tag:

```
<definitions
xmlns:http="http://schemas.ionas.com/transport/http"
xmlns:http-conf="http://schemas.ionas.com/transport/http/configuration" ... >
...
<service name="...">
  <port binding="...">
    <soap:address ...>
      <http-conf:server ...
        ServerPrivateKeyPassword="MyKeyPassword"
        TrustedRootCertificates="RootCertPath"
        ... />
      </port>
    </service>
```

Alternatively, you can set the `plugins:http:server:server_private_key_password` variable in the Artix configuration file, `artix.cfg`, as follows:

```
# Artix Configuration File
...
SecureServerScope {
  plugins:http:server:server_private_key_password = "MyKeyPassword";
  ...
};
```

Certificate Pass Phrase for IIOP/TLS

Overview

Once you have specified a PKCS#12 certificate, you must also provide its *pass phrase*. The pass phrase is needed to decrypt the certificate's private key (which is used during the TLS security handshake to prove the certificate's authenticity).

For the IIOP/TLS transport, the pass phrase can be provided in one of the following ways:

- [From a dialog prompt.](#)
- [In a password file.](#)
- [Directly in configuration.](#)

From a dialog prompt

If the pass phrase is not specified in any other way, Artix will prompt the user for the pass phrase as the application starts up. This approach is suitable for persistent (that is, manually-launched) servers.

C++ Applications

When a C++ application starts up, the user is prompted for the pass phrase at the command line as follows:

```
Initializing the ORB
Enter password :
```

In a password file

The pass phrase is stored in a password file whose location is specified in the `principal_sponsor:auth_method_data` configuration variable using the `password_file` option. In the following example, the *SecureApp* scope configures the principal sponsor as follows:

```
# Artix Configuration File
SecureApp {
  ...
  principal_sponsor:use_principal_sponsor = "true";
  principal_sponsor:auth_method_id = "pkcs12_file";
  principal_sponsor:auth_method_data =
    ["filename=X509Deploy/certs/administrator.p12",
     "password_file=X509Deploy/certs/administrator.pwf"];
  ...
};
```

In this example, the pass phrase for the `bank_server.p12` certificate is stored in the `administrator.pwf` file, which contains the following pass phrase:

```
administratorpass
```

WARNING: Because the password file stores the pass phrase in plain text, the password file should not be readable by anyone except the administrator. For greater security, you could supply the pass phrase from a dialog prompt instead.

Directly in configuration

For a PKCS #12 file, the pass phrase can be specified directly in the `principal_sponsor:auth_method_data` configuration variable using the `password` option. For example, the `bank_server` demonstration configures the principal sponsor as follows:

```
# Artix Configuration File
bank_server {
    ...
    principal_sponsor:use_principal_sponsor = "true";
    principal_sponsor:auth_method_id = "pkcs12_file";
    principal_sponsor:auth_method_data =
        [ "filename=ASPInstallDir\asp\6.0\etc\tls\x509\certs\demos\bank
        _server.p12", "password=bankserverpass" ];
};
```

In this example, the pass phrase for the `bank_server.p12` certificate is `bankserverpass`.

WARNING: Storing the pass phrase directly in configuration is not recommended for deployed systems. The pass phrase is in plain text and could be read by anyone.

Advanced IIOP/TLS Configuration Options

Overview

For added security, the IIOP/TLS transport allows you to apply extra conditions on certificates. Before reading this section you might find it helpful to consult [“Managing Certificates” on page 141](#), which provides some background information on the structure of certificates.

In this section

This section discusses the following advanced IIOP/TLS configuration options:

Setting a Maximum Certificate Chain Length	page 186
Applying Constraints to Certificates	page 187

Setting a Maximum Certificate Chain Length

Max chain length policy

You can use the maximum chain length policy to enforce the maximum length of certificate chains presented by a peer during handshaking.

A certificate chain is made up of a root CA at the top, an application certificate at the bottom and any number of CA intermediaries in between. The length that this policy applies to is the (inclusive) length of the chain from the application certificate presented to the first signer in the chain that appears in the list of trusted CA's (as specified in the `TrustedCAListPolicy`).

Example

For example, a chain length of 2 mandates that the certificate of the immediate signer of the peer application certificate presented must appear in the list of trusted CA certificates.

Configuration variable

You can specify the maximum length of certificate chains used in maximum chain length policy with the `policies:iio_tls:max_chain_length_policy` configuration variable. For example:

```
policies:iio_tls:max_chain_length_policy = "4";
```

Default value

The default value is 2 (that is, the application certificate and its signer, where the signer must appear in the list of trusted CA's).

Applying Constraints to Certificates

Certificate constraints policy

You can use the certificate constraints policy to apply constraints to peer X.509 certificates. 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 the certificate constraints policy through the `policies:iioptls:certificate_constraints_policy` configuration variable. For example:

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

Constraint language

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

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

Example

This is an example list of constraints:

```
policies:iioptls: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 [“ASN.1 and Distinguished Names” on page 283](#).

Configuring IIOP/TLS Secure Associations

The Artix IIOP/TLS transport layer offers additional functionality that enables you to customize client-server connections by specifying secure invocation policies and security mechanism policies.

In this chapter

This chapter discusses the following topics:

Overview of Secure Associations	page 190
Setting IIOP/TLS Association Options	page 192
Specifying IIOP/TLS Cipher Suites	page 204
Caching IIOP/TLS Sessions	page 213

Overview of Secure Associations

Secure association

A *secure association* is a term that has its origins in the CORBA Security Service and refers to any link between a client and a server that enables invocations to be transmitted securely. In the present context, a secure association is an IIOP/TLS connection augmented by a collection of security policies that govern the behavior of the connection.

TLS session

A *TLS session* is the TLS implementation of a secure client-server association. The TLS session is accompanied by a *session state* that stores the security characteristics of the association.

A TLS session underlies each secure association in Artix.

Colocation

For *colocated invocations*, that is where the calling code and called code share the same address space, Artix supports the establishment of colocated secure associations. A special interceptor, `TLS_CoLoc`, is provided by the security plug-in to optimize the transmission of secure, colocated invocations.

Configuration overview

The security characteristics of an association can be configured through the following CORBA policy types:

- *Client secure invocation policy*—enables you to specify the security requirements on the client side by setting association options. See [“Choosing Client Behavior” on page 196](#) for details.
- *Target secure invocation policy*—enables you to specify the security requirements on the server side by setting association options. See [“Choosing Target Behavior” on page 198](#) for details.
- *Mechanism policy*—enables you to specify the security mechanism used by secure associations. In the case of TLS, you are required to specify a list of cipher suites for your application. See [“Specifying IIOP/TLS Cipher Suites” on page 204](#) for details.

Figure 28 illustrates all of the elements that configure a secure association. The security characteristics of the client and the server can be configured independently of each other.

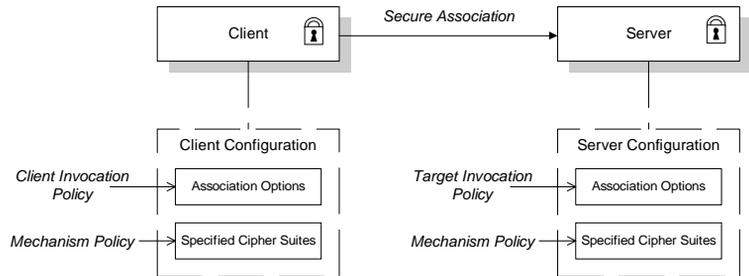


Figure 28: Configuration of a Secure Association

Setting IIOP/TLS Association Options

Overview

This section explains the meaning of the various IIOP/TLS association options and describes how you can use the IIOP/TLS association options to set client and server secure invocation policies for IIOP/TLS connections.

In this section

The following subsections discuss the meaning of the settings and flags:

Secure Invocation Policies	page 193
Association Options	page 194
Choosing Client Behavior	page 196
Choosing Target Behavior	page 198
Hints for Setting Association Options	page 200

Secure Invocation Policies

Secure invocation policies

You can set the minimum security requirements for the applications in your system with two types of security policy:

- *Client secure invocation policy*—specifies the client association options.
- *Target secure invocation policy*—specifies the association options on a target object.

These policies can only be set through configuration; they cannot be specified programmatically by security-aware applications.

Configuration example

For example, to specify that client authentication is required for IIOp/TLS connections, you can set the following target secure invocation policy for your server:

```
# Artix Configuration File
secure_server_enforce_client_auth
{
  policies:iioptls:target_secure_invocation_policy:requires =
  ["EstablishTrustInClient", "Confidentiality"];

  policies:iioptls:target_secure_invocation_policy:supports =
  ["EstablishTrustInClient", "Confidentiality", "Integrity",
  "DetectReplay", "DetectMisordering",
  "EstablishTrustInTarget"];

  // Other settings (not shown)...
};
```

Association Options

Available options

You can use *association options* to configure IIOP/TLS secure associations. They can be set for clients or servers where appropriate. These are the available options:

- `NoProtection`
 - `Integrity`
 - `Confidentiality`
 - `DetectReplay`
 - `DetectMisordering`
 - `EstablishTrustInTarget`
 - `EstablishTrustInClient`
-

NoProtection

Use the `NoProtection` flag to set minimal protection. This means that insecure bindings are supported, and (if the application supports something other than `NoProtection`) the target can accept secure and insecure invocations.

Integrity

Use the `Integrity` flag to indicate that your application supports integrity-protected invocations. Setting this flag implies that your TLS cipher suites support message digests (such as MD5, SHA1).

Confidentiality

Use the `Confidentiality` flag if your application requires or supports at least confidentiality-protected invocations. The object can support this feature if the cipher suites specified by the `MechanismPolicy` support confidentiality-protected invocations.

DetectReplay

Use the `DetectReplay` flag to indicate that your application supports or requires replay detection on invocation messages. This is determined by characteristics of the supported TLS cipher suites.

DetectMisordering

Use the `DetectMisordering` flag to indicate that your application supports or requires error detection on fragments of invocation messages. This is determined by characteristics of the supported TLS cipher suites.

EstablishTrustInTarget

The `EstablishTrustInTarget` flag is set for client policies only. Use the flag to indicate that your client supports or requires that the target authenticate its identity to the client. This is determined by characteristics of the supported TLS cipher suites. This is normally set for both client `supports` and `requires` unless anonymous cipher suites are supported.

EstablishTrustInClient

Use the `EstablishTrustInClient` flag to indicate that your target object requires the client to authenticate its privileges to the target. This option cannot be required as a client policy.

If this option is supported on a client's policy, it means that the client is prepared to authenticate its privileges to the target. On a target policy, the target supports having the client authenticate its privileges to the target.

Choosing Client Behavior

Client secure invocation policy

The client secure invocation policy type determines how a client handles security issues.

IIOP/TLS configuration

You can set this policy for IIOP/TLS connections through the following configuration variables:

```
policies:iiop_tls:client_secure_invocation_policy:requires
    Specifies the minimum security features that the client requires to
    establish an IIOP/TLS connection.
```

```
policies:iiop_tls:client_secure_invocation_policy:supports
    Specifies the security features that the client is able to support on
    IIOP/TLS connections.
```

Association options

In both cases, you provide the details of the security levels in the form of `AssociationOption` flags—see [“Association Options” on page 194](#).

Default value

The default value for the client secure invocation policy is:

```
supports      Integrity, Confidentiality, DetectReplay,
              DetectMisordering, EstablishTrustInTarget
requires      Integrity, Confidentiality, DetectReplay,
              DetectMisordering, EstablishTrustInTarget
```

Example

The following example shows some sample settings for the client secure invocation policy:

```
# Artix Configuration File
bank_client {
  ...
  policies:iiop_tls:client_secure_invocation_policy:requires =
    ["Confidentiality", "EstablishTrustInTarget"];

  policies:iiop_tls:client_secure_invocation_policy:supports =
    ["Confidentiality", "Integrity", "DetectReplay",
     "DetectMisordering", "EstablishTrustInTarget"];
};
...
};
```

Choosing Target Behavior

Target secure invocation policy

The target secure invocation policy type operates in a similar way to the client secure invocation policy type. It determines how a target handles security issues.

IIOP/TLS configuration

You can set the target secure invocation policy for IIOP/TLS connections through the following configuration variables:

```
policies:iiop_tls:target_secure_invocation_policy:requires
```

Specifies the minimum security features that your targets require, before they accept an IIOP/TLS connection.

```
policies:iiop_tls:target_secure_invocation_policy:supports
```

Specifies the security features that your targets are able to support on IIOP/TLS connections.

Association options

In both cases, you can provide the details of the security levels in the form of `AssociationOption` flags—see [“Association Options” on page 194](#).

Default value

The default value for the target secure invocation policy is:

```
supports      Integrity, Confidentiality, DetectReplay,
              DetectMisordering, EstablishTrustInTarget
```

```
requires      Integrity, Confidentiality, DetectReplay,
              DetectMisordering
```

Example

The following example shows some sample settings for the target secure invocation policy:

```
# Artix Configuration File
...
bank_server {
  ...
  policies:iiop_tls:target_secure_invocation_policy:requires =
    ["Confidentiality"];

  policies:iiop_tls:target_secure_invocation_policy:supports =
    ["Confidentiality", "Integrity", "DetectReplay",
     "DetectMisordering", "EstablishTrustInTarget"];
  ...
};
...
```

Hints for Setting Association Options

Overview

This section gives an overview of how association options can be used in real applications.

Rules of thumb

The following rules of thumb should be kept in mind:

- If an association option is *required* by a particular invocation policy, it must also be *supported* by that invocation policy. It makes no sense to require an association option without supporting it.
 - It is important to be aware that the secure invocation policies and the security mechanism policy mutually interact with each other. That is, the association options effective for a particular secure association depend on the available cipher suites (see [“Constraints Imposed on Cipher Suites” on page 210](#)).
 - The `NoProtection` option must appear alone in a list of *required* options. It does not make sense to require other security options in addition to `NoProtection`.
-

Types of association option

Association options can be categorized into the following different types, as shown in [Table 2](#).

Table 2: *Description of Different Types of Association Option*

Description	Relevant Association Options
Request or require TLS peer authentication.	EstablishTrustInTarget and EstablishTrustInClient .
Quality of protection.	Confidentiality , Integrity , DetectReplay , and DetectMisordering .
Allow or require insecure connections.	NoProtection .

EstablishTrustInTarget and EstablishTrustInClient

These association options are used as follows:

- `EstablishTrustInTarget`—determines whether a server sends its own X.509 certificate to a client during the SSL/TLS handshake. In practice, secure Orbix applications must enable `EstablishTrustInTarget`, because all of the cipher suites supported by Orbix require it.

The `EstablishTrustInTarget` association option should appear in all of the configuration variables shown in the relevant row of [Table 3](#).

- `EstablishTrustInClient`—determines whether a client sends its own X.509 certificate to a server during the SSL/TLS handshake. The `EstablishTrustInClient` feature is optional and various combinations of settings are possible involving this association option.

The `EstablishTrustInClient` association option can appear in any of the configuration variables shown in the relevant row of [Table 3](#).

Table 3: *Setting EstablishTrustInTarget and EstablishTrustInClient Association Options*

Association Option	Client side—can appear in...	Server side—can appear in...
<code>EstablishTrustInTarget</code>	<p><code>policies:client_secure_invocation_policy:supports</code></p> <p><code>policies:client_secure_invocation_policy:requires</code></p>	<p><code>policies:target_secure_invocation_policy:supports</code></p>
<code>EstablishTrustInClient</code>	<p><code>policies:client_secure_invocation_policy:supports</code></p>	<p><code>policies:target_secure_invocation_policy:supports</code></p> <p><code>policies:target_secure_invocation_policy:requires</code></p>

Note: The SSL/TLS client authentication step can also be affected by the `policies:allow_unauthenticated_clients_policy` configuration variable. See “[policies Namespace](#)” on page 252.

Confidentiality, Integrity, DetectReplay, and DetectMisordering

These association options can be considered together, because normally you would require either all or none of these options. Most of the cipher suites supported by Orbix support all of these association options, although there are a couple of integrity-only ciphers that do not support Confidentiality (see [Table 7 on page 211](#)). As a rule of thumb, if you want security you generally would want *all* of these association options.

Table 4: *Setting Quality of Protection Association Options*

Association Options	Client side—can appear in...	Server side—can appear in...
Confidentiality, Integrity, DetectReplay, and DetectMisordering	<code>policies:client_secure_invocation_policy:supports</code>	<code>policies:target_secure_invocation_policy:supports</code>
	<code>policies:client_secure_invocation_policy:requires</code>	<code>policies:target_secure_invocation_policy:requires</code>

A typical secure application would list *all* of these association options in *all* of the configuration variables shown in [Table 4](#).

Note: Some of the sample configurations appearing in the generated configuration file require Confidentiality, but not the other qualities of protection. In practice, however, the list of required association options is implicitly extended to include the other qualities of protection, because the cipher suites that support Confidentiality also support the other qualities of protection. This is an example of where the security mechanism policy interacts with the secure invocation policies.

NoProtection

The NoProtection association option is used for two distinct purposes:

- *Disabling security selectively*—security is disabled, either in the client role or in the server role, if NoProtection appears as the sole *required* association option and as the sole *supported* association option in a secure invocation policy. This mechanism is selective in the sense that the client role and the server role can be independently configured as either secure or insecure.

Note: In this case, the `orb_plugins` configuration variable should include the `iiop` plug-in to enable insecure communication.

- *Making an application semi-secure*—an application is semi-secure, either in the client role or in the server role, if `NoProtection` appears as the sole *required* association option and as a *supported* association option along with other secure association options. The meaning of semi-secure in this context is, as follows:
 - ◆ *Semi-secure client*—the client will open either a secure or an insecure connection, depending on the disposition of the server (that is, depending on whether the server accepts only secure connections or only insecure connections). If the server is semi-secure, the type of connection opened depends on the order of the bindings in the `binding:client_binding_list`.
 - ◆ *Semi-secure server*—the server accepts connections either from a secure or an insecure client.

Note: In this case, the `orb_plugins` configuration variable should include both the `iiop_tls` plug-in and the `iiop` plug-in.

Table 5 shows the configuration variables in which the `NoProtection` association option can appear.

Table 5: *Setting the NoProtection Association Option*

Association Option	Client side—can appear in...	Server side—can appear in...
<code>NoProtection</code>	<p><code>policies:client_secure_invocation_policy:supports</code></p> <p><code>policies:client_secure_invocation_policy:requires</code></p>	<p><code>policies:target_secure_invocation_policy:supports</code></p> <p><code>policies:target_secure_invocation_policy:requires</code></p>

References

For more information about setting association options, see the following:

- [“Securing Communications with SSL/TLS” on page 97.](#)
- The `demons.tls` scope in a generated Orbix configuration file.

Specifying IIOP/TLS Cipher Suites

Overview

This section explains how to specify the list of cipher suites that are made available to an application (client or server) for the purpose of establishing IIOP/TLS secure associations. During a security handshake, the client chooses a cipher suite that matches one of the cipher suites available to the server. The cipher suite then determines the security algorithms that are used for the secure association.

In this section

This section contains the following subsections:

Supported Cipher Suites	page 205
Setting the Mechanism Policy	page 208
Constraints Imposed on Cipher Suites	page 210

Supported Cipher Suites

Artix cipher suites

The following cipher suites are supported by Artix IIOP/TLS:

- Null encryption, integrity-only ciphers:

```
RSA_WITH_NULL_MD5
RSA_WITH_NULL_SHA
```

- Standard ciphers

```
RSA_EXPORT_WITH_RC4_40_MD5
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
```

Security algorithms

Each cipher suite specifies a set of three security algorithms, which are used at various stages during the lifetime of a secure association:

- *Key exchange algorithm*—used during the security handshake to enable authentication and the exchange of a symmetric key for subsequent communication. Must be a public key algorithm.
 - *Encryption algorithm*—used for the encryption of messages after the secure association has been established. Must be a symmetric (private key) encryption algorithm.
 - *Secure hash algorithm*—used for generating digital signatures. This algorithm is needed to guarantee message integrity.
-

Key exchange algorithms

The following key exchange algorithms are supported by Artix IIOP/TLS:

RSA	Rivest Shamir Adleman (RSA) public key encryption using X.509v3 certificates. No restriction on the key size.
RSA_EXPORT	RSA public key encryption using X.509v3 certificates. Key size restricted to 512 bits.

Encryption algorithms

The following encryption algorithms are supported by Artix IIOPTLS:

RC4_40	A symmetric encryption algorithm developed by RSA data security. Key size restricted to 40 bits.
RC4_128	RC4 with a 128-bit key.
DES40_CBC	Data encryption standard (DES) symmetric encryption. Key size restricted to 40 bits.
DES_CBC	DES with a 56-bit key.
3DES_EDE_CBC	Triple DES (encrypt, decrypt, encrypt) with an effective key size of 168 bits.

Secure hash algorithms

The following secure hash algorithms are supported by Artix IIOPTLS:

MD5	Message Digest 5 (MD5) hash algorithm. This algorithm produces a 128-bit digest.
SHA	Secure hash algorithm (SHA). This algorithm produces a 160-bit digest, but is somewhat slower than MD5.

Cipher suite definitions

The Artix IIOPTLS cipher suites are defined as follows:

Table 6: *Cipher Suite Definitions*

Cipher Suite	Key Exchange Algorithm	Encryption Algorithm	Secure Hash Algorithm	Exportable?
RSA_WITH_NULL_MD5	RSA	NULL	MD5	yes
RSA_WITH_NULL_SHA	RSA	NULL	SHA	yes
RSA_EXPORT_WITH_RC4_40_MD5	RSA_EXPORT	RC4_40	MD5	yes
RSA_WITH_RC4_128_MD5	RSA	RC4_128	MD5	no
RSA_WITH_RC4_128_SHA	RSA	RC4_128	SHA	no
RSA_EXPORT_WITH_DES40_CBC_SHA	RSA_EXPORT	DES40_CBC	SHA	yes
RSA_WITH_DES_CBC_SHA	RSA	DES_CBC	SHA	no
RSA_WITH_3DES_EDE_CBC_SHA	RSA	3DES_EDE_CBC	SHA	no

Reference

For further details about cipher suites in the context of TLS, see RFC 2246 from the Internet Engineering Task Force (IETF). This document is available from the IETF Web site: <http://www.ietf.org>.

Setting the Mechanism Policy

Mechanism policy

To specify IIOP/TLS cipher suites, use the *mechanism policy*. The mechanism policy is a client and server side security policy that determines

- Whether SSL or TLS is used, and
- Which specific cipher suites are to be used.

The `protocol_version` configuration variable

You can specify whether SSL or TLS is used with a transport protocol by setting the `policies:iiop_tls:mechanism_policy:protocol_version` configuration variable for IIOP/TLS. For example:

```
# Artix Configuration File
policies:iiop_tls:mechanism_policy:protocol_version = "SSL_V3";
```

You can set the `protocol_version` configuration variable to one of the following alternatives:

```
TLS_V1
SSL_V3
```

And a special setting for interoperating with an application deployed on the OS/390 platform (to work around a bug in IBM's System/SSL toolkit):

```
SSL_V2V3
```

The cipher suites configuration variable

You can specify the cipher suites available to a transport protocol by setting the `policies:iiop_tls:mechanism_policy:ciphersuites` configuration variable for IIOP/TLS. For example:

```
# Artix Configuration File
policies:iiop_tls:mechanism_policy:ciphersuites =
[ "RSA_WITH_NULL_MD5",
  "RSA_WITH_NULL_SHA",
  "RSA_EXPORT_WITH_RC4_40_MD5",
  "RSA_WITH_RC4_128_MD5" ];
```

Cipher suite order

The order of the entries in the mechanism policy's cipher suites list is important.

During a security handshake, the client sends a list of acceptable cipher suites to the server. The server then chooses the first of these cipher suites that it finds acceptable. The secure association is, therefore, more likely to use those cipher suites that are near the beginning of the `ciphersuites` list.

Valid cipher suites

You can specify any of the following cipher suites:

- Null encryption, integrity only ciphers:

```
RSA_WITH_NULL_MD5,  
RSA_WITH_NULL_SHA
```

- Standard ciphers

```
RSA_EXPORT_WITH_RC4_40_MD5,  
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
```

Default values

If no cipher suites are specified through configuration or application code, the following apply:

```
RSA_WITH_RC4_128_SHA,  
RSA_WITH_RC4_128_MD5,  
RSA_WITH_3DES_EDE_CBC_SHA,  
RSA_WITH_DES_CBC_SHA
```

Constraints Imposed on Cipher Suites

Effective cipher suites

Figure 29 shows that cipher suites initially specified in the configuration are *not* necessarily made available to the application. Artix checks each cipher suite for compatibility with the specified association options and, if necessary, reduces the size of the list to produce a list of *effective cipher suites*.

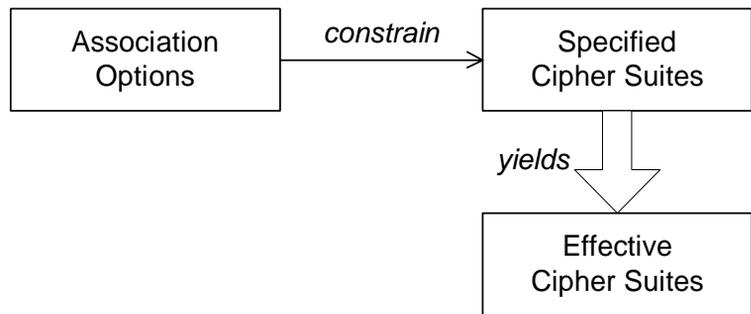


Figure 29: Constraining the List of Cipher Suites

Required and supported association options

For example, in the context of the IIOP/TLS protocol the list of cipher suites is affected by the following configuration options:

- *Required association options*—as listed in `policies:iiop_tls:client_secure_invocation_policy:requires` ON the client side, or `policies:iiop_tls:target_secure_invocation_policy:requires` ON the server side.
- *Supported association options*—as listed in `policies:iiop_tls:client_secure_invocation_policy:supports` ON the client side, or `policies:iiop_tls:target_secure_invocation_policy:supports` ON the server side.

Cipher suite compatibility table

Use [Table 7](#) to determine whether or not a particular cipher suite is compatible with your association options.

Table 7: *Association Options Supported by Cipher Suites*

Cipher Suite	Supported Association Options
RSA_WITH_NULL_MD5	Integrity, DetectReplay, DetectMisordering
RSA_WITH_NULL_SHA	Integrity, DetectReplay, DetectMisordering
RSA_EXPORT_WITH_RC4_40_MD5	Integrity, DetectReplay, DetectMisordering, Confidentiality
RSA_WITH_RC4_128_MD5	Integrity, DetectReplay, DetectMisordering, Confidentiality
RSA_WITH_RC4_128_SHA	Integrity, DetectReplay, DetectMisordering, Confidentiality
RSA_EXPORT_WITH_DES40_CBC_SHA	Integrity, DetectReplay, DetectMisordering, Confidentiality
RSA_WITH_DES_CBC_SHA	Integrity, DetectReplay, DetectMisordering, Confidentiality
RSA_WITH_3DES_EDE_CBC_SHA	Integrity, DetectReplay, DetectMisordering, Confidentiality

Determining compatibility

The following algorithm is applied to the initial list of cipher suites:

1. For the purposes of the algorithm, ignore the `EstablishTrustInClient` and `EstablishTrustInTarget` association options. These options have no effect on the list of cipher suites.
2. From the initial list, remove any cipher suite whose supported association options (see [Table 7](#)) do not satisfy the configured required association options.
3. From the remaining list, remove any cipher suite that supports an option (see [Table 7](#)) not included in the configured supported association options.

No suitable cipher suites available

If no suitable cipher suites are available as a result of incorrect configuration, no communications will be possible and an exception will be raised. Logging also provides more details on what went wrong.

Example

For example, specifying a cipher suite such as `RSA_WITH_RC4_128_MD5` that supports `Confidentiality`, `Integrity`, `DetectReplay`, `DetectMisordering`, `EstablishTrustInTarget` (and optionally `EstablishTrustInClient`) but specifying a `secure_invocation_policy` that supports only a subset of those features results in that cipher suite being ignored.

Caching IIOPTLS Sessions

Session caching policy

You can use the IIOPTLS session caching policy to control TLS session caching and reuse for both the client side and the server side.

Configuration variable

You can set the session caching policy with the `policies:iiop_tls:session_caching_policy` or `policies:https:session_caching_policy` configuration variables. For example:

```
policies:iiop_tls:session_caching_policy = "CACHE_CLIENT";
```

Valid values

You can apply the following values to the session caching policy:

```
CACHE_NONE,
CACHE_CLIENT,
CACHE_SERVER,
CACHE_SERVER_AND_CLIENT
```

Default value

The default value is `CACHE_NONE`.

Configuration variable

```
plugins:atli_tls_tcp:session_cache_validity_period
```

This allows control over the period of time that SSL/TLS session caches are valid for.

Valid values

`session_cache_validity_period` is specified in seconds.

Default value

The default value is 1 day.

Configuration variable

```
plugins:atli_tls_tcp:session_cache_size
```

`session_cache_size` is the maximum number of SSL/TLS sessions that are cached before sessions are flushed from the cache.

Default value

This defaults to no limit specified for C++.

This defaults to 100 for Java.

Principal Propagation

Principal propagation is a compatibility feature of Artix that is designed to facilitate interoperability with legacy Orbix applications.

In this chapter

This chapter discusses the following topics:

Introduction to Principal Propagation	page 216
Configuring	page 217
Programming	page 220
Interoperating with .NET	page 223

Introduction to Principal Propagation

Overview

Artix principal propagation is a transport-neutral mechanism that can be used to transmit a secure identity from a client to a server. It is *not* recommended that you use this feature in new applications. Principal propagation is provided primarily in order to facilitate interoperability with legacy Orbix applications.

WARNING: By default, the principal is propagated across the wire in plaintext. Hence, the principal is vulnerable to snooping. To protect against this possibility, you should enable SSL for your application.

Supported bindings/transports

Support for principal propagation is limited to the following bindings and transports:

- CORBA binding—the principal is sent in a GIOP service context.
- SOAP over HTTP—the principal is sent in a SOAP header.

Note: If a CORBA call is colocated, the principal is not propagated unless you remove the `POA_Colloc` interceptor from the binding lists in the `artix.cfg` file. This has the effect of disabling the CORBA colocated binding optimization.

Interoperability

The primary purpose of Artix principal propagation is to facilitate interoperability with legacy Orbix applications, in particular for applications running on the mainframe.

Because Artix uses standard mechanisms to propagate the principal, this feature ought to be compatible with third-party products as well.

Configuring

Overview

This section describes how to configure Artix to use principal propagation. The following aspects of configuration are described:

- [CORBA](#).
- [SOAP over HTTP](#).
- [Routing](#).

Note: Principal configuration is not supported for any other bindings, apart from CORBA and SOAP over HTTP.

CORBA

To use principal propagation with a CORBA binding, you must set the following configuration variables in your `artix.cfg` file (located in the `ArtixInstallDir/artix/2.0/etc/domains` directory):

Example 42: Configuring Principal Propagation for a CORBA Binding

```
policies:giop:interop_policy:send_principal = "true";
policies:giop:interop_policy:enable_principal_service_context =
    "true";
```

You can either add these settings to the global scope or to a specific sub-scope (in which case you must specify the sub-scope to the `-ORBname` command line switch when running the Artix application).

SOAP over HTTP

SOAP over HTTP requires no special configuration to support principal propagation. The Artix SOAP binding will always add a principal header, if you switch on message attributes in your code. The following cases arise:

- *Message attributes enabled and principal set explicitly*—the specified principal is sent in the principal header.
- *Message attributes enabled and principal not set*—Artix reads the username from the operating system and sends this username in the principal header.
- *Message attributes not enabled*—no principal header appears in the request message.

If you want a SOAP server to authenticate a propagated principal using the iS2 security service, however, you do need to add some settings to the server's configuration scope in your `artix.cfg` file, as shown in [Example 43](#).

Example 43: *Configuring Principal Authentication for SOAP*

```
# Security Layer Settings
policies:asp:enable_authorization = "true";
plugins:is2_authorization:action_role_mapping =
  "file://C:\artix\artix\1.2\demos\secure_hello_world\http_soap
  /config/helloworld_action_role_mapping.xml";
plugins:asp:authorization_realm = "IONAGlobalRealm";

plugins:asp:security_type = "PRINCIPAL";
plugins:asp:default_password = "default_password";
```

Setting `plugins:asp:security_type` equal to `PRINCIPAL` specifies that the received principal serves as the username for the purpose of authentication. The `plugins:asp:default_password` value serves as the password for the purpose of authentication. This latter setting is necessary because, although the iS2 service requires a password, there is no password propagated with the principal.

WARNING: The procedure of supplying a default password for the principal enables you to integrate principals with the iS2 service. Users identified in this way, however, do *not* have the same status as properly authenticated users. For security purposes, such users should enjoy lesser privileges and be treated in the same way as unauthenticated users.

The net effect of the configuration shown in [Example 43](#) is that the SOAP server performs authentication by contacting the central iS2 security service. See also “[Security Layer](#)” on [page 18](#) and “[Configuring the iS2 server](#)” on [page 99](#) for more details about configuring the iS2 service.

Routing

If you are using the Artix routing feature, you need to modify the WSDL by adding a `<routing:propagateInputAttribute>` tag, as shown in [Example 44](#).

Example 44: *Configuring a Router to Support Principal Propagation*

```
<definitions ... >
  ...
  <routing:route name="route_from_corba_to_soap">
    <routing:source service="tns:client"
port="CorbaClient"/>
    <routing:destination service="tns:server"
port="SoapServer"/>
    <routing:propagateInputAttribute name="Principal"/>
  </routing:route>
</definitions>
```

Programming

Overview

This section describes how to program an Artix client and server to set (client side) and get (server side) a principal value. The code examples are written using the transport-neutral message attributes API.

Client example

[Example 45](#) shows how to set the principal prior to invoking an operation, `echoString()`, on a proxy object, of `MyProxy` type.

Example 45: Setting a Principal on the Client Side

```
// C++
...
MyProxy proxy;

// Switch message attributes on.
1 proxy.get_port().use_input_message_attributes(true);

// Set the "Principal" attribute.
2 MessageAttributes& input_attributes =
  proxy.get_port().get_input_message_attributes();
3 input_attributes.set_string("Principal", "theprincipal");

// Now use the proxy as normal.
4 proxy.echoString();
```

The preceding client example can be explained as follows:

1. You must call `use_message_attributes()` on the proxy's port object to enable message attributes (which are responsible for propagating the principal). Because message attributes add a performance penalty, they are disabled by default.
2. This line gets a reference to the proxy's input message attributes object.
3. This line uses a transport-neutral API to set the `Principal` message attribute.

4. This line invokes a remote WSDL operation, `echoString()`, which includes the `Principal` attribute in the input message. The precise mechanism used for propagating the principal value is transport specific.

Server example

[Example 46](#) shows how to read the principal on the server side, when the servant is invoked by a client that uses principal propagation.

Example 46: Reading the Principal on the Server Side

```

1 // C++
  // Override the base Port activation method.
void MyImpl::activate(IT_Bus::Port& port)
{
2     port.use_input_message_attributes(true);
}

3 // in operation..
void MyImpl::echoString(const IT_Bus::String& inputString,
                        IT_Bus::String& Response)
IT_THROW_DECL((IT_Bus::Exception))
{
    Response = inputString;

    try
    {
4        Current& current=get_bus()->get_current();
5        Port& port=current.get_operation().get_port();
6        const String& the_principal =
            port().get_input_message_attributes().get_string(
                "Principal");
7    }
    catch (IT_Bus::NoSuchAttributeException) { }
}

```

The preceding server example can be explained as follows:

1. By overriding the port's virtual activation method, you ensure that each port created for this servant will have its attributes set properly.
2. You must call `use_message_attributes()` on the servant base class to enable message attributes. Because message attributes add a performance penalty, they are disabled by default.

3. This is the implementation of the `echoString()` operation that was called in [Example 45](#).
4. Get the `Current` object from the `Bus`. The `Current` object holds references to the port.
5. Get a reference to the port from the `Current` object.
6. This line uses the transport-neutral message attribute API to read the `Principal` value received from the client.
7. If the client has not sent a `Principal` attribute, the `IT_Bus:NoSuchAttributeException` exception is thrown.

Interoperating with .NET

Overview

If your Artix applications must interoperate with other Web service products, for example .NET, you need to modify your WSDL contract in order to make the principal header interoperable. This section describes the changes you can make to a WSDL contract to facilitate interoperability with other Web services platforms.

In this section

This section contains the following subsections:

Explicitly Declaring the Principal Header	page 224
Modifying the SOAP Header	page 226

Explicitly Declaring the Principal Header

Overview

Artix applications do not require any modifications to the WSDL contract in order to use principal headers. Whenever input message attributes are enabled (set by programming), an Artix service expects to receive the user's principal in a SOAP header.

In contrast to this, non-Artix services, for example, .NET services, require the principal header to be declared *explicitly* in the WSDL contract. Otherwise, the non-Artix services would be unable to access the principal.

Declaring the principal header in WSDL

[Example 47](#) shows the typical modifications you must make to a WSDL contract in order to make the principal value accessible to non-Artix applications.

Example 47: WSDL Declaration of the Principal Header

```

<definitions ... >
  <types>
    <schema targetNamespace="TypeSchema" ... >
      ...
      1     <element name="principal" type="xsd:string"/>
      ...
    </schema>
  </type>
  ...
  2 <message targetNamespace="http://schemas.iona.com/security"
      name="principal">
  3   <part element="TypePrefix:principal" name="principal"/>
  </message>
  ...
  4 <binding ... xmlns:sec="http://schemas.iona.com/security">
    ...
  5   <operation ...>
    ...
    <input>
      <soap:body ...>
      6     <soap:header message="sec:principal"
          part="principal" use="literal">
    </input>
    </operation>
  </binding>

```

Example 47: WSDL Declaration of the Principal Header

```
...  
</definitions>
```

The preceding WSDL extract can be explained as follows:

1. Declare a `<principal>` element in the type schema, which must be declared to be of type, `xsd:string`. In this example, the `<principal>` element belongs to the *TypeSchema* namespace.
2. Add a new `<message>` element.
3. The `<part>` tag's `element` attribute is set equal to the QName of the preceding `principal` element. Hence, in this example the *TypePrefix* appearing in `element="TypePrefix:principal"` must be a prefix associated with the *TypeSchema* namespace.
4. Edit the binding, or bindings, for which you might need to access the principal header. You should define a prefix for the `http://schemas.iona.com/security` namespace within the `<binding>` tag, which in this example is `sec`.
5. Edit each operation for which you might need to access the principal header.
6. Add a `<soap:header>` tag to the operation's input part, as shown.

Modifying the SOAP Header

Overview

It is possible to change the default format of the principal header by making appropriate modifications to the WSDL contract. It is usually not necessary to modify the header format in this way, but in some cases it could facilitate interoperability.

Default SOAP header

By default, when a client uses principal propagation with SOAP over HTTP, the input message sent over the wire includes the following form of header:

```
<SOAP-ENV:Header>
  <sec:principal xmlns:sec="http://schemas.ionas.com/security"
    xsi:type="xsd:string">my_principal</sec:principal>
</SOAP-ENV:Header>
```

Custom SOAP header

You can change the form of the SOAP header that is sent over the wire to have the following custom format (replacing `<sec:principal>` by a custom tag, `<sec:PrincipalTag>`):

```
<SOAP-ENV:Header>
  <sec:PrincipalTag xmlns:sec="http://schemas.ionas.com/security"
    xsi:type="xsd:string">my_principal</sec:PrincipalTag>
</SOAP-ENV:Header>
```

WSDL modifications

To change the tag that is sent in the SOAP header to be *PrincipalTag*, you can modify your WSDL contract as shown in [Example 48](#).

Example 48: Customizing the Form of the Principal Header

```
<definitions ... >
  <types>
    <schema targetNamespace="TypeSchema" ... >
      ...
      <element name="PrincipalTag" type="xsd:string"/>
      ...
    </schema>
  </types>
  ...
  <message targetNamespace="http://schemas.ionas.com/security"
```

Example 48: *Customizing the Form of the Principal Header*

```

    name="principal">
2      <part element="TypePrefix:PrincipalTag" name="principal"/>
</message>
...
<binding ... xmlns:sec="http://schemas.iona.com/security">
...
  <operation ...>
    ...
    <input>
3      <soap:body ...>
        <soap:header message="sec:principal"
          part="principal" use="literal">
        </input>
    </operation>
  </binding>
...
</definitions>

```

The preceding WSDL extract can be explained as follows:

1. Modify the `<principal>` element in the type schema to give it an arbitrary QName. In this example, the `<PrincipalTag>` element belongs to the *TypeSchema* namespace.
2. The `<part>` tag's `element` attribute is set equal to the QName of the preceding `principal` element. Hence, in this example the *TypePrefix* appearing in `element="TypePrefix:PrincipalTag"` must be a prefix associated with the *TypeSchema* namespace.
3. The `<soap:header>` tag must be defined precisely as shown here. That is, when writing or reading a principal header, Artix looks for the `principal` part of the message with QName, `principal`, in the namespace, `http://schemas.iona.com/security`.

Programming Authentication

To ensure that Web services and Web service clients developed using Artix can interoperate with the widest possible array of Web services, Artix supports the WS Security specification for propagating Kerberos security tokens and username/password security tokens in SOAP message headers. The security tokens are placed into the SOAP message header using Artix APIs that format the tokens and place them in the header correctly.

In this chapter

This chapter discusses the following topics:

Propagating a Username/Password Token	page 230
Propagating a Kerberos Token	page 232

Propagating a Username/Password Token

Overview

Many Web services use simple username/password authentication to ensure that only preapproved clients can access them. Artix provides a simple client side API for embedding the username and password into the SOAP message header of requests in a WS Security compliant manner.

Procedure

Embedding a username and password token into the SOAP header of a request using the Artix APIs requires you to do the following:

1. Instruct the proxy's port object to use the message attributes. The message attributes are responsible for propagating the token. Because the use of message attributes results in a performance hit, they are not used by default.
 2. Get a reference to the input message's message attributes.
 3. Set the `WSSEUsernameToken` property on the message attributes using the `set_string()` method to specify the username.
 4. Set the `WSSEPasswordToken` property on the message attributes using the `set_string()` method to specify the password.
-

Example

[Example 49](#) shows how to set the username/password token prior to invoking an operation on a proxy object of `MyProxy` type.

Example 49: Setting a Username/Password Token on the Client Side

```
// C++
...
MyProxy proxy;

// Switch message attributes on.
proxy.get_port().use_input_message_attributes(true);

// Get the message attributes.
MessageAttributes& input_attributes =
    proxy.get_port().get_input_message_attributes();
```

Example 49: *Setting a Username/Password Token on the Client Side*

```
//Set the username message attribute.  
input_attributes.set_string("WSSEUsernameToken",  
                            "artix_user");  
  
//Set the username message attribute.  
input_attributes.set_string("WSSEPasswordToken",  
                            "artix");
```

Propagating a Kerberos Token

Overview

Using the Kerberos Authentication Service requires you to make a few changes to your client code. First you need to acquire a valid Kerberos token. Then you need to embed it into the SOAP message header of all the request being made on the secure server.

Acquiring a Kerberos Token

To get a security token from the Kerberos Authentication Service is you must use platform specific APIs and then base64 encode the returned binary token so that it can be placed into the SOAP header.

On UNIX platforms use the GSS APIs to contact Kerberos and get a token for the service you wish to make requests upon. On Windows platforms use the Microsoft Security Framework APIs to contact Kerberos and get a token for the service you wish to contact.

Embedding the token in the SOAP header

Embedding a Kerberos token into the SOAP header of a request using the Artix APIs requires you to do the following:

1. Instruct the proxy's port object to use the message attributes. The message attributes are responsible for propagating the token. Because the use of message attributes results in a performance hit, they are not used by default.
2. Get a reference to the input message's message attributes.
3. Set the Kerberos token property in the message headers using the message attributes' `set_string()` method. The Kerberos token property is named `WSSEKerberosv5SToken`. The property's value is the base64 encoded string generated from the token obtained from the Kerberos Authentication Service.

[Example 50](#) shows how to set the Kerberos token prior to invoking an operation on a proxy object of `MyProxy` type.

Example 50: *Setting a Kerberos Token on the Client Side*

```
// C++
...
MyProxy proxy;

// The value of the token string placed in the SOAP header is a
// base64 encoded string created from the token recieved from
// Kerberos
String token_string = base64EncodedKerberosToken;

// Switch message attributes on.
proxy.get_port().use_input_message_attributes(true);

// Set the Kerberos token attribute.
MessageAttributes& input_attributes =
    proxy.get_port().get_input_message_attributes();
input_attributes.set_string("WSSEKerberosv5SToken",
                           token_string);
```


Setting Security Properties in Artix Contracts

Artix allows you to configure a number of security features directly from the Artix contract describing your system.

Overview

Ocasionally you will need finer grained control of your systems security than is provided through the standard Artix and security configuration. Artix provides the ability to control security on a per-port basis by describing the service's security settings in the Artix contract that describes it. This is done by using the `<bus:security>` extension in the `<port>` element describing the service's address and transport details.

Namespace

The XML namespace defining `<bus:security>` is `http://schemas.ionas.com/bus`. You will need to add the following line to the definitions element of any contracts that use the `<bus:security>` element:

```
xmlns:bus="http://schemas.ionas.com/bus"
```

<bus:security> attributes

All of the attributes to <bus:security> map directly to Artix configuration variables controlling security. The settings specified in the contract override the settings specified in the Artix configuration file, `artix.cfg`. They are all optional and are listed in [Table 8](#).

Table 8: *Contract Security Attributes*

Configuration Variable	Contract Attribute
<code>plugins:is2_authorization:action_role_mapping</code>	<code>is2AuthorizationActionRoleMapping</code>
<code>policies:asp:enable_security</code>	<code>enableSecurity</code>
<code>policies:asp:enable_authorization</code>	<code>enableAuthorization</code>
<code>plugins:asp:authentication_cache_size</code>	<code>authenticationCacheSize</code>
<code>plugins:asp:authentication_cache_timeout</code>	<code>authenticationCacheTimeout</code>
<code>plugins:asp:security_type</code>	<code>securityType</code>
<code>plugins:asp:security_level</code>	<code>securityLevel</code>
<code>plugins:asp:authorization_realm</code>	<code>authorizationRealm</code>
<code>plugins:asp:default_password</code>	<code>defaultPassword</code>

For a description of security configuration see [“Security Configuration” on page 245](#).

Examples

Disabling security for a service

[Example 51](#) shows how to disable security for the service `widgetService`.

Example 51: *Disabling Security in an Artix Contract*

```
<definitions ....
  xmlns:bus="http://schemas.ionas.com/bus"
  ...>
  ...
  <service name="widgetService">
    <port name="widgetServicePort" binding="tns:widgetSOAPBinding">
      <soap:address location="http://localhost:8080"/>
      <bus:security enableSecurity="false" />
    </port>
  </service>
</definitions>
```

Enabling security for a service

[Example 52](#) shows how to enable security for the service `personalInfoService`. For this example, it is assumed that no security configuration was specified in the Artix configuration.

Example 52: *Enabling Security in an Artix Contract*

```
<definitions ....
  xmlns:bus="http://schemas.ionas.com/bus"
  ...>
  ...
  <service name="personalInfoService">
    <port name="personalInfoServicePort" binding="tns:infoSOAPBinding">
      <soap:address location="http://localhost:8080"/>
      <bus:security enableSecurity="true"
        is2AuthorizationActionRoleMapping="file://c:/ionas/artix/2.0/bin/action_role.xml"
        enableAuthorization="true"
        securityLevel="REQUEST_LEVEL"
        securityType="USERNAME_PASSWORD"
        authenticationCacheSize="5"
        authenticationChaceTimeout="10" />
    </port>
  </service>
</definitions>
```

The `<bus:security>` element in [Example 52](#) fully configures `personalInfoService` to use WS Security compliant username/password authentication.

Overriding specific security properties for a service

[Example 53](#) shows how to specify that a particular service, `kerberosWidgetService`, is to use WS Security compliant Kerberos token for authentication while the remaining services in the domain are using HTTPS authentication.

Example 53: *Changing Security Configuration in an Artix Contract*

```
<definitions ...
  xmlns:bus="http://schemas.ionas.com/bus"
  ...>
  ...
  <service name="kerberosWidgetService">
    <port name="kerberosWidgetServicePort" binding="tns:widget.SOAPBinding">
      <soap:address location="http://localhost:8080"/>
      <bus:security securityLevel="REQUEST_LEVEL"
                    securityType="KERBEROS" />
    </port>
  </service>
</definitions>
```

Security Configuration

This appendix provides details of Artix security configuration variables.

In this appendix

This appendix contains the following sections:

initial_references Namespace	page 240
plugins Namespace	page 241
policies Namespace	page 252
principal_sponsor Namespace	page 261
principal_sponsor:csi Namespace	page 264

initial_references Namespace

List of configuration variables

The `initial_references` namespace contains the following configuration variables:

`initial_references:IT_TLS_Toolkit:plugin`

(Windows only.) This configuration variable enables you to specify the underlying SSL/TLS toolkit to be used by Orbix. 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 Namespace

List of configuration variables

The `plugins` namespace contains the following configuration variables.

`plugins:artix_security:shlib_name`

Specifies the root name of the shared library containing the `artix_security` plug-in.

This configuration variable is always initialized as follows:

```
plugins:artix_security:shlib_name = "it_security_plugin";
```

`plugins:asp: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.

`plugins:asp: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 iS2 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.

`plugins:asp: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`.

`plugins:asp: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 iS2 security service.

The default value is the string, `default_password`.

`plugins:asp:security_type`

Specifies the source of the user identity that is sent to the iS2 server 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:

<code>USERNAME_PASSWORD</code>	Authenticate the username and password propagated as WSDL message attributes. For example, you can configure these values on the client side using the <code>UserName</code> and <code>Password</code> attributes in the <code><http-conf:client></code> tag in the WSDL contract.
<code>CERT_SUBJECT</code>	Authenticate the Common Name (CN) from the client certificate's subject DN.
<code>ENCODED_TOKEN</code>	<i>Reserved for future use.</i>
<code>KERBEROS_TOKEN</code>	Authenticate the Kerberos token. You must have the Kerberos adapter configured to use this option. For more information.
<code>PRINCIPAL</code>	Authenticate the CORBA principal. This is needed to support interoperability with legacy CORBA applications. This options can be used in combination with the <code>plugins:asp:default_password</code> setting.

`plugins:asp: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:use_jsse_tk`

(Java only) Specifies whether or not to use the JSSE/JCE architecture with Orbix Java applications. 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:baltimore_toolkit:shlib_name`

(Windows only) Specifies the root name of the shared library containing the Baltimore SSL/TLS toolkit.

This configuration variable is always initialized as follows:

```
plugins:baltimore_toolkit:shlib_name = "it_tls_baltimore";
```

`plugins:gsp: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.

`plugins:gsp: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 iS2 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.

`plugins:gsp: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).

`plugins:gsp:enable_authorization`

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

Default is `true`.

`plugins:gsp:enable_gssup_sso`

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

`plugins:gsp:enable_x509_sso`

Enables certificate-based SSO when set to `true`.

`plugins:gsp: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`.

`plugins:gsp:enable_security_service_cert_authentication`

A boolean GSP policy that enables X.509 certificate-based authentication using the iS2 server.

Default is `false`.

`plugins:gsp: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 338](#).

`plugins:http: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`.

`plugins:http: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`.

`plugins:http: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`.

`plugins:http: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`.

`plugins:http: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`.

`plugins:http: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`.

`plugins:http: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`.

`plugins:http: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`.

`plugins:http: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`.

`plugins:http: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`.

`plugins:http: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`.

`plugins:http: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: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.

`plugins:iiop_tls: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.

`plugins:iiop_tls: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.

`plugins:iiop_tls: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.

`plugins:iiop_tls: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.

```
plugins:iiop_tls:enable_iiop_1_0_client_support
```

When this variable is set, the `iiop_tls` plug-in reads this variable's value instead of the `plugins:iiop:enable_iiop_1_0_client_support` variable's value.

```
plugins:iiop_tls:incoming_connections:hard_limit
```

Specifies the maximum number of incoming (server-side) connections permitted to IOP. IOP 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.

```
plugins:iiop_tls:incoming_connections:soft_limit
```

Specifies the number of connections at which IOP 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.

```
plugins:iiop_tls: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.

```
plugins:iiop_tls: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.

```
plugins:is2_authorization: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: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 338](#) 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_admin` 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.

```
plugins:kdm:iiop_tls:port
```

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

```
plugins:kdm: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_admin: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 338](#) 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:shlib_name
```

Specifies the root name of the shared library containing the `login_client` plug-in.

This configuration variable is always initialized as follows:

```
plugins:login_client:shlib_name = "it_login_client";
```

```
plugins:login_client: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:shlib_name
```

Specifies the root name of the shared library containing the `login_service` plug-in.

This configuration variable is always initialized as follows:

```
plugins:login_service:shlib_name = "it_login_service";
```

```
plugins:login_service: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: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`, Orbix 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:schannel_toolkit:shlib_name`

(Windows only) Specifies the root name of the shared library containing the Schannel SSL/TLS toolkit.

This configuration variable is always initialized as follows:

```
plugins:schannel_toolkit:shlib_name = "it_tls_schannel";
```

`plugins:security: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 Namespace

List of configuration variables

The policies namespace defines the default Artix policies. Many of these policies can also be set programmatically from within an application.

`policies: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.

`policies:asp:enable_authorization`

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

`policies:asp:enable_sso`

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

`policies: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.

`policies: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.

`policies: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.

`policies:csi:attribute_service:client_supports`

A client-side policy that specifies the association options supported by the CSIV2 attribute service (principal propagation). The only association option that can be specified is `IdentityAssertion`. This policy is normally specified in an intermediate server so that it propagates CSIV2 identity tokens to a target server. For example:

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

`policies:csi:attribute_service:target_supports`

A server-side policy that specifies the association options supported by the CSIV2 attribute service (principal propagation). The only association option that can be specified is `IdentityAssertion`. For example:

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

`policies:csi: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.

```
policies:csi:auth_over_transport:client_supports
```

A client-side policy that specifies the association options supported by CSlv2 authorization over transport. The only association option that can be specified is `EstablishTrustInClient`. For example:

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

```
policies:csi:auth_over_transport:server_domain_name
```

The iSF security domain (CSlv2 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 CSlv2 client about to open a connection to this server would check that the domain name in its own CSlv2 credentials matches the domain name embedded in the IOR.

```
policies:csi:auth_over_transport:target_requires
```

A server-side policy that specifies the association options required for CSlv2 authorization over transport. The only association option that can be specified is `EstablishTrustInClient`. For example:

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

```
policies:csi:auth_over_transport:target_supports
```

A server-side policy that specifies the association options supported by CSlv2 authorization over transport. The only association option that can be specified is `EstablishTrustInClient`. For example:

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

```
policies:iiop_tls: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).

```
policies:iiop_tls: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.

`policies:iiop_tls: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.

`policies:iiop_tls:certificate_constraints_policy`

A list of constraints applied to peer certificates—see [“Applying Constraints to Certificates” on page 338](#) 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.

`policies:iiop_tls: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 [“Association Options” on page 535](#). For defaults, see [“Choosing Client Behavior” on page 296](#).

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

`policies:iiop_tls: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 [“Association Options” on page 535](#). For defaults, see [“Choosing Client Behavior” on page 296](#).

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

`policies:iiop_tls:client_version_policy`

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

`policies:iiop_tls:connection_attempts`

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

`policies:iiop_tls:connection_retry_delay`

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

`policies:iiop_tls:max_chain_length_policy`

The maximum certificate chain length that an ORB will accept (see [“Certificate Chaining” on page 251](#)).

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

`policies:iiop_tls:mechanism_policy:ciphersuites`

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

Table 9: *Mechanism Policy Cipher Suites*

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

`policies:iiop_tls:mechanism_policy:protocol_version`

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 Orbix application deployed on the OS/390 platform. Orbix

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";
```

```
policies:iiop_tls: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.

```
policies:iiop_tls: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.

```
policies:iiop_tls: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.

```
policies:iiop_tls: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.

```
policies:iiop_tls: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.

```
policies:iiop_tls:session_caching_policy
```

When this policy is set, the `iiop_tls` plug-in reads this policy's value instead of the `policies:session_caching` policy's value (C++) or `policies:session_caching_policy` policy's value (Java).

`policies:iiop_tls: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—see “Association Options” on page 535. For defaults, see “Choosing Target Behavior” on page 298.

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

`policies:iiop_tls:target_secure_invocation_policy:supports`

Specifies the maximum level of security supported by a server. The value of this variable is specified as a list of association options—see “Association Options” on page 535. For defaults, see “Choosing Target Behavior” on page 298.

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

`policies:iiop_tls: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.

`policies:iiop_tls: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.

`policies:iiop_tls: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.

`policies:iiop_tls:trusted_ca_list_policy`

Contains a list of filenames (or a single filename), each of which contains a concatenated list of CA certificates in PEM format. The aggregate of the CAs in all of the listed files is the set of trusted CAs. For example, you might specify two files containing CA lists as follows:

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

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

See also [“Certificate Chaining” on page 251](#).

```
policies:max_chain_length_policy
```

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

```
policies:mechanism_policy:ciphersuites
```

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

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.

```
policies:mechanism_policy:protocol_version
```

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

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.

```
policies:session_caching_policy
```

(Java only)

Specifies whether an ORB caches the session information for secure associations when acting in a client role, a server role, or both. The

purpose of session caching is to enable closed connections to be re-established quickly. The following values are supported:

```
CACHE_NONE
CACHE_CLIENT
CACHE_SERVER
CACHE_SERVER_AND_CLIENT
```

The policy can also be set programmatically using the

```
IT_TLS_API::SessionCachingPolicy CORBA policy. Default is
CACHE_NONE.
```

`policies:session_caching`
(C++ only)

Same effect as the `policies:session_caching_policy` variable, except it affects C++ applications instead of Java applications.

`policies: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.)
```

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.

`policies: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.)
```

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.

`policies:trusted_ca_list_policy`

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

principal_sponsor Namespace

List of configuration variables

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 may 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.

`principal_sponsor:use_principal_sponsor`

A boolean value that determines 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.

`principal_sponsor:auth_method_id`

A string that selects 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";
```

`principal_sponsor:auth_method_data`

A string array containing information to be interpreted by the authentication method represented by the `auth_method_id`.

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

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

<code>provider</code>	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 <code>dkck132.d11</code> (from Baltimore).
<code>slot</code>	The number of a particular slot on the smart card (for example, 0) containing the user's credentials.
<code>pin</code>	A PIN to gain access to the smart card— <i>optional</i> . 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, the following authentication data can be provided in `auth_method_data`:

<code>label</code>	(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 as specified by `filename=` is not found, it is searched for on the classpath.
- ◆ The file as 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 a couple of 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.

Java

```
principal_sponsor:callback_handler:ClassName
```

A string that provides the class name of an interface that implements the interface `com.ionacorba.tls.auth.CallbackHandler`.

```
principal_sponsor:callback_handler:ClassName
    ="demos.tls.login.DemoCallbackHandler";
```

Java

```
principal_sponsor:login_attempts
```

An integer that specifies how many times a user will be 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`.

principal_sponsor:csi Namespace

List of configuration variables

The `principal_sponsor:csi` 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 may 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.

`principal_sponsor:csi: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`.

`principal_sponsor:csi:use_principal_sponsor`

A boolean value that switches the CSI principal sponsor on or off. If `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`.

`principal_sponsor:csi:auth_method_id`

A string that selects the authentication method to be used by the CSI application. The following authentication methods are available:

<code>GSSUPMech</code>	The Generic Security Service Username/Password (GSSUP) mechanism.
------------------------	---

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

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

```
principal_sponsor:csi:auth_method_data
```

A string array containing information to be interpreted by the authentication method represented by the `auth_method_id`.

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

<code>username</code>	The username for CSIV2 authorization over transport. Note that authentication of CSIV2 usernames and passwords is performed on the server side.
<code>password</code>	The password associated with <code>username</code> . It is not recommended to supply the password from configuration for deployed systems.
<code>domain</code>	The CSIV2 authentication domain in which the username/password pair is authenticated. When the client is about to open a new connection, this domain name is compared with the domain name embedded in the relevant IOR (see <code>policies:csi:auth_over_transport:server_domain_name</code>). The domain names must match. Note: If <code>domain</code> is an empty string, it matches any target domain. That is, an empty domain string is equivalent to a wildcard.

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

For example, to log on to a CSIV2 application as the `administrator` user in the `US-SantaClara` domain:

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

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

Note: It is currently not possible to customize the login prompt associated with the CSIV2 principal sponsor. As an alternative, you could implement your own login GUI by programming and pass the user input directly to the principal authenticator.

iS2 Configuration

This appendix provides details of how to configure the iS2 server.

In this appendix

This appendix contains the following sections:

Properties File Syntax	page 268
iS2 Properties File	page 269
log4j Properties File	page 281

Properties File Syntax

Overview

The iS2 server uses standard Java property files for its configuration. Some aspects of the Java properties file syntax are summarized here for your convenience.

Property definitions

A property is defined with the following syntax:

```
<PropertyName>=<PropertyValue>
```

The *<PropertyName>* is a compound identifier, with each component delimited by the . (period) character. For example, *is2.current.server.id*. The *<PropertyValue>* is an arbitrary string, including all of the characters up to the end of the line (embedded spaces are allowed).

Specifying full pathnames

When setting a property equal to a filename, you normally specify a full pathname, as follows:

UNIX

```
/home/data/securityInfo.xml
```

Windows

```
D:/iona/securityInfo.xml
```

or, if using the backslash as a delimiter, it must be escaped as follows:

```
D:\\iona\\securityInfo.xml
```

Specifying relative pathnames

If you specify a relative pathname when setting a property, the root directory for this path must be added to the iS2 server's classpath. For example, if you specify a relative pathname as follows:

UNIX

```
securityInfo.xml
```

The iS2 server's classpath must include the file's parent directory. For example:

```
CLASSPATH = /home/data/:<rest_of_classpath>
```

iS2 Properties File

Overview

An iS2 properties file is used to store the properties that configure a specific iS2 server instance. Generally, every iS2 server instance should have its own iS2 properties file. This section provides descriptions of all the properties that can be specified in an iS2 properties file.

File location

The default location of the iS2 properties file is the following:

ASPInstallDir/etc/domains/DomainName/is2.properties

In general, the iS2 properties file location is specified in the Orbix configuration by setting the `is2.properties` property in the `plugins:java_server:system_properties` property list.

For example, on UNIX the security server's property list is normally initialized in the `iona_services.security` configuration scope as follows:

```
# Orbix configuration file
...
iona_services {
    ...
    security {
        ...
        plugins:java_server:system_properties =
["org.omg.CORBA.ORBClass=com.iona.corba.art.artimpl.ORBImpl",
"org.omg.CORBA.ORBSingletonClass=com.iona.corba.art.artimpl.O
RBSingleton",
"is2.properties=ASPInstallDir/etc/domains/DomainName/is2.pro
perties"];
        ...
    };
};
```

List of properties

The following properties can be specified in the iS2 properties file:

`com.iona.isp.adapters`

Specifies the iS2 adapter type to be loaded by the iS2 server at runtime. Choosing a particular adapter type is equivalent to choosing an iSF security domain. Currently, you can specify one of the following adapter types:

- ◆ file
- ◆ LDAP
- ◆ SiteMinder

For example, you can select the LDAP adapter as follows:

```
com.iona.isp.adapters=LDAP
```

```
com.iona.isp.adapter.file.class
```

Specifies the Java class that implements the file adapter.

For example, the default implementation of the file adapter provided with Orbix E2A ASP is selected as follows:

```
com.iona.isp.adapter.file.class=com.iona.security.is2adapter.  
file.FileAuthAdapter
```

```
com.iona.isp.adapter.file.param.filename
```

Specifies the name and location of a file that is used by the file adapter to store user authentication data.

For example, you can specify the file,

`C:/is2_config/security_info.xml`, as follows:

```
com.iona.isp.adapter.file.param.filename=C:/is2_config/securi  
ty_info.xml
```

```
com.iona.isp.adapter.file.params
```

Obsolete. This property was needed by earlier versions of the iS2 server, but is now ignored.

```
com.iona.isp.adapter.LDAP.class
```

Specifies the Java class that implements the LDAP adapter.

For example, the default implementation of the LDAP adapter provided with Orbix E2A ASP is selected as follows:

```
com.iona.isp.adapter.LDAP.class=com.iona.security.is2adapter.  
ldap.LdapAdapter
```

```
com.iona.isp.adapter.LDAP.param.CacheSize
```

Specifies the maximum LDAP cache size in units of bytes. This maximum applies to the *total* LDAP cache size, including all LDAP connections opened by this iS2 server instance.

Internally, the iS2 server uses a third-party toolkit (currently the *iPlanet SDK*) to communicate with an LDAP server. The cache referred to here is one that is maintained by the LDAP third-party toolkit. Data

retrieved from the LDAP server is temporarily stored in the cache in order to optimize subsequent queries.

For example, you can specify a cache size of 1000 as follows:

```
com.iona.isp.adapter.LDAP.param.CacheSize=1000
```

```
com.iona.isp.adapter.LDAP.param.CacheTimeToLive
```

Specifies the LDAP cache time to-live in units of seconds.

```
com.iona.isp.adapter.LDAP.param.CacheTimeToLive=60
```

```
com.iona.isp.adapter.LDAP.param.GroupBasedDN
```

Specifies the base DN of the tree in the LDAP directory that stores user groups.

For example, you could use the RDN sequence, DC=iona,DC=com, as a base DN by setting this property as follows:

```
com.iona.isp.adapter.LDAP.param.GroupBasedDN=dc=iona,dc=com
```

Note: The order of the RDNs is significant. The order should be based on the LDAP schema configuration.

```
com.iona.isp.adapter.LDAP.param.GroupNameAttr
```

Specifies the attribute type whose corresponding attribute value gives the name of the user group.

Default is CN.

For example, you can use the common name, CN, attribute type to store the user group's name by setting this property as follows:

```
com.iona.isp.adapter.LDAP.param.GroupNameAttr=cn
```

```
com.iona.isp.adapter.LDAP.param.GroupObjectClass
```

Specifies the object class that applies to user group entries in the LDAP directory structure. An object class defines the required and allowed attributes of an entry.

Default is `groupOfUniqueNames`.

For example, to specify that all user group entries belong to the `groupOfUniqueNames` object class:

```
com.iona.isp.adapter.LDAP.param.GroupObjectClass=groupofunique  
names
```

`com.iona.isp.adapter.LDAP.param.GroupSearchScope`

Specifies the group search scope. The search scope is the starting point of a search and the depth from the base DN to which the search should occur. This property can be set to one of the following values:

- ◆ `BASE`—Search a single entry (the base object).
- ◆ `ONE`—Search all entries immediately below the base DN.
- ◆ `SUB`—Search all entries from a whole subtree of entries.

Default is `SUB`.

For example:

```
com.iona.isp.adapter.LDAP.param.GroupSearchScope=SUB
```

`com.iona.isp.adapter.LDAP.param.host.<cluster_index>`

For the `<cluster_index>` LDAP server replica, specifies the IP hostname where the LDAP server is running. The `<cluster_index>` is 1 for the primary server, 2 for the first failover replica, and so on.

For example, you could specify that the primary LDAP server is running on host 10.81.1.100 as follows:

```
com.iona.isp.adapter.LDAP.param.host.1=10.81.1.100
```

`com.iona.isp.adapter.LDAP.param.MaxConnectionPoolSize`

Specifies the maximum LDAP connection pool size for the iS2 server (a strictly positive integer). The maximum connection pool size is the maximum number of LDAP connections that would be opened and cached by the iS2 server.

Default is 1.

For example, to limit the iS2 server to open a maximum of 50 LDAP connections at a time:

```
com.iona.isp.adapter.LDAP.param.MaxConnectionPoolSize=50
```

`com.iona.isp.adapter.LDAP.param.MemberDNAttr`

Specifies which LDAP attribute is used to retrieve group members. The LDAP adapter uses the `MemberDNAttr` property to construct a query to find out which groups a user belongs to.

The list of the user's groups is needed to determine the complete set of roles assigned to the user. The LDAP adapter determines the complete set of roles assigned to a user as follows:

- i. The adapter retrieves the roles assigned directly to the user.
- ii. The adapter finds out which groups the user belongs to, and retrieves all the roles assigned to those groups.

Default is `uniqueMember`.

For example, you can select the `uniqueMember` attribute as follows:

```
com.iona.isp.adapter.LDAP.param.MemberDNAttr=uniqueMember
```

```
com.iona.isp.adapter.LDAP.param.MemberFilter
```

Specifies how to search for members in a group. The value specified for this property must be an LDAP search filter (can be a custom filter).

No default.

```
com.iona.isp.adapter.LDAP.param.MinConnectionPoolSize
```

Specifies the minimum LDAP connection pool size for the iS2 server. The minimum connection pool size specifies the number of LDAP connections that are opened during initialization of the iS2 server.

Default is `1`.

For example, to specify a minimum of 10 LDAP connections at a time:

```
com.iona.isp.adapter.LDAP.param.MinConnectionPoolSize=10
```

```
com.iona.isp.adapter.LDAP.param.port.<cluster_index>
```

For the `<cluster_index>` LDAP server replica, specifies the IP port where the LDAP server is listening. The `<cluster_index>` is `1` for the primary server, `2` for the first failover replica, and so on.

Default is `389`.

For example, you could specify that the primary LDAP server is listening on port `636` as follows:

```
com.iona.isp.adapter.LDAP.param.port.1=636
```

```
com.iona.isp.adapter.LDAP.param.PrincipalUserDN.<cluster_index>
```

For the `<cluster_index>` LDAP server replica, specifies the username that is used to login to the LDAP server (in distinguished name format). This property need only be set if the LDAP server is configured to require username/password authentication.

No default.

`com.iona.isp.adapter.LDAP.param.PrincipalUserPassword.<cluster_index>`

For the `<cluster_index>` LDAP server replica, specifies the password that is used to login to the LDAP server. This property need only be set if the LDAP server is configured to require username/password authentication.

No default.

WARNING: Because the password is stored in plaintext, you must ensure that the `is2.properties` file is readable and writable only by users with administrator privileges.

`com.iona.isp.adapter.LDAP.param.RetrieveAuthInfo`

Specifies whether or not the iS2 server retrieves authorization information from the LDAP server. This property selects one of the following alternatives:

- ◆ `yes`—the iS2 server retrieves authorization information from the LDAP server.
- ◆ `no`—the iS2 server retrieves authorization information from the iS2 authorization manager..

Default is `no`.

For example, to use the LDAP server's authorization information:

`com.iona.isp.adapter.LDAP.param.RetrieveAuthInfo=yes`

`com.iona.isp.adapter.LDAP.param.RoleNameAttr`

Specifies the attribute type that the LDAP server uses to store the role name.

Default is `CN`.

For example, you can specify the common name, `CN`, attribute type as follows:

`com.iona.isp.adapter.LDAP.param.RoleNameAttr=cn`

`com.iona.isp.adapter.LDAP.param.SSLCACertDir.<cluster_index>`

For the `<cluster_index>` LDAP server replica, specifies the directory name for trusted CA certificates. All certificate files in this directory are loaded and set as trusted CA certificates, for the purpose of opening an

SSL connection to the LDAP server. The CA certificates can either be in DER-encoded X.509 format or in PEM-encoded X.509 format.

No default.

For example, to specify that the primary LDAP server uses the `d:/certs/test` directory to store CA certificates:

```
com.iona.isp.adapter.LDAP.param.SSLCACertDir.1=d:/certs/test
```

```
com.iona.isp.adapter.LDAP.param.SSLClientCertFile.<cluster_index>
```

Specifies the client certificate file that is used to identify the iS2 server to the `<cluster_index>` LDAP server replica. This property is needed only if the LDAP server requires SSL/TLS mutual authentication. The certificate must be in PKCS#12 format.

No default.

```
com.iona.isp.adapter.LDAP.param.SSLClientCertPassword.<cluster_index>
```

Specifies the password for the client certificate that identifies the iS2 server to the `<cluster_index>` LDAP server replica. This property is needed only if the LDAP server requires SSL/TLS mutual authentication.

No default.

WARNING: Because the password is stored in plaintext, you must ensure that the `is2.properties` file is readable and writable only by users with administrator privileges.

```
com.iona.isp.adapter.LDAP.param.SSLEnabled.<cluster_index>
```

Enables SSL/TLS security for the connection between the iS2 server and the `<cluster_index>` LDAP server replica. The possible values are `yes` or `no`.

Default is `no`.

For example, to enable an SSL/TLS connection to the primary LDAP server:

```
com.iona.isp.adapter.LDAP.param.SSLEnabled.1=yes
```

```
com.iona.isp.adapter.LDAP.param.UseGroupAsRole
```

Specifies whether a user's groups should be treated as roles. The following alternatives are available:

- ♦ `yes`—each group name is interpreted as a role name.
- ♦ `no`—for each of the user’s groups, retrieve all roles assigned to the group.

This option is useful for some older versions of LDAP, such as iPlanet 4.0, that do not have the role concept.

Default is `no`.

For example:

```
com.iona.isp.adapter.LDAP.param.UseGroupAsRole=no
```

```
com.iona.isp.adapter.LDAP.param.UserBaseDN
```

Specifies the base DN (an ordered sequence of RDNs) of the tree in the LDAP directory that stores user object class instances.

For example, you could use the RDN sequence, `DC=iona,DC=com`, as a base DN by setting this property as follows:

```
com.iona.isp.adapter.LDAP.param.UserBaseDN=dc=iona,dc=com
```

```
com.iona.isp.adapter.LDAP.param.UserCertAttrName
```

Specifies the attribute type that stores a user certificate.

Default is `userCertificate`.

For example, you can explicitly specify the attribute type for storing user certificates to be `userCertificate` as follows:

```
com.iona.isp.adapter.LDAP.param.UserCertAttrName=userCertificate
```

```
com.iona.isp.adapter.LDAP.param.UserNameAttr=uid
```

Specifies the attribute type whose corresponding value uniquely identifies the user. This is the attribute used as the user’s login ID.

Default is `uid`.

For example:

```
com.iona.isp.adapter.LDAP.param.UserNameAttr=uid
```

```
com.iona.isp.adapter.LDAP.param.UserObjectClass
```

Specifies the attribute type for the object class that stores users.

Default is `organizationalPerson`.

For example:

```
com.iona.isp.adapter.LDAP.param.UserObjectClass=organizationalPerson
```

`com.iona.isp.adapter.LDAP.param.UserRoleDNAttr`

Specifies the attribute type that stores a user's role DN.

Default is `nsRoleDn` (from the Netscape LDAP directory schema).

For example:

```
com.iona.isp.adapter.LDAP.param.UserRoleDNAttr=nsroledn
```

`com.iona.isp.adapter.LDAP.param.UserSearchFilter`

Custom filter for retrieving users. In the current version, `$USER_NAME$` is the only replaceable parameter supported. This parameter would be replaced during runtime by the LDAP adapter with the current User's login ID. This property uses the standard LDAP search filter syntax.

For example:

```
&(uid=$USER_NAME$(objectclass=organizationalPerson)
```

`com.iona.isp.adapter.LDAP.param.UserSearchScope`

Specifies the user search scope. This property can be set to one of the following values:

- ◆ `BASE`—Search a single entry (the base object).
- ◆ `ONE`—Search all entries immediately below the base DN.
- ◆ `SUB`—Search all entries from a whole subtree of entries.

Default is `SUB`.

For example:

```
com.iona.isp.adapter.LDAP.param.UserSearchScope=SUB
```

`com.iona.isp.adapter.LDAP.param.version`

Specifies the LDAP protocol version that the iS2 server uses to communicate with LDAP servers. The possible values are 2 (for LDAP v2, <http://www.ietf.org/rfc/rfc1777.txt>) or 3 (for LDAP v3, <http://www.ietf.org/rfc/rfc2251.txt>).

Default is 3.

For example, to select the LDAP protocol version 3:

```
com.iona.isp.adapter.LDAP.param.version=3
```

`com.iona.isp.adapter.LDAP.params`

Obsolete. This property was needed by earlier versions of the iS2 server, but is now ignored.

`com.iona.security.azmgr.adminUserName`

Specifies the name of the user who has privileges to administer the authorization manager. The specified user's details must be stored in the enterprise security backend accessed through an iS2 adapter (for example, file adapter, LDAP adapter, or SiteMinder adapter).

For example, you can specify that the `IONAAdmin` user administers the authorization manager by setting this property as follows:

```
com.iona.security.azmgr.adminUserName=IONAAdmin
```

`com.iona.security.azmgr.PersistencePropertyFileName`

Specifies the file that stores the configuration properties of the authorization manager.

For example, you can store the authorization manager configuration properties in a file, `C:/azmgr_config/azmgr.properties`, as follows:

```
com.iona.security.azmgr.PersistencePropertyFileName=C:/azmgr_
config/azmgr.properties
```

`com.iona.security.azmgr.RealmName`

Specifies the authorization manager's realm name. Only a single realm name can be specified.

For example, to specify the realm name, `Engineering`:

```
com.iona.security.azmgr.RealmName=Engineering
```

`com.iona.security.azmgr.useIONAAuthorizationManager`

Enables the authorization manager component of the iS2 server.

Possible values are `yes` or `no`.

For example:

```
com.iona.security.azmgr.useIONAAuthorizationManager=yes
```

`is2.current.server.id`

Specifies the current iS2 server's ID (required for clustering). When a secure application obtains a single sign-on (SSO) token from this iS2 server, the server ID is embedded into the SSO token. Subsequently, if the SSO token is passed to a *second* iS2 server instance, the second iS2 server recognizes that the SSO token originates from the first iS2 server and delegates security operations to the first iS2 server.

For example, to assign a server ID of `1` to the current iS2 server:

```
is2.current.server.id=1
```

```
is2.cluster.properties.filename
```

Specifies the file that stores the configuration properties for clustering.

```
is2.cluster.properties.filename=C:/is2_config/cluster.properties
```

```
is2.sso.cache.size
```

Specifies the maximum cache size (number of user sessions) associated with single sign-on (SSO) feature. The SSO caches user information, including the user's group and role information. If the maximum cache size is reached, the oldest sessions are deleted from the session cache.

No default.

For example:

```
is2.sso.cache.size=1000
```

```
is2.sso.enabled
```

Enables the single sign-on (SSO) feature of the iS2 server. The possible values are `yes` (enabled) and `no` (disabled).

Default is `yes`.

For example:

```
is2.sso.enabled=yes
```

```
is2.sso.session.idle.timeout
```

Sets the session idle time-out in units of seconds for the single sign-on (SSO) feature of the iS2 server. A zero value implies no time-out.

If a user logs on to the IONA Security Framework (supplying username and password) with SSO enabled, the iS2 server returns an SSO token for the user. The next time the user needs to access a resource, there is no need to log on again because the SSO token can be used instead. However, if no secure operations are performed using the SSO token for the length of time specified in the idle time-out, the SSO token expires and the user must log on again.

Default is `0` (no time-out).

For example:

```
is2.sso.session.idle.timeout=0
```

`is2.sso.session.timeout`

Sets the absolute session time-out in units of seconds for the single sign-on (SSO) feature of the iS2 server. A zero value implies no time-out.

This is the maximum length of time since the time of the original user login for which an SSO token remains valid. After this time interval elapses, the session expires irrespective of whether the session has been active or idle. The user must then login again.

Default is 0 (no time-out).

For example:

```
is2.sso.session.timeout=0
```

`log4j.configuration`

Specifies the log4j configuration filename. You can use the properties in this file to customize the level of debugging output from the iS2 server. See also [“log4j Properties File” on page 281](#).

For example,

```
log4j.configuration=d:/temp/myconfig.txt
```

log4j Properties File

Overview

The log4j properties file configures log4j logging for your iS2 server. This section describes a minimal set of log4j properties that can be used to configure basic logging.

log4j version

The iS2 server is built with log4j version.

log4j documentation

For complete log4j documentation, see the following Web page:
<http://jakarta.apache.org/log4j/docs/documentation.html>

File location

The location of the log4j properties file is specified by the `log4j.configuration` property in the iS2 properties file. For ease of administration, different iS2 server instances can optionally share a common log4j properties file.

List of properties

To give you some idea of the capabilities of log4j, the following is an incomplete list of properties that can be specified in a log4j properties file:

```
log4j.appender.<AppenderHandle>
```

This property specifies a log4j appender class that directs `<AppenderHandle>` logging messages to a particular destination. For example, one of the following standard log4j appender classes could be specified:

```
org.apache.log4j.ConsoleAppender
org.apache.log4j.FileAppender
org.apache.log4j.RollingFileAppender
org.apache.log4j.DailyRollingFileAppender
org.apache.log4j.AsyncAppender
org.apache.log4j.WriterAppender
```

For example, to log messages to the console screen for the `A1` appender handle:

```
log4j.appender.A1=org.apache.log4j.ConsoleAppender
```

```
log4j.appender.<AppenderHandle>.layout
```

This property specifies a log4j layout class that is used to format `<AppenderHandle>` logging messages. One of the following standard log4j layout classes could be specified:

```
org.apache.log4j.PatternLayout
org.apache.log4j.HTMLLayout
org.apache.log4j.SimpleLayout
org.apache.log4j.TTCCLayout
```

For example, to use the pattern layout class for log messages processed by the `A1` appender:

```
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
```

```
log4j.appender.<AppenderHandle>.layout.ConversionPattern
```

This property is used only in conjunction with the `org.apache.log4j.PatternLayout` class (when specified by the `log4j.appender.<AppenderHandle>.layout` property) to define the format of a log message.

For example, you can specify a basic conversion pattern for the `A1` appender as follows:

```
log4j.appender.A1.layout.ConversionPattern=%-4r [%t] %-5p %c
    %x - %m%n
```

```
log4j.rootCategory
```

This property is used to specify the logging level of the root logger and to associate the root logger with one or more appenders. The value of this property is specified as a comma separated list as follows:

```
<LogLevel>, <AppenderHandle01>, <AppenderHandle02>, ...
```

The logging level, `<LogLevel>`, can have one of the following values:

```
DEBUG
INFO
WARN
ERROR
FATAL
```

An appender handle is an arbitrary identifier that associates a logger with a particular logging destination.

For example, to select all messages at the `DEBUG` level and direct them to the `A1` appender, you can set the property as follows:

```
log4j.rootCategory=DEBUG, A1
```

ASN.1 and Distinguished Names

The OSI Abstract Syntax Notation One (ASN.1) and X.500 Distinguished Names play an important role in the security standards that define X.509 certificates and LDAP directories.

In this appendix

This appendix contains the following section:

ASN.1	page 284
Distinguished Names	page 285

ASN.1

Overview

The *Abstract Syntax Notation One* (ASN.1) was defined by the OSI standards body in the early 1980s to provide a way of defining data types and structures that is independent of any particular machine hardware or programming language. In many ways, ASN.1 can be considered a forerunner of the OMG's IDL, because both languages are concerned with defining platform-independent data types.

ASN.1 is important, because it is widely used in the definition of standards (for example, SNMP, X.509, and LDAP). In particular, ASN.1 is ubiquitous in the field of security standards—the formal definitions of X.509 certificates and distinguished names are described using ASN.1 syntax. You do not require detailed knowledge of ASN.1 syntax to use these security standards, but you need to be aware that ASN.1 is used for the basic definitions of most security-related data types.

BER

The OSI's Basic Encoding Rules (BER) define how to translate an ASN.1 data type into a sequence of octets (binary representation). The role played by BER with respect to ASN.1 is, therefore, similar to the role played by GIOP with respect to the OMG IDL.

DER

The OSI's Distinguished Encoding Rules (DER) are a specialization of the BER. The DER consists of the BER plus some additional rules to ensure that the encoding is unique (BER encodings are not).

References

You can read more about ASN.1 in the following standards documents:

- ASN.1 is defined in X.208.
- BER is defined in X.209.

Distinguished Names

Overview

Historically, distinguished names (DN) were defined as the primary keys in an X.500 directory structure. In the meantime, however, DNs have come to be used in many other contexts as general purpose identifiers. In the Artix Security Framework, DNs occur in the following contexts:

- X.509 certificates—for example, one of the DNs in a certificate identifies the owner of the certificate (the security principal).
- LDAP—DNs are used to locate objects in an LDAP directory tree.

String representation of DN

Although a DN is formally defined in ASN.1, there is also an LDAP standard that defines a UTF-8 string representation of a DN (see RFC 2253). The string representation provides a convenient basis for describing the structure of a DN.

Note: The string representation of a DN does *not* provide a unique representation of DER-encoded DN. Hence, a DN that is converted from string format back to DER format does not always recover the original DER encoding.

DN string example

The following string is a typical example of a DN:

```
C=US,O=IONA Technologies,OU=Engineering,CN=A. N. Other
```

Structure of a DN string

A DN string is built up from the following basic elements:

- [OID](#).
- [Attribute types](#).
- [AVA](#).
- [RDN](#).

OID

An OBJECT IDENTIFIER (OID) is a sequence of bytes that uniquely identifies a grammatical construct in ASN.1.

Attribute types

The variety of attribute types that could appear in a DN is theoretically open-ended, but in practice only a small subset of attribute types are used. [Table 10](#) shows a selection of the attribute types that you are most likely to encounter:

Table 10: *Commonly Used Attribute Types*

String Representation	X.500 Attribute Type	Size of Data	Equivalent OID
C	countryName	2	2.5.4.6
O	organizationName	1...64	2.5.4.10
OU	organizationalUnitName	1...64	2.5.4.11
CN	commonName	1...64	2.5.4.3
ST	stateOrProvinceName	1...64	2.5.4.8
L	localityName	1...64	2.5.4.7
STREET	streetAddress		
DC	domainComponent		
UID	userid		

AVA

An *attribute value assertion* (AVA) assigns an attribute value to an attribute type. In the string representation, it has the following syntax:

```
<attr-type>=<attr-value>
```

For example:

```
CN=A. N. Other
```

Alternatively, you can use the equivalent OID to identify the attribute type in the string representation (see [Table 10](#)). For example:

```
2.5.4.3=A. N. Other
```

RDN

A *relative distinguished name* (RDN) represents a single node of a DN (the bit that appears between the commas in the string representation).

Technically, an RDN might contain more than one AVA (it is formally defined as a set of AVAs); in practice, however, this almost never occurs. In the string representation, an RDN has the following syntax:

`<attr-type>=<attr-value>[+<attr-type>=<attr-value> ...]`

Here is an example of a (very unlikely) multiple-value RDN:

`OU=Eng1+OU=Eng2+OU=Eng3`

Here is an example of a single-value RDN:

`OU=Engineering`

Action-Role Mapping DTD

This appendix presents the document type definition (DTD) for the action-role mapping XML file.

DTD file

The action-role mapping DTD is shown in [Example 54](#).

Example 54:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT action-name (#PCDATA)>
<!ELEMENT role-name (#PCDATA)>
<!ELEMENT server-name (#PCDATA)>
<!ELEMENT action-role-mapping (server-name, interface+)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT interface (name, action-role+)>
<!ELEMENT action-role (action-name, role-name+)>
<!ELEMENT allow-unlisted-interfaces (#PCDATA)>
<!ELEMENT secure-system (allow-unlisted-interfaces*,
    action-role-mapping+)>
```

Action-role mapping elements

The elements of the action-role mapping DTD can be described as follows:

`<!ELEMENT action-name (#PCDATA)>`

Specifies the action name to which permissions are assigned. In Artix, the action name is equivalent to a WSDL operation name. That is, the *OperationName* from a tag `<operation name="OperationName">`.

You can also use the wildcard, `*`, to match all action names (WSDL operation names) in an interface (WSDL port type). Use the wildcard to assign roles to all actions in an interface.

`<!ELEMENT action-role (action-name, role-name+)>`

Groups together a particular action and all of the roles permitted to perform that action.

`<!ELEMENT action-role-mapping (server-name, interface+)>`

Contains all of the permissions that apply to a particular server application.

`<!ELEMENT allow-unlisted-interfaces (#PCDATA)>`

Specifies the default access permissions that apply to interfaces (WSDL port types) not explicitly listed in the action-role mapping file. The element contents can have the following values:

- ◆ `true`—for any interfaces not listed, access to all of the interfaces' actions is allowed for all roles. If the remote user is unauthenticated (in the sense that no credentials are sent by the client), access is also allowed.

Note: However, if `<allow-unlisted-interfaces>` is `true` and a particular interface is listed, then only the actions explicitly listed within that interface's `<interface>` element are accessible. Unlisted actions from the listed interface are not accessible.

- ◆ `false`—for any interfaces not listed, access to all of the interfaces' actions is denied for all roles. Unauthenticated users are also denied access.

Default is `false`.

`<!ELEMENT interface (name, action-role+)>`

In the case of an Artix server, the `<interface>` element contains all of the access permissions for one particular WSDL port type.

<!ELEMENT name (#PCDATA)>

Within the scope of an <interface> element, identifies the interface (WSDL port type) with which permissions are being associated. In Artix, the interface name is a WSDL port type name specified in the format *NamespaceURI:PortTypeName*. That is, the *PortTypeName* comes from a tag, <portType name="*PortTypeName*">, defined in the *NamespaceURI* namespace. The *NamespaceURI* is usually defined in the <definitions targetNamespace="*NamespaceURI*" ...> tag of the WSDL contract.

<!ELEMENT role-name (#PCDATA)>

Specifies a role to which permission is granted. The role name can be any role that belongs to the server's Artix authorization realm (for SOAP bindings, the realm name is specified by the `plugins:asp:authorization_realm` configuration variable in the server's configuration scope) or to the `IONAGlobalRealm` realm. The roles themselves are defined in the iS2 server backend; for example, in a file adapter file or in an LDAP backend.

<!ELEMENT secure-system (allow-unlisted-interfaces*,
action-role-mapping+)>

The outermost scope of an action-role mapping file groups together a collection of <action-role-mapping> elements.

<!ELEMENT server-name (#PCDATA)>

The <server-name> element specifies the configuration scope (that is, the ORB name) used by the server in question. This is normally the value of the `-ORBname` parameter passed to the server executable on the command line.

OpenSSL Utilities

The `openssl` program consists of a large number of utilities that have been combined into one program. This appendix describes how you use the `openssl` program with Orbix when managing X.509 certificates and private keys.

In this appendix

This appendix contains the following sections:

Using OpenSSL Utilities	page 294
The OpenSSL Configuration File	page 303

Using OpenSSL Utilities

The OpenSSL package

Orbix ships a version of the OpenSSL program that is available with Eric Young's openssl package. OpenSSL is a publicly available implementation of the SSL protocol. Consult [“License Issues” on page 537](#) for information about the copyright terms of OpenSSL.

Note: For complete documentation of the OpenSSL utilities, consult the documentation at the OpenSSL web site <http://www.openssl.org/docs>.

Command syntax

An `openssl` command line takes the following form:

```
openssl utility arguments
```

For example:

```
openssl x509 -in OrbixCA -text
```

The `openssl` utilities

This appendix describes four `openssl` utilities:

- `x509` Manipulates X.509 certificates.
 - `req` Creates and manipulates certificate signing requests, and self-signed certificates.
 - `rsa` Manipulates RSA private keys.
 - `ca` Implements a Certification Authority (CA).
-

The `-help` option

To get a list of the arguments associated with a particular command, use the `-help` option as follows:

```
openssl utility -help
```

For example:

```
openssl x509 -help
```

Examples

A number of examples using `openssl` commands are described in [“Managing Certificates” on page 141](#).

The x509 Utility

Purpose of the x509 utility

In Orbix the x509 utility is mainly used for:

- Printing text details of certificates you wish to examine.
 - Converting certificates to different formats.
-

Options

The options supported by the openssl x509 utility are as follows:

```
-inform arg      - input format - default PEM
                  (one of DER, NET or PEM)

-outform arg     - output format - default PEM
                  (one of DER, NET or PEM)

-keyform arg     - private key format - default PEM

-CAform arg      - CA format - default PEM

-CAkeyform arg   - CA key format - default PEM

-in arg          - input file - default stdin

-out arg         - output file - default stdout

-serial         - print serial number value

-hash           - print serial number value

-subject        - print subject DN

-issuer         - print issuer DN

-startdate      - notBefore field

-enddate        - notAfter field

-dates          - both Before and After dates

-modulus        - print the RSA key modulus

-fingerprint    - print the certificate fingerprint

-noout          - no certificate output

-days arg       - How long till expiry of a signed certificate
                  - def 30 days

-signkey arg    - self sign cert with arg

-x509toreq      - output a certification request object
```

```

-req           - input is a certificate request, sign and
                output
-CA arg       - set the CA certificate, must be PEM format
-CAkey arg    - set the CA key, must be PEM format. If missing
                it is assumed to be in the CA file
-CAcreateserial - create serial number file if it does not exist
-CAserial     - serial file
-text        - print the certificate in text form
-C           - print out C code forms
-md2/-md5/-sha1/ - digest to do an RSA sign with
-mdc2

```

Using the x509 utility

To print the text details of an existing PEM-format X.509 certificate, use the x509 utility as follows:

```
openssl x509 -in MyCert.pem -inform PEM -text
```

To print the text details of an existing DER-format X.509 certificate, use the x509 utility as follows:

```
openssl x509 -in MyCert.der -inform DER -text
```

To change a certificate from PEM format to DER format, use the x509 utility as follows:

```
openssl x509 -in MyCert.pem -inform PEM -outform DER -out
    MyCert.der
```

The req Utility

Purpose of the `x509` utility

The `req` utility is used to generate a self-signed certificate or a certificate signing request (CSR). A CSR contains details of a certificate to be issued by a CA. When creating a CSR, the `req` command prompts you for the necessary information from which a certificate request file and an encrypted private key file are produced. The certificate request is then submitted to a CA for signing.

If the `-nodes` (no DES) parameter is not supplied to `req`, you are prompted for a pass phrase which will be used to protect the private key.

Note: It is important to specify a validity period (using the `-days` parameter). If the certificate expires, applications that are using that certificate will not be authenticated successfully.

Options

The options supported by the `openssl req` utility are as follows:

<code>-inform arg</code>	input format - one of DER TXT PEM
<code>-outform</code>	arg output format - one of DER TXT PEM
<code>-in arg</code>	input file
<code>-out arg</code>	output file
<code>-text</code>	text form of request
<code>-noout</code>	do not output REQ
<code>-verify</code>	verify signature on REQ
<code>-modulus</code>	RSA modulus
<code>-nodes</code>	do not encrypt the output key
<code>-key file</code>	use the private key contained in file
<code>-keyform arg</code>	key file format
<code>-keyout arg</code>	file to send the key to
<code>-newkey rsa:bits</code>	generate a new RSA key of 'bits' in size
<code>-newkey dsa:file</code>	generate a new DSA key, parameters taken from CA in 'file'
<code>-[digest]</code>	Digest to sign with (md5, sha1, md2, mdc2)

<code>-config file</code>	request template file
<code>-new</code>	new request
<code>-x509</code>	output an x509 structure instead of a certificate req. (Used for creating self signed certificates)
<code>-days</code>	number of days an x509 generated by <code>-x509</code> is valid for
<code>-asn1-kludge</code>	Output the 'request' in a format that is wrong but some CA's have been reported as requiring [It is now always turned on but can be turned off with <code>-no-asn1-kludge</code>]

Using the req Utility

To create a self-signed certificate with an expiry date a year from now, the `req` utility can be used as follows to create the certificate `CA_cert.pem` and the corresponding encrypted private key file `CA_pk.pem`:

```
openssl req -config ssl_conf_path_name -days 365
-out CA_cert.pem -new -x509 -keyout CA_pk.pem
```

This following command creates the certificate request `MyReq.pem` and the corresponding encrypted private key file `MyEncryptedKey.pem`:

```
openssl req -config ssl_conf_path_name -days 365
-out MyReq.pem -new -keyout MyEncryptedKey.pem
```

The rsa Utility

Purpose of the `rsa` utility

The `rsa` command is a useful utility for examining and modifying RSA private key files. Generally RSA keys are stored encrypted with a symmetric algorithm using a user-supplied pass phrase. The OpenSSL `req` command prompts the user for a pass phrase in order to encrypt the private key. By default, `req` uses the triple DES algorithm. The `rsa` command can be used to change the password that protects the private key and to convert the format of the private key. Any `rsa` command that involves reading an encrypted `rsa` private key will prompt for the PEM pass phrase used to encrypt it.

Options

The options supported by the `openssl rsa` utility are as follows:

<code>-inform arg</code>	input format - one of DER NET PEM
<code>-outform arg</code>	output format - one of DER NET PEM
<code>-in arg</code>	input file
<code>-out arg</code>	output file
<code>-des</code>	encrypt PEM output with cbc des
<code>-des3</code>	encrypt PEM output with ede cbc des using 168 bit key
<code>-text</code>	print the key in text
<code>-noout</code>	do not print key out
<code>-modulus</code>	print the RSA key modulus

Using the `rsa` Utility

Converting a private key to PEM format from DER format involves using the `rsa` utility as follows:

```
openssl rsa -inform DER -in MyKey.der -outform PEM -out MyKey.pem
```

Changing the pass phrase which is used to encrypt the private key involves using the `rsa` utility as follows:

```
openssl rsa -inform PEM -in MyKey.pem -outform PEM -out MyKey.pem
-des3
```

Removing encryption from the private key (which is not recommended) involves using the `rsa` command utility as follows:

```
openssl rsa -inform PEM -in MyKey.pem -outform PEM -out MyKey2.pem
```

Note: Do not specify the same file for the `-in` and `-out` parameters, because this can corrupt the file.

The ca Utility

Purpose of the `ca` utility

You can use the `ca` utility create X.509 certificates by signing existing signing requests. It is imperative that you check the details of a certificate request before signing. Your organization should have a policy with respect to the issuing of certificates. Before implementing CAs, refer to [“Managing Certificates” on page 141](#) for more information.

The `ca` utility is used to sign certificate requests thereby creating a valid X.509 certificate which can be returned to the request submitter. It can also be used to generate Certificate Revocation Lists (CRLS). For information on the `ca -policy` and `-name` options, refer to [“The OpenSSL Configuration File” on page 303](#).

Creating a new CA

To create a new CA using the `openssl ca` utility, two files (`serial` and `index.txt`) need to be created in the location specified by the `openssl` configuration file that you are using. See also [“Set Up Your Own CA” on page 152](#).

Options

The options supported by the `openssl ca` utility are as follows:

<code>-verbose</code>	- Talk alot while doing things
<code>-config file</code>	- A config file
<code>-name arg</code>	- The particular CA definition to use
<code>-gencrl</code>	- Generate a new CRL
<code>-crl days</code>	- Days is when the next CRL is due
<code>-crl hours</code>	- Hours is when the next CRL is due
<code>-days arg</code>	- number of days to certify the certificate for
<code>-md arg</code>	- md to use, one of md2, md5, sha or sha1
<code>-policy arg</code>	- The CA 'policy' to support
<code>-keyfile arg</code>	- PEM private key file
<code>-key arg</code>	- key to decode the private key if it is encrypted
<code>-cert</code>	- The CA certificate
<code>-in file</code>	- The input PEM encoded certificate request(s)

```

-out file           - Where to put the output file(s)
-outdir dir        - Where to put output certificates
-infiles...       - The last argument, requests to process
-spkac file       - File contains DN and signed public key and
                  challenge
-preserveDN       - Do not re-order the DN
-batch            - Do not ask questions
-msie_hack        - msie modifications to handle all thos
                  universal strings

```

Note: Most of the above parameters have default values as defined in `openssl.cnf`.

Using the `ca` Utility

Converting a private key to PEM format from DER format involves using the `ca` utility as shown in the following example. To sign the supplied CSR `MyReq.pem` to be valid for 365 days and create a new X.509 certificate in PEM format, use the `ca` utility as follows:

```

openssl ca -config ssl_conf_path_name -days 365
-in MyReq.pem -out MyNewCert.pem

```

The OpenSSL Configuration File

Overview

A number of OpenSSL commands (for example, `req` and `ca`) take a `-config` parameter that specifies the location of the openssl configuration file. This section provides a brief description of the format of the configuration file and how it applies to the `req` and `ca` commands. An example configuration file is listed at the end of this section.

Structure of `openssl.cnf`

The `openssl.cnf` configuration file consists of a number of sections that specify a series of default values that are used by the openssl commands.

In this section

This section contains the following subsections:

[req] Variables	page 304
[ca] Variables	page 305
[policy] Variables	page 306
Example openssl.cnf File	page 307

[req] Variables

Overview of the variables

The `req` section contains the following variables:

```
default_bits = 1024
default_keyfile = privkey.pem
distinguished_name = req_distinguished_name
attributes = req_attributes
```

`default_bits` configuration variable

The `default_bits` variable is the default RSA key size that you wish to use. Other possible values are 512, 2048, and 4096.

`default_keyfile` configuration variable

The `default_keyfile` variable is the default name for the private key file created by `req`.

`distinguished_name` configuration variable

The `distinguished_name` variable specifies the section in the configuration file that defines the default values for components of the distinguished name field. The `req_attributes` variable specifies the section in the configuration file that defines defaults for certificate request attributes.

[ca] Variables

Choosing the CA section

You can configure the file `openssl.cnf` to support a number of CAs that have different policies for signing CSRs. The `-name` parameter to the `ca` command specifies which CA section to use. For example:

```
openssl ca -name MyCa ...
```

This command refers to the CA section `[MyCa]`. If `-name` is not supplied to the `ca` command, the CA section used is the one indicated by the `default_ca` variable. In the [“Example openssl.cnf File” on page 307](#), this is set to `CA_default` (which is the name of another section listing the defaults for a number of settings associated with the `ca` command). Multiple different CAs can be supported in the configuration file, but there can be only one default CA.

Overview of the variables

Possible `[ca]` variables include the following

`dir`: The location for the CA database

The database is a simple text database containing the following tab separated fields:

`status`: A value of 'R' - revoked, 'E' -expired or 'V' valid

`issued date`: When the certificate was certified

`revoked date`: When it was revoked, blank if not revoked

`serial number`: The certificate serial number

`certificate`: Where the certificate is located

`CN`: The name of the certificate

The `serial number` field should be unique, as should the `CN/status` combination. The `ca` utility checks these at startup.

`certs`: This is where all the previously issued certificates are kept

[policy] Variables

Choosing the policy section

The policy variable specifies the default policy section to be used if the `-policy` argument is not supplied to the `ca` command. The CA policy section of a configuration file identifies the requirements for the contents of a certificate request which must be met before it is signed by the CA.

There are two policy sections defined in the [“Example openssl.cnf File” on page 307](#): `policy_match` and `policy_anything`.

Example policy section

The `policy_match` section of the example `openssl.cnf` file specifies the order of the attributes in the generated certificate as follows:

```
countryName
stateOrProvinceName
organizationName
organizationalUnitName
commonName
emailAddress
```

The `match` policy value

Consider the following value:

```
countryName = match
```

This means that the country name must match the CA certificate.

The `optional` policy value

Consider the following value:

```
organisationalUnitName = optional
```

This means that the `organisationalUnitName` does not have to be present.

The `supplied` policy value

Consider the following value:

```
commonName = supplied
```

This means that the `commonName` must be supplied in the certificate request.

Example openssl.cnf File

Listing

The following listing shows the contents of an example `openssl.cnf` configuration file:

```
#####
# openssl example configuration file.
# This is mostly used for generation of certificate requests.
#####
[ ca ]
default_ca= CA_default          # The default ca section
#####

[ CA_default ]

dir=/opt/iona/OrbixSSL1.0c/certs # Where everything is kept

certs=$dir # Where the issued certs are kept
crl_dir= $dir/crl # Where the issued crl are kept
database= $dir/index.txt # database index file
new_certs_dir= $dir/new_certs # default place for new certs
certificate=$dir/CA/OrbixCA # The CA certificate
serial= $dir/serial # The current serial number
crl= $dir/crl.pem # The current CRL
private_key= $dir/CA/OrbixCA.pk # The private key
RANDFILE= $dir/.rand # private random number file
default_days= 365 # how long to certify for
default_crl_days= 30 # how long before next CRL
default_md= md5 # which message digest to use
preserve= no # keep passed DN ordering

# A few different ways of specifying how closely the request
# should conform to the details of the CA

policy= policy_match

# For the CA policy

[policy_match]
countryName= match
stateOrProvinceName= match
organizationName= match
organizationalUnitName= optional
commonName= supplied
```

```

emailAddress= optional

# For the 'anything' policy
# At this point in time, you must list all acceptable 'object'
# types

[ policy_anything ]
countryName = optional
stateOrProvinceName= optional
localityName= optional
organizationName = optional
organizationalUnitName = optional
commonName= supplied
emailAddress= optional

[ req ]
default_bits = 1024
default_keyfile= privkey.pem
distinguished_name = req_distinguished_name
attributes = req_attributes

[ req_distinguished_name ]
countryName= Country Name (2 letter code)
countryName_min= 2
countryName_max = 2
stateOrProvinceName= State or Province Name (full name)
localityName = Locality Name (eg, city)
organizationName = Organization Name (eg, company)
organizationalUnitName = Organizational Unit Name (eg, section)
commonName = Common Name (eg. YOUR name)
commonName_max = 64
emailAddress = Email Address
emailAddress_max = 40

[ req_attributes ]
challengePassword = A challenge password
challengePassword_min = 4
challengePassword_max = 20
unstructuredName= An optional company name

```

License Issues

This appendix contains the text of licenses relevant to Artix.

In this appendix

This appendix contains the following section:

OpenSSL License	page 310
---------------------------------	--------------------------

OpenSSL License

Overview

The licence agreement for the usage of the OpenSSL command line utility shipped with Artix SSL/TLS is as follows:

LICENSE ISSUES

=====

The OpenSSL toolkit stays under a dual license, i.e. both the conditions of the OpenSSL License and the original SSLeay license apply to the toolkit. See below for the actual license texts. Actually both licenses are BSD-style Open Source licenses. In case of any license issues related to OpenSSL please contact openssl-core@openssl.org.

OpenSSL License

```
/* =====
 * Copyright (c) 1998-1999 The OpenSSL Project. All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 * notice, this list of conditions and the following disclaimer in
 * the documentation and/or other materials provided with the
 * distribution.
 *
 * 3. All advertising materials mentioning features or use of this
 * software must display the following acknowledgment:
 * "This product includes software developed by the OpenSSL Project
 * for use in the OpenSSL Toolkit. (http://www.openssl.org/)"
 *
 * 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
 * endorse or promote products derived from this software without
 * prior written permission. For written permission, please contact
 * openssl-core@openssl.org.
 *
 * 5. Products derived from this software may not be called "OpenSSL"
 * nor may "OpenSSL" appear in their names without prior written
 * permission of the OpenSSL Project.
```

```

*
* 6. Redistributions of any form whatsoever must retain the following
* acknowledgment:
* "This product includes software developed by the OpenSSL Project
* for use in the OpenSSL Toolkit (http://www.openssl.org/)"
*
* THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY
* EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
* PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OpenSSL PROJECT OR
* ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
* NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
* STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
* OF THE POSSIBILITY OF SUCH DAMAGE.
* =====
*
* This product includes cryptographic software written by Eric Young
* (eay@cryptsoft.com). This product includes software written by Tim
* Hudson (tjh@cryptsoft.com).
*
*/

```

Original SSLeay License

```

-----
/* Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)
* All rights reserved.
*
* This package is an SSL implementation written
* by Eric Young (eay@cryptsoft.com).
* The implementation was written so as to conform with Netscapes SSL.
*
* This library is free for commercial and non-commercial use as long as
* the following conditions are aheared to. The following conditions
* apply to all code found in this distribution, be it the RC4, RSA,
* lhash, DES, etc., code; not just the SSL code. The SSL documentation
* included with this distribution is covered by the same copyright terms
* except that the holder is Tim Hudson (tjh@cryptsoft.com).
*
* Copyright remains Eric Young's, and as such any Copyright notices in
* the code are not to be removed.
* If this package is used in a product, Eric Young should be given attribution
* as the author of the parts of the library used.

```

```

* This can be in the form of a textual message at program startup or
* in documentation (online or textual) provided with the package.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
* 1. Redistributions of source code must retain the copyright
*   notice, this list of conditions and the following disclaimer.
* 2. Redistributions in binary form must reproduce the above copyright
*   notice, this list of conditions and the following disclaimer in the
*   documentation and/or other materials provided with the distribution.
* 3. All advertising materials mentioning features or use of this software
*   must display the following acknowledgement:
*   "This product includes cryptographic software written by
*    Eric Young (eay@cryptsoft.com)"
*   The word 'cryptographic' can be left out if the rouines from the library
*   being used are not cryptographic related :-).
* 4. If you include any Windows specific code (or a derivative thereof) from
*   the apps directory (application code) you must include an acknowledgement:
*   "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"
*
* THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ``AS IS'' AND
* ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
* FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
* DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
* LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
* OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
* SUCH DAMAGE.
*
* The licence and distribution terms for any publically available version or
* derivative of this code cannot be changed.  i.e. this code cannot simply be
* copied and put under another distribution licence
* [including the GNU Public Licence.]
*/

```

Index

Symbols

.NET

and principal propagation 223

<action-role-mapping> tag 137

<interface> tag 137

<name> tag 137

<realm> tag 129

<role> tag 129

<server-name> tag 137

<users> tag 129

A

access control

wsdltoacl utility 138

ACL

<action-role-mapping> tag 137

<interface> tag 137

<name> tag 137

<server-name> tag 137

action_role_mapping configuration variable 135

action-role mapping file 135

action-role mapping file, example 136

action-role mapping

and role-based access control 123

action_role_mapping configuration variable 70, 135

action-role mapping file

<action-role-mapping> tag 137

<interface> tag 137

<name> tag 137

<server-name> tag 137

CORBA

configuring 135

example 136

administration

OpenSSL command-line utilities 151

AgentSecret property 109

Artix security layer

and certificate-based authentication 48

Artix security plug-in

and security layer 31

authentication_cache_size configuration

variable 34

Artix security plug-in plug-in

authentication_cache_timeout configuration
variable 34

ASN.1 143

ASP plug-in

caching of credentials 33

asp plug-in

default_password configuration value 218

security_type configuration variable 218

ASP security layer

and HTTP 37

and SOAP binding 55

association options

and cipher suite constraints 210

and mechanism policy 200

client secure invocation policy, default 196

compatibility with cipher suites 211

EstablishTrustInClient 62, 76

NoProtection 65

rules of thumb 200

SSL/TLS

Confidentiality 194

DetectMisordering 194

DetectReplay 194

EstablishTrustInClient 195

EstablishTrustInTarget 195

Integrity 194

NoProtection 194

setting 192

target secure invocation policy, default 198

authentication

and security layer 31

caching of credentials 33

certificate-based 28

CSI 28

HTTP Basic Authentication 28

iSF

process of 67

own certificate, specifying 179

pass phrase

dialog prompt, C++ 183

in configuration 184

password file, from 183

SSL/TLS

- mutual 175
 - target only 171
 - trusted CA list 178
 - authentication_cache_size configuration variable 33, 34
 - authentication_cache_timeout configuration variable 33, 34
 - authorization
 - and security layer 31
 - caching of credentials 33
 - role-based access control 123
 - roles
 - creating 125
 - special 127
 - authorization realm
 - adding a server 124
 - IONAGlobalRealm realm 127
 - iSF 123
 - iSF, setting in server 70
 - roles in 125
 - servers in 124
 - special 127
 - authorization realms
 - creating 125
- B**
- backward trust 77
 - Baltimore toolkit
 - selecting for C++ applications 240
 - shlib name configuration variable 243
 - bus:security 235
 - bus-security:security interceptor 47, 92
- C**
- CA 142
 - choosing a host 146
 - commercial CAs 145
 - index file 153
 - list of trusted 148
 - multiple CAs 148
 - private CAs 146
 - private key, creating 154
 - security precautions 146
 - See *A/socertificate authority*
 - self-signed 154
 - serial file 153
 - trusted list 160, 178
 - 305
 - CA, setting up 152
 - CACHE_CLIENT session caching value 213
 - CACHE_NONE session caching value 213
 - CACHE_SERVER_AND_CLIENT session caching value 213
 - CACHE_SERVER session caching value 213
 - caching
 - authentication_cache_size configuration variable 33, 34
 - authentication_cache_timeout configuration variable 33, 34
 - CACHE_CLIENT session caching value 213
 - CACHE_NONE session caching value 213
 - CACHE_SERVER_AND_CLIENT session caching value 213
 - CACHE_SERVER session caching value 213
 - of credentials 33
 - SSL/TLS
 - cache size 213
 - validity period 213
 - Caching sessions 213
 - CAs 152
 - ca utility 301
 - certificate authority
 - and certificate signing 142
 - certificate-based authentication 28
 - and HTTP 38
 - example scenario 48, 78
 - file adapter, configuring 129
 - LDAP adapter, configuring 131
 - certificate constraints policy
 - three-tier target server 77
 - certificate_constraints_policy variable 187
 - Certificates
 - chain length 186
 - constraints 187
 - certificates
 - chaining 147
 - constraint language 187
 - constraints policy 77
 - contents of 143
 - creating and signing 155
 - deployment, 159
 - importing and exporting 150
 - length limit 148
 - own, specifying 179
 - pass phrase 183
 - peer 147
 - PKCS#12 file 149

- public key 143
 - public key encryption 205
 - security handshake 171, 176
 - self-signed 147, 154
 - serial number 143
 - signing 142, 156
 - signing request 155
 - trusted CA list 160, 178
 - X.509 142
 - certificate signing request 155
 - common name 156
 - signing 156
 - chaining of certificates 147
 - ciper suites
 - order of 209
 - cipher suites
 - ciphersuites configuration variable 208
 - compatibility algorithm 211
 - compatibility with association options 211
 - default list 209
 - definitions 206
 - effective 210
 - encryption algorithm 205
 - exportable 206
 - integrity-only ciphers 205
 - key exchange algorithm 205
 - mechanism policy 208
 - secure hash algorithm 205
 - secure hash algorithms 206
 - security algorithms 205
 - specifying 204
 - standard ciphers 205
 - ciphersuites configuration variable 208
 - client_binding_list configuration variable
 - iSF, client configuration 68
 - secure client 61
 - ClientCertificate attribute 42
 - ClientPrivateKey attribute 42
 - ClientPrivateKeyPassword attribute 42
 - client secure invocation policy 210
 - IIOP/TLS 196
 - ClientSecureInvocationPolicy policy 193
 - colocated invocations
 - and secure associations 190
 - colocation
 - incompatibility with principal propagation 216
 - common names
 - uniqueness 156
 - Confidentiality association option 194
 - hints 202
 - Confidentiality option 194
 - Configuration file 303
 - constraint language 187
 - Constraints
 - for certificates 187
 - CORBA
 - action-role mapping file 135
 - action-role mapping file, example 136
 - configuring principal propagation 217
 - intermediate server configuration 73
 - iSF, three-tier system 72
 - principal propagation 216
 - security, overview 58
 - SSL/TLS
 - client configuration 60
 - securing communications 60
 - server configuration 62
 - three-tier target server configuration 75
 - CORBA binding
 - CSI authorization over transport 28
 - CSI identity assertion 28
 - protocol layers 30
 - SSO overview 86
 - CORBA Principal 27, 56
 - CORBA security
 - CSlv2 plug-in 59
 - GSP plug-in 59
 - IIOP/TLS plug-in 59
 - CSI
 - authorization over transport 28
 - identity assertion 28
 - CSI interceptor 68
 - CSlv2
 - certificate constraints policy 77
 - principal sponsor
 - client configuration 69
 - CSlv2 plug-in
 - CORBA security 59
 - CSR 155
- ## D
- data encryption standard
 - see DES
 - default_password configuration value 218
 - DES
 - symmetric encryption 206
 - DetectMisordering association option 194
 - hints 202

DetectMisordering option 194
 DetectReplay association option 194
 hints 202
 DetectReplay option 194
 domain name
 ignored by iSF 67

E

effective cipher suites
 definition 210
 enable_principal_service_context configuration
 variable 217
 encryption algorithm
 RC4 206
 encryption algorithms 205
 DES 206
 symmetric 206
 triple DES 206
 enterprise security service
 and iSF security domains 121
 EstablishTrustInClient association option 62, 195
 hints 201
 three-tier target server 76
 EstablishTrustInClient option 195
 EstablishTrustInTarget association option 195
 hints 201
 EstablishTrustInTarget option 195
 exportable cipher suites 206

F

file adapter 100
 configuring certificate-based authentication 129
 properties 100
 file domain
 <realm> tag 129
 <users> tag 129
 example 128
 file location 128
 managing 128

G

GroupBaseDN property 104
 GroupNameAttr property 104
 GroupObjectClass property 104
 GroupSearchScope property 105
 GSP plug-in
 and security layer 31
 and the login service 86

authentication_cache_size configuration
 variable 33
 authentication_cache_timeout configuration
 variable 33
 caching of credentials 33
 CORBA security 59

H

HTTP
 ASP security layer 37
 security layers 36
 HTTP Basic Authentication 28, 38
 overview 45
 HTTP-compatible binding
 compatible bindings 37
 overview 36
 protocol layers 29
 HTTPS
 ciphersuites configuration variable 208
 mutual authentication 41
 target-only authentication 39
 HTTPS security
 overview 39

I

identity assertion 28
 IIOP/TLS
 ciphersuites configuration variable 208
 IIOP/TLS plug-in
 CORBA security 59
 IIOP plug-in
 and semi-secure clients 61
 IIOP_TLS interceptor 61
 index file 153
 Integrity association option 194
 hints 202
 integrity-only ciphers 205
 Integrity option 194
 interceptors
 artix security 47
 bus-security 92
 login_client 91
 interoperability
 explicit principal header 224
 OS/390, SSL/TLS 208
 with .NET 223
 with Orbix applications 216
 invocation policies

- interaction with mechanism policy 200
- IONAGlobalRealm realm 127
- IONAUserRole 138
- is2.properties file 100
- iS2 adapters
 - enterprise security service 121
 - file domain
 - managing 128
 - LDAP domain
 - managing 131
 - SiteMinder domain
 - managing 132
- iS2 server
 - configuring 99
 - file adapter 100
 - is2.properties file 100
 - LDAP adapter 102
 - LDAP adapter, properties 103
 - log4j logging 116
 - security information file 100
 - SiteMinder adapter, configuring 108
- iSF
 - action_role_mapping configuration variable 70
 - and certificate-based authentication 78
 - authorization realm
 - setting in server 70
 - client configuration
 - CSI interceptor 68
 - CORBA
 - three-tier system 72
 - three-tier target server configuration 75
 - two-tier scenario description 67
 - CORBA security 58
 - domain name, ignoring 67
 - intermediate server configuration 73
 - security domain
 - creating 122
 - server configuration
 - server_binding_list 68
 - server_domain_name configuration variable 70
 - three-tier scenario description 73
 - user account
 - creating 122

J

- JCE architecture
 - enabling 243

K

- kdc property 111
- Kerberos 110
 - token 27
- Kerberos adapter
 - Kerberos KDC server 111
 - properties 110
- Kerberos property
 - RetrieveAuthInfo 112
- Kerberos Realm Name property 111
- Kerberos token
 - setting 233
- key exchange algorithms 205

L

- LDAP adapter 102
 - basic properties 105
 - configuring certificate-based authentication 131
 - GroupBaseDN property 104
 - GroupNameAttr property 104
 - GroupObjectClass property 104, 105
 - LDAP server replicas 106
 - MemberDNAttr property 105
 - PrincipalUserDN property 107
 - PrincipalUserPassword property 107
 - properties 103
 - replica index 106
 - RoleNameAttr property 104
 - SSLCACertDir property 107
 - SSLClientCertFile property 107
 - SSLClientCertPassword property 107
 - SSLEnabled property 107
 - UserBaseDN property 104
 - UserNameAttr property 104
 - UserObjectClass property 104
 - UserRoleDNAttr property 104
- LDAP domain
 - managing 131
- Lightweight Directory Access Protocol
 - see LDAP
- log4j 116
 - documentation 116
 - properties file 116
- logging
 - in secure client 62
 - log4j 116
- login_client:login_client interceptor 91
- login_client plug-in 91

- and the login service 86
- login service
 - and single sign-on 86
 - standalone deployment 87
 - WSDL contract for 95
- login_service plug-in
 - configuring 93

M

- max_chain_length_policy configuration variable 186
- MD5 194, 206
- mechanism policy
 - interaction with invocation policies 200
- MechanismPolicy 194
- mechanism policy 208
- MemberDNAttr property 105
- message attributes
 - and routing 218
- message digest 5
 - see MD5
- message digests 194
- message fragments 194
- MESSAGE_LEVEL security level 93
- mixed configurations, SSL/TLS 65
- multiple CAs 148
- mutual authentication 175
 - HTTPS 41

N

- NoProtection association option
 - rules of thumb 200
- NoProtection association option 65, 194
 - hints 202
 - semi-secure applications 203
- NoProtection option 194
- NoSuchAttributeException exception 222

O

- OpenSSL 146, 293
- openssl
 - configuration file 303
 - utilities 294
- openssl.cnf example file 307
- OpenSSL command-line utilities 151
- ORBname argument 138
- orb_plugins configuration variable 61
 - client configuration 68
- orb_plugins variable

- and the NoProtection association option 202
- semi-secure configuration 203
- OS/390
 - interoperability with 208

P

- pass phrase 183
 - dialog prompt, C++ 183
 - in configuration 184
 - password file, from 183
- Password attribute 46
- peer certificate 147
- performance
 - caching of credentials 33
- PKCS#12 files
 - creating 150, 155
 - definition 149
 - importing and exporting 150
 - pass phrase 183
 - viewing 150
- plug-ins
 - CSlv2, in CORBA security 59
 - GSP, in CORBA security 59
 - IIOp 61
 - IIOp/TLS, in CORBA security 59
- plugins:asp:default_password configuration variable 50
- plugins:asp:security_level 243
- plugins:asp:security_level configuration variable 47
- plugins:asp:security_type configuration variable 47
- plugins:login_client:wSDL_url configuration variable 91
- plugins:login_service:wSDL_url configuration variable 93
- POA_Coloc interceptor 216
- policies
 - client secure invocation 210
 - ClientSecureInvocationPolicy 193
 - IIOp/TLS
 - client secure invocation 196
 - target secure invocation 198
 - target secure invocation 210
 - TargetSecureInvocationPolicy 193
- policies:asp:enable_authorization configuration variable 47
- policies:asp:enable_sso configuration variable 92 306
- Principal 27
- principals

- and colocation 216
- configuring propagation 217
- explicit principal header 224
- from O/S username 217
- interoperability 216
- interoperating with .NET 223
- NoSuchAttributeException exception 222
- overview 216
- reading on the server side 221
- routing configuration 218
- setting on the client side 220
- principal sponsor
 - CSiv2
 - client configuration 69
 - SSL/TLS
 - enabling 64
 - SSL/TLS, disabling 62
- principal_sponsor
 - auth_method_data 261, 265
 - callback_handler (Java only) 263
 - login_attempts (Java only) 263
 - use_principal_sponsor 261, 264
- principal_sponsor Namespace Variables 261, 264
- PrincipalUserDN property 107
- PrincipalUserPassword property 107
- private key 154
- propagateInputAttribute WSDL tag 218
- protocol version
 - interoperability with OS/390 208
- protocol_version configuration variable 208
- public key encryption 205
- public keys 143

R

- RC4 encryption 206
- realm
 - see authorization realm
- realm property 111
- Replay detection 194
 - 304
- REQUEST_LEVEL security level 92
- req utility 297
- req Utility command 297
- Rivest Shamir Adleman
 - see RSA
- role-based access control 123
 - example 126
- RoleNameAttr property 104
- role-properties file 139

- roles
 - creating 125
 - special 127
- root certificate directory 148
- routing
 - and principal propagation 218
- RSA 205
 - symmetric encryption algorithm 206
- RSA_EXPORT_WITH_DES40_CBC_SHA cipher suite 205, 211
- RSA_EXPORT_WITH_RC4_40_MD5 cipher suite 205, 211
- rsa utility 299
- rsa Utility command 299
- RSA_WITH_3DES_EDE_CBC_SHA cipher suite 205, 211
- RSA_WITH_DES_CBC_SHA cipher suite 205, 211
- RSA_WITH_NULL_MD5 cipher suite 205, 211
- RSA_WITH_NULL_SHA cipher suite 205, 211
- RSA_WITH_RC4_128_MD5 cipher suite 205, 211
- RSA_WITH_RC4_128_SHA cipher suite 205, 211

S

- Schannel toolkit
 - selecting for C++ applications 240
- secure associations
 - client behavior 196
 - definition 190
 - TLS_Coloc interceptor 190
- secure hash algorithms 205, 206
- security algorithms
 - and cipher suites 205
- security domain
 - creating 122
- security domains
 - architecture 121
 - iSF 122
- security handshake
 - cipher suites 204
 - SSL/TLS 171, 176
- security information file 100
- security layer
 - overview 31
- security levels
 - MESSAGE_LEVEL 93
 - REQUEST_LEVEL 92
- security_type configuration variable 218
- security types
 - USERNAME_PASSWORD 93

- self-signed CA 154
- self-signed certificate 147
- semi-secure applications
 - and NoProtection 203
- send_principal configuration variable 217
- serial file 153
- serial number 143
- ServerAddress property 109
- ServerAuthnPort property 109
- server_binding_list configuration variable 68
- ServerCertificate attribute 44
- server_domain_name configuration variable
 - iSF, ignored by 70
- ServerPrivateKey attribute 44
- ServerPrivateKeyPassword attribute 44
- session_cache_size configuration variable 213
- session_cache_validity_period configuration variable 213
- session_caching_policy configuraion variable 213
- session_caching_policy variable 213
- session_idle timeout
 - SSO 88
- session timeout
 - SSO 88
- SHA 206
- SHA1 194
- signing certificates 142
- Single sign-on
 - and security layer 31
- single sign-on
 - SSO token 28
 - token timeouts 88
- SiteMinder adapter
 - AgentSecret property 109
 - configuring 108
 - properties 108
 - ServerAddress property 109
 - ServerAuthnPort property 109
- SiteMinder domain
 - managing 132
- SOAP
 - principal propagation 216
- SOAP binding
 - ASP security layer 55
 - configuring principal propagation 217
 - protocol layers 30, 54
 - SOAP protocol layer 55
 - SSO overview 86
- Specifying ciphersuites 204
- SSL/TLS
 - association options
 - setting 192
 - caching validity period 213
 - cipher suites 204
 - client configuration 60
 - colocated invocations 190
 - encryption algorithm 205
 - IIOPTLS interceptor 61
 - key exchange algorithm 205
 - logging 62
 - mechanism policy 208
 - mixed configurations 65
 - orb_plugins list 61
 - principal sponsor
 - disabling 62
 - enabling 64
 - protocol_version configuration variable 208
 - secure associations 190
 - secure hash algorithm 205
 - secure hash algorithms 206
 - securing communications 60
 - security handshake 171, 176
 - selecting a toolkit, C++ 240
 - semi-secure client
 - IIOPT plug-in 61
 - server configuration 62
 - session cache size 213
 - TLS session 190
- SSLCACertDir property 107
- SSLClientCertFile property 107
- SSLClientCertPassword property 107
- SSLeay 146
- SSLEnabled property 107
- SSO
 - advantages 87
 - CORBA binding 86
 - login_client plug-in 91
 - login_service WSDL 95
 - sample configurations 96
 - session idle timeout 88
 - session timeout 88
 - SOAP binding 86
 - username/password-based authentication 90
 - _SSO_TOKEN_ 90
- SSO token 28
 - and the login service 86
 - automatic refresh 88
 - timeouts 88

standard ciphers 205
 symmetric encryption algorithms 206

T

Target
 choosing behavior 198
 target authentication 171
 target-only authentication
 HTTPS 39
 target secure invocation policy 210
 IIOP/TLS 198
 TargetSecureInvocationPolicy policy 193
 three-tier scenario description 73
 TLS
 session caching 213
 TLS_Coloc interceptor 190
 TLS security
 and HTTP 37
 TLS session
 definition 190
 toolkit replaceability
 enabling JCE architecture 243
 selecting the toolkit, C++ 240
 triple DES 206
 trusted CA list 160
 trusted CA list policy 178
 trusted CAs 148
 TrustedRootCertificates attribute 40, 44

U

use_jsse_tk configuration variable 243
 use_message_attributes() function 220, 221
 user account
 creating 122
 UserBaseDN property 104
 username/password-based authentication
 overview 89
 SSO 90
 UserName attribute 46
 UserNameAttr property 104
 USERNAME_PASSWORD security type 93
 UserObjectClass property 104
 UserRoleDNAttr property 104
 UserSearchScope property
 LDAP adapter
 UserObjectClass property 104
 UseSecureSockets attribute 40, 43
 utilities

wsdltoacl 138

V

Variables 304, 305, 306

W

Web service security extension
 opage see WSSE 27
 wsdltoacl utility 138
 role-properties file 139
 WSSE
 Kerberos token 27
 UsernameToken 27
 WSSE Kerberos credentials 56
 WSSEKerberosv5SToken property 233
 WSSEPasswordToken property 230
 WSSE UsernameToken credentials 56
 WSSEUsernameToken property 230

X

X.509
 public key encryption 205
 X.509 certificate
 definition 142
 X.509 certificates 141
 x509 utility 295

