



Security Guide

Version 1.2, October 2003

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 © 2001–2003 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: 31-Oct-2003

M 3 1 1 4

Contents

List of Tables	vii
List of Figures	ix
Preface	xi
Chapter 1 Introduction to Security	1
Security for SOAP Bindings	2
Secure Hello World Example	3
HTTPS Connection	6
IIOP/TLS Connection	11
Security Layer	18
Chapter 2 Configuring the iS2 Server	25
Configuring the File Adapter	26
Configuring the LDAP Adapter	28
Configuring the SiteMinder Adapter	34
Additional iS2 Configuration	36
Configuring the Log4J Logging	37
Chapter 3 Managing Users, Roles and Domains	39
Introduction to Domains and Realms	40
iSF Security Domains	41
iSF Authorization Realms	43
Managing a File Security Domain	48
Managing an LDAP Security Domain	50
Managing a SiteMinder Security Domain	51
Chapter 4 Managing Access Control Lists	53
Overview of Artix ACL Files	54
Artix Action-Role Mapping ACL	55

Chapter 5	Managing Certificates	59
	What are X.509 Certificates?	60
	Certification Authorities	62
	Commercial Certification Authorities	63
	Private Certification Authorities	64
	Certificate Chaining	65
	PKCS#12 Files	67
	Creating Your Own Certificates	69
	Set Up Your Own CA	70
	Use the CA to Create Signed Certificates	73
	Deploying Certificates	76
	Overview of Certificate Deployment	77
	Deploying Trusted Certificate Authority Certificates	78
	Deploying Application Certificates	82
Chapter 6	Configuring HTTPS and IIOP/TLS Authentication	85
	Requiring Authentication	86
	Target-Only Authentication	87
	Mutual Authentication	90
	Specifying Trusted CA Certificates	93
	Specifying an Application's Own Certificate	94
	Providing a Certificate Pass Phrase	95
	Certificate Pass Phrase for HTTPS	96
	Certificate Pass Phrase for IIOP/TLS	98
	Advanced IIOP/TLS Configuration Options	100
	Setting a Maximum Certificate Chain Length	101
	Applying Constraints to Certificates	102
Chapter 7	Configuring IIOP/TLS Secure Associations	105
	Overview of Secure Associations	106
	Setting IIOP/TLS Association Options	108
	Secure Invocation Policies	109
	Association Options	110
	Choosing Client Behavior	112
	Choosing Target Behavior	114
	Specifying IIOP/TLS Cipher Suites	116
	Supported Cipher Suites	117
	Setting the Mechanism Policy	120

Constraints Imposed on Cipher Suites	122
Caching IIOP/TLS Sessions	125
Chapter 8 Principal Propagation	127
Introduction to Principal Propagation	128
Configuring	129
Programming	132
Interoperating with .NET	135
Explicitly Declaring the Principal Header	136
Modifying the SOAP Header	138
Appendix A Security Configuration	141
plugins Namespace	142
policies Namespace	146
principal_sponsor Namespace	153
principal_sponsor:csi Namespace	155
Appendix B iS2 Configuration	157
Properties File Syntax	158
iS2 Properties File	159
Cluster Properties File	172
log4j Properties File	174
Appendix C ASN.1 and Distinguished Names	177
ASN.1	178
Distinguished Names	179
Appendix D Action-Role Mapping DTD	183
Appendix E OpenSSL Utilities	187
Using OpenSSL Utilities	188
The x509 Utility	189
The req Utility	191
The rsa Utility	193
The ca Utility	195
The OpenSSL Configuration File	197
[req] Variables	198

CONTENTS

[ca] Variables	199
[policy] Variables	200
Example openssl.cnf File	201
Appendix F License Issues	203
OpenSSL License	204
Index	207

List of Tables

Table 1: LDAP Properties in the com.ionas.adapter.LDAP.param Scope	32
Table 2: Cipher Suite Definitions	118
Table 3: Association Options Supported by Cipher Suites	123
Table 4: Mechanism Policy Cipher Suites	151
Table 5: Commonly Used Attribute Types	180

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: Architecture of an iSF Security Domain	41
Figure 6: Server View of iSF Authorization Realms	44
Figure 7: Role View of iSF Authorization Realms	45
Figure 8: Assignment of Realms and Roles to Users Janet and John	46
Figure 9: A Certificate Chain of Depth 2	65
Figure 10: A Certificate Chain of Depth 3	66
Figure 11: Elements in a PKCS#12 File	67
Figure 12: Target Authentication Only	87
Figure 13: Mutual Authentication	90
Figure 14: Configuration of a Secure Association	107
Figure 15: Constraining the List of Cipher Suites	122

LIST OF FIGURES

Preface

Audience

This guide is aimed at C++ developers who are developing Artix client and server applications. The C++ API described in this guide 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.

Related documentation

The document set for Artix includes the following related documents:

- *Artix Tutorial.*
- *Getting Started with Artix Encompass.*
- *Getting Started with Artix Relay.*
- *Artix User's Guide.*
- *Artix C++ Programmer's Guide.*
- *Artix Thread Library Reference.*

The latest updates to the Artix documentation can be found at [http://
iona.com/docs](http://iona.com/docs).

Additional resources

The [IONA knowledge base](http://www.iona.com/support/knowledge_base/index.xml) (http://www.iona.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.iona.com/support/updates/index.xml) (<http://www.iona.com/support/updates/index.xml>) contains the latest releases and patches for IONA products.

If you need help with this or any other IONA products, contact IONA at support@iona.com. Comments on IONA documentation can be sent to docs-support@iona.com.

Typographical conventions

This guide uses the following typographical conventions:

- | | |
|-----------------------|---|
| <i>Constant width</i> | <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> |
| <i>Italic</i> | <p>Italic words in normal text represent <i>emphasis</i> and <i>new terms</i>.</p> <p>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:</p> <pre>% cd /users/<i>your_name</i></pre> <p>Note: Some command examples may use angle brackets to represent variable values you must supply. This is an older convention that is replaced with <i>italic</i> words or characters.</p> |

Keying conventions

This guide may use 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.

PREFACE

Introduction to 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 IONA Security Framework (iSF) provides security for SOAP bindings between Artix applications. The iSF 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 IONA 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.

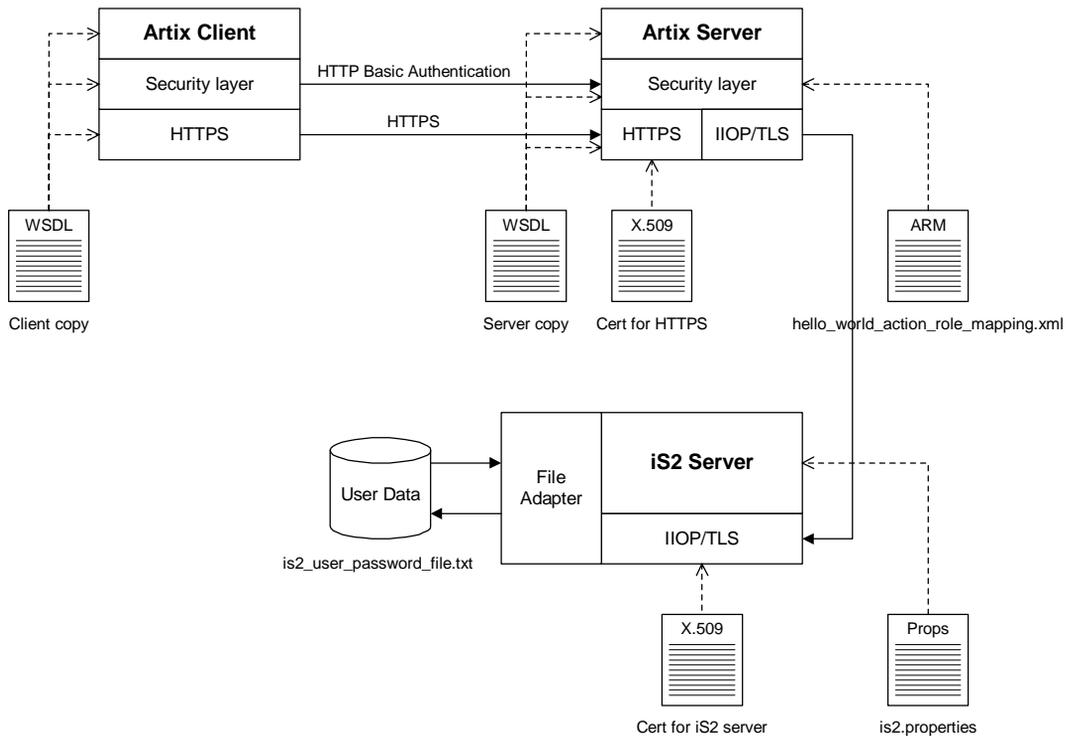


Figure 1: Overview of the Secure HelloWorld Example

Location	The secure HelloWorld demonstration is located in the following directory: <i>ArtixInstallDir/artix/Version/demos/secure_hello_world/http_soap</i>
Main elements of the example	The main elements of the secure HelloWorld example shown in Figure 1 are, as follows: <ul style="list-style-type: none">• 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: <ul style="list-style-type: none">• 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 for iS2). This example uses the iS2 <i>file adapter</i> , 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.

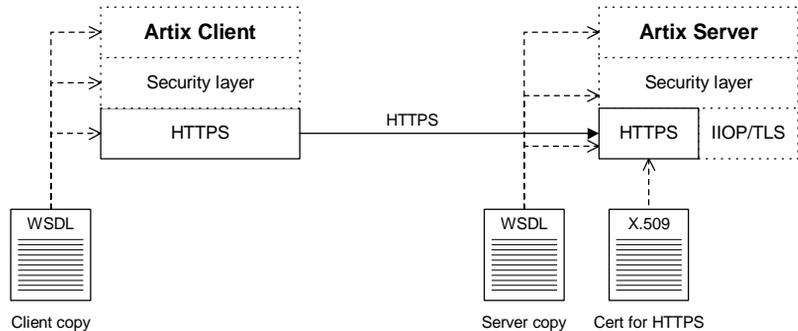


Figure 2: A HTTPS Connection in the HelloWorld Example

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/configuration" ... >
  ...
  <service name="HelloWorldService">

```

Example 2: WSDL Contract with Server HTTPS Configuration

```

1   <port binding="tns:HelloWorldPortBinding"
2   name="HelloWorldPort">
3       <soap:address location="https://localhost:55012"/>
4       <http-conf:server
5           UseSecureSockets="true"
6
7           ServerCertificate="../certificates/openssl/x509/certs/key.cert.pem"
8
9           ServerPrivateKey="../certificates/openssl/x509/certs/privkey.pem"
10
11          ServerPrivateKeyPassword="testaspen"
12
13          TrustedRootCertificates="../certificates/openssl/x509/ca/cacert.pem"
14      />
15  </port>
16  </service>
17 </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 59](#).
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.

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.

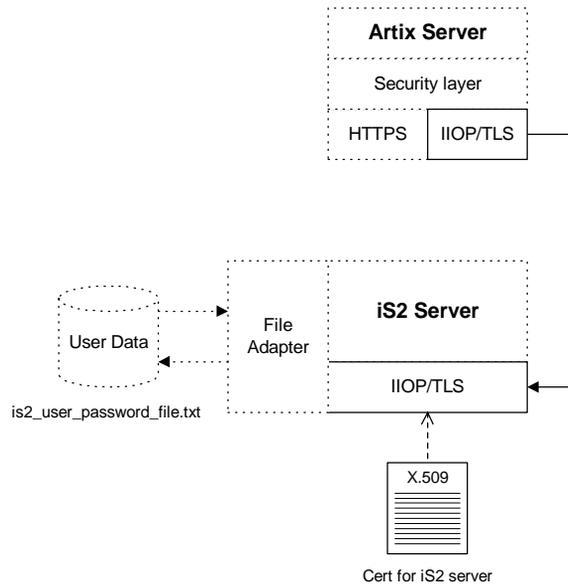


Figure 3: An IIOP/TLS Connection in the HelloWorld Example

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 117.

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/Version/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 IIOP/TLS Configuration*

```

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

  policies:iiop_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 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 Artix server.
For more details, see [“Specifying Trusted CA Certificates” on page 93](#).
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 IIOP/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 IIOP/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 `iioptls` plug-in is included in the list (thus enabling IIOPTLS connections), whereas the `iiopt` plug-in is excluded (thus disabling plain IIOPT 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 IIOPT/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 IIOPT/TLS connections the Artix server can open when it acts in a client role. In particular, these client invocation policies impose conditions on the IIOPT/TLS connection to the iS2 server.

For more details about the client secure invocation policy, see [“Setting IIOPT/TLS Association Options”](#) on page 108.

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 IIOPT/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"];
        ...
3   policies:target_secure_invocation_policy:requires =
        ["NoProtection"];
        policies:target_secure_invocation_policy:supports =
        ["NoProtection", "Confidentiality", "EstablishTrustInTarget",
        "EstablishTrustInClient", "DetectMisordering",
        "DetectReplay", "Integrity"];
        ...
4   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 153](#) and [“Providing a Certificate Pass Phrase” on page 95](#).

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 108](#).

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 90](#).

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.

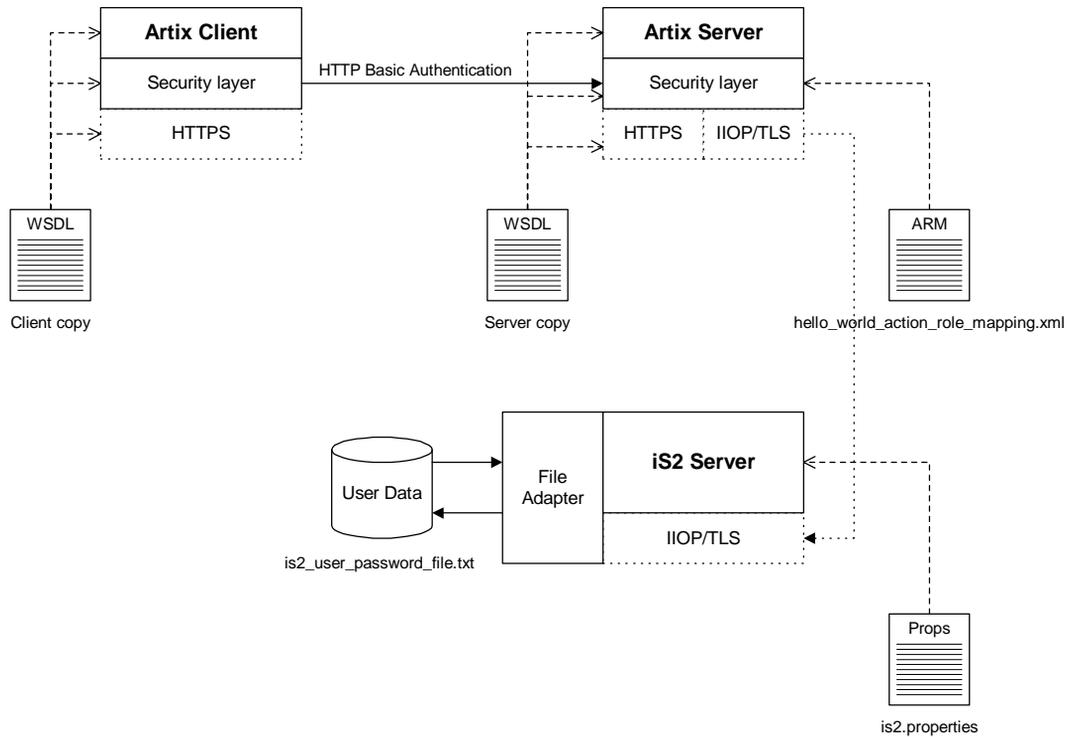


Figure 4: *The Security Layer in the HelloWorld Example*

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 iS2 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 iS2 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 iS2 file adapter, see [“Managing a File Security Domain” on page 48](#).

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

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

```

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

1. The `policies:asp:enable_security` variable is set to `true` to enable login security (enables authentication support and is a prerequisite for authorization support).
2. The `policies:asp:enable_authorization` variable is set to `true` to enable authorization.
3. This setting specifies the location of an *action-role mapping file* that provides fine-grained access control to operations and port types.
4. The iSF authorization realm determines which of the user's roles will be considered during an access control decision. iSF authorization realms provide a way of grouping user roles together. The `IONAGlobalRealm` (the default) includes all user roles.
5. 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 iSF 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 39](#).

Configuring the iS2 Server

This chapter describes how to configure the properties of the iS2 security 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 26
Configuring the LDAP Adapter	page 28
Configuring the SiteMinder Adapter	page 34
Additional iS2 Configuration	page 36

Configuring the File Adapter

Overview

The iS2 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 iS2 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 iS2 file adapter:

- `is2.properties` file—the default location of the iS2 properties file is as follows:
`ASPInstallDir/etc/domains/DomainName/is2.properties`
 See “[iS2 Properties File](#)” on page 159 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 8](#) shows the properties to set for a file adapter.

Example 8: 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
   e.FileAuthAdapter
3  com.iona.isp.adapter.file.param.filename=ASPInstallDir/etc/domain
   s/DomainName/is2_user_password_role_file.txt

```

Example 8: *Sample File Adapter Properties*

```
#####  
## 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 iS2 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.

See [“Managing a File Security Domain” on page 98](#) for details of how to create or modify the security information file.

4. (*Optionally*) You might also want to edit the general iS2 server properties.

See [“Additional iS2 Configuration” on page 36](#) 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 Orbix E2A Application Server Platform, 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:
ASPInstallDir/etc/domains/DomainName/is2.properties
See [“iS2 Properties File” on page 159](#) for details of how to customize the default iS2 properties file location.

Minimal LDAP configuration

Example 9 shows the minimum set of iS2 properties that can be used to configure an LDAP adapter.

Example 9: 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 159](#) 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.

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

See [“IS2 Properties File” on page 159](#) for more details.

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

In addition to these basic properties, you must also set a number of LDAP parameters, which are prefixed by `com.ionas.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 157](#).

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 IONA 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 Orbix E2A Application Server Platform, but is available from Netegrity at <http://www.netegrity.com>. See the *OrbixE2A ASP Installation Guide* for details.

File location

The following file configures the SiteMinder adapter:

- `is2.properties` file—the default location of the iS2 properties file is as follows:
`ASPInstallDir/etc/domains/DomainName/is2.properties`
 See “iS2 Properties File” on page 159 for details of how to customize the default iS2 properties file location.

SiteMinder adapter properties

[Example 10](#) shows the properties to set for the SiteMinder adapter.

Example 10: SiteMinder Adapter Properties

```

1  com.iona.isp.adapters=SiteMinder
   #####
   ##
   ## SiteMinder Adapter Properties
   ##
   #####
2  com.iona.isp.adapter.SiteMinder.class=com.iona.security.is2adapter.
   smadapter.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 10: 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 iS2 Configuration"](#) on page 36 for details.

Additional iS2 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 iS2 adapter type.

In this section

This section contains the following subsections:

Configuring the Log4J Logging

page 37

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 iS2 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 IONA 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 40
Managing a File Security Domain	page 48
Managing an LDAP Security Domain	page 50
Managing a SiteMinder Security Domain	page 51

Introduction to Domains and Realms

Overview

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

In this section

This section contains the following subsections:

iSF Security Domains	page 41
iSF Authorization Realms	page 43

iSF Security Domains

Overview

This subsection introduces the concept of an iSF security domain.

Domain architecture

Figure 5 shows the architecture of an iSF security domain. The iSF security domain is identified with an enterprise security service that plugs into the iS2 server through an iS2 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 iSF security domain.

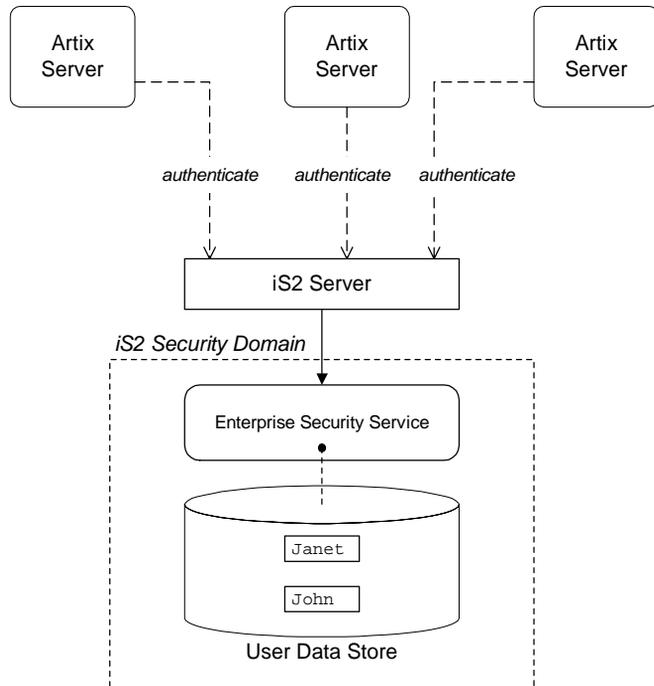


Figure 5: Architecture of an iSF Security Domain

iSF security domain

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

Here are some specific examples of iSF 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 iSF security domain

Effectively, you create an iSF security domain by configuring the iS2 server to link to an enterprise security service through an iS2 adapter (such as a SiteMinder adapter or an LDAP adapter). The enterprise security service is the implementation of the iSF 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 48](#).

iSF Authorization Realms

Overview

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

iSF authorization realm

An *iSF 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 IONA 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 iSF authorization realm is a way of grouping servers with similar authorization requirements. Figure 6 shows two iSF authorization realms, `Engineering` and `Finance`, each containing a collection of server applications.

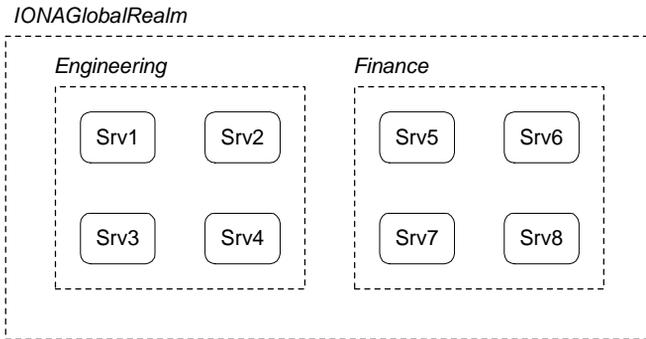


Figure 6: Server View of iSF Authorization Realms

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 iSF 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 iSF authorization realm acts as a namespace for roles. For example, [Figure 7](#) shows two iSF authorization realms, `Engineering` and `Finance`, each associated with a set of roles.

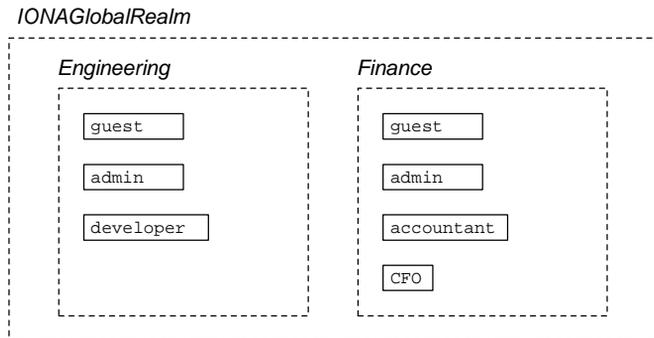


Figure 7: *Role View of iSF Authorization Realms*

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 8](#) 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.

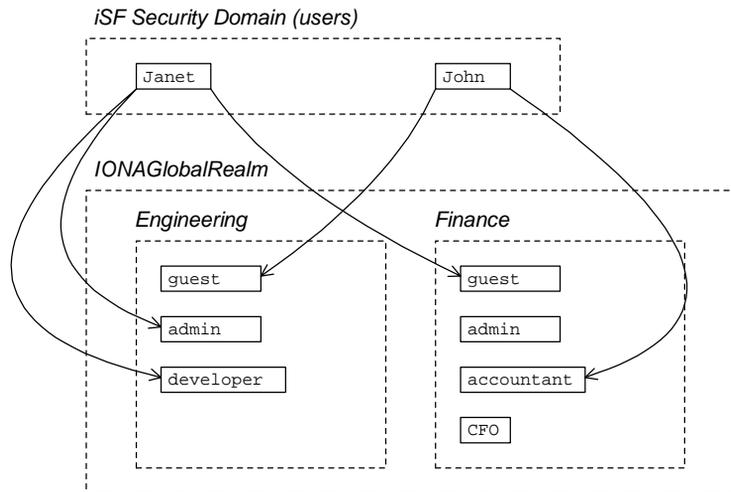


Figure 8: Assignment of Realms and Roles to Users Janet and John

Special realms and roles

The following special realms and roles are supported by the IONA security framework:

- `IONAGlobalRealm` realm—a special realm that encompasses every iSF authorization realm. Roles defined within the `IONAGlobalRealm` are valid within every iSF 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 26](#)). 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 11](#) 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 11: 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">
5       <realm name="Financials"
          description="Financial Department">
```

Example 11: *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.

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 IONA Security Framework is that the iS2 server be configured to use the LDAP adapter.

See [“Configuring the LDAP Adapter” on page 28](#).

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 and mapping users to roles. Using Netegrity tools you can administer users, roles, and realms.

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 IONA Security Framework is that the iS2 server be configured to use the SiteMinder adapter.

See [“Configuring the SiteMinder Adapter” on page 34](#).

References

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

Managing Access Control Lists

The IONA 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 54
Artix Action-Role Mapping ACL	page 55

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/Version/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 12](#).

Example 12: 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 13](#) shows how you might configure an action-role mapping file for the `HelloWorldPortType` port type given in the preceding [Example 12](#) on [page 56](#).

Example 13: 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 13: *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 12 on page 56](#) 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 183](#) for details.

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 60
Certification Authorities	page 62
Certificate Chaining	page 65
PKCS#12 Files	page 67
Creating Your Own Certificates	page 69
Deploying Certificates	page 76

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 177](#) 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 63
Private Certification Authorities	page 64

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 203](#). 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 69](#).

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 9 shows an example of a simple certificate chain.

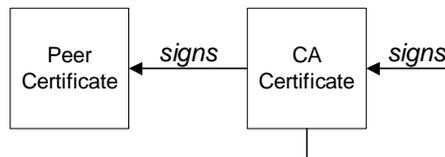


Figure 9: A Certificate Chain of Depth 2

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 10](#) shows what this certificate chain looks like.

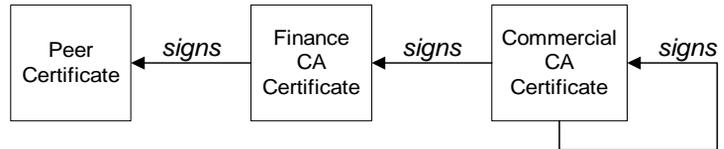


Figure 10: A Certificate Chain of Depth 3

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 78](#).

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 11 shows the typical elements in a PKCS#12 file.

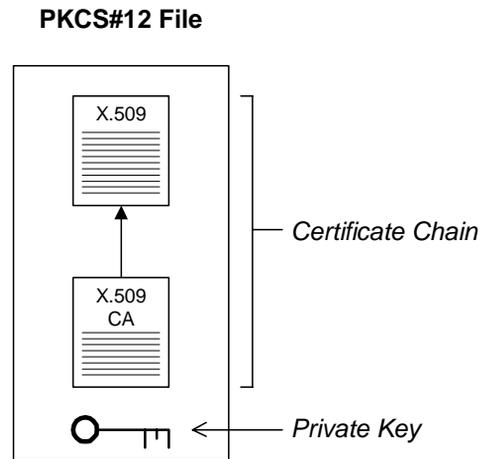


Figure 11: Elements in a PKCS#12 File

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 73](#).

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 187. 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 70
Use the CA to Create Signed Certificates	page 73

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 64](#).

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/crl
```

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 197](#).

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 70](#), 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) []:Orbix
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 70.

```

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 Orbix 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 77
Deploying Trusted Certificate Authority Certificates	page 78
Deploying Application Certificates	page 82

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/Version/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 70](#)). 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>
```

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.ionas.com/transport/http"
xmlns:http-conf="http://schemas.ionas.com/transport/http/co
nfiguration" ... >
...
<service name="...">
  <port binding="...">
    ...
    <http-conf:server ...
      TrustedRootCertificates="X509CA/ca/ca_list01.pem"
    ... />
  </port>
</service>
```

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 70](#)). 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/Version/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"
ClientCertificateChain="X509Deploy/certs/applications/CertName.pem"
ClientPrivateKey="X509Deploy/certs/applications/CertPrivKey.pem"
ClientPrivateKeyPassword="MyKeyPassword"
TrustedRootCertificates="RootCertPath"
... />
    </port>
  </service>
```

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/transport/http"
xmlns:http-conf="http://schemas.iona.com/transport/http/configuration" ... >
...
<service name="...">
  <port binding="...">
    <soap:address ...>
      <http-conf:server UseSecureSockets="true"
ServerCertificate="X509Deploy/certs/applications/CertName.pem"
ServerCertificateChain="X509Deploy/certs/applications/CertName.pem"
ServerPrivateKey="X509Deploy/certs/applications/CertPrivKey.pem"
ServerPrivateKeyPassword="MyKeyPassword"
TrustedRootCertificates="RootCertPath"
... />
    </port>
  </service>
```

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/Version/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 95](#).

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 86
Specifying Trusted CA Certificates	page 93
Specifying an Application's Own Certificate	page 94
Providing a Certificate Pass Phrase	page 95
Advanced IIOP/TLS Configuration Options	page 100

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 87
Mutual Authentication	page 90

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 12](#).

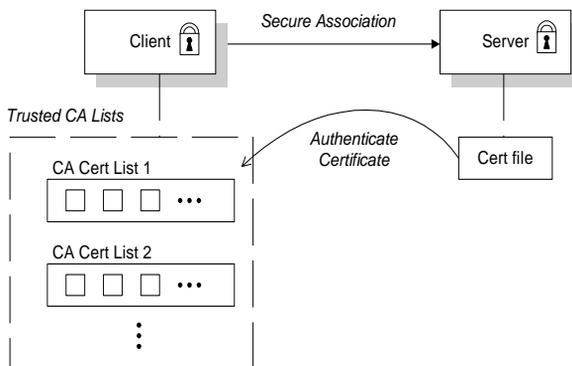


Figure 12: *Target Authentication Only*

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 94](#).
- One or more lists of trusted certification authorities (CA) are made available to the client—see [“Deploying Trusted Certificate Authority Certificates” on page 78](#).

During the security handshake, the server sends its certificate chain to the client—see [Figure 12](#). 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.ionaproducts.com/transport/http"
xmlns:http-conf="http://schemas.ionaproducts.com/transport/http/configuration" ... >
...
<service name="...">
  <port binding="...">
    <soap:address ...>
      <http-conf:client UseSecureSockets="true"
        TrustedRootCertificates="RootCertPath"
        ... />
    </port>
  </service>
```

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.ionaproducts.com/transport/http"
xmlns:http-conf="http://schemas.ionaproducts.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>
```

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:iioptls:mechanism_policy:protocol_version = "SSL_V3";
policies:iioptls:mechanism_policy:ciphersuites =
    ["RSA_WITH_RC4_128_SHA", "RSA_WITH_RC4_128_MD5"];

bank_server {
    policies:iioptls:target_secure_invocation_policy:requires =
        ["Confidentiality"];
    policies:iioptls:target_secure_invocation_policy:supports =
        ["Confidentiality", "Integrity", "DetectReplay",
        "DetectMisordering", "EstablishTrustInTarget"];
    ...
};

bank_client {
    ...
    policies:iioptls:client_secure_invocation_policy:requires =
        ["Confidentiality", "EstablishTrustInTarget"];
    policies:iioptls: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 13. In this case, the server and the client each require an X.509 certificate for the security handshake.

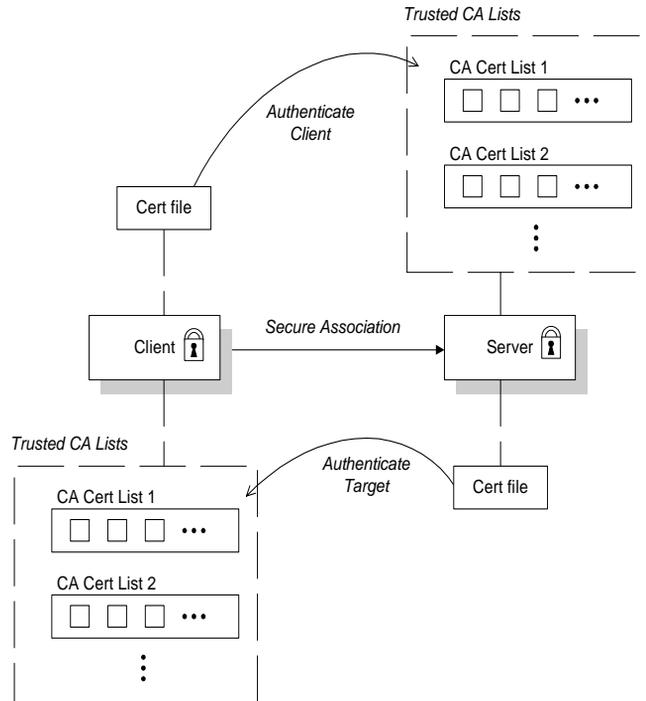


Figure 13: Mutual Authentication

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 94](#).
- Both client and server are configured with lists of trusted certification authorities (CA)—see [“Deploying Trusted Certificate Authority Certificates” on page 78](#).

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 12](#).

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 82](#).

IIOPTLS 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 IIOPTLS.

```
# 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:iioptls:client_secure_invocation_policy:requires =
  ["Confidentiality", "EstablishTrustInTarget"];
  policies:iioptls: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 78.](#)
- [“Deploying for the IIOP/TLS transport” on page 80.](#)

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 82.](#)

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 96
Certificate Pass Phrase for IIOP/TLS	page 98

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>
```

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>
```

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 207](#), 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 101
Applying Constraints to Certificates	page 102

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:iiop_tls:max_chain_length_policy` configuration variable. For example:

```
policies:iiop_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:iiop_tls:certificate_constraints_policy` configuration variable. For example:

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

Constraint language

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

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

Example

This is an example list of constraints:

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

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

```
If
  The OU is unit1 or IT_SSL
And
```

```

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

```

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

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

Distinguished names

For more information on distinguished names, see [“ASN.1 and Distinguished Names”](#) on page 177.

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 106
Setting IIOP/TLS Association Options	page 108
Specifying IIOP/TLS Cipher Suites	page 116
Caching IIOP/TLS Sessions	page 125

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 112](#) 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 114](#) 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 116](#) for details.

Figure 14 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.

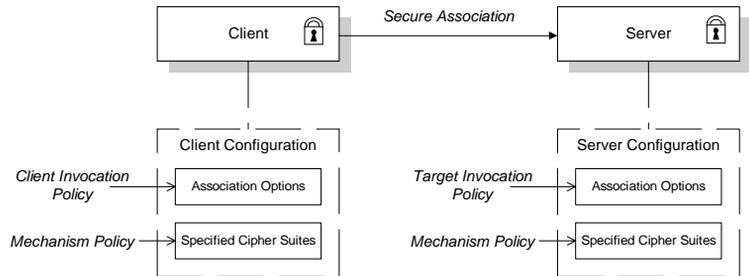


Figure 14: Configuration of a Secure Association

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 109
Association Options	page 110
Choosing Client Behavior	page 112
Choosing Target Behavior	page 114

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:iiop_tls:target_secure_invocation_policy:requires =
  ["EstablishTrustInClient", "Confidentiality"];

  policies:iiop_tls: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 110](#).

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 110](#).

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"];
  ...
};
...
```

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 117
Setting the Mechanism Policy	page 120
Constraints Imposed on Cipher Suites	page 122

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 IIOP/TLS:

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 IIOP/TLS:

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 IIOP/TLS cipher suites are defined as follows:

Table 2: *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 15 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*.

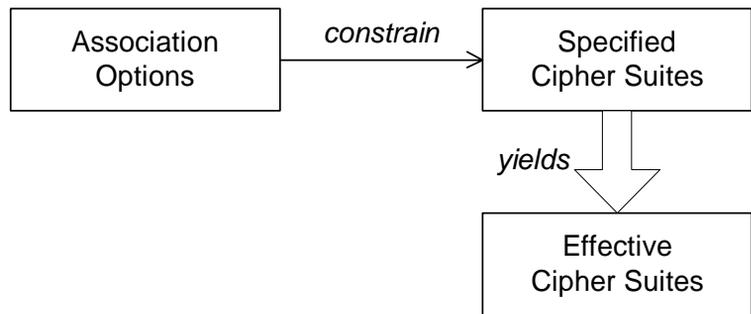


Figure 15: Constraining the List of Cipher Suites

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 3](#) to determine whether or not a particular cipher suite is compatible with your association options.

Table 3: *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 3](#)) do not satisfy the configured required association options.
3. From the remaining list, remove any cipher suite that supports an option (see [Table 3](#)) 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 128
Configuring	page 129
Programming	page 132
Interoperating with .NET	page 135

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/Version/etc/domains` directory):

Example 14: 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 15](#).

Example 15: *Configuring Principal Authentication for SOAP*

```
# Security Layer Settings
policies:asp:enable_security = "true";
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 15](#) 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 25 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 16](#).

Example 16: *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 17](#) shows how to set the principal prior to invoking an operation, `echoString()`, on a proxy object, of `MyProxy` type.

Example 17: 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 18](#) shows how to read the principal on the server side, when the servant is invoked by a client that uses principal propagation.

Example 18: Reading the Principal on the Server Side

```

// C++
// Implementation constructor.
//
MyImpl::MyImpl(IT_Bus::Bus_ptr bus, IT_Bus::Port* port) :
    BaseServer(bus, port)
1  {
    get_port().use_input_message_attributes(true);
}

// in operation..
2  void MyImpl::echoString(const IT_Bus::String&
    inputString, IT_Bus::String& Response)
    IT_THROW_DECL((IT_Bus::Exception))
    {
        Response = inputString;

        try {
            const String & the_principal =
3            get_port().get_input_message_attributes().get_string(
                "Principal"
            );
4        }
        catch (IT_Bus::NoSuchAttributeException) { }
    }

```

The preceding server example can be explained as follows:

1. 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.
2. This is the implementation of the `echoString()` operation that was called in [Example 18](#).

3. This line uses the transport-neutral message attribute API to read the `Principal` value received from the client.
4. 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 136
Modifying the SOAP Header	page 138

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 19 shows the typical modifications you must make to a WSDL contract in order to make the principal value accessible to non-Artix applications.

Example 19: 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 19: 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.iona.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.iona.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 20](#).

Example 20: Customizing the Form of the Principal Header

```
<definitions ... >
  <types>
    <schema targetNamespace="TypeSchema" ... >
      ...
      <element name="PrincipalTag" type="xsd:string"/>
      ...
    </schema>
  </types>
  ...
  <message targetNamespace="http://schemas.iona.com/security"
```

Example 20: *Customizing the Form of the Principal Header*

```

    name="principal">
2      <part element="TypePrefix:PrincipalTag" name="principal"/>
</message>
...
<binding ... xmlns:sec="http://schemas.ionas.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.ionas.com/security`.

Security Configuration

This appendix provides details of Artix security configuration variables.

In this appendix

This appendix contains the following sections:

plugins Namespace	page 142
policies Namespace	page 146
principal_sponsor Namespace	page 153
principal_sponsor:csi Namespace	page 155

plugins Namespace

List of configuration variables

The `plugins` namespace contains the following configuration variables.

`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 iSF 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 `PRINCIPAL`, 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 is propagated without an accompanying password.

When the `PRINCIPAL` security type is selected, the `asp` 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 IONA 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 `asp` 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>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:gsp:authentication_cache_size`

For CORBA bindings, specifies 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`

For CORBA bindings, specifies 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: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: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 IIOP. IIOP does not accept new connections above this limit. Defaults to -1 (disabled).

When this variable is set, the `iiop_tls` plug-in reads this variable's value instead of the

`plugins:iiop:incoming_connections:hard_limit` variable's value.

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

```
plugins:iiop_tls:incoming_connections:soft_limit
```

Specifies the number of connections at which IIOP should begin closing incoming (server-side) connections. Defaults to -1 (disabled). When this variable is set, the `iiop_tls` plug-in reads this variable's value instead of the

```
plugins:iiop:incoming_connections:soft_limit
```

 variable's value.

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

```
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";
```

policies Namespace

List of configuration variables

The policies namespace defines the default CORBA policies for an ORB. Many of these policies can also be set programmatically from within an application.

`policies: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:asp:enable_authorization`

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

Note: This feature requires that the `policies:asp:enable_security` variable is also set to `true`.

`policies:asp:enable_security`

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

`policies:certificate_constraints_policy`

A list of constraints applied to peer certificates—see [“Applying Constraints to Certificates” on page 102](#) for the syntax of the pattern constraint language.

`policies: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. For defaults, see [“Choosing Client Behavior” on page 112](#).

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

`policies: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. For defaults, see [“Choosing Client Behavior” on page 112](#).

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

`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:client_supports`

A client-side policy that specifies the association options supported by CSIV2 authorization over transport. The only association option that can be specified is `EstablishTrustInClient`. For example:

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

`policies:csi:auth_over_transport:server_domain_name`

The iSF security domain (CSIV2 authentication domain) to which this server application belongs. The iSF security domains are administered within an overall security technology domain.

`policies:csi:auth_over_transport:target_requires`

A server-side policy that specifies the association options required for CSIV2 authorization over transport. The only association option that can be specified is `EstablishTrustInClient`. For example:

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

`policies:csi:auth_over_transport:target_supports`

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

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

`policies:gsp:enable_authorization`

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

Default is `true`.

`policies:gsp:enable_security_service_cert_authentication`

A boolean GSP policy that enables X.509 certificate-based authentication using the iS2 server.

Default is `false`.

`policies:iiop_tls:allow_unauthenticated_clients_policy`

When this policy is set, the `iiop_tls` plug-in reads this policy's value instead of the [policies:allow_unauthenticated_clients_policy](#) policy's value.

`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`

When this policy is set, the `iiop_tls` plug-in reads this policy's value instead of the [policies:certificate_constraints_policy](#) policy's value.

`policies:iiop_tls:client_secure_invocation_policy:requires`

When this policy is set, the `iiop_tls` plug-in reads this policy's value instead of the [policies:client_secure_invocation_policy:requires](#) policy's value.

`policies:iiop_tls:client_secure_invocation_policy:supports`

When this policy is set, the `iiop_tls` plug-in reads this policy's value instead of the [policies:client_secure_invocation_policy:supports](#) policy's value.

`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:max_chain_length_policy`

When this policy is set, the `iiop_tls` plug-in reads this policy's value instead of the [policies:max_chain_length_policy](#) policy's value.

`policies:iiop_tls:mechanism_policy:ciphersuites`

When this policy is set, the `iiop_tls` plug-in reads this policy's value instead of the [policies:mechanism_policy:ciphersuites](#) policy's value.

`policies:iiop_tls:mechanism_policy:protocol_version`

When this policy is set, the `iiop_tls` plug-in reads this policy's value instead of the [policies:mechanism_policy:protocol_version](#) policy's value.

`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_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++).

`policies:iiop_tls:target_secure_invocation_policy:requires`

When this policy is set, the `iiop_tls` plug-in reads this policy's value instead of the [policies:target_secure_invocation_policy:requires](#) policy's value.

`policies:iiop_tls:target_secure_invocation_policy:supports`

When this policy is set, the `iiop_tls` plug-in reads this policy's value instead of the [policies:target_secure_invocation_policy:supports](#) policy's value.

`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`

When this policy is set, the `iiop_tls` plug-in reads this policy's value instead of the [policies:trusted_ca_list_policy](#) policy's value.

`policies:max_chain_length_policy`

The maximum certificate chain length that an ORB will accept (see [“Certificate Chaining” on page 65](#)).

`policies: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 4: *Mechanism Policy Cipher Suites*

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

`policies: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.

For example:

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

`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`

Specifies the minimum level of security required by a server. The value of this variable is specified as a list of association options. For defaults, see [“Choosing Target Behavior” on page 114](#).

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

`policies: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. For defaults, see [“Choosing Target Behavior” on page 114](#).

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

`policies: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 65](#).

principal_sponsor Namespace

List of configuration variables

The `principal_sponsor` namespace stores configuration information to be used when obtaining credentials. Artix provides an implementation of a principal sponsor that creates credentials for applications automatically. The principal sponsor automatically calls the `authenticate()` operation on the `PrincipalAuthenticator` object after determining the data to supply.

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 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"];
```

principal_sponsor:csi Namespace

List of configuration variables

The `principal_sponsor:csi` namespace stores configuration information to be used when obtaining credentials. Artix provides an implementation of a principal sponsor that creates credentials for applications automatically. The principal sponsor automatically calls the `authenticate()` operation on the `PrincipalAuthenticator` object after determining the data to supply.

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_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:

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

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

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

```
principal_sponsor: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.

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: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 158
iS2 Properties File	page 159
Cluster Properties File	page 172
log4j Properties File	page 174

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.ionacorba.art.artimpl.ORBImpl",
"org.omg.CORBA.ORBSingletonClass=com.ionacorba.art.artimpl.ORBSingleton",
"is2.properties=ASPInstallDir/etc/domains/DomainName/is2.properties"];
        ...
    };
};
```

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.GroupBaseDN`

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.GroupBaseDN=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.isp.adapter.SiteMinder.class
```

Specifies the Java class that implements the SiteMinder adapter.

For example, the default implementation of the SiteMinder adapter provided with Orbix E2A ASP is selected as follows:

```
com.iona.isp.adapter.SiteMinder.class=com.iona.security.is2adapter.smadapter.SiteMinderAgent
```

```
com.iona.isp.adapter.SiteMinder.param.AgentName
```

Specifies the SiteMinder agent's name.

For example:

```
com.iona.isp.adapter.SiteMinder.param.AgentName=web
```

```
com.iona.isp.adapter.SiteMinder.param.AgentSecret
```

Specifies the SiteMinder agent's password.

For example:

```
com.iona.isp.adapter.SiteMinder.param.AgentSecret=secret
```

```
com.iona.isp.adapter.SiteMinder.param.ServerAddress
```

Specifies the IP hostname where the SiteMinder server is running.

For example:

```
com.iona.isp.adapter.SiteMinder.param.ServerAddress=localhost
```

`com.iona.isp.adapter.SiteMinder.param.ServerAuthnPort`
 Specifies the IP port where the SiteMinder server is listening.

For example,

```
com.iona.isp.adapter.SiteMinder.param.ServerAuthnPort=44442
```

`com.iona.isp.adapter.SiteMinder.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 174](#).

For example,

```
log4j.configuration=d:/temp/myconfig.txt
```

Cluster Properties File

Overview

The cluster properties file is used to store properties common to a group of iS2 server instances that operate as a cluster. This section provides descriptions of all the properties that can be specified in a cluster file.

File location

The location of the cluster properties file is specified by the `is2.cluster.properties.filename` property in the iS2 properties file. All of the iS2 server instances in a specific cluster must share the same cluster properties file.

List of properties

The following properties can be specified in the cluster properties file:

`com.iona.security.common.securityInstanceURL.<server_ID>`

For the `<server_ID>` iS2 server instance, specifies the server's URL.

When single sign-on (SSO) and clustering are both enabled, the iS2 server instances use the specified instance URLs to communicate with each other. By specifying the URL for a particular iS2 server instance, you are instructing the instance to listen for messages at that URL.

Because the iS2 server instances share the same cluster file, they can read each other's URLs and open connections to each other.

The connections between iS2 server instances can either be insecure (HTTP cluster) or secure (HTTPS cluster). To enable SSL/TLS security, use the `https:` prefix in each of the instance URLs.

For example, to configure two iS2 server instances to operate in a cluster using *insecure* communications (HTTP):

```
com.iona.security.common.securityInstanceURL.1=http://localhost:8080/isp/AuthService
com.iona.security.common.securityInstanceURL.2=http://localhost:8081/isp/AuthService
```

Alternatively, to configure two iS2 server instances to operate in a cluster using *secure* communications (HTTPS):

```
com.iona.security.common.securityInstanceURL.1=https://localhost:8080/isp/AuthService
com.iona.security.common.securityInstanceURL.2=https://localhost:8081/isp/AuthService
```

In the secure case, you must also configure the certificate-related cluster properties for each iS2 server instance.

`com.iona.security.common.securityInstanceCACertDir.<server_ID>`

For the <server_ID> iS2 server instance in a HTTPS cluster, specifies the directory containing trusted CA certificates. The CA certificates can either be in DER-encoded X.509 format or in PEM-encoded X.509 format.

For example, to specify `d:/temp/cert` as the CA certificate directory for the primary iS2 server instance:

```
com.iona.security.common.securityInstanceCACertDir.1=d:/temp/cert
```

`com.iona.security.common.securityInstanceClientCert.<server_ID>`

For the <server_ID> iS2 server instance in a HTTPS cluster, specifies the client certificate file that identifies the iS2 server to its peers within a cluster. The certificate must be in PKCS#12 format.

No default.

`com.iona.security.common.securityInstanceClientCertPasswd.<server_ID>`

For the <server_ID> iS2 server instance in a HTTPS cluster, specifies the password for the client certificate that identifies the iS2 server to its peers within a cluster.

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.

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 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 178
Distinguished Names	page 179

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 IONA 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 5](#) shows a selection of the attribute types that you are most likely to encounter:

Table 5: *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 5](#)). 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 21](#).

Example 21:

```
<?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 iSF 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 188
The OpenSSL Configuration File	page 197

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 203](#) 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 207](#).

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 207](#) 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 197](#).

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 219](#).

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 198
[ca] Variables	page 199
[policy] Variables	page 200
Example openssl.cnf File	page 201

[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 201](#), 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 201](#): `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 Orbix.

In this appendix

This appendix contains the following section:

OpenSSL License	page 204
---------------------------------	--------------------------

OpenSSL License

Overview

The licence agreement for the usage of the OpenSSL command line utility shipped with Orbix2000 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 135

<action-role-mapping> tag 57

<interface> tag 57

<name> tag 57

<realm> tag 49

<role> tag 49

<server-name> tag 57

<users> tag 49

A

ACL

<action-role-mapping> tag 57

<interface> tag 57

<name> tag 57

<server-name> tag 57

action_role_mapping configuration variable 55

action_role_mapping file 55

action_role_mapping file, example 56

action-role mapping

and role-based access control 43

action_role_mapping configuration variable 55

action-role mapping file

<action-role-mapping> tag 57

<interface> tag 57

<name> tag 57

<server-name> tag 57

CORBA

configuring 55

example 56

administration

OpenSSL command-line utilities 69

AgentSecret property 35

ASN.1 61

asp plug-in

default_password configuration value 130

security_type configuration variable 130

association options

and cipher suite constraints 122

client secure invocation policy, default 112

compatibility with cipher suites 123

SSL/TLS

Confidentiality 110

DetectMisordering 110

DetectReplay 110

EstablishTrustInClient 111

EstablishTrustInTarget 111

Integrity 110

NoProtection 110

setting 108

target secure invocation policy, default 114

authentication

own certificate, specifying 94

pass phrase

dialog prompt, C++ 98

in configuration 99

password file, from 98

SSL/TLS

mutual 90

target only 87

trusted CA list 93

authorization

role-based access control 43

roles

creating 45

special 47

authorization realm

adding a server 44

IONAGlobalRealm realm 47

iSF 43

roles in 45

servers in 44

special 47

authorization realms

creating 45

C

CA 60

choosing a host 64

commercial CAs 63

index file 71

list of trusted 66

multiple CAs 66

private CAs 64

private key, creating 72

- security precautions 64
- See A/socertificate authority
- self-signed 72
- serial file 71
- trusted list 78, 93
- 199
- CA, setting up 70
- CACHE_CLIENT session caching value 125
- CACHE_NONE session caching value 125
- CACHE_SERVER_AND_CLIENT session caching value 125
- CACHE_SERVER session caching value 125
- caching
 - CACHE_CLIENT session caching value 125
 - CACHE_NONE session caching value 125
 - CACHE_SERVER_AND_CLIENT session caching value 125
 - CACHE_SERVER session caching value 125
- SSL/TLS
 - cache size 125
 - validity period 125
- Caching sessions 125
- CAs 70
- ca utility 195
- certificate authority
 - and certificate signing 60
- certificate_constraints_policy variable 102
- Certificates
 - chain length 101
 - constraints 102
- certificates
 - chaining 65
 - constraint language 102
 - contents of 61
 - creating and signing 73
 - deployment, 77
 - importing and exporting 68
 - length limit 66
 - own, specifying 94
 - pass phrase 98
 - peer 65
 - PKCS#12 file 67
 - public key 61
 - public key encryption 117
 - security handshake 87, 91
 - self-signed 65, 72
 - serial number 61
 - signing 60, 74
 - signing request 73
 - trusted CA list 78, 93
 - X.509 60
- certificate signing request 73
 - common name 74
 - signing 74
- chaining of certificates 65
- ciper suites
 - order of 121
- cipher suites
 - ciphersuites configuration variable 120
 - compatibility algorithm 123
 - compatibility with association options 123
 - default list 121
 - definitions 118
 - effective 122
 - encryption algorithm 117
 - exportable 118
 - integrity-only ciphers 117
 - key exchange algorithm 117
 - mechanism policy 120
 - secure hash algorithm 117
 - secure hash algorithms 118
 - security algorithms 117
 - specifying 116
 - standard ciphers 117
- ciphersuites configuration variable 120
- client secure invocation policy 122
 - IIOPTLS 112
- ClientSecureInvocationPolicy policy 109
- colocated invocations
 - and secure associations 106
- colocation
 - incompatibility with principal propagation 128
- common names
 - uniqueness 74
- Confidentiality association option 110
- Confidentiality option 110
- Configuration file 197
- constraint language 102
- Constraints
 - for certificates 102
- CORBA
 - action-role mapping file 55
 - action-role mapping file, example 56
 - configuring principal propagation 129
 - principal propagation 128
- CSR 73

D

- data encryption standard
 - see DES
- default_password configuration value 130
- DES
 - symmetric encryption 118
- DetectMisordering association option 110
- DetectMisordering option 110
- DetectReplay association option 110
- DetectReplay option 110

E

- effective cipher suites
 - definition 122
- enable_principal_service_context configuration variable 129
- encryption algorithm
 - RC4 118
- encryption algorithms 117
 - DES 118
 - symmetric 118
 - triple DES 118
- enterprise security service
 - and iSF security domains 41
- EstablishTrustInClient association option 111
- EstablishTrustInClient option 111
- EstablishTrustInTarget association option 111
- EstablishTrustInTarget option 111
- exportable cipher suites 118

F

- file adapter 26
 - properties 26
- file domain
 - <realm> tag 49
 - <users> tag 49
 - example 48
 - file location 48
 - managing 48

G

- GroupBaseDN property 30
- GroupNameAttr property 30
- GroupObjectClass property 30
- GroupSearchScope property 31

H

- HTTPS
 - ciphersuites configuration variable 120

I

- IIOPTLS
 - ciphersuites configuration variable 120
- index file 71
- Integrity association option 110
- integrity-only ciphers 117
- Integrity option 110
- interoperability
 - explicit principal header 136
 - OS/390, SSL/TLS 120
 - with .NET 135
 - with Orbix applications 128
- IONAGlobalRealm realm 47
- is2.properties file 26
- iS2 adapters
 - enterprise security service 41
 - file domain
 - managing 48
 - LDAP domain
 - managing 50
 - SiteMinder domain
 - managing 51
- iS2 server
 - configuring 25
 - file adapter 26
 - is2.properties file 26
 - LDAP adapter 28
 - LDAP adapter, properties 29
 - log4j logging 37
 - security information file 26
 - SiteMinder adapter, configuring 34
- iSF
 - security domain
 - creating 42
 - user account
 - creating 42

K

- key exchange algorithms 117

L

- LDAP adapter 28
 - basic properties 31
 - GroupBaseDN property 30

- GroupNameAttr property 30
- GroupObjectClass property 30, 31
- LDAP server replicas 32
- MemberDNAttr property 31
- PrincipalUserDN property 33
- PrincipalUserPassword property 33
- properties 29
- replica index 32
- RoleNameAttr property 30
- SSLCACertDir property 33
- SSLClientCertFile property 33
- SSLClientCertPassword property 33
- SSEnabled property 33
- UserBaseDN property 30
- UserNameAttr property 30
- UserObjectClass property 30
- UserRoleDNAttr property 30
- LDAP domain
 - managing 50
- Lightweight Directory Access Protocol
 - see LDAP
- log4j 37
 - documentation 37
 - properties file 37
- logging
 - log4j 37

M

- max_chain_length_policy configuration variable 101
- MD5 110, 118
- MechanismPolicy 110
- mechanism policy 120
- MemberDNAttr property 31
- message attributes
 - and routing 131
- message digest 5
 - see MD5
- message digests 110
- message fragments 110
- multiple CAs 66
- mutual authentication 90

N

- NoProtection association option 110
- NoProtection option 110
- NoSuchAttributeException exception 134

O

- OpenSSL 64, 187
- openssl
 - configuration file 197
 - utilities 188
- openssl.cnf example file 201
- OpenSSL command-line utilities 69
- OS/390
 - interoperability with 120

P

- pass phrase 98
 - dialog prompt, C++ 98
 - in configuration 99
 - password file, from 98
- peer certificate 65
- PKCS#12 files
 - creating 68, 73
 - definition 67
 - importing and exporting 68
 - pass phrase 98
 - viewing 68
- POA_Coloc interceptor 128
- policies
 - client secure invocation 122
 - ClientSecureInvocationPolicy 109
 - IIOP/TLS
 - client secure invocation 112
 - target secure invocation 114
 - target secure invocation 122
 - TargetSecureInvocationPolicy 109
 - 200
- principals
 - and colocation 128
 - configuring propagation 129
 - explicit principal header 136
 - from O/S username 129
 - interoperability 128
 - interoperating with .NET 135
 - NoSuchAttributeException exception 134
 - overview 128
 - reading on the server side 133
 - routing configuration 131
 - setting on the client side 132
- principal_sponsor
 - auth_method_data 154, 156
 - use_principal_sponsor 153, 155
- principal_sponsor Namespace Variables 153, 155

PrincipalUserDN property 33
 PrincipalUserPassword property 33
 private key 72
 propagateInputAttribute WSDL tag 131
 protocol version
 interoperability with OS/390 120
 protocol_version configuration variable 120
 public key encryption 117
 public keys 61

R

RC4 encryption 118
 realm
 see authorization realm
 Replay detection 110
 198
 req utility 191
 req Utility command 191
 Rivest Shamir Adleman
 see RSA
 role-based access control 43
 example 46
 RoleNameAttr property 30
 roles
 creating 45
 special 47
 root certificate directory 66
 routing
 and principal propagation 131
 RSA 117
 symmetric encryption algorithm 118
 RSA_EXPORT_WITH_DES40_CBC_SHA cipher
 suite 117, 123
 RSA_EXPORT_WITH_RC4_40_MD5 cipher
 suite 117, 123
 rsa utility 193
 rsa Utility command 193
 RSA_WITH_3DES_EDE_CBC_SHA cipher
 suite 117, 123
 RSA_WITH_DES_CBC_SHA cipher suite 117, 123
 RSA_WITH_NULL_MD5 cipher suite 117, 123
 RSA_WITH_NULL_SHA cipher suite 117, 123
 RSA_WITH_RC4_128_MD5 cipher suite 117, 123
 RSA_WITH_RC4_128_SHA cipher suite 117, 123

S

secure associations
 client behavior 112

 definition 106
 TLS_Coloc interceptor 106
 secure hash algorithms 117, 118
 security algorithms
 and cipher suites 117
 security domain
 creating 42
 security domains
 architecture 41
 iSF 42
 security handshake
 cipher suites 116
 SSL/TLS 87, 91
 security information file 26
 security_type configuration variable 130
 self-signed CA 72
 self-signed certificate 65
 send_principal configuration variable 129
 serial file 71
 serial number 61
 ServerAddress property 35
 ServerAuthnPort property 35
 session_cache_size configuration variable 125
 session_cache_validity_period configuration
 variable 125
 session_caching_policy configuraion variable 125
 session_caching_policy variable 125
 SHA 118
 SHA1 110
 signing certificates 60
 SiteMinder adapter
 AgentSecret property 35
 configuring 34
 properties 34
 ServerAddress property 35
 ServerAuthnPort property 35
 SiteMinder domain
 managing 51
 SOAP
 principal propagation 128
 SOAP binding
 configuring principal propagation 129
 Specifying ciphersuites 116
 SSL/TLS
 association options
 setting 108
 caching validity period 125
 cipher suites 116
 colocated invocations 106

INDEX

- encryption algorithm 117
- key exchange algorithm 117
- mechanism policy 120
- protocol_version configuration variable 120
- secure associations 106
- secure hash algorithm 117
- secure hash algorithms 118
- security handshake 87, 91
- session cache size 125
- TLS session 106
- SSLCACertDir property 33
- SSLClientCertFile property 33
- SSLClientCertPassword property 33
- SSLLeay 64
- SSLEnabled property 33
- standard ciphers 117
- symmetric encryption algorithms 118

T

- Target
 - choosing behavior 114
- target authentication 87
- target secure invocation policy 122
 - IIOPTLS 114
- TargetSecureInvocationPolicy policy 109
- TLS
 - session caching 125
- TLS_Coloc interceptor 106
- TLS session
 - definition 106
- triple DES 118
- trusted CA list 78
- trusted CA list policy 93
- trusted CAs 66

U

- use_message_attributes() function 132, 133
- user account
 - creating 42
- UserBaseDN property 30
- UserNameAttr property 30
- UserObjectClass property 30
- UserRoleDNAttr property 30
- UserSearchScope property
 - LDAP adapter
 - UserObjectClass property 30

V

- Variables 198, 199, 200

X

- X.509
 - public key encryption 117
- X.509 certificate
 - definition 60
- X.509 certificates 59
- x509 utility 189

