WebSphere® Application Server – Express, Version 6

# Securing applications and their environment

**Note**

Before using this information, be sure to read the general information under “Notices” on page 981.

**Compilation date: December 6, 2004**

**© Copyright International Business Machines Corporation 2004. All rights reserved.**
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

# Contents

**How to send your comments** . . . . . . . . . . ix

**Chapter 1. Overview and new features for securing applications and their environment** . . . . . 1
Overview of securing applications and their environment . . . . . 2
What is new for security specialists . . . . . 3
Enabling security for WSIF . . . . . 4

**Chapter 2. How do I secure applications and their environments?** . . . . . 7

**Chapter 3. Securing applications and their environment** . . . . . 13

**Chapter 4. Integrating IBM WebSphere Application Server security with existing security systems** . . . . . 15
Interoperability issues for security . . . . . 18
Interoperating with a C++ common object request broker architecture client . . . . . 18
Interoperating with previous product versions . . . . . 20
Security: Resources for learning . . . . . 21

**Chapter 5. Planning to secure your environment** . . . . . 23
Security considerations when adding a Base Application Server node to Network Deployment . . . . 30
Creating login key files . . . . . 31
Preparing truststore files . . . . . 32
Configuring the application server for interoperability . . . . . 32

**Chapter 6. Implementing security considerations at installation time** . . . . . 33
Securing your environment before installation . . . . . 33
Securing your environment after installation . . . . . 34
Protecting plain text passwords . . . . . 35
PropFilePasswordEncoder command reference . . . . . 37

**Chapter 7. Migrating security configurations from previous releases** . . . . . 39
Migrating custom user registries . . . . . 39
Migrating trust association interceptors . . . . . 42
Migrating Common Object Request Broker Architecture programmatic login to Java Authentication and Authorization Service . . . . . 45
Migrating from the CustomLoginServlet class to servlet filters . . . . . 48

**Chapter 8. Developing secured applications** . . . . . 51
Developing with programmatic security APIs for Web applications . . . . . 51
- Example: Web applications code . . . . . 53
- Developing servlet filters for form login processing . . . . . 54

Developing form login pages . . . . . 59
- Example: Form login . . . . . 60

Developing with programmatic APIs for EJB applications . . . . . 62
- Example: Enterprise bean application code . . . . . 64

Programmatic login . . . . . 66
Developing programmatic logins with the Java Authentication and Authorization Service . . . . . 74
- Example: Programmatic logins . . . . . 76

Custom login module development for a system login configuration . . . . . 78
Example: Customizing a server-side Java Authentication and Authorization Service authentication and login configuration . . . . . 94
Example: Getting the Caller Subject from the Thread . . . . . 99
Example: Getting the RunAs Subject from the Thread . . . . . 100

© Copyright IBM Corp. 2004

Example: Overriding the RunAs Subject on the Thread . . . 101
Example: User revocation from a cache . . . 101
Developing your own J2C principal mapping module . . . 102
Developing custom user registries . . . 104
Example: Custom user registries . . . 105
UserRegistry interface methods . . . 106
Trust association interceptor support for Subject creation . . . 112

**Chapter 9. Assembling secured applications** . . . 115
Enterprise bean component security . . . 115
Securing enterprise bean applications . . . 116
Web component security . . . 117
Securing Web applications using an assembly tool . . . 118
Role-based authorization . . . 120
Admin roles . . . 122
Naming roles . . . 123
Adding users and groups to roles using an assembly tool . . . 124
Mapping users to RunAs roles using an assembly tool . . . 125

**Chapter 10. Deploying secured applications** . . . 127
Assigning users and groups to roles . . . 128
Security role to user and group mappings . . . 129
Security role to user and group selections . . . 130
Look up users and groups settings . . . 131
Delegations . . . 132
Assigning users to RunAs roles . . . 134
Unprotected EJB 2.0 methods protection settings . . . 135
EJB 2.1 method protection level settings . . . 136
RunAs roles to users mapping . . . 136
Updating and redeploying secured applications . . . 137

**Chapter 11. Testing security** . . . 139

**Chapter 12. Administering security** . . . 141
Global security . . . 141
Configuring global security. . . . 142
Enabling global security. . . . 144
Configuring server security . . . 148
Server-level security settings . . . 150
Administrative console and naming service authorization . . . 151
Assigning users to administrator roles . . . 153
Console groups and CORBA naming service groups . . . 154
Assigning users to naming roles . . . 156
Console users settings and CORBA naming service user settings . . . 156
Authentication mechanisms . . . 158
Configuring authentication mechanisms . . . 160
Simple WebSphere authentication mechanism . . . 160
Lightweight Third Party Authentication . . . 160
Configuring Lightweight Third Party Authentication . . . 161
Trust associations . . . 165
Configuring trust association interceptors . . . 169
Single signon . . . 171
Configuring single signon . . . 172
Single signon using WebSEAL or the Tivoli Access Manager plug-in for Web servers . . . 179
Creating a trusted user account in Tivoli Access Manager . . . 179
Configuring WebSEAL for use with WebSphere Application Server . . . 179

Configuring Tivoli Access Manager plug-in for Web servers for use with WebSphere Application Server . . . 180
Configuring single signon using the trust association interceptor . . . 181
Configuring single signon using trust association interceptor ++ . . . 182
Global signon principal mapping . . . 185
Configuring global signon principal mapping . . . 186
User registries . . . 189
Configuring user registries . . . 190
Local operating system user registries . . . 191
Configuring local operating system user registries . . . 195
Lightweight Directory Access Protocol . . . 197
Configuring Lightweight Directory Access Protocol user registries . . . 198
Configuring Lightweight Directory Access Protocol search filters . . . 204
Using specific directory servers as the LDAP server . . . 206
Locating a user's group memberships in Lightweight Directory Access Protocol . . . 209
Dynamic groups and nested group support . . . 210
Dynamic and nested group support for the SunONE or iPlanet Directory Server . . . 210
Configuring dynamic and nested group support for the SunONE or iPlanet Directory Server. . . . 211
Dynamic groups and nested group support for the IBM Directory Server . . . 211
Configuring dynamic and nested group support for the IBM Directory Server . . . 211
Custom user registries . . . 211
Configuring custom user registries . . . 213
Java Authentication and Authorization Service . . . 240
Java Authentication and Authorization Service authorization . . . 240
Configuring application logins for Java Authentication and Authorization Service . . . 242
Login configuration for Java Authentication and Authorization Service . . . 245
Configuration entry settings for Java Authentication and Authorization Service . . . 247
System login configuration entry settings for Java Authentication and Authorization Service . . . 248
Login module settings for Java Authentication and Authorization Service . . . 254
Login module order settings for Java Authentication and Authorization Service . . . 255
Login configuration settings for Java Authentication and Authorization Service . . . 255
J2EE Connector security . . . 256
Managing J2EE Connector Architecture authentication data entries . . . 259
Identity mapping . . . 260
Configuring inbound identity mapping . . . 261
Example: Custom login module for inbound mapping . . . 267
Configuring outbound mapping to a different target realm . . . 270
Example: Using the WSLogin configuration to create a basic authentication subject . . . 271
Example: Sample login configuration for RMI_OUTBOUND . . . 273
Security attribute propagation . . . 275
Enabling security attribute propagation . . . 279
Default PropagationToken . . . 282
Implementing a custom PropagationToken . . . 287
Example: com.ibm.wsspi.security.token.PropagationToken implementation . . . 288
Example: custom PropagationToken login module . . . 294
Default AuthorizationToken . . . 297
Implementing a custom AuthorizationToken . . . 300
Example: com.ibm.wsspi.security.token.AuthorizationToken implementation . . . 301
Example: custom AuthorizationToken login module . . . 306
Default SingleSignonToken . . . 309
Implementing a custom SingleSignonToken . . . 310
Example: com.ibm.wsspi.security.token.SingleSignonToken implementation . . . 311
Example: custom SingleSignonToken login module . . . 316
Example: HTTP cookie retrieval . . . 318
Default AuthenticationToken . . . 321
Implementing a custom AuthenticationToken . . . 322

Example: com.ibm.wsspi.security.token.AuthenticationToken implementation . . . . . . . . . . 323
Example: custom AuthenticationToken login module . . . . . . . . . . . . . . . . . . . . . 329
Propagating a custom Java serializable object . . . . . . . . . . . . . . . . . . . . . . . 331
Authorization in WebSphere Application Server . . . . . . . . . . . . . . . . . . . . . . . 335
JACC providers. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 336
Authorization providers settings . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 336
JACC support in WebSphere Application Server. . . . . . . . . . . . . . . . . . . . . . . 337
JACC policy context handlers . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 340
JACC policy context identifiers (ContextID) format . . . . . . . . . . . . . . . . . . . . 340
JACC policy propagation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 341
JACC registration of the provider implementation classes . . . . . . . . . . . . . . . . . 342
Enabling an external JACC provider . . . . . . . . . . . . . . . . . . . . . . . . . . . 342
External Java Authorization Contract for Containers provider settings . . . . . . . . . . . . 343
Propagating security policy of installed applications to a JACC provider using wsadmin . . . . . . 345
Configuring a JACC provider . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 346
Interfaces used to support JACC . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 348
Tivoli Access Manager integration as the JACC provider. . . . . . . . . . . . . . . . . . . 351
Tivoli Access Manager security for WebSphere Application Server . . . . . . . . . . . . . . 352
Creating the security administrative user . . . . . . . . . . . . . . . . . . . . . . . . . 355
Tivoli Access Manager JACC provider configuration . . . . . . . . . . . . . . . . . . . . . 355
Configuring the JACC provider for Tivoli Access Manager using the wsadmin utility . . . . . . . . 356
Configuring the JACC provider for Tivoli Access Manager using the administrative console . . . . . 358
Tivoli Access Manager JACC provider settings . . . . . . . . . . . . . . . . . . . . . . . 360
Enabling the JACC provider for Tivoli Access Manager . . . . . . . . . . . . . . . . . . . . 361
Configuring additional authorization servers . . . . . . . . . . . . . . . . . . . . . . . . 362
Role-based security with embedded Tivoli Access Manager . . . . . . . . . . . . . . . . . . 362
Administering security users and roles with Tivoli Access Manager . . . . . . . . . . . . . . 363
Configuring Tivoli Access Manager groups . . . . . . . . . . . . . . . . . . . . . . . . . 364
Tivoli Access Manager JACC provider configuration properties . . . . . . . . . . . . . . . . 364
Static role caching properties . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 365
Dynamic role caching properties . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 365
Object caching properties . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 366
Role-based policy framework properties. . . . . . . . . . . . . . . . . . . . . . . . . . 367
System-dependent configuration properties . . . . . . . . . . . . . . . . . . . . . . . . 368
Logging Tivoli Access Manager security . . . . . . . . . . . . . . . . . . . . . . . . . . 368
Enabling embedded Tivoli Access Manager . . . . . . . . . . . . . . . . . . . . . . . . . 369
Disabling embedded Tivoli Access Manager client . . . . . . . . . . . . . . . . . . . . . . 370
Disabling embedded Tivoli Access Manager client using the Administration Console . . . . . . . 370
Disabling embedded Tivoli Access Manager client using wsadmin . . . . . . . . . . . . . . . 371
Forcing the unconfiguration of the Tivoli Access Manager JACC provider . . . . . . . . . . . . 371
Updating console users and groups . . . . . . . . . . . . . . . . . . . . . . . . . . . . 372
The Tivoli Access Manager migrateEAR utility . . . . . . . . . . . . . . . . . . . . . . . 372
Troubleshooting authorization providers . . . . . . . . . . . . . . . . . . . . . . . . . . 375
Authentication protocol for EJB security . . . . . . . . . . . . . . . . . . . . . . . . . . 378
Common Secure Interoperability Version 2 features . . . . . . . . . . . . . . . . . . . . . 382
Identity assertion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 382
Message layer authentication . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 383
Secure Sockets Layer client certificate authentication . . . . . . . . . . . . . . . . . . . 384
Supported authentication protocols . . . . . . . . . . . . . . . . . . . . . . . . . . . 386
Configuring Common Secure Interoperability Version 2 and Security Authentication Service authentication protocols . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 386
Common Secure Interoperability Version 2 and Security Authentication Service client configuration 387
Configuring Common Secure Interoperability Version 2 inbound authentication . . . . . . . . 390
Configuring Common Secure Interoperability Version 2 outbound authentication . . . . . . . . 395
Configuring inbound transports . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 400
Configuring outbound transports . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 402

Example: Common Secure Interoperability Version 2 scenarios . . . 404
Secure Sockets Layer . . . 411
Authenticity . . . 413
Confidentiality . . . 414
Integrity . . . 416
Configuring Secure Sockets Layer . . . 417
Configuring Secure Sockets Layer for Web client authentication . . . 418
Configuring Secure Sockets Layer for the Lightweight Directory Access Protocol client . . . 419
Configuring IBM HTTP Server for Secure Sockets Layer mutual authentication . . . 421
Configuring the Web server plug-in for Secure Sockets Layer . . . 422
Configuring Secure Sockets Layer for Java client authentication . . . 427
Secure Sockets Layer configuration repertoire settings . . . 430
Creating a Secure Sockets Layer repertoire configuration entry . . . 437
Configuring Federal Information Processing Standard Java Secure Socket Extension files . . . 438
Digital certificates . . . 439
Managing digital certificates . . . 443
Changes to IBM Developer Kit for Java Technology Edition Version 1.4.x . . . 452
Cryptographic token support . . . 454
Opening a cryptographic token using the key management utility (iKeyman) . . . 455
Configuring to use cryptographic tokens . . . 455
Cryptographic token settings . . . 458
Using Java Secure Socket Extension and Java Cryptography Extension with Servlets and enterprise bean files . . . 459
Java 2 security . . . 463
Access control exception . . . 468
Configuring Java 2 security . . . 469
Using PolicyTool to edit policy files . . . 471
Migrating Java 2 security policy . . . 494

**Chapter 13. Configuring security with scripting** . . . 497
Enabling and disabling global security using scripting . . . 497
Enabling and disabling Java 2 security using scripting . . . 498

**Chapter 14. Learn about WebSphere applications** . . . 501
Web applications . . . 501
Security constraints . . . 501
EJB applications . . . 502
Configuring security for message-driven beans that use listener ports . . . 502
Configuring security for EJB 2.1 message-driven beans . . . 502
Client applications . . . 503
Accessing secure resources using SSL and applet clients . . . 503
Web services . . . 504
Transport level security . . . 504
Configuring HTTP outbound transport level security with the administrative console . . . 504
Configuring HTTP outbound transport level security with an assembly tool . . . 505
Configuring HTTP outbound transport-level security using Java properties . . . 506
HTTP basic authentication . . . 507
Configuring HTTP basic authentication with the administrative console . . . 507
Configuring HTTP basic authentication with an assembly tool . . . 508
Configuring HTTP basic authentication programmatically . . . 509
Configuring additional HTTP transport properties using the administrative console . . . 510
Configuring additional HTTP transport properties with an assembly tool . . . 511
Configuring additional HTTP transport properties using wsadmin . . . 512
Provide HTTP endpoint URL information . . . 513
Publish WSDL zip files settings . . . 514
Securing Web services for version 6 applications based on WS-Security . . . 515

Securing Web services for version 5.x applications based on WS-Security . . . . . . . . . . . 801
Configuring UDDI Security Roles . . . . . . . . . . . . . . . . . . . . . . . . . 936
Security API for the UDDI V3 Registry . . . . . . . . . . . . . . . . . . . . . . . 937
Data access resources . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 938
Security of lookups with component managed authentication . . . . . . . . . . . . . . . . 938
Messaging resources . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 938
Configuring authorization security for a Version 5 default messaging provider . . . . . . . . . 938
Securing WebSphere MQ messaging directories and log files . . . . . . . . . . . . . . . . 943
Configuring security for message-driven beans that use listener ports . . . . . . . . . . . . 944
Configuring security for EJB 2.1 message-driven beans . . . . . . . . . . . . . . . . . . 945
Mail, URLs, and other J2EE resources . . . . . . . . . . . . . . . . . . . . . . . . 945
JavaMail security permissions best practices . . . . . . . . . . . . . . . . . . . . . 945
Learn about WebSphere programming extensions . . . . . . . . . . . . . . . . . . . . 946
Scheduler . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 946

**Chapter 15. Tuning security configurations** . . . . . . . . . . . . . . . . . . . . 949
Tuning CSIv2 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 949
Tuning LDAP authentication . . . . . . . . . . . . . . . . . . . . . . . . . . . 950
Tuning Web authentication . . . . . . . . . . . . . . . . . . . . . . . . . . . . 950
Tuning authorization . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 951
Security cache properties . . . . . . . . . . . . . . . . . . . . . . . . . . . . 951
Secure Sockets Layer performance tips . . . . . . . . . . . . . . . . . . . . . . . 952
Tuning security . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 953

**Chapter 16. Troubleshooting security configurations** . . . . . . . . . . . . . . . . . 955
Errors when trying to configure or enable security . . . . . . . . . . . . . . . . . . . 955
Access problems after enabling security . . . . . . . . . . . . . . . . . . . . . . . 957
Errors after enabling security . . . . . . . . . . . . . . . . . . . . . . . . . . . 960
Errors trying to enable or configure Secure Socket Layer (SLL) encrypted access . . . . . . . . 965
Errors after configuring or enabling Secure Sockets Layer . . . . . . . . . . . . . . . . . 965
Security components troubleshooting tips . . . . . . . . . . . . . . . . . . . . . . 968

**Notices** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 981

**Trademarks and service marks** . . . . . . . . . . . . . . . . . . . . . . . . . 983

# How to send your comments

Your feedback is important in helping to provide the most accurate and highest quality information.

- To send comments on articles in the WebSphere Application Server Information Center
  1. Display the article in your Web browser and scroll to the end of the article.
  2. Click on the **Feedback** link at the bottom of the article, and a separate window containing an e-mail form appears.
  3. Fill out the e-mail form as instructed, and click on **Submit feedback** .
- To send comments on PDF books, you can e-mail your comments to: **wasdoc@us.ibm.com** or fax them to 919-254-0206.

  Be sure to include the document name and number, the WebSphere Application Server version you are using, and, if applicable, the specific page, table, or figure number on which you are commenting.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright IBM Corp. 2004

# Chapter 1. Overview and new features for securing applications and their environment

This topic summarizes the contents and organization of the security documentation, including links to conceptual overviews and descriptions of new features.

- "Overview of securing applications and their environment" on page 2
- What is new for administrators

**Sections in the security documentation:**

**Chapter 4, "Integrating IBM WebSphere Application Server security with existing security systems," on page 15**
: This provides interoperability information. WebSphere Application Server security is an integral part of your multiple-tier enterprise computing framework. WebSphere Application Server adopts the open architecture paradigm and provides many plug-in points to integrate with enterprise software components to provide end-to-end security. WebSphere Application Server plug-in points are based on standard J2EE specifications wherever applicable. WebSphere Application Server is actively involved in various standard bodies to externalize and to standardize plug-in interfaces.

**Chapter 5, "Planning to secure your environment," on page 23**
: This examines some typical configuration and common security practices. There are several communication links from a browser on the Internet, through web servers and product servers, to the enterprise data at the back end. WebSphere Application Server security is built on a layered security architecture. This also examines the security protection offered by each security layer and common security practice for good quality of protection in end-to-end security.

**Chapter 6, "Implementing security considerations at installation time," on page 33**
: This describes how to implement security before, during, and after installing the product.

**Chapter 7, "Migrating security configurations from previous releases," on page 39**
: This describes how to migrate your security configurations from a previous product release.

**Chapter 8, "Developing secured applications," on page 51**
: This describes how to implement declarative and programmatic security while developing, assembling, and deploying your applications. The product security components provide or collaborate with other services to provide authentication, authorization, delegation, and data protection. The product also supports the security features described in the Java 2 Enterprise Edition (J2EE) specification.

**Chapter 9, "Assembling secured applications," on page 115**
: This describes how to use assembly tools to secure applications and the EJB and Web modules that comprise them.

**Chapter 9, "Assembling secured applications," on page 115**
: This describes security tasks and considerations as you are deploying applications onto the application server and testing that users can access the secured applications.

**Chapter 12, "Administering security," on page 141**
: This describes how to configure and administer security features, including:

  - Global security
  - Authentication mechanisms (directories and user registries)
  - Authorization policies and providers, including Java Authentication and Authorization Service (JAAS)
  - Trust association interceptors
  - Single signon
  - Common Secure Interoperability Version 2 (CSIv2)

© Copyright IBM Corp. 2004

- Secure Sockets Layer (SSL)
- Java 2 Security manager
- Security attribute propagation

**Chapter 14, “Learn about WebSphere applications,” on page 501**
This provides security instructions that are specific to the various types of applications, such as Web applications or Web services.

**“Tuning security” on page 953**
Enabling security decreases performance. This describes considerations for increasing performance.

**Chapter 16, “Troubleshooting security configurations,” on page 955**
This describes how to troubleshoot errors related to security.

## Overview of securing applications and their environment

This topic provides links to conceptual overviews of securing applications and the application serving environment.

**“What is new for security specialists” on page 3**

This topic provides an overview of new and changed features in security.

**WebSphere security architecture**

This Education on Demand presentation provides an overview of the security architecture. Additional presentations are available that focus on the following concepts:

- CSIv2 security overview
- JACC overview
- JAAS client overview
- Resource security overview

**Introduction: Security**

This topic describes how IBM WebSphere Application Server provides security infrastructure and mechanisms to protect sensitive Java 2 Platform, Enterprise Edition (J2EE) resources and administrative resources and to address enterprise end-to-end security requirements on authentication, resource access control, data integrity, confidentiality, privacy, and secure interoperability.

**Chapter 4, “Integrating IBM WebSphere Application Server security with existing security systems,” on page 15**

This topic describes how the product security features relate to the security features of the environment into which you have added application serving capability.

**Chapter 5, “Planning to secure your environment,” on page 23**

Several communication links from a browser on the Internet, through Web servers and product servers, to the enterprise data at the back-end. This topic examines some typical configuration and common security practices. WebSphere Application Server security is built on a layered security architecture. This section also examines the security protection offered by each security layer and common security practice for good quality of protection in end-to-end security.

**Tutorials**

Education on Demand offers:

- Secure WebSphere Bank application

### Samples

The Samples Gallery offers:

- **Login - Form Login**

  The Form Login Sample demonstrates a very simple example of how to use the WebSphere login facilities to implement and configure login applications. The Sample uses the J2EE form-based login technology to customize the look and feel of the login screens. It uses servlet filters to log the user information and the date information. The Sample finishes the session by using the form-based logout function, an IBM extension to the J2EE specification.
- **Login - JAAS Login**

  The JAAS Login Sample demonstrates how to use the Java Authentication and Authorization Service (JAAS) with WebSphere Application Server. The Sample uses server-side login with JAAS to authenticate a real user to the WebSphere security run time. Based upon a successful login, the WebSphere security run time uses the authenticated Subject to perform authorization checks on a protected stateless session enterprise bean. If the Sample runs successfully, it displays all the principals and public credentials of the authenticated user.

## What is new for security specialists

This topic highlights what is new or changed in Version 6 for users who are responsible for securing applications and the application serving environment.

The biggest improvement in security involves the set of supported specifications.

*External JACC provider support*

The Java Authorization Contract for Containers specification (JACC) version 1.0, introduced in WebSphere Application Server Version 6 and defined by Java 2 Platform, Enterprise Edition (J2EE) Version 1.4, defines a contract between J2EE containers and external authorization providers. Based on this specification, WebSphere Application Server enables you to plug in an external provider to make authorization decisions when you are accessing a J2EE resource. When you use this feature, WebSphere Application Server supports Tivoli Access Manager as the default JACC provider.

For more information, see “JACC providers” on page 336.

*Java 2 security manager*

WebSphere Application Server Version 6 provides you with greater control over the permission granted to applications for manipulating non-system threads. You can permit applications to manipulate non-system threads using the was.policy file. However, these thread control permissions are disabled, by default.

For more information, see“Configuring the was.policy file” on page 482.

*JCA 1.5 support*

WebSphere Application Server Version 6 supports the J2EE Connector Architecture (JCA) Version 1.5 specification, which provides new features such as the inbound resource adapter. For more information, see J2EE Connector Architecture resource adapters.

From a security perspective, Version 6 provides an enhanced custom principal and credential programming interface and custom mapping properties at the resource reference level. The custom JAAS LoginModule, which was developed for JCA principal and credential mapping for WebSphere Application Server Version 5.x, continues to be supported.

*SSL channel framework*

The Secure Sockets Layer channel framework incorporates the new IBMJSSE2 implementation and separates the security function of Java Secure Sockets Extension (JSSE) from the network communication function.

*Web authentication using the Java Authentication and Authorization Service programming model*

WebSphere Application Server Version 6 enables you to use the Java Authentication and Authorization Service (JAAS) programming model to perform Web authentication in your application code. To use this function, you must create your own JAAS login configuration by cloning the WEB_INBOUND login configuration and define a cookie=true login option. After a successful login using your login configuration, the Web login session is tracked by single signon (SSO) token cookies. This option replaces the SSOAuthenticator interface, which was deprecated in WebSphere Application Server Version 4.

For more information, see “Java Authentication and Authorization Service authorization” on page 240.

*Web services security*

WebSphere Application Server Version 6 increases the extensibility of Web services security by providing a pluggable architecture. The implementation in WebSphere Application Server includes many of the features described in the Organization for the Advancement of Structured Information Standards (OASIS) Web Services Security Version 1 standard. As part of this standard, WebSphere Application Server supports custom, pluggable tokens that are used for signing and encryption; pluggable signing and encryption algorithms; pluggable key locators for locating a key that is used for digital signature or encryption; signing or encrypting elements in a Simple Object Access Protocol (SOAP) message; and specifying the order of the signing or encryption processes.

## Enabling security for WSIF

The Web Services Invocation Framework (WSIF) interacts with a security manager in the following ways:

- WSIF runs in the Java 2 platform, Enterprise Edition (J2EE) security context without modification.
- When WSIF is run under a J2EE container, port implementations can use the security context to pass on security tokens or credentials as necessary.

- WSIF implementations can automatically convert J2EE security context into appropriate context for onward services.

For WSIF to interact effectively with the WebSphere Application Server security manager, enable the following permission in the `was.policy` file: **FilePermission** to load the WSDL. This permission is required when a WSDL file is referred to using the `file://` protocol.

# Chapter 2. How do I secure applications and their environments?

- Develop and deploy secure applications
- Secure the application hosting environment
- Troubleshoot security

**Legend for ″How do I?...″ links**

| Documentation | Show me | Tell me | Guide me | Teach me |
|---|---|---|---|---|
| Refer to the detailed steps and reference | Watch a brief multimedia demonstration | View the presentation for an overview | Be led through the console pages | Perform the tutorial with sample code |
| **Approximate time:** Varies | **Approximate time:** 3 to 5 minutes | **Approximate time:** 10 minutes+ | **Approximate time:** 1/2 hour+ | **Approximate time:** 1 hour+ |

### Develop and deploy secure applications

These tasks involve securing your applications during development (optional, programmatic security), assembly (declarative security), and after deploying them on the application server.

------------------------------------------------------------------------

**Secure Web applications: Authentication and authorization**

Most of the security for an application is configured during the assembly stage. The security configured during the assembly stage is called declarative security because the security is declared or defined in the deployment descriptors. The declarative security is enforced by the security run time. For some applications, declarative security is not sufficient to express the security model of the application. For these applications, you can use programmatic security.

Documentation | Tell me

- Declarative
- Programmatic

Related documentation topics:

- Session security support
- Chapter 8, “Developing secured applications,” on page 51

------------------------------------------------------------------------

**Secure EJB applications: J2EE authorization**

Most of the security for an application is configured during the assembly stage. The security configured during the assembly stage is called declarative security because the security is declared or defined in the deployment descriptors. The declarative security is enforced by the security run time. For some applications, declarative security is not sufficient to express the security model of the application. For these applications, you can use programmatic security.

© Copyright IBM Corp. 2004

Documentation | | Tell me
- Declarative
- Programmatic

--------------------------------------------------------------------------

**Use Web services security (WS-Security)**

Use any of many methods to integrate message-level security into an application serving environment. Web services security for WebSphere Application Server is based on standards included in the Web services security (WS-Security) specification. These standards address how to provide protection for messages exchanged in a Web service environment. The specification defines the core facilities for protecting the integrity and confidentiality of a message and provides mechanisms for associating security-related claims with the message.

Documentation | Show me | Tell me

--------------------------------------------------------------------------

**Enable Java 2 security**

Java 2 security is disabled by default, but is enabled automatically when global security is enabled. Whether you use it is independent of your decision to use J2EE role-based authorization. It provides an extra level of access control protection on top of the J2EE role-based authorization. It particularly addresses the protection of system resources and APIs.

Documentation | | Tell me
- Console
- Scripting

--------------------------------------------------------------------------

**Develop JAAS clients**

If you write a login module that adds information to the Subject of a system login, refer to this topic for the main Java Authentication and Authorization Service (JAAS) plug in points for configuring system logins.

Documentation | | Tell me

--------------------------------------------------------------------------

**Enable resource security (overview)**

Applications access many resources for data access, messaging, mail, and other purposes.

Tell me | | Teach me

--------------------------------------------------------------------------

**Enable resource security: J2C and JDBC data sources**

Secure the Java DataBase Connectivity (JDBC) data sources and Java 2 Connector (J2C) resources used by applications to access data.

Show me | Tell me

Related documentation topics:

- "Security of lookups with component managed authentication" on page 938
- "JavaMail security permissions best practices" on page 945

--------------------------------------------------------------------------

### Enable resource security: JMS resources

Secure the Java Message Service (JMS) resources used by applications to obtain messaging support.

| Documentation | Tell me |
| --- | --- |

--------------------------------------------------------------------------

## Secure the application hosting environment

The counterpart of secure your applications, before and after deployment, is to secure the server hosting environment into which the applications are deployed.

--------------------------------------------------------------------------

### Secure the administrative environment

Use the administrative console to assign users to administrative roles.

| Documentation | Tell me | Guide me |
| --- | --- | --- |
| | • Security for system administrator<br>• Securing administrative environment | |

Related documentation topics:

- "Securing your environment before installation" on page 33 (installing with proper authority)
- "Securing your environment after installation" on page 34 (passwords and such)

--------------------------------------------------------------------------

### Configure security with wsadmin scripting (overview)

Scripting is a non-graphical alternative that you can use to configure and manage WebSphere Application Server. The WebSphere Application Server wsadmin tool provides the ability to run scripts. The wsadmin tool supports a full range of product administrative activities.

| Documentation |
| --- |

--------------------------------------------------------------------------

### Configure global security

Configure global security, which applies to all applications running in the environment and determines whether security is used at all, the type of registry against which authentication takes place, and other values, many of which act as defaults.

| Documentation | | Tell me | Guide me |
|---|---|---|---|
| • Console<br>• Scripting | | | |

--------------------------------------------------------------------------

### Authenticate users with the local operating system user registry

Configure the product to authenticate users against the local operating system user registry. The product provides and supports the implementation for Windows operating system registries, AIX, Solaris and multiple versions of Linux operating systems. The respective operating system APIs are called by the product processes (servers) for authenticating a user and other security-related tasks (for example, getting user or group information).

| Documentation | Show me | Tell me | Guide me |
|---|---|---|---|
| | • SWAM<br>• LTPA | | |

--------------------------------------------------------------------------

### Authenticate users with an LDAP user registry

Configure the product to authenticate users against a Lightweight Directory Access Protocol (LDAP) user registry. The product security provides and supports implementation of most major LDAP directory servers, which can act as the repository for user and group information. These LDAP servers are called by the product processes (servers) for authenticating a user and other security-related tasks (for example, getting user or group information). This support is provided by using different user and group filters to obtain the user and group information. These filters have default values that you can modify to fit your needs. The custom LDAP feature enables you to use any other LDAP server (which is not in the product supported list of LDAP servers) for its user registry by using the appropriate filters.

| Documentation | Show me | Tell me | Guide me |
|---|---|---|---|

--------------------------------------------------------------------------

### Authenticate with a custom user registry

After you have implemented and built the UserRegistry interface, you can configure the product to use your custom user registry to authenticate users.

| Documentation | Show me | Tell me | Guide me |
|---|---|---|---|

--------------------------------------------------------------------------

### Set up Single Sign On (SSO)

With single signon (SSO) support, Web users can authenticate once when accessing Web resources across multiple WebSphere Application Servers. Form login mechanisms for Web applications require that SSO is enabled.

| Documentation | | | |
|---|---|---|---|

--------------------------------------------------------------------------

### Set up Secure Sockets Layer (SSL) between remote servers or clients and servers

Secure Sockets Layer (SSL) is used by multiple components within WebSphere Application Server to provide trust and privacy.

Documentation Show me Tell me

Related documentation topics:

- “Accessing secure resources using SSL and applet clients” on page 503

--------------------------------------------------------------------------

### Set up CSIv2

Configure Common Secure Interoperability Version 2 (CSIv2) features including SSL client certificate authentication, message layer authentication, identity assertion, and security attribute propagation.

Documentation Tell me

--------------------------------------------------------------------------

### Configure an authorization provider (JACC)

Configure the product to use an external security provider you have set up to work with WebSphere Application Server that can support Java 2 Platform, Enterprise Edition (J2EE) authorization based on the JACC specification.

Documentation Tell me

--------------------------------------------------------------------------

## Troubleshoot security problems

Troubleshoot several types of problems related to enabling or configuring security.

--------------------------------------------------------------------------

### Troubleshoot the security subsystem

Troubleshoot several types of problems related to enabling or configuring security.

Documentation

--------------------------------------------------------------------------

# Chapter 3. Securing applications and their environment

WebSphere Application Server supports the J2EE model for creating, assembling, securing, and deploying applications. This article provides a high-level description of what is involved in securing resources in a J2EE environment. Applications are often created, assembled and deployed in different phases and by different teams.

Consult the J2EE specifications for complete details.

1. Plan to secure your applications and environment. For more information, see Chapter 5, “Planning to secure your environment,” on page 23. Complete this step before you install the WebSphere Application Server.
2. Consider pre-installation and post-installation requirements. For more information, see Chapter 6, “Implementing security considerations at installation time,” on page 33. For example, during this step, you learn how to protect security configurations after you install the product.
3. Migrate your existing security systems. For more information, see Chapter 7, “Migrating security configurations from previous releases,” on page 39.
4. Develop secured applications. For more information, see Chapter 8, “Developing secured applications,” on page 51.
5. Assemble secured applications. For more information, see Chapter 9, “Assembling secured applications,” on page 115. Development tools, such as the Assembling applications are used to assemble J2EE modules and to set the attributes in the deployment descriptors.

   Most of the steps in assembling J2EE applications involve deployment descriptors; deployment descriptors play a central role in application security in a J2EE environment.

   Application assemblers combine J2EE modules, resolve references between them, and create from them a single deployment unit, typically an Enterprise Archive (EAR) file. Component providers and application assemblers can be represented by the same person but do not have to be.
6. Deploy secured applications. For more information, see Chapter 10, “Deploying secured applications,” on page 127.

   Deployer link entities referred to in an enterprise application are mapped to the runtime environment. The deployer:
   - Maps actual users and groups to application roles
   - Installs the enterprise application into the environment
   - Makes the final adjustments needed to run the application
7. Test secured applications. For more information, see Chapter 11, “Testing security,” on page 139.
8. Manage security configurations. For more information, see Chapter 12, “Administering security,” on page 141.
9. Improve performance by tuning security configurations. For more information, see Chapter 15, “Tuning security configurations,” on page 949.
10. Troubleshoot security configurations. For more information, see Chapter 16, “Troubleshooting security configurations,” on page 955.

Your applications and production environment are secured.

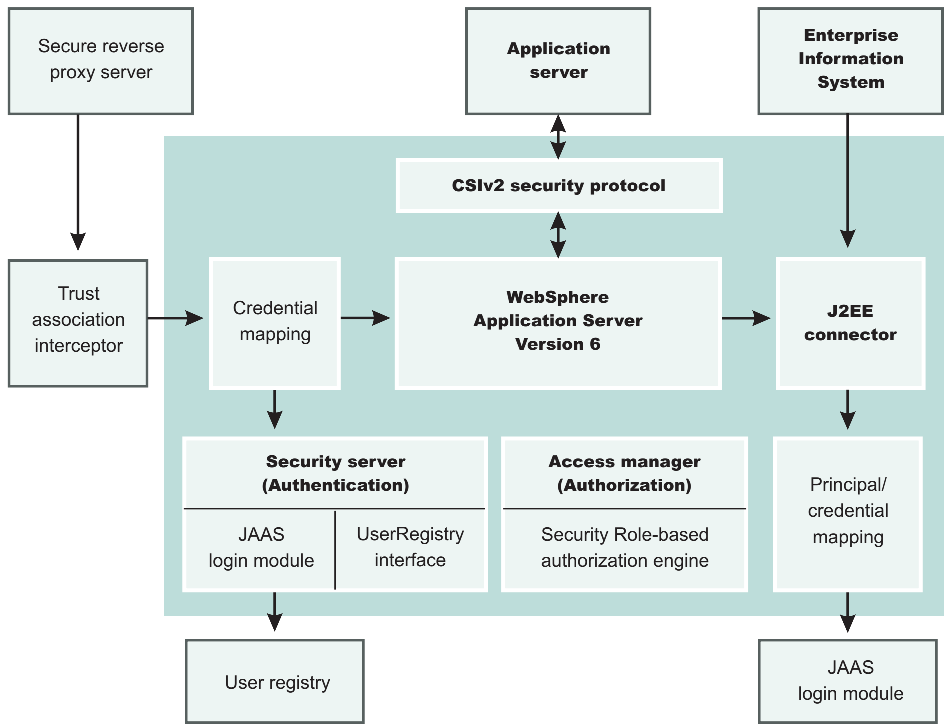
See “Security: Resources for learning” on page 21 for more information on the WebSphere Application Server security architecture.

© Copyright IBM Corp. 2004

# Chapter 4. Integrating IBM WebSphere Application Server security with existing security systems

WebSphere Application Server plays an integral part of the multiple-tier enterprise computing framework. WebSphere Application Server adopts the open architecture paradigm and provides many plug-in points to integrate with enterprise software components to provide end-to-end security. WebSphere Application Server plug-in points are based on standard J2EE specifications wherever applicable. WebSphere Application Server is actively involved in various standard bodies to externalize and to standardize plug-in interfaces.

In the following example, several typical multiple-tier enterprise network configurations are discussed. In each case, various WebSphere Application Server plug-in points are used to integrate with other business components. The discussion starts with a basic multiple-tier enterprise network configuration:




A list of terms used in this discussion follows:

**Protocol firewall**
: Prevents unauthorized access from the Internet to the demilitarized zone. The role of this node is to provide the Internet traffic access only on certain ports and to block other IP ports.

**WebSphere Application Server plug-in**
: Redirects all the requests for servlets and JSP pages. Also referred to in WebSphere Application Server literature as *Web server redirector* was introduced to separate Web server from application server. The advantage of using Web server redirector is that you can move an application server and all the application business logic behind the domain firewall.

**Domain firewall**
: Prevents unauthorized access from the demilitarized zone to an internal network. The role of this firewall is to allow the network traffic originating from the demilitarized zone and note from the Internet.

**Directory**
: Provides information about the users and their rights in the Web application. The information can

© Copyright IBM Corp. 2004

contain user IDs, passwords, certificates, access groups, and so forth. This node supplies the information to the security services like authentication and authorization service.

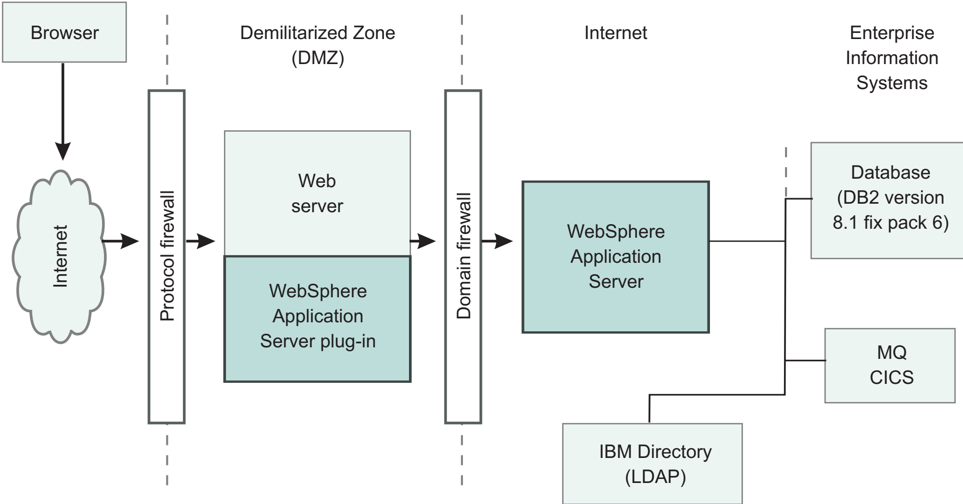
**Enterprise information system**
Represents existing enterprise applications and business data in back-end databases.

WebSphere Application Server provides the infrastructure to run application business logic and communicate with internal back-end systems and databases Web applications and enterprise beans can access. WebSphere Application Server has a built in HTTPS server that can accept client requests. A typical configuration, however, places WebSphere Application Server behind the domain firewall for better protection. A WebSphere Application Server plug-in to Web server configuration can redirect Web requests to WebSphere Application Server. WebSphere Application Server provides plug-ins for many popular Web servers.

You can configure WebSphere Application Server and the Web server plug-in to communicate through secure SSL channels. You can configure a WebSphere Application Server HTTP server to open communication channels only with a restricted set of Web server plug-ins. You can configure the HTTP server to require client certificate authentication with self-signed certificates and to trust only the signer certificate.

For more information, refer to

For instructions on how to generate self-signed certificates and how to set up secure communications channels between an HTTP server and the WebSphere Application Server plug-in, refer to Configuring IHS plug-in and the Internal Web Server for SSL and Configuring IHS for SSL Mutual Authentication.



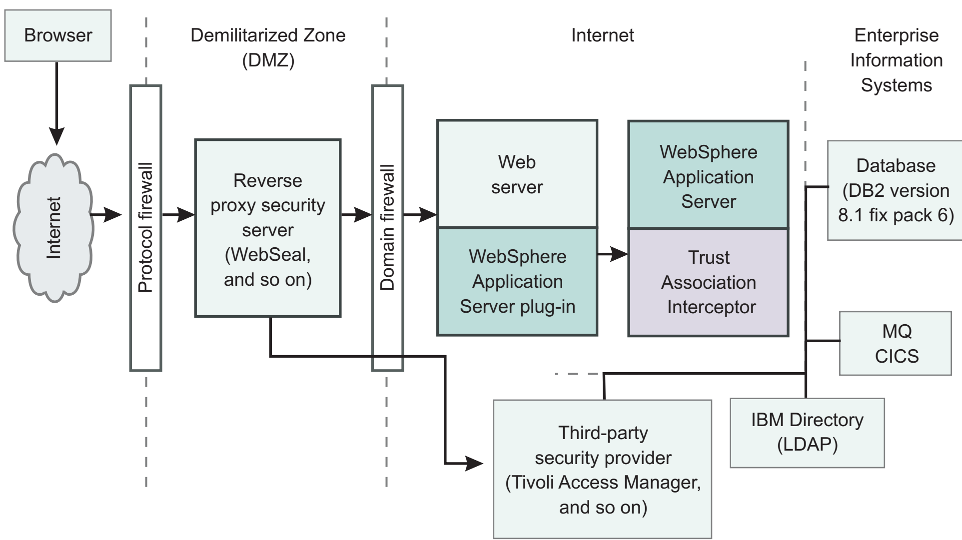

The WebSphere Application Server plug-in routes HTTP requests according to the virtual host and port configuration and the URL pattern matching. Client authentication and finer grained access control are handled by WebSphere Application Server behind the firewall.

In cases where the Web server can contain sensitive data and direct access is not desirable, the following configuration uses Tivoli WebSEAL to shield a Web server from unauthorized requests. WebSEAL is a Reverse Proxy Security Server (RPSS) that uses Tivoli Access Manager to perform coarse-grained access control to filter out unauthorized requests before they reach the domain firewall. WebSEAL uses Tivoli Access Manager to perform access control as illustrated in the picture. WebSphere Application Server supports various user registry implementations through the pluggable user registry interface.

WebSphere Application Server ships a Local OS user registry implementation for Windows, AIX, AS/400, and Lightweight Directory Access Protocol (LDAP).

WebSphere Application Server also supports users in developing their own custom registry and plug-in through the pluggable user registry interface. When integrated with a third party security provider, WebSphere Application Server can share the user registry with the third-party security provider. In the particular example of integrating with WebSEAL, you can configure WebSphere Application Server to use the LDAP user registry, which can be shared with WebSEAL and Tivoli Access Manager. Moreover, you can configure WebSphere Application Server to use the Light Weight Third Party (LTPA) authentication mechanism, which supports the Trust Association Interceptor plug-in point.

Basically, the RPSS performs authentication and adds proper authentication data into the request header and then redirects the request to Web server. A trust relationship is formed between an RPSS and WebSphere Application Server, and the RPSS can assert client identity to WebSphere Application Server to achieve single signon (SSO) between RPSS and WebSphere Application Server. When the request is forward to WebSphere Application Server, WebSphere Application Server uses the TAI plug-in for the particular RPSS server to evaluate the trust relationship and to extract the authenticated client identity. WebSphere Application Server then maps the client identity to a WebSphere Application Server security credential. For instructions on setting up a trust association interceptor, refer to Trust associations, Configuring trust association interceptors.




When configured to use the LDAP user registry, WebSphere Application Server uses LDAP to perform authentication. The client ID and password are passed from WebSphere Application Server to the LDAP server. You can configure WebSphere Application Server to set up an SSL connection to LDAP so that passwords are not passed in plain text. To set up an SSL connection from WebSphere Application Server to the LDAP server, refer to Configuring SSL for the LDAP client. WebSphere Application Server Version 5 supports the J2EE Connector Architecture (JCA). The connector architecture defines a standard interface for WebSphere Application Server to connect to heterogeneous enterprise information systems (EIS). Examples of EIS includes database systems, transaction processing such as CICS, and messaging such as Message Queue (MQ). The EIS implementation can perform authentication and access control to protect business data and resources. Resource Adapters authenticate EIS. The authentication data can be provided either by application code or by WebSphere Application Server. WebSphere Application Server provides a principal mapping plug-in point. A principal mapping module plug-in maps the authenticated client principal to a password credential, (that is, user ID and password, for the EIS security domain). WebSphere Application Server ships a default principal mapping module, which maps any authenticated client principal to a configured pair of user IDs and passwords.

Each connector can be configured to use a different set of IDs and passwords. For a description on how to configure JCA principal mapping user IDs and passwords, refer to Managing J2C Authentication Data Entries. A principal mapping module is a special purpose Java Authentication and Authorization Service

(JAAS) login module. You can develop your own principal mapping module to fit your particular business application environment. For detailed steps on developing and configuring a custom principal mapping module, refer to the articles, Developing your own Java 2 security mapping module underneath JAAS Programmatic Login and Managing Java Authentication and Authorization Service (JAAS) Login Configuration.

**Security and WebSphere MQseries**

It is important to note that security logging information on UNIX systems is not protected because of the world-writeable files in the `/var` file system of MQseries. MQseries ships the following files with its product:
- -rw-rw-rw- /var/mqm/errors/AMQERR01.LOG
- -rw-rw-rw- /var/mqm/errors/AMQERR02.LOG
- -rw-rw-rw- /var/mqm/errors/AMQERR03.LOG

The previously mentioned files are world-writeable and enable any users on the system to fill up the `/var` file system where all the security logging information is stored. This leaves the security information unprotected because anyone can access the logging information without being tracked.

To work around this problem, create a file system for the embedded messaging component working data on UNIX. Before you install the embedded messaging component of WebSphere Application Server on UNIX platforms, consider creating and mounting a journalized file system called `/var/mqm`. Use a partition strategy with a separate volume for the WebSphere MQ data. This means that other system activity is not affected if a large amount of WebSphere MQ work builds up.

To determine the size of the `/var/mqm` file system for a server installation, consider the following:
- Maximum number of messages in the system at one time
- Contingency for message buildups, if there is a system problem
- Average size of the message data, plus 500 bytes for the message header
- Number of queues
- Size of log files and error messages

Allow 50MB as a minimum for a WebSphere MQ server. You need less space in the `/var/mqm` file system for a WebSphere MQ client (typically 15MB).

## Interoperability issues for security

To have interoperability of Security Authentication Service (SAS) between C++ and WebSphere Application Server, use the Common Secure Interoperability Version 2 (CSIv2) authentication protocol over Remote Method Invocation over the Internet Inter-ORB Protocol (RMI-IIOP).

## Interoperating with a C++ common object request broker architecture client

You can achieve interoperability of Security Authentication Service between the C++ Common Object Request Broker Architecture (CORBA) client and WebSphere Application Server using Common Secure Interoperability Version 2 (CSIv2) authentication protocol over Remote Method Invocation over the Internet Inter-ORB Protocol (RMI-IIOP). The CSIv2 security service protocol has authentication, attribute and transport layers. Among the three layers, transport authentication is conceptually simple, however, cryptographically based transport authentication is the strongest. WebSphere Application Server Enterprise has implemented the transport authentication layer, so that C++ secure CORBA clients can use it effectively in making CORBA clients and protected enterprise bean resources work together.

**Security authentication from non-Java based C++ client to enterprise beans**. WebSphere Application Server supports security in the CORBA C++ client to access protected enterprise beans. If configured,

C++ CORBA clients can access protected enterprise bean methods using client certificate to achieve mutual authentication on WebSphere Application Server Enterprise applications.

To support the C++ CORBA client in accessing protected enterprise beans:

- Create an environment file for the client, such as current.env. Set the variables listed below (security_sslKeyring, client_protocol_user, client_protocol_password) in the file.
- Point to the environment file using the fully qualified path name through the environment variable WAS_CONFIG_FILE. For example, in the test shell script test.sh, export WAS_CONFIG_FILE=/WebSphere/V5R0M0/AppServer/bin/current.env.

| C++ security setting | Description |
| --- | --- |
| client_protocol_password | Specifies the password for the user ID. |
| client_protocol_user | Specifies the user ID to be authenticated at the target server. |
| security_sslKeyring | Specifies the name of the RACF keyring the client will use. The keyring must be defined under the user ID that is issuing the command to run the client. |

To support the C++ CORBA client in accessing protected enterprise beans:

1. Obtain a valid certificate to represent the client and export its public key to the target enterprise bean server.

   A valid certificate is needed to represent the C++ client. Request a certificate from the certificate authority (CA) or create a self-signed certificate for testing purposes.

   Use the Key Management Utility from the Global Security Kit (GSKit) to extract the public key from the personal certificate and save it in the `.arm` format. For details, see the related information about how to extract the personal certificate of the public key.
2. Prepare a truststore file for WebSphere Application Server.

   Add the extracted client public key in the `.arm` file from the client to the server key truststore file. The server can now authenticate the client.

   **Note:** This is done by invoking the Key Management Utility through ikeyman.bat or ikeyman.sh from WebSphere Application Server installation.

   For details, see the article on Adding truststore files.
3. Configure WebSphere Application Server to support SSL as the authentication mechanism.
   a. Start the administrative console.
   b. Locate the application server that has the target enterprise bean deployed and configure it to use SSL client certificate authentication.

      If it is a base installation, complete the following steps:
      1) Click **Security > Global security**. Under Authentication, click **Authentication protocol > CSIv2 inbound authentication**. Select **Supported** for the Basic authentication and Client certificate authentication options. Leave the rest of the options as defaults.
      2) Click **Security > Global security**. Under Authentication, click **Authentication protocol > CSIv2 inbound transport** and verify that the **SSL-supported** option is selected.

      If it is a Network Deployment setting, complete the following steps:
      1) Click **Server > Application Server >***server_name_where_the_EJB_resides*. Under security, click **Server security**. Under Additional properties, click **CSI inbound authentication**. Select **Supported** for the Basic authentication and Client certificate authentication options. Leave the rest of the options as defaults.

2) Click **Server > Application Server >***server_name_where_the_EJB_resides*. Under security, click **Server security**. Under Additional properties, click **CSI inbound transport**. Verify that the **SSL-Supported** option is selected.

For details, see the security articles Configuring CSIv2 inbound authentication and Configuring CSIv2 inbound transport.

c. Restart the application server.

The WebSphere Application Server is ready to take a C++ CORBA security client and a mutually authenticated server and client by using SSL in the transport layer.

4. Configure the C++ CORBA client to use a certificate in performing the mutual authentication.

Client users are accustomed to using property files in their applications because they are helpful in specifying configuration settings. The following list presents important C++ security settings:

| C++ security setting | Description |
|---|---|
| com.ibm.CORBA.bootstrapHostName=ricebella.austin.ibm.com | Specifies the target host name. |
| com.ibm.CORBA.securityEnabled=yes | Enables security. |
| com.ibm.CSI.performTLClientAuthenticationSupported=yes | Ensures client is supporting mutual authentication by certificate |
| com.ibm.CSI.performTransportAssocSSLTLSSupported=yes | Ensures SSL is used, not TCP/IP |
| com.ibm.ssl.keyFile=C:/ricebella/etc/DummyKeyRingFile.KDB | Specifies which key database file to use. |
| com.ibm.ssl.keyPassword=WebAS | Specifies the password for opening the key database file. WebSphere Application Server supports a utility called `PasswordEncode4cpp` to encode the plain password. |
| com.ibm.CORBA.translationEnabled=1 | Enables the valueType conversion. |

To use the property files in running a C++ client, an environment variable WASPROPS, is used to indicate where a property file or a list of property files exist.

For the complete set of C++ client properties, see the sample property file `scclient.props`, which is shipped with the product located in the *install_root*\profiles*profile_name*\etc directory.

## Interoperating with previous product versions

IBM WebSphere Application Server, Version 5.x or later interoperates with the previous product versions (such as Version 4 and Version 3.5). Interoperability is achieved only when the Lightweight Third Party Authentication (LTPA) authentication mechanism and Lightweight Directory Access Protocol (LDAP) user registry are used. Credentials derived from Simple WebSphere Authentication Mechanisms (SWAM) are not forwardable.

1. Enable security with the LTPA authentication mechanism and the LDAP user registry. Make sure that the same LDAP user registry is shared by all the product versions.
2. Extract and add Version 5 server certificates into the server key ring file of the previous version.
   a. Open the Version 5 server key ring file using the key management utility (iKeyman) and extract the server certificate to a file.
   b. Open the server key ring of the previous product version, using the key management utility and add the certificate extracted from product Version 5.
3. Extract and add Version 5 server certificates into the server key ring file of the previous version.
   a. Open the Version 5 server key ring file using the key management utility (iKeyman) and extract the server certificate to a file.
   b. Open the server key ring of the previous product version, using the key management utility and add the certificate extracted from product Version 5.
4. Extract and add Version 5 trust certificates into the trust key ring file of the previous product version.

a. Open the Version 5 trust key ring file using the key management utility and extract the trust certificate to a file.
b. Open the trust key ring file of the previous product version using the key management utility and add the certificate extracted from Version 5.

5. If single signon (SSO) is enabled, export keys from the Version 5 product and import them into the previous product version. The Version 4 product requires the fix, PQ61779, and the Version 3.5 product requires the fix, PQ59667, for SSO to function.
6. Verify that the application uses the correct JNDI name. In Version 5, the enterprise beans are registered with long JNDI names like, `(top)/nodes/`*`node_name`*`/servers/`*`server_name`*`/HelloHome`. Whereas in previous releases, enterprise beans are registered under a root like, `(top)/HelloHome`. Therefore, EJB applications from previous versions perform a lookup on the Version 5 enterprise beans.

   You can also create EJB name bindings in Version 5 that are compatible with the previous version. To create an EJB name binding at the root Version 5, start the administrative console and click **Environment** > **Naming** > **Naming Space Bindings** > **New** > **EJB** > **Next**. Complete all the fields and enter a short name (for example, `-HelloHome`) as the JNDI Name. Click **Next** and **Finish**.
7. Stop and restart all the servers.
8. Make sure that the correct naming bootstrap port is used to perform naming lookup. In previous product versions, the naming bootstrap port is **900**. In Version 5, the bootstrap port is **2809**.

## Security: Resources for learning

Use the following links to find relevant supplemental information about Securing applications and their environment. The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

These links are provided for convenience. Often, the information is not specific to the IBM WebSphere Application Server product, but is useful all or in part for understanding the product. When possible, links are provided to technical papers and Redbooks that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas.

View links to additional information about:
- "Planning, business scenarios and IT architecture"
- "Programming model and decisions"
- "Programming specifications" on page 22
- "Administration" on page 22

**Planning, business scenarios and IT architecture**
- WebSphere Application Server Library
- WebSphere Application Server Support
- WebSphere Application Server Version 5 Security Redbook
- Accessing the Samples (Samples Gallery)

  The technology sample in the WebSphere Application Server Samples Gallery contains several security-related samples including the form login sample and the Java Authentication and Authorization Service (JAAS) login sample.
- WebSphere Application Server security: Presentation series

**Programming model and decisions**
- JSSE Documentation.

  Refer to the http://www.ibm.com/developerworks/java/jdk/security/jsseDocs.zip file for the Javadoc of the APIs, JSSE Reference Guide, and JSSE samples.
- iKeyman documentation.

  Look in the http://www.ibm.com/developerworks/java/jdk/security/iKeymanDocs.zip file for the Secure Sockets Layer (SSL) Introduction and iKeyman documentation.

- JCE documentation.
  - For the JCA spec and JCE API usage refer to the http://www.ibm.com/developerworks/java/jdk/security/jceDocs.zip file.
  - For JCE sample applications refer to the http://www.ibm.com/developerworks/java/jdk/security/jceDocs.zip file.
  - For Java Cryptography Architecture Reference refer to the http://www.ibm.com/developerworks/java/jdk/security/jceDocs.zip file.
  - For how to implement a JCE provider refer to the http://www.ibm.com/developerworks/java/jdk/security/jceDocs.zip file.
  - For the Javadoc of JCE APIs refer to the http://www.ibm.com/developerworks/java/jdk/security/jceDocs.zip file.
  - For the 1.4.2 release of the IBM developer kit for the Java platform refer to the http://www-106.ibm.com/developerworks/java/library/j-ibmsecurity.html file.

**Programming specifications**

- J2EE Specifications
- EJB Specifications
- Servlet Specifications
- Common Secure Interoperability Version 2 (CSIv2) Specification
- JAAS Specification.

  For programming and usage in JAAS, refer to the specification located at http://www.ibm.com/developerworks/java/jdk/security/ and scroll down to find the JAAS documentation for your platform. This document contains the following when unpacked:
  - login.html - LoginModule Developer's Guide
  - api.html - Developer's Guide (JAAS JavaDoc)
  - HelloWorld.tar - Sample JAAS Application
- Java 2 Platform, Standard Edition, v 1.4.2 API Specification
- Java Authorization Contract for Containers (JSR 115) Specification

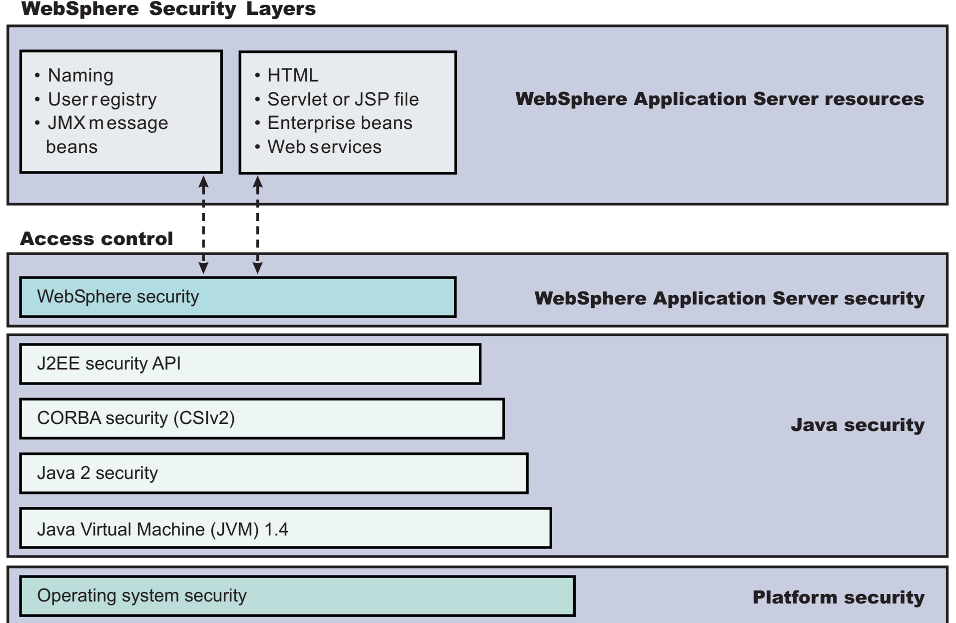
**Administration**

- WebSphere Application Server Version 4.0 Security Redbook: WebSphere Security Model.
- IBM HTTP Server Support and Documentation
- IBM Directory Server Support and Documentation
- IBM developer kits

  This Web site provides access to the IBM developer kits provided by the IBM Centre for Java Technology Development. Using this Web site, you can find various security and diagnostic information including information on the Federal Information Processing Standard, Java Version 1.4.1, Java Version 1.4.2, the iKeyman tool, and the Public Key Cryptography Standards (PKCS).
- IBM cryptographic hardware devices
- Supported hardware, software and APIs prerequisite Web site
- WebSphere education on demand: Enabling security best practice tutorials
- http://www.redbooks.ibm.com/abstracts/sg244986.html?Open

# Chapter 5. Planning to secure your environment

There are several communication links from a browser on the Internet, through Web servers and product servers, to the enterprise data at the back-end. This section examines some typical configurations and common security practices. WebSphere Application Server security is built on a layered security architecture as showed in the following figure. This section also examines the security protection that is offered by each security layer and common security practice for good quality of protection in end-to-end security. The following figure illustrates the building blocks that comprise the operating environment for security within WebSphere Application Server:




- **Operating System Security** -

  The security infrastructure of the underlying operating system provides certain security services for WebSphere Application Server. These services include the file system security support that secure sensitive files in the product installation for WebSphere Application Server. The system administrator can configure the product to obtain authentication information directly from the operating system user registry.
- **Network Security** - The Network Security layers provide transport level authentication and message integrity and encryption. You can configure the communication between separate application servers to use Secure Sockets Layer (SSL) and HTTPS. Additionally, you can use IP Security and Virtual Private Network (VPN) for added message protection.
- **JVM 1.4** - The JVM security model provides a layer of security above the operating system layer.
- **Java 2 Security** - The Java 2 Security model offers fine-grained access control to system resources including file system, system property, socket connection, threading, class loading, and so on. Application code must explicitly grant the required permission to access a protected resource.
- **OMG CSIv2 Security** - Any calls made among secure Object Request Brokers (ORB) are invoked over the Common Security Interoperability Version 2 (CSIv2) security protocol that sets up the security context and the necessary quality of protection. After the session is established, the call is passed up to the enterprise bean layer. For backward compatibility, WebSphere Application Server supports the Secure Authentication Service (SAS) security protocol, which was used in prior releases of WebSphere Application Server and other IBM products.
- **J2EE Security** - The security collaborator enforces Java 2 Platform, Enterprise Edition (J2EE)-based security policies and supports J2EE security APIs.

© Copyright IBM Corp. 2004

- **WebSphere Security** - WebSphere Application Server security enforces security policies and services in a unified manner on access to Web resources, enterprise beans, and JMX administrative resources. It consists of WebSphere Application Server security technologies and features to support the needs of a secure enterprise environment.

**WebSphere Application Server Network Deployment installation**: The following figure shows a typical multiple-tier business computing environment for a WebSphere Application Server Network Deployment installation.

**Important:** There is a node agent instance on every computer node.

Each product application server consists of a Web container, an EJB container, and the administrative subsystem. The WebSphere Application Server deployment manager contains only WebSphere administrative code and the administrative console. The administrative console is a special J2EE Web application that provides the interface for performing administrative functions. WebSphere Application Server configuration data is stored in XML descriptor files, which must be protected by operating system security. Passwords and other sensitive configuration data can be modified using the administrative console. However, you must protect these passwords and sensitive data. For more information, see "Protecting plain text passwords" on page 35.

The administrative console Web application has a setup data constraint that requires the administrative console servlets and JSP files to be accessed only through an SSL connection when global security is enabled.

After installation, the administrative console HTTPS port is configured to use **DummyServerKeyFile.jks** and **DummyServerTrustFile.jks** with the default self-signed certificate. Using the dummy key and trust file certificate is not safe and you need to generate your own certificate to replace dummy ones immediately. It is more secure if you first enable global security and complete other configuration tasks after global security is enforced.

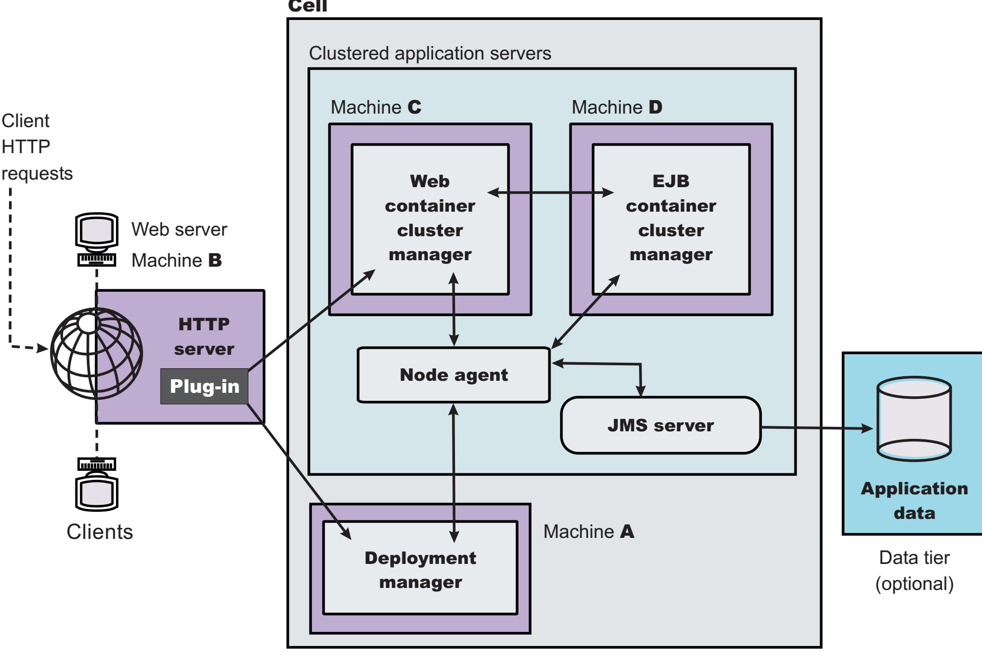



*Figure 1. Multiple-tier business computing environment.*

**Global and administrative security**:

WebSphere Application Servers interact with each other through CSIv2 and Secure Authentication Services (SAS) security protocols as well as HTTP and HTTPS protocols.

You can configure these protocols to use Secure Sockets Layer (SSL) when you enable WebSphere Application Server global security. The WebSphere Application Server administrative subsystem in every server uses Simple Object Access Protocol (SOAP) Java Management Extensions (JMX) connectors and Remote Method Invocation over the Internet Inter-ORB Protocol (RMI/IIOP) JMX connectors to pass administrative commands and configuration data. When global security is disabled, the SOAP JMX connector uses HTTP protocol and the RMI/IIOP connector uses the TCP/IP protocol. When global security is enabled, the SOAP JMX connector always uses HTTPS protocol. When global security is enabled, you can configure the RMI/IIOP JMX connector to either use SSL or to use TCP/IP. It is recommended that you enable global security and enable SSL to protect the sensitive configuration data.

Global security and administrative security configuration is at the cell level.

When global security is enabled, you can disable application security at each individual application server by clearing the **Enable global security** option on the global security panel. The Global security panel is accessed through the administrative console by clicking **Security > Global security**. Disabling application server security does not affect the administrative subsystem in that application server, which is controlled by the global security configuration only. Both administrative subsystem and application code in an application server share the optional per server security protocol configuration. For more information, see "Configuring server security" on page 148.

**Security for J2EE resources**: Security for J2EE resources is provided by the Web container and the EJB container. Each container provides two kinds of security: declarative security and programmatic security.

In declarative security, an application security structure includes data integrity and confidentiality, authentication requirements, security roles, and access control. Access control is expressed in a form that is external to the application. In particular, the deployment descriptor is the primary vehicle for declarative security in the J2EE platform. WebSphere Application Server maintains J2EE security policy, including information derived from the deployment descriptor and specified by deployers and administrators in a set of XML descriptor files. At run time, the container uses the security policy that is defined in the XML descriptor files to enforce data constraints and access control.

When declarative security alone is not sufficient to express the security model of an application, you might use "Programmatic login" on page 66 to make access decisions. When global security is enabled and application server security is not disabled at the server level, J2EE applications security is enforced. When the security policy is specified for a Web resource, the Web container performs access control when the resource is requested by a Web client. The Web container challenges the Web client for authentication data if none is present according to the specified authentication method, ensures the data constraints are met, and determines whether the authenticated user has the required security role. The Web security collaborator enforces role-based access control by using an access manager implementation. An access manager makes authorization decisions that are based on security policy derived from the deployment descriptor. An authenticated user principal can access the requested servlet or JavaServer Pages (JSP) file if it has one of the required security roles. Servlets and JSP pages can use the `HttpServletRequest` methods `isUserInRole` and `getUserPrincipal`.

When global security is enabled and application server security is not disabled, the EJB container enforces access control on EJB method invocation.

The authentication takes place regardless of whether method permission is defined for the specific EJB method. The EJB security collaborator enforces role-based access control by using an access manager implementation. An access manager makes authorization decisions that are based on security policy derived from the deployment descriptor. An authenticated user principal can access the requested EJB

method if it has one of the required security roles. EJB code can use the `EJBContext` methods `isCallerInRole` and `getCallerPrincipal`. Use the J2EE role-based access control to protect valuable business data from access by unauthorized users from both the Internet and the intranet. Refer to "Securing Web applications using an assembly tool" on page 118 and "Securing enterprise bean applications" on page 116.

**Role-based security**: WebSphere Application Server extends the security, role-based access control to administrative resources including the JMX system management subsystem, user registries, and JNDI name space. WebSphere administrative subsystem defines four administrative security roles:

**Monitor role**
: A monitor can view the configuration information and status, but cannot make any changes.

**Operator role**
: An operator can trigger run-time state changes, such as start an application server or stop an application, but cannot make configuration changes.

**Configurator role**
: A configurator can modify the configuration information, but cannot change the state of the run time.

**Administrator role**
: An operator as well as a configurator, which additionally can modify sensitive security configuration and security policy such as setting server ID and password, enable or disable global security and Java 2 security, and map users and groups to the administrator role.

A user with the configurator role can perform most administrative work including installing new applications and application servers. There are certain configuration tasks a configurator does not have sufficient authority to do when global security is enabled, including modifying a WebSphere Application Server identity and password, LTPA password and keys, and assigning users to administrative security roles.

Those sensitive configuration tasks require the administrative role because the server ID is mapped to the administrator role.

WebSphere Application Server administrative security is enforced when global security is enabled. It is recommended that WebSphere Application Server global security be enabled to protect administrative subsystem integrity. Application server security can be selectively disabled if there is no sensitive information to protect. For securing administrative security, refer to "Assigning users to administrator roles" on page 153 and "Assigning users and groups to roles" on page 128.

**Java 2 security permissions**: WebSphere Application Server uses the Java 2 security model to create a secure environment to run application code. Java 2 security provides a fine-grained and policy-based access control to protect system resources such as files, system properties, opening socket connections, loading libraries, and so on. The J2EE Version 1.4 specification defines a typical set of Java 2 security permissions that Web and EJB components expect to have. These permissions are shown in the following table.

*Table 1. J2EE security permissions set for Web components*

| Security Permission | Target | Action |
| --- | --- | --- |
| java.lang.RuntimePermission | loadLibrary | |
| java.lang.RuntimePermission | queuePrintJob | |
| java.net.SocketPermission | * | connect |
| java.io.FilePermission | * | read, write |
| java.util.PropertyPermission | * | read |

*Table 2. J2EE security permissions set for EJB components*

| Security Permission | Target | Action |
|---|---|---|
| java.lang.RuntimePermission | queuePrintJob | |
| java.net.SocketPermission | * | connect |
| java.util.PropertyPermission | * | read |

The WebSphere Application Server Java 2 security implementation is based on the J2EE Version 1.4 specification. The specification granted Web components read and write file access permission to any file in the file system, which might be too broad. The WebSphere Application Server default policy gives Web components read and write permission to the subdirectory and the subtree where the Web module is installed. The default Java 2 security policy for all Java virtual machines and WebSphere Application Server processes are contained in the following policy files:

**${java.home}/jre/lib/security/java.policy**
Used as the default policy for the Java virtual machine (JVM).

**${USER_INSTALL_ROOT}/properties/server.policy**
Used as the default policy for all product server processes

To simplify policy management, WebSphere Application Server policy is based on resource type rather than code base (location). The following files are the default policy files for WebSphere Application Server subsystem. These policy files, which are an extension of WebSphere Application Server run time and are referred to as *Service Provider Programming Interfaces (SPI)*, are shared by multiple J2EE applications:

**${WAS_INSTALL_ROOT}/profiles/***profile_name***/config/cells/***cell_name***/nodes/***node_name***/spi.policy**
Used for embedded resources defined in the `resources.xml` file, such as the Java Message Service (JMS), JavaMail, and JDBC drivers.

**${WAS_INSTALL_ROOT}/profiles/***profile_name***/config/cells/***cell_name***/nodes/***node_name***/library.policy**
Used by the shared library that is defined by the WebSphere Application Server administrative console.

**${WAS_INSTALL_ROOT}/profiles/***profile_name***/config/cells/***cell_name***/nodes/***node_name***/app.policy**
Used as the default policy for J2EE applications.

In general, applications should not require more permissions to run than those recommended by the J2EE specification to be portable among various application servers. However, some applications might require more permissions. WebSphere Application Server supports a per application policy file, `was.policy`, to be packaged together with each application from granting extra permissions to that application.

**Attention:** Grant extra permissions to an application after careful consideration because of the potential of compromising the system integrity.

WebSphere Application Server uses a permission filtering policy file to alert users when an application requires permissions that are on the filter list during application installation and causes the offended application installation to fail. For example, it is recommended that you not give the `java.lang.RuntimePermission` *exitVM* permission to an application so that application code cannot terminate WebSphere Application Server. The filtering policy is defined by the `filterMask` in `${WAS_INSTALL_ROOT}/profiles/`***profile_name***`/config/cells/`***cell_name***`/filter.policy`. Moreover, WebSphere Application Server also performs run-time permission filtering that is based on the run-time filtering policy to ensure that application code is not granted a permission that is considered harmful to system integrity.

WebSphere Application Server Version 4 supported Java 2 Security, but enforced only three permissions checking against `exitVM`, create and set the security manager. Other permission checking is disabled by default.

Therefore, many applications developed for prior releases of WebSphere Application Server might not be Java 2 Security ready. To migrate those applications to WebSphere Application Server Version 6 quickly, you might temporarily give those applications `java.security.AllPermission` in the `was.policy` file. It is recommended to test or make those applications Java 2 Security ready; for example, identity what extra permissions, if any, are required and to grant only those permissions to a particular application. Not granting applications `AllPermission` can certainly reduce the risk of compromising system integrity. For more information on migrating applications to WebSphere Application Server Version 6, refer to "Migrating Java 2 security policy" on page 494.

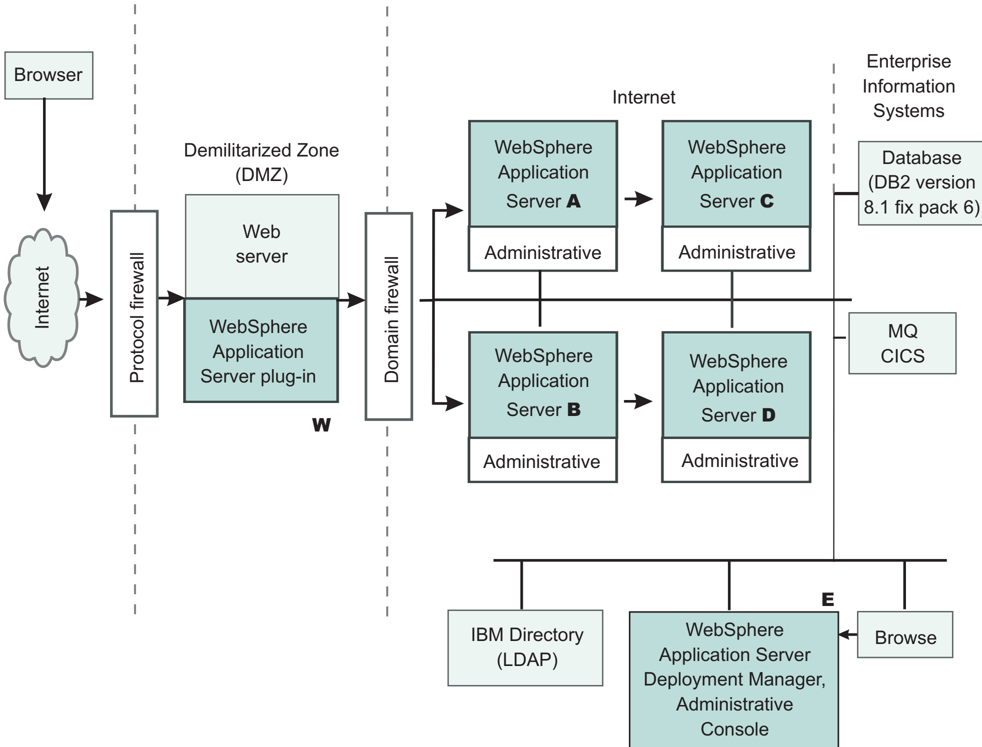
The WebSphere Application Server run time uses Java 2 Security to protect sensitive run-time functions; therefore, it is recommended that you enforce Java 2 security. Applications that are granted with `AllPermission` not only have access to sensitive system resources, but also WebSphere Application Server run-time resources and can potentially cause damage to both. In cases where an application can be trusted to be safe, WebSphere Application Server allows Java 2 Security to be disabled on a per application server basis. You can enforce Java 2 security by default in the security center and disable the per application server Java 2 Security flag to disable it at the particular application server.

When you specify the **Enable global security** and **Enable Java 2 Security** options on the Global security panel of the administrative console, the information, along with other sensitive configuration data, are stored in a set of XML configuration files. Both role-based access control and Java 2 Security permission-based access control are employed to protect the integrity of the configuration data. The example uses configuration data protection to illustrate how system integrity is maintained.
- When Java 2 security is enforced, the application code cannot access the WebSphere Application Server run-time classes that manage the configuration data unless it is granted the required WebSphere Application Server run-time permissions.
- When Java 2 security is enforced, application code cannot access the WebSphere Application Server configuration XML files unless it has been granted the required file read and write permission.
- The JMX administrative subsystem provides SOAP over HTTP or HTTPS and RMI/IIOP remote interface to enable application programs to extract and to modify configuration files and data. When global security is enabled, an application program can modify the WebSphere Application Server configuration if the application program has presented valid authentication data and the security identity has the required security roles.
- If a user can disable Java 2 security, then that user can modify the WebSphere Application Server configuration including the WebSphere Application Server security identity and authentication data along with other sensitive data. Only users with the administrator security role can disable Java 2 security.
- Because WebSphere Application Server security identity is given to the administrator role, only users with the administrator role can disable global security, to change server ID and password, and to map users and groups to administrative roles, and so on.

**Other Runtime resources**: Other WebSphere Application Server run time resources are protected by a similar mechanism as described previously. It is very important to enable WebSphere Application Server global security and to enforce Java 2 Security. J2EE Specification defines several authentication methods for Web components: HTTP Basic Authentication, Form-Based Authentication, and HTTPS Client Certificate Authentication. When you use client certificate login, it is more convenient for the browser client if the Web resources have integral or confidential data constraint. If a browser uses HTTP to access the Web resource, the Web container automatically redirects the browser to the HTTPS port. The CSIv2 security protocol also supports client certificate authentication. You can also use SSL client authentication to setup secure communication among a selected set of servers based on a trust relationship.

If you start from the WebSphere Application Server plug-in at the Web server, you can configure SSL mutual authentication between it and the WebSphere Application Server HTTPS server. When using a self-signed certificate, you can restrict the WebSphere Application Server plug-in to communicate with only the selected two WebSphere Application Server servers as shown in the following figure. Suppose you want to restrict the HTTPS server in WebSphere Application Server **A** and in WebSphere Application Server **B** to accept secure socket connections only from the WebSphere Application Server plug-in **W**. You can generate three self-signed certificates using the IKEYMAN and the certificate management utilities.

For example, use certificate **W** and trust certificate **A** and **B**. The HTTPS server of WebSphere Application Server **A** is configured to use certificate **A** and to trust certificate **W**. The HTTPS server of WebSphere Application Server **B** is configured to use certificate **B** and to trust certificate **W**. For more information on IKEYMAN, refer to “Starting the key management utility (iKeyman)” on page 446.




The trust relationship depicted in the previous picture is shown in the following table.

| Server | Key | Trust |
| --- | --- | --- |
| WebSphere Application Server plug-in | W | A, B |
| WebSphere Application Server A | A | W |
| WebSphere Application Server B | B | W |

When WebSphere Application Server is configured to use an Lightweight Directory Access Protocol (LDAP) user registry, you also can configure SSL with mutual authentication between every application server and the LDAP server with self-signed certificate so that a password is not passed in clear text from WebSphere Application Server to the LDAP server. In this example, the node agent processes are not discussed. Each node agent must communicate with application servers on the same node and with the Deployment Manager. Node agents also must communicate with LDAP servers when they are configured to use an LDAP user registry. It is reasonable to let the deployment manager and the node agents use the same certificate. Suppose application server **A** and **C** are on the same computer node. The Node agent on that node needs to have certificates **A** and **C** in its trust file. WebSphere Application Server does not provide a user registry configuration or management utility. In addition, it does not dictate the user registry password policy. It is recommended that you use the password policy recommended by your user registry, including the password length and expiration period.

1. Determine which versions of WebSphere Application Server you are using.
2. Review the WebSphere Application Server security architecture.
3. Review each of the following topics as also defined in Related reference.
   - “Authentication protocol for EJB security” on page 378

- “Supported authentication protocols” on page 386
- “Common Secure Interoperability Version 2 features” on page 382
- “Identity assertion” on page 382
- “Authentication mechanisms” on page 158
  - “Lightweight Third Party Authentication settings” on page 163
  - “Trust associations” on page 165
  - “Single signon” on page 171
- “User registries” on page 189
  - “Local operating system user registries” on page 191
  - “Lightweight Directory Access Protocol” on page 197
- “Custom user registries” on page 211
- “Java 2 security” on page 463
  - “Java 2 security policy files” on page 472
- “Java Authentication and Authorization Service” on page 240
  - “Programmatic login” on page 66
- “J2EE Connector security” on page 256
- “Access control exception” on page 468
  - “Role-based authorization” on page 120
  - “Administrative console and naming service authorization” on page 151
- “Secure Sockets Layer” on page 411
  - “Authenticity” on page 413
  - “Confidentiality” on page 414
  - “Integrity” on page 416

## Security considerations when adding a Base Application Server node to Network Deployment

At some point, you might decide to centralize the configuration of your stand-alone base application servers by adding them into a Network Deployment cell. If your base application server is currently configured with security, there are some issues to consider. The major issue when adding a node to the cell is whether the user registries between the base application server and the Deployment Manager are the same.

When adding a node to the cell, you automatically inherit both the user registry and the authentication mechanism of the cell.

For distributed security, all servers in the cell must use the same user registry and authentication mechanism. To recover from a user registry change, you must modify your applications so that the user and group to role mappings are correct for the new user registry. To do this, see the article on “Assigning users and groups to roles” on page 128.

Another major issue is the SSL public-key infrastructure. Prior to performing `addNode` with the Deployment Manager, verify that `addNode` can communicate as an SSL client with the Deployment Manager. This requires that the addNode truststore (configured in `sas.client.props`) contains the signer certificate of the Deployment Manager personal certificate as found in the keystore (specified in the administrative console).

See the article, “Managing digital certificates” on page 443.

The following are other issues to consider when running the `addNode` command with security:

1. When attempting to run system management commands such as `addNode`, you need to explicitly specify administrative credentials to perform the operation. The `addNode` command accepts `-username` and `-password` parameters to specify the userid and password, respectively. The user ID and password that are specified must be an administrative user; for example, a user that is a member of the console users with **Operator** or **Administrator** privileges or the administrative user ID configured in the User Registry. An example for `addNode`, `addNode CELL_HOST 8879 -includeapps -username user -password`

`pass`. `-includeapps` is optional, but this option attempts to include the server applications into the Deployment Manager. The `addNode` command might fail if the user registries used by the WebSphere Application Server and the Deployment Manager are not the same. To correct this problem, either make the user registries the same or turn off security. If you change the user registries, remember to verify that the users to roles and groups to roles mappings are correct. See addNode command for more information on the addNode syntax.
2. Adding a secured remote node through the administrative console is not supported. You can either disable security on the remote node before performing the operation or perform the operation from the command line using the addNode script.
3. Before running the `addNode` command, you must verify that the truststore files on the nodes can communicate with the keystore files from the Deployment Manager and vice versa. When using the default `DummyServerKeyFile` and `DummyServerTrustFile`, you should not see this problem as these are already able to communicate. However, never use these dummy files in a production environment or anytime sensitive data is being transmitted.
4. After running addNode, the application server is in a new SSL domain. It might contain SSL configurations that point to keystore and truststore files that are not prepared to interoperate with other servers in the same domain. Consider which servers will be intercommunicating and ensure that the servers are trusted within your truststore files.

Proper understanding of the security interactions between distributed servers greatly reduces problems encountered with secure communications. Security adds complexity because additional function needs to be managed. For security to function, it needs thorough consideration during the planning of your infrastructure. This document helps to reduce the problems that could occur due to inherent security interactions.

When you have security problems related to the WebSphere Application Server Network Deployment environment, check the Chapter 16, “Troubleshooting security configurations,” on page 955 section to see if you can get information about the problem. When trace is needed to solve a problem, because servers are distributed, quite often it is required to gather trace on all servers simultaneously while recreating the problem. This trace can be enabled dynamically or statically, depending on the type problem occurring.

## Creating login key files

1. Create a login key file. The authenticating user IDs, passwords, and target realms for each different target server are specified in the login key file, which is an ASCII file. When the security authentication service processes the login key file, the passwords in the file are encoded.
2. Add information to the login key file in the following format:

   Realm_name User_ID Password
3. Make sure that the data conforms to the following rules:
   - One realm name
   - One user ID, and one password defined in each entry
   - One entry per line
   - No blank lines between entries
   - Comments on separate lines only
   - Begin any comment with a pound sign (#):

     Example:

     ```
     # Sample key file
     #
     # First target realm
     #
     TargetRealm serverID serverPassword
     #
     ```

```
# Second target realm
#
TargetRealm2 serverID2 serverPassword2
#
# End of key file
```

A sample file named `wsserver.key` also contains these instructions. After installation, you can locate this sample file in the *install_root*/`properties` directory. You can use or modify the sample file as needed for testing.

**Note:** You can place the login key file anywhere on a host machine running the application server. However, it is recommended that you place the login key file under a securable file system .

After creating the login key files, read the article entitled, "Preparing truststore files."

## Preparing truststore files

Secure Sockets Layer (SSL) protocol protects the communication between WebSphere Application Servers. To complete the SSL connection, establish a valid truststore file for the WebSphere Application Server. A truststore file is a key database file that contains the public keys (See "Creating login key files" on page 31 for information about how to create a new keystore file.)

1. Extract the public key of the server by using the key management tool from WebSphere Application Server. For details, see "Configuring the server for request decryption: choosing the decryption method" on page 891.
2. Add the public key from the WebSphere Application Server as a signer certificate into the requesting WebSphere Application Server truststore file. For details, see the related information about how to "Importing signer certificates" on page 451.

The WebSphere Application Server truststore file is now ready to use for SSL connections with the WebSphere Application Server.

See "Configuring the application server for interoperability" for interoperability.

## Configuring the application server for interoperability

After the truststore file is ready, complete the following steps to configure the WebSphere Application Server.

1. Configure the enterprise beans that access WebSphere Application Server. Before deploying the enterprise beans, configure the RunAs Identity.
2. Enable security.
3. Enable outbound SAS authentication protocol.
4. Specify the truststore file in an Secure Sockets Layer (SSL) configuration alias and configure the WebSphere Application Server with that alias.
5. Set the **Request timeout** and **Locate request timeout** values to zero for the Object Request Broker (ORB) service.
6. Specify a security property named com.ibm.CORBA.keyFileName for the absolute path of the login key file created earlier.
7. Restart the WebSphere Application Server.

# Chapter 6. Implementing security considerations at installation time

Complete the following tasks to implement security before, during, and after installing WebSphere Application Server.

1. "Securing your environment before installation." This step describes how to install WebSphere Application Server with the proper authority.
2. Install the WebSphere Application Server. This step describes how to install WebSphere Application Server as the root user on a UNIX platform or as an administrator on a Windows platform.

   During installation you are prompted to Chapter 7, "Migrating security configurations from previous releases," on page 39.
3. "Securing your environment after installation" on page 34. This step provides information on how to protect password information after you install WebSphere Application Server.

## Securing your environment before installation

The following instructions explain how to perform a product installation with proper authority on UNIX platforms, Linux platforms, Solaris operating environments, and Windows platforms.

**UNIX platforms**

On UNIX platforms, log on as **root** and verify that the umask value is **022**.

To verify that the umask value is **022**, execute the **umask** command.

To set up the umask value as **022**, execute the **umask 022** command.

**Linux platforms and Solaris operating environments**

On Linux platforms or Solaris operating environments, make sure that the `/etc` directory contains a shadow password file. The shadow password file is named shadow and is in the `/etc` directory. If the shadow password file does not exist, an error occurs after enabling global security and configuring the user registry as local operating system.

To create the shadow file, run the pwconv command (with no parameters). This command creates an `/etc/shadow` file from the `/etc/passwd` file. After creating the shadow file, you can configure local operating system security.

**Windows platforms**

On Windows platforms, the logon user must be a member of the administrator group with the rights of **Act as part of the operating system** and **Log on as a service**.

To add the rights to a user on a Windows 2000 platform:

1. Click **Start** > **Programs** > **Administrative Tools** > **Local Security Policy** (for domain configuration, select **Domain Security Policies**, instead).
2. From the Local Security Settings Panel, click **Local Policies** > **User Rights Assignment** and add the following rights to the user ID:
   - Act as part of the operating system
   - Log on as a service

© Copyright IBM Corp. 2004

## Securing your environment after installation

WebSphere Application Server depends on several configuration files created during installation. These files contain password information and need protection. Although the files are protected to a limited degree during installation, this basic level of protection is probably not sufficient for your site. Verify that these files are protected in compliance with the policies of your site.

The files in the *install_root*\profiles*profile_name*\config and *install_root*\profiles*profile_name*\properties , except for those in the following list, need protection. For example, give permission to the user who logs onto the system for WebSphere Application Server primary administrative tasks. Other users or groups, such as WebSphere Application Server console users and console groups, who perform partial WebSphere Application Server administrative tasks, like configuring, starting servers and stopping servers, need permissions as well.

The files in the *install_root*\profiles*profile_name*\properties directory that should not be protected are:

- TraceSettings.properties
- client.policy
- client_types.xml
- implfactory.properties
- sas.client.props
- sas.stdclient.properties
- sas.tools.properties
- soap.client.props
- wsadmin.properties
- wsjaas_client.conf

1. Secure files on a Windows system:
   a. Open the browser for a view of the files and directories on the machine.
   b. Locate and right-click the file or the directory that you want to protect.
   c. Click **Properties**.
   d. Click the **Security** tab.
   e. Remove the Everyone entry and any other user or group that you do not want to have access to the file.
   f. Add the users who can access the files with the proper permission.
2. Secure files on UNIX systems. This procedure applies only to the ordinary UNIX file system. If your site uses access-control lists, secure the files by using that mechanism. Any site-specific requirements can affect the owner, group, and corresponding privileges. For example, on AIX,
   a. Go to the *install_root* directory and change the ownership of the directory configuration and properties to the user who logs onto the system for WebSphere Application Server primary administrative tasks. Run the following command: chown -R *logon_name directory_name*

      Where:
      - *login_name* is a specified user or group.
      - *directory_name* is the name of the directory that contains the files.

      It is recommended that you assign ownership of the files that contain password information to the user who runs the application server. If more than one user runs the application server, provide permission to the group in which the users are assigned in the user registry.
   b. Set up the permission by running the following command: chmod -R 770 *directory_name*.
   c. Go to the *install_root*\profiles*profile_name*\properties directory and set the following file permission to **everybody** by running the following command: chmod 777 *file_names*. where *file_names* are the following files:
      - TraceSettings.properties
      - client.policy

- `client_types.xml`
- `implfactory.properties`
- `sas.client.props`
- `sas.stdclient.properties`
- `sas.tools.properties`
- `soap.client.props`
- `wsadmin.properties`
- `wsjaas_client.conf`

d. Create a group for WebSphere Application Server and put the users who perform full or partial WebSphere Application Server administrative tasks in that group.

e. If you want to use WebSphere MQ as a JMS provider, restrict access to the `/var/mqm` directories and log files used. Give write access to the user ID mqm or members of the mqm user group only.

After securing your environment, only the users given permission can access the files. Failure to adequately secure these files can lead to a breach of security in your WebSphere Application Server applications.

If failures occur that are caused by file accessing permissions, check the permission settings.

## Protecting plain text passwords

The WebSphere Application Server has several plain text passwords. These passwords are not encrypted, but are encoded. The following is a list of files with encoded passwords:

**Important:** *WAS_INSTALL_ROOT* is a WebSphere Application Server Environment variable that you can configure through the administrative console by clicking **Environment > WebSphere variables**.

| File name | Additional information |
| --- | --- |
| *WAS_INSTALL_ROOT*\profiles*profile_name*\config\cells*cell_name*\security.xml | The following fields contain encoded passwords:<br>• **LTPA password**<br>• **JAAS authentication data**<br>• **User registry server password**<br>• **LDAP user registry bind password**<br>• **Key file password**<br>• **Trust file password**<br>• **Cryptographic token device password** |
| *WAS_INSTALL_ROOT*\profiles*profile_name*\properties\sas.client.props | Specifies passwords for:<br>• com.ibm.ssl.keyStorePassword<br>• com.ibm.ssl.trustStorePassword<br>• com.ibm.CORBA.loginPassword |
| war/WEB-INF/ibm_web_bnd.xml | Specify passwords for the default basic authentication for the ″resource-ref″ bindings within all descriptors (except in the Java cryptography architecture) |
| ejb jar/META-INF/ibm_ejbjar_bnd.xml | Specify passwords for the default basic authentication for the ″resource-ref″ bindings within all descriptors (except in the Java crytography architecture) |
| client jar/META-INF/ibm-appclient_bnd.xml | Specify passwords for the default basic authentication for the ″resource-ref″ bindings within all descriptors (except in the Java crytography architecture) |

| File name | Additional information |
|---|---|
| ear/META-INF/ibm_application_bnd.xml | Specify passwords for the default basic authentication for the ″run as″ bindings within all descriptors |
| *WAS_INSTALL_ROOT* \profiles*profile_name*\config\cells*cell_name*\nodes*node_name*\servers\server1\security.xml | The following fields contain encoded passwords:<br>• **Key file password**<br>• **Trust file password**<br>• **Cryptographic token device password**<br>• **Authentication target password**<br>• **Session persistence password**<br>• **DRS Client data replication password** |
| *WAS_INSTALL_ROOT*\profiles*profile_name*\config\cells*cell_name*\nodes*node_name*\servers\server1\resources.xml | The following fields contain encoded passwords:<br>• **WAS40Datasource password**<br>• **mailTransport password**<br>• **mailStore password**<br>• **MQQueue queue mgr password** |
| For WebSphere Application Server and WebSphere Application Server Express:<br>• *WAS_INSTALL_ROOT*\profiles*profile_name*\config\cells*cell_name*\ws-security.xml<br>• *WAS_INSTALL_ROOT*\profiles*profile_name*\config\cells*cell_name*\nodes*node_name*\servers\server1\ws-security<br>For Network Deployment:<br>*WAS_INSTALL_ROOT*\profiles*profile_name*\config\cells*cell_name*\ws-security.xml | |
| ibm-webservices-bnd.xmi | |
| ibm-webservicesclient-bnd.xmi | |
| *WAS_INSTALL_ROOT* \profiles*profile_name*\properties\soap.client.props com.ibm.ssl.trustStorePassword | Specifies passwords for:<br>• com.ibm.ssl.keyStorePassword<br>• com.ibm.ssl.trustStorePassword<br>• com.ibm.SOAP.loginPassword |
| *WAS_INSTALL_ROOT* \profiles*profile_name*\properties\sas.tools.properties | Specifies passwords for:<br>• com.ibm.ssl.keyStorePassword<br>• com.ibm.ssl.trustStorePassword<br>• com.ibm.CORBA.loginPassword |
| *WAS_INSTALL_ROOT* \profiles*profile_name*\properties\sas.stdclient.properties com.ibm.ssl.keyStorePassword | Specifies passwords for:<br>• com.ibm.ssl.keyStorePassword<br>• com.ibm.ssl.trustStorePassword<br>• com.ibm.CORBA.loginPassword |
| *WAS_INSTALL_ROOT* \profiles*profile_name*\properties\wsserver.key | |

To re-encode a password in one of the previous files, complete the following steps:

1. Access the file using a text editor and type over the encoded password in plain text. The new password is shown in plain text and must be encoded.
2. Use the `PropFilePasswordEncoder.bat` or `PropFilePasswordEncode.sh` file in the *WAS_INSTALL_ROOT*\profiles*profile_name*\bin\ directory to re-encode the password.

   If you are re-encoding SAS properties files, type `PropFilePasswordEncoder` *file_name* `-sas` and the `PropFilePasswordEncoder` file encodes the known SAS properties.

If you are encoding files that are not SAS properties files, type `PropFilePasswordEncoder` *file_name password_properties_list*

*file_name* is the name of the z/SAS properties file. *password_properties_list* is the name of the properties to encode within the file.

Use the `PropFilePasswordEncoder` utility to encode WebSphere Application Server password files only. The utility cannot encode passwords contained in XML files or other files that contain open and close tags.

If you reopen the affected file or files, the passwords do not display in plain text. Instead, the passwords appear encoded. WebSphere Application Server does not provide a utility for decoding the passwords.

## PropFilePasswordEncoder command reference

### Purpose

The **PropFilePasswordEncoder** command encodes passwords located in plain text property files. This command encodes both Secure Authentication Server (SAS) property files and non-SAS property files. After you have encoded the passwords, note that a decoding command does not exist. To encode passwords, you must run this command from the *install_dir*/bin directory of a WebSphere Application Server installation.

### Syntax

The command syntax is as follows:

```
PropFilePasswordEncoder file_name
```

### Parameters

The following option is available for the PropFilePasswordEncoder command:

**-sas**
Encodes SAS property files.

The following examples demonstrate the correct syntax.

```
PropFilePasswordEncoder file_name password_properties_list
PropFilePasswordEncoder file_name -SAS
```

# Chapter 7. Migrating security configurations from previous releases

This article addresses the need to migration your security configurations from a previous release of IBM WebSphere Application Server to WebSphere Application Server, Version 6. Complete the following steps to migrate your security configurations:

- Before migrating your configurations, verify that the administrative server of the previous release is running.
- If security is enabled in the previous release, obtain the server ID and password of the previous release. This information is needed to log onto the administrative server of the previous release during migration.
- You can optionally disable security in the previous release before migrating the installation. There is no logon required during the installation.

1. Start the First steps wizard by launching the `firststeps.bat` or `firststeps.sh` file. The first steps file is located in the following directory:
   - Linux UNIX ./*install_root*/profiles/*profile_name*/firststeps/firststeps.sh
   - Windows *install_root*\profiles*profile_name*\firststeps\firststeps.bat
2. On the First steps wizard panel, click **Migration wizard**.
3. Follow the instructions provided in the First steps wizard to complete the migration.

   For more information on the Migration wizard, see Using the Migration wizard.

The security configuration of previous WebSphere Application Server releases and its applications are migrated to the new installation of WebSphere Application Server Version 6.

This task is for migrating an installation.

If custom user registry is used in the previous version, the migration process does not migrate the class files used by the custom user registry in the <*previous_install_root*>\classes directory. Therefore, after migration, copy your custom user registry implementation classes to the *install_root*\classes directory.

If you upgrade from WebSphere Application Server, Version 5.x or 4.0.x to WebSphere Application Server, Version 6, data associated with Version 5.x or 4.0.x trust associations is not automatically migrated to Version 6. To migrate trust associations, see “Migrating trust association interceptors” on page 42.

## Migrating custom user registries

Before you perform this task, it is assumed that you already have a custom user registry implemented and are working with WebSphere Application Server Version 5.x or 4.x. The custom registry in WebSphere Application Server Version 4 is based on the CustomRegistry interface. For WebSphere Application Server Version 5.x and later, the interface is called the UserRegistry interface. The WebSphere Application Server Version 4-based custom registry works without any changes to the implementation in WebSphere Application Server Version 5.x or later except when the implementation is using data sources to connect to a database during initialization. If the previous implementation is using a data source to access a database, change the implementation to use JDBC connections to connect to the database. The WebSphere Application Server Version 4 version of the CustomRegistry interface was deprecated in WebSphere Application Server Version 5. So, moving your implementation to the WebSphere Application Server Version 5.x and later based interface is expected.

In WebSphere Application Server Version 5.x and later, in addition to the UserRegistry interface, the custom user registry requires the Result object to handle user and group information. This file is already provided in the package and you are expected to use it for the getUsers, getGroups and the getUsersForGroup methods.

© Copyright IBM Corp. 2004

In WebSphere Application Server Version 4.x, it might have been possible to use other WebSphere Application Server components (for example, datasources) to initialize the custom registry. This is no longer possible in WebSphere Application Server Version 5 or later, because other components like the containers are initialized after security and are not available during the registry initialization. In WebSphere Application Server Version 5, a custom registry implementation is a pure custom implementation, independent of other WebSphere Application Server components.

In WebSphere Application Server Version 4, if you had display names for users the EJB method getCallerPrincipal( ) and the servlet methods getUserPrincipal( ) and getRemoteUser( ) returned the display names. This behavior changed in WebSphere Application Server Version 5.x. By default, these methods now return the security name instead of the display name. However, if you need the display names to return, set the WAS_UseDisplayName property to **true**. See the getUserDisplayName method description or the Javadoc, for more information.

If the migration tool was used to migrate the WebSphere Application Server Version 4 configuration to WebSphere Application Server Version 5.x or later, be aware that this migration does not involve any changes to your existing code. Since the WebSphere Application Server Version 4 custom registry works in WebSphere Application Server Version 5.x or later without any changes to the implementation (except when using data sources) you can use the Version 4-based custom registry after the migration without modifying the code. Consider that the migration tool might not copy your implementation files from Version 4 to Version 5.x or later. You might have to copy them to the class path in the Version 6 setup (preferably to the `classes` subdirectoy, just like in Version 4). If you are using the WebSphere Application Server Network Deployment version, copy the files to the cell and to each of the nodes class paths.

In Version 5.x or later, a case insensitive authorization can occur when using the custom registry. This authorization is in effect only on the authorization check. This function is useful in cases where your custom registry returns inconsistent (in terms of case) results for user and group unique IDs.

**Note:** Setting this flag does not have any effect on the user names or passwords. Only the unique IDs returned from the registry are changed to lower-case before comparing them with the information in the authorization table, which is also converted to lowercase during run time.

Before proceeding, look at the UserRegistry interface. See “Developing custom user registries” on page 104 for a description of each of these methods in detail and the changes from Version 4.

The following steps go through in detail all the changes required to move your WebSphere Application Server Version 4 custom user registry to the Version 5.x or later custom user registry. The steps are very simple and involve minimal code changes. The sample implementation file is used as an example when describing some of the steps.

1. Change your implementation to `UserRegistry` instead of `CustomRegistry`. Change:

   ```
   public class FileRegistrySample implements CustomRegistry
   ```
   to
   ```
   public class FileRegistrySample implements UserRegistry
   ```
2. Throw the java.rmi.RemoteException in the constructors public FileRegistrySample() throws java.rmi.RemoteException
3. Change the mapCertificate method to take a certificate chain instead of a single certificate. Change

   ```
   public String mapCertificate(X509Certificate cert)
   ```
   to
   ```
   public String mapCertificate(X509Certificate[]cert)
   ```

   Having a certificate chain gives you the flexibility to act on the chain instead of one certificate. If you are only interested in the first certificate just take the first certificate in the chain before processing. In Version 5, the mapCertificate method is called to map the user in a certificate to a valid user in the

registry, when certificates are used for authentication by the Web or the Java clients (transport layer certificates, Identity Assertion certificates). In Version 4, this was only called by Web clients since the Common Secure Interoperability Version 2 (CSIv2) protocol was not supported.

4. Remove the getUsers() method.
5. Change the signature of the getUsers(String) method to return a Result object and accept an additional parameter (int). Change:

```
public List getUsers(String pattern)
```
to
```
public Result getUsers(String pattern, int limit)
```

In your implementation, construct the Result object from the list of the users obtained from the registry (whose number is limited to the value of the limit parameter) and call the setHasMore() method on the Result object if the total number of users in the registry exceeds the limit value.

6. Change the signature of the getUsersForGroup(String) method to return a Result object and accept an additional parameter (int) and throw a new exception called NotImplementedException. Change the following:

```
public List getUsersForGroup(String groupName)
          throws CustomRegistryException,
                 EntryNotFoundException {
```

to

```
public  Result  getUsersForGroup(String  groupSecurityName,  int  limit)
             throws  NotImplementedException,
                     EntryNotFoundException,
                     CustomRegistryException  {
```

In Version 5.x and later, this method is not called directly by the WebSphere Application Server Security component. However, other components of the WebSphere Application Server like the WebSphere Business Integration Server Foundation Process Choreographer use this method when staff assignments are modeled using groups. Since this already is implemented in WebSphere Application Server Version 4, it is recommended that you change the implementation similar to the `getUsers` method as explained in step 5.

7. Remove the getUniqueUserIds(String) method.
8. Remove the getGroups() method.
9. Change the signature of the getGroups(String) method to return a Result object and accept an additional parameter (int). change the following:

```
public List getGroups(String pattern)
```

to

```
public Result getGroups(String pattern, int limit)
```

In your implementation, construct the Result object from the list of the groups obtained from the registry (whose number is limited to the value of the limit parameter) and call the setHasMore() method on the Result object if the total number of groups in the registry exceeds the limit value.

10. Add the `createCredential` method. This method is not called at this time, so return as `null`.

```
public com.ibm.websphere.security.cred.WSCredential
   createCredential(String userSecurityName)
         throws CustomRegistryException,
                NotImplementedException,
```

```
                EntryNotFoundException {
        return null;
    }
```

The first and second lines of the previous code example normally appear on one line. However, it extended beyond the width of the page.

11. To build the Version 5.x and later implementation make sure you have the `sas.jar` and `wssec.jar` in your class path.

    ```
    %install_root%\java\bin\javac -classpath %WAS_HOME%\lib\wssec.jar;
    %WAS_HOME%\lib\sas.jar FileRegistrySample.java
    ```

    Type the previous lines as one continuous line.
12. Copy the implementation classes to the product class path. The *%install_root%*`/lib/ext` directory is the preferred location. If you are using the Network Deployment product, make sure that you copy these files to the cell and all the nodes. Without the files in each of the node class paths the nodes and the application servers in those nodes cannot start when security is on.
13. Use the administrative console to set up the custom registry. Follow the instructions in the "Configuring custom user registries" on page 213 article to set up the custom registry including the IgnoreCase flag. Make sure that you add the `WAS_UseDisplayName` properties, if required.

Migrates a Version 4 custom registry to the Version 5.x and later custom registry.

This step is required to migrate a custom registry from WebSphere Application Server Version 4 to WebSphere Application Server Version 5.x and later.

If you are enabling security, make sure you complete the remaining steps. Once completed, save the configuration and restart all the servers. Try accessing some J2EE resources to verify that the custom registry migration was successful.

## Migrating trust association interceptors

The following topics are addressed in this document:
- Changes to the product-provided trust association interceptors
- Migrating product-provided trust association interceptors
- Changes to the custom trust association interceptors
- Migrating custom trust association interceptors

**Changes to the product-provided trust association interceptors**

For the product provided implementation for the WebSeal server a new optional property `com.ibm.websphere.security.webseal.ignoreProxy` has been added. If this property is set to `true` or `yes`, the implementation does not check for the proxy host names and the proxy ports to match any of the host names and ports listed in the `com.ibm.websphere.security.webseal.hostnames` and the `com.ibm.websphere.security.webseal.ports` property respectively. For example, if the VIA header contains the following information:

HTTP/1.1 Fred (Proxy), 1.1 Sam (Apache/1.1),
HTP/1.1 webseal1:7002, 1.1 webseal2:7001

**Note:** The previous VIA header information was split onto two lines due to the width of the printed page.

and the `com.ibm.websphere.security.webseal.ignoreProxy` is set to `true` or `yes`, the host name `Fred` is not be used when matching the host names. By default, this property is not set, which implies that any

proxy host names and ports expected in the VIA header should be listed in the host names and the ports properties to satisfy the `isTargetInterceptor` method.

### Migrating product-provided trust association interceptors

*The properties located in the* `webseal.properties` *and* `trustedserver.properties` *files are not migrated from previous versions of the WebSphere Application Server. You must migrate the appropriate properties to WebSphere Application Server, Version 5 using the trust association panels in the administrative console.* For more information, see Configuring trust association interceptors.

### Changes to the custom trust association interceptors

If the custom interceptor extends, `com.ibm.websphere.security.WebSphereBaseTrustAssociationInterceptor`, then implement the following new method to initialize the interceptor:

```
public int init (java.util.Properties props);
```

WebSphere Application Server checks the return status before using the Trust Association implementation. Zero (0) is the default value for indicating the interceptor was successfully initialized.

However, if a previous implementation of the trust association interceptor returns a different error status you can either change your implementation to match the expectations or make one of the following changes:

**Method 1:**
: Add the `com.ibm.websphere.security.trustassociation.initStatus` property in the trust association interceptor custom properties. Set the property to the value that indicates that the interceptor is successfully initialized. All of the other possible values imply failure. In case of failure, the corresponding trust association interceptor is not used.

**Method 2:**
: Add the `com.ibm.websphere.security.trustassociation.ignoreInitStatus` property in the trust association interceptor custom properties. Set the value of this property to **true**, which tells WebSphere Application Server to ignore the status of this method. If you add this property to the custom properties, WebSphere Application Server does not check the return status, which is similar to previous versions of WebSphere Application Server.

The `public int init (java.util.Properties props);` method replaces the `public int init (String propsFile)` method.

The init(Properties) method accepts a java.util.Properties object which contains the set of properties required to initialize the interceptor. All the properties set for an interceptor (by using the Custom Properties link for that interceptor or using scripting) will be sent to this method. The interceptor can then use these properties to initialize itself. For example, in the product provided implementation for the WebSEAL server, this method reads the hosts and ports so that a request coming in can be verified to come from trusted hosts and ports. A return value of 0 implies that the interceptor initialization is successful. Any other value implies that the initialization was not successful and the interceptor will not be used.

All the properties set for an interceptor (by using the **Custom Properties** link in the administrative console for that interceptor or using scripting) is sent to this method. The interceptor can then use these properties to initialize itself. For example, in the product-provided implementation for the WebSEAL server, this method reads the hosts and ports so that an incoming request can be verified to come from trusted hosts and ports. A return value of **0** implies that the interceptor initialization is successful. Any other value implies that the initialization was not successful and the interceptor is ignored.

**Note:** The `init(String)` method still works if you want to use it instead of implementing the init(Properties) method. The only requirement is that the file name containing the custom trust association properties should now be entered using the **Custom Properties** link of the interceptor in the administrative console or by using scripts. You can enter the property using *either* of the following methods. The first method is used for backward compatibility with previous versions of WebSphere Application Server.

**Method 1:**

The same property names used in the previous release are used to obtain the file name. The file name is obtained by concatenating the `.config` to the `com.ibm.websphere.security.trustassociation.types` property value. If the file name is called `myTAI.properties` and is located in the `C:/WebSphere/AppServer/properties` directory, set the following properties:

- `com.ibm.websphere.security.trustassociation.types = myTAItype`
- `com.ibm.websphere.security.trustassociation.myTAItype.config = C:/WebSphere/AppServer/properties/myTAI.properties`

**Method 2:**

You can set the `com.ibm.websphere.security.trustassociation.initPropsFile` property in the trust association custom properties to the location of the file. For example, set the following property:

```
com.ibm.websphere.security.trustassociation.initPropsFile=
C:/WebSphere/AppServer/properties/myTAI.properties
```

The previous line of code was split into two lines due to the width of the screen. Type as one continuous line.

However, it is highly recommended that your implementation be changed to implement the init(Properties) method instead of relying on init (String propsfile) method.

### Migrating custom trust association interceptors

The trust associations from previous versions of WebSphere Application Server are not automatically migrated to version 5.x and later. Users can manually migrate these trust associations using the following steps:

1. Recompile the implementation file, if necessary.

   For more information, refer to the ″Changes to the custom trust association interceptors″ section previously discussed in this document.

   To recompile the implementation file, type the following:

   ```
   %WAS_HOME%/java/bin/javac -classpath %WAS_HOME%/lib/wssec.jar;
   %WAS_HOME%/lib/j2ee.jar <your implementation file>.java
   ```

   **Note:** The previous line of code was broken into two lines due to the width of the page. Type the code as one continuous line.
2. Copy the custom trust association interceptor class files to a location in your product class path. It is suggested that you copy these class files into the `%WAS_HOME%/lib/ext` directory.
3. Start the WebSphere Application Server
4. Enable security to use the trust association interceptor. *The properties located in your custom trust association properties file and in the `trustedserver.properties` file are not migrated from previous versions of WebSphere Application Server to version 5. You must migrate the appropriate properties to WebSphere Application Server, version 5.x or later using the trust association panels in the GUI.* For more information, see Configuring trust association interceptors.

## Migrating Common Object Request Broker Architecture programmatic login to Java Authentication and Authorization Service

WebSphere Application Server fully supports the Java Authentication and Authorization Service (JAAS) as programmatic login APIs. See “Configuring application logins for Java Authentication and Authorization Service” on page 242 and “Developing programmatic logins with the Java Authentication and Authorization Service” on page 74, for more details on JAAS support.

This document outlines the deprecated Common Object Request Broker Architecture (CORBA) programmatic login APIs and the alternatives provided by JAAS. The following are the deprecated CORBA programmatic login APIs:

- `${user.install.root}/installedApps/sampleApp.ear/default_app.war/WEB-INF/classes/LoginHelper.java`.

  The `sampleApp` is not included in Version 5.x and later.
- `${user.install.root}/installedApps/sampleApp.ear/default_app.war/WEB-INF/classes/ServerSideAuthenticator.java`.

  The `sampleApp` is not included in Version 5.x and later.
- **com.ibm.IExtendedSecurity._LoginHelper**.

  This API is included with the product, but is deprecated.
- **org.omg.SecurityLevel2.Credentials**.

  This API is included with the product, but not recommended to use.

The APIs provided in WebSphere Application Server Version 5.x and later are a combination of standard JAAS APIs and a product implementation of standard JAAS interfaces.

The following information is only a summary; refer to the JAAS documentation for your platform located at: http://www.ibm.com/developerworks/java/jdk/security/ .

- Programmatic login APIs:
  - javax.security.auth.login.LoginContext
  - javax.security.auth.callback.CallbackHandler interface: The WebSphere Application Server product provides the following implementation of the javax.security.auth.callback.CallbackHandler interface:

    **com.ibm.websphere.security.auth.callback.WSCallbackHandlerImpl**
    Provides a non-prompt CallbackHandler when the application pushes basic authentication data (user ID, password, and security realm) or token data to product LoginModules. This API is recommended for server-side login.

    **com.ibm.websphere.security.auth.callback.WSGUICallbackHandlerImpl**
    Provides a login prompt CallbackHandler to gather basic authentication data (user ID, password, and security realm). This API is recommended for client-side login.

    **Note:** If this API is used on the server side, the server is blocked for input.
  - javax.security.auth.callback.Callback interface:

    **javax.security.auth.callback.NameCallback**
    Provided by JAAS to pass the user name to the LoginModules interface.

    **javax.security.auth.callback.PasswordCallback**
    Provided by JAAS to pass the password to the LoginModules interface.

    **com.ibm.websphere.security.auth.callback.WSCredTokenCallbackImpl**
    Provided by the product to perform a token-based login. With this API, an application can pass a token-byte array to the LoginModules interface.
  - **javax.security.auth.spi.LoginModule interface**

    WebSphere Application Server provides LoginModules implementation for client and server-side login. Refer to “Configuring application logins for Java Authentication and Authorization Service” on page 242 for details.
- javax.security.Subject:

**com.ibm.websphere.security.auth.WSSubject**
An extension provided by the product to invoke remote J2EE resources using the credentials in the javax.security.Subject

**com.ibm.websphere.security.cred.WSCredential**
After a successful JAAS login with the WebSphere Application Server LoginModules intefaces, a `com.ibm.websphere.security.cred.WSCredential` credential is created and stored in the Subject.

**com.ibm.websphere.security.auth.WSPrincipal**
An authenticated principal, that is created and stored in a Subject that is authenticated by the WebSphere LoginModules interface.

1. Use the following as an example of how to perform programmatic login using the CORBA-based programmatic login APIs: The CORBA-based programmatic login APIs are replaced by JAAS login.

```
public class TestClient {
...
private void performLogin() {
// Get the ID and password of the user.
String userid = customGetUserid();
String password = customGetPassword();

// Create a new security context to hold authentication data.
LoginHelper loginHelper = new LoginHelper();
try {
// Provide the ID and password of the user for authentication.
org.omg.SecurityLevel2.Credentials credentials =
loginHelper.login(userid, password);

// Use the new credentials for all future invocations.
loginHelper.setInvocationCredentials(credentials);
// Retrieve the name of the user from the credentials
// so we can tell the user that login succeeded.

String username = loginHelper.getUserName(credentials);
System.out.println("Security context set for user: "+username);
} catch (org.omg.SecurityLevel2.LoginFailed e) {
// Handle the LoginFailed exception.
}
}
...
}
```

2. Use the following example to migrate the CORBA-based programmatic login APIs to the JAAS programmatic login APIs. The following example assumes that the application code is granted for the required Java 2 security permissions. For more information, see “Configuring application logins for Java Authentication and Authorization Service” on page 242, “Configuring Java 2 security” on page 469 and the JAAS documentation located at: http://www.ibm.com/developerworks/java/jdk/security/.

```
public class TestClient {
...
private void performLogin() {
// Create a new JAAS LoginContext.
javax.security.auth.login.LoginContext lc = null;

try {
// Use GUI prompt to gather the BasicAuth data.
lc = new javax.security.auth.login.LoginContext("WSLogin",
```

```
new com.ibm.websphere.security.auth.callback.WSGUICallbackHandlerImpl());

// create a LoginContext and specify a CallbackHandler implementation
// CallbackHandler implementation determine how authentication data is collected
// in this case, the authentication date is collected by  login prompt
//   and pass to the authentication mechanism implemented by the LoginModule.
} catch (javax.security.auth.login.LoginException e) {
System.err.println("ERROR: failed to instantiate a LoginContext and the exception: "
+ e.getMessage());
e.printStackTrace();

// may be javax.security.auth.AuthPermission "createLoginContext" is not granted
//   to the application, or the JAAS Login Configuration is not defined.
}

if (lc != null)
try {
lc.login();  // perform login
javax.security.auth.Subject s = lc.getSubject();
// get the authenticated subject

// Invoke a J2EE resources using the authenticated subject
com.ibm.websphere.security.auth.WSSubject.doAs(s,
new java.security.PrivilegedAction() {
public Object run() {
try {
bankAccount.deposit(100.00);  // where bankAccount is an protected EJB
} catch (Exception e) {
System.out.println("ERROR: error while accessing EJB resource, exception: "
+ e.getMessage());
e.printStackTrace();
}
return null;
}
}
);

// Retrieve the name of the principal from the Subject
// so we can tell the user that login succeeded,
// should only be one WSPrincipal.
java.util.Set ps =
s.getPrincipals(com.ibm.websphere.security.auth.WSPrincipal.class);
java.util.Iterator it = ps.iterator();
while (it.hasNext()) {
com.ibm.websphere.security.auth.WSPrincipal p =
(com.ibm.websphere.security.auth.WSPrincipal) it.next();
System.out.println("Principal: " + p.getName());
}
} catch (javax.security.auth.login.LoginException e) {
System.err.println("ERROR: login failed with exception: " + e.getMessage());
e.printStackTrace();

// login failed, might want to provide relogin logic
}
```

```
}
...
}
```

## Migrating from the CustomLoginServlet class to servlet filters

The CustomLoginServlet class was deprecated in Version 5. Those applications using the CustomLoginServlet class to perform authentication now need to use form-based login. Using the form-based login mechanism, you can control the look and feel of the login screen. In form-based login, a login page is specified that displays when retrieving the user ID and password information. You also can specify an error page that displays when authentication fails.

If login and error pages are not enough to implement the CustomLoginServlet class, use servlet filters. Servlet filters can dynamically intercept requests and responses to transform or use the information contained in the requests or responses. One or more servlet filters attach to a servlet or a group of servlets. Servlet filters also can attach to JSP files and HTML pages. All the attached servlet filters are called before invoking the servlet.

Both form-based login and servlet filters are supported by any Servlet 2.3 specification-compliant Web container. A form login servlet performs the authentication and servlet filters can perform additional authentication, auditing, or logging tasks.

To perform pre-login and post-login actions using servlet filters, configure these servlet filters for either form login page or for `/j_security_check` URL. The `j_security_check` is posted by the form login page with the `j_username` parameter, containing the user name and the `j_password` parameter containing the password. A servlet filter can use user name and password information to perform more authentication or meet other special needs.

1. Develop a form login page and error page for the application, as described in “Developing form login pages” on page 59.
2. Configure the form login page and the error page for the application as described in “Securing Web applications using an assembly tool” on page 118.
3. Develop servlet filters if additional processing is required before and after form login authentication. Refer to “Developing servlet filters for form login processing” on page 54 for details.
4. Configure the servlet filters developed in the previous step for either the form login page URL or for the `/j_security_check` URL. Use an assembly tool or development tools like Rational Application Developer to configure filters. After configuring the servlet filters, the `web-xml` file contains two stanzas. The first stanza contains the servlet filter configuration, the servlet filter, and its implementation class. The second stanza contains the filter mapping section and a mapping of the servlet filter to the URL. In this case, the servlet filter maps to /j_security_check.

```
<filter id="Filter_1">
   <filter-name>LoginFilter</filter-name>
   <filter-class>LoginFilter</filter-class>
   <description>Performs pre-login and post-login operation</description>
   <init-param>
    <param-name>ParamName</param-name>
   <param-value>ParamValue</param-value>
  <init-param>
 </filet>

 <filter-mapping>
  <filter-name>LoginFilter</filter-name>
  <url-pattern>/j_security_check</url-pattern>
  </filter-mapping>
```

This migration results in an application that uses form-based login and servlet filters without the use of the CustomLoginServlet class.

The use of form-based login and servlet filters by the new application are used to replace the CustomLoginServlet class. Servlet filters also are used to perform additional authentication, auditing and logging.

# Chapter 8. Developing secured applications

IBM WebSphere Application Server provides security components that provide or collaborate with other services to provide authentication, authorization, delegation, and data protection. WebSphere Application Server also supports the security features described in the Java 2 Platform, Enterprise Edition (J2EE) specification. An application goes through three stages before it is ready to run:
- Development
- Assembly
- Deployment

Most of the security for an application is configured during the assembly stage. The security configured during the assembly stage is called *declarative security* because the security is *declared* or *defined* in the deployment descriptors. The declarative security is enforced by the security run time. For some applications, declarative security is not sufficient to express the security model of the application. For these applications, you can use *programmatic security*.

1. Develop secure Web applications.
2. Develop secure Web applications. For more information, see "Developing with programmatic security APIs for Web applications."
3. Develop servlet filters for form login processing. For more information, see "Developing servlet filters for form login processing" on page 54.
4. Develop form login pages. For more information, see "Developing form login pages" on page 59.
5. Develop enterprise bean component applications. For more information, see "Developing with programmatic APIs for EJB applications" on page 62.
6. Develop with Java Authentication and Authorization Service to log in programmatically. For more information, see "Developing programmatic logins with the Java Authentication and Authorization Service" on page 74.
7. Develop your own Java 2 security mapping module. For more information, see "Configuring application logins for Java Authentication and Authorization Service" on page 242.
8. Develop custom user registries. For more information, see "Developing custom user registries" on page 104.
9. Develop a custom interceptor for trust associations. For more information, see "Trust association interceptor support for Subject creation" on page 112

## Developing with programmatic security APIs for Web applications

Programmatic security is used by security-aware applications when declarative security alone is not sufficient to express the security model of the application. Programmatic security consists of the following methods of the HttpServletRequest interface:

**getRemoteUser()**
: Returns the user name the client used for authentication. Returns **null** if no user is authenticated.

**isUserInRole**
: (String role name): Returns **true** if the remote user is granted the specified security role. If the remote user is not granted the specified role, or if no user is authenticated, it returns **false**.

**getUserPrincipal()**
: Returns the java.security.Principal object containing the remote user name. If no user is authenticated, it returns **null**.

When the `isUserInRole()` method is used, declare a security-role-ref element in the deployment descriptor with a `role-name` subelement containing the role name passed to this method. Since actual roles are created during the assembly stage of the application, you can use a logical role as the role name and provide enough hints to the assembler in the description of the security-role-ref element to link that role to the actual role. During assembly, the assembler creates a role-link subelement to link the role name to the

actual role. Creation of a security-role-ref element is possible if development tools such as Rational Web Developer is used. You also can create the security-role-ref element during assembly stage using an assembly tool.

1. Add the required security methods in the servlet code.
2. Create a security-role-ref element with the **role-name** field. If a security-role-ref element is not created during development, make sure it is created during the assembly stage.

A programmatically secured servlet application.

This step is required to secure an application programmatically. This action is particularly useful is when a Web application wants to access external resources and wants to control the access to external resources using its own authorization table (external-resource to remote-user mapping). In this case, use the `getUserPrincipal()` or `getRemoteUser()` methods to get the remote user and then it can consult its own authorization table to perform authorization. The remote user information also can help retrieve the corresponding user information from an external source such as a database or from an enterprise bean. You can use the isUserInRole() method in a similar way.

After development, a security-role-ref element can be created:

```
<security-role-ref>
<description>Provide hints to assembler for linking this role
name to an actual role here<\description>
<role-name>Mgr<\role-name>
</security-role-ref>
```

During assembly, the assembler creates a role-link element:

```
<security-role-ref>
<description>Hints provided by developer to map the role
name to the role-link</description>
<role-name>Mgr</role-name>
<role-link>Manager</role-link>
</security-role-ref>
```

You can add programmatic servlet security methods inside any servlet doGet(), doPost(), doPut(), doDelete() service methods. The following example depicts using a programmatic security API:

```
public void doGet(HttpServletRequest request,
HttpServletResponse response) {

   ....

   // to get remote user using getUserPrincipal()
   java.security.Principal principal = request.getUserPrincipal();
   String remoteUser = principal.getName();

   // to get remote user using getRemoteUser()
   remoteUser = request.getRemoteUser();

   // to check if remote user is granted Mgr role
   boolean isMgr = request.isUserInRole("Mgr");

   // use the above information in any way as needed by
   // the application
```

```
    ....

}
```

After developing an application, use an assembly tool to create roles and to link the actual roles to role names in the security-role-ref elements. For more information, see “Securing Web applications using an assembly tool” on page 118.

## Example: Web applications code

The following example depicts a Web application or servlet using the programmatic security model. The following example is one usage and not necessarily the only usage of the programmatic security model. The application can use the information returned by the `getUserPrincipal()`, `isUserInRole()` and `getRemoteUser()` methods in any other way that is meaningful to that application. Using the declarative security model whenever possible is strongly recommended.

File : HelloServlet.java

```
public class HelloServlet extends javax.servlet.http.HttpServlet {

 public void doPost(
  javax.servlet.http.HttpServletRequest request,
  javax.servlet.http.HttpServletResponse response)
  throws javax.servlet.ServletException, java.io.IOException {
 }
public void doGet(
  javax.servlet.http.HttpServletRequest request,
  javax.servlet.http.HttpServletResponse response)
  throws javax.servlet.ServletException, java.io.IOException {

        String s = "Hello";


        // get remote user using getUserPrincipal()
        java.security.Principal principal = request.getUserPrincipal();
        String remoteUserName = "";
        if( principal != null )
          remoteUserName = principal.getName();
// get remote user using getRemoteUser()
        String remoteUser = request.getRemoteUser();

        // check if remote user is granted Mgr role
        boolean isMgr = request.isUserInRole("Mgr");

        // display Hello username for managers and bob.
        if ( isMgr || remoteUserName.equals("bob") )
            s = "Hello " + remoteUserName;

   String message =  "<html> \n" +
              "<head><title>Hello Servlet</title></head>\n" +
        "<body> /n +"
    "<h1> "  +s+ </h1>/n " +
  byte[] bytes = message.getBytes();

  // displays "Hello" for ordinary users
        // and displays "Hello username" for managers and "bob".
        response.getOutputStream().write(bytes);
```

```
  }
}
```

After developing the servlet, you can create a security role reference for the HelloServlet as shown in the following example:

```
<security-role-ref>
<description> </description>
<role-name>Mgr</role-name>
</security-role-ref>
```

## Developing servlet filters for form login processing

You can control the look and feel of the login screen using the form-based login mechanism. In form-based login, you specify a login page that is used to retrieve the user ID and password information. You also can specify an error page that displays when authentication fails.

If additional authentication or additional processing is required before and after authentication, servlet filters are an option. Servlet filters can dynamically intercept requests and responses to transform or use the information contained in the requests or responses. One or more servlet filters can attach to a servlet or a group of servlets. Servlet filters also can attach to JSP files and HTML pages. All the attached servlet filters are called before the servlet is invoked.

Both form-based login and servlet filters are supported by any servlet version 2.3 specification complaint Web container. The form login servlet performs the authentication and servlet filters perform additional authentication, auditing, or logging information.

To perform pre-login and post-login actions using servlet filters, configure these filters for either form login page support or for the `/j_security_check` URL. The j_security_check is posted by a form login page with the `j_username` parameter containing the user name and the j_password parameter containing the password. A servlet filter can use the user name parameter and password information to perform more authentication or other special needs.

A servlet filter implements the javax.servlet.Filter class. There are three methods in the filter class that need implementing:

- **init(javax.servlet.FilterConfig cfg)**. This method is called by the container exactly once when the servlet filter is placed into service. The `FilterConfig` passed to this method contains the init-parameters of the servlet filter. Specify the init-parameters for a servlet filter during configuration using the assembly tool.
- **destroy()**. This method is called by the container when the servlet filter is taken out of a service.
- **doFilter(ServletRequest req, ServletResponse res, FilterChain chain)**. This method is called by the container for every servlet request that maps to this filter before invoking the servlet. FilterChain passed to this method can be used to invoke the next filter in the chain of filters. The original requested servlet executes when the last filter in the chain calls the `chain.doFilter()` method. Therefore, all filters should call the chain.doFilter() method for the original servlet to execute after filtering. If an additional authentication check is implemented in the filter code and results in failure, the original servlet does not be execute. The chain.doFilter() method is not called and can be redirected to some other error page.

If a servlet maps to many servlet filters, servlet filters are called in the order that is listed in the deployment descriptor of the application (`web.xml`).

An example of a servlet filter follows: This login filter can map to /j_security_check to perform pre-login and post-login actions.

```
import javax.servlet.*;

     public class LoginFilter implements Filter {

     protected FilterConfig filterConfig;

     // Called once when this filter is instantiated.
     // If mapped to j_security_check, called
     // very first time j_security_check is invoked.
     public void init(FilterConfig filterConfig) throws ServletException {
        this.filterConfig = filterConfig;
        }

     public void destroy() {
        this.filterConfig = null;
        }

     // Called for every request that is mapped to this filter.
     // If mapped to j_security_check,
     // called for every  j_security_check action
     public void doFilter(ServletRequest request,
     ServletResponse response, FilterChain chain)
         throws java.io.IOException, ServletException   {

         // perform pre-login action here

         chain.doFilter(request, response);
         // calls the next filter in chain.

         // j_security_check if this filter is

         // mapped to j_security_check.

        // perform post-login action here.

                  }
       }
```

Place the servlet filter class file in the `WEB-INF/classes` directory of the application.

## Configuring servlet filters

IBM Rational Application Developer or an assembly tool can configure the servlet filters. There are two steps in configuring a servlet filter.

1. Name the servlet filter and assign the corresponding implementation class to the servlet filter.

   Optionally, assign initialization parameters that get passed to the `init()` method of the servlet filter.After configuring the servlet filter, the application deployment descriptor, `web.xml`, contains a servlet filter configuration similar to the following example:

```
<filter id="Filter_1">
   <filter-name>LoginFilter</filter-name>
   <filter-class>LoginFilter</filter-class>
   <description>Performs pre-login and post-login
        operation</description>
   <init-param>// optional
     <param-name>ParameterName</param-name>
```

```
      <param-value>ParameterValue</param-value>
    </init-param>
 </filter>
```

2. Map the servlet filter to URL or servlet.

   After mapping the servlet filter to a servlet or a URL, the application deployment descriptor (web.xml) contains servlet mapping similar to the following example:

   ```
   <filter-mapping>
       <filter-name>LoginFilter</filter-name>
       <url-pattern>/j_security_check</url-pattern>
                     // can be servlet <servlet>servletName</servlet>
   </filter-mapping>
   ```

You can use servlet filters to replace the CustomLoginServlet, and to perform additional authentication, auditing, and logging.

The WebSphere Application Server Samples Gallery provides a form login sample that demonstrates how to use the WebSphere Application Server login facilities to implement and configure form login procedures. The sample integrates the following technologies to demonstrate the WebSphere Application Server and Java 2 Platform, Enterprise Edition (J2EE) login functionality:

- J2EE form-based login
- J2EE servlet filter with login
- IBM extension: form-based login

The form login sample is part of the Technology Samples package. For more information on how to access the form login sample, see Accessing the Samples (Samples Gallery).

## Example: Servlet filters

This example illustrates one way the servlet filters can perform pre-login and post-login processing during form login.

```
Servlet filter source code: LoginFilter.java
/**
 * A servlet filter example: This example filters j_security_check and
 * performs pre-login action to determine if the user trying to log in
 * is in the revoked list. If the user is on the revoked list, an error is
 * sent back to the browser.
 *
 * This filter reads the revoked list file name from the FilterConfig
 * passed in the init() method. It reads the revoked user list file and
 * creates a revokedUsers list.
 *
 * When the doFilter method is called, the user logging in is checked
 * to make sure that the user is not on the revoked Users list.
 *
 */

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class LoginFilter implements Filter {

   protected FilterConfig filterConfig;
```

```
   java.util.List revokeList;


   /**
    * init() : init() method called when the filter is instantiated.
    * This filter is instantiated the first time j_security_check is
    * invoked for the application (When a protected servlet in the
    * application is accessed).
    */
   public void init(FilterConfig filterConfig) throws ServletException {
      this.filterConfig = filterConfig;

      // read revoked user list
      revokeList = new java.util.ArrayList();
      readConfig();
   }


   /**
    * destroy() : destroy() method called when the filter is taken
    * out of service.
    */
   public void destroy() {
      this.filterConfig = null;
      revokeList = null;
   }

   /**
    * doFilter() : doFilter() method called before the servlet to
    * which this filteris mapped is invoked. Since this filter is
    * mapped to j_security_check,this method is called before
    * j_security_check action is posted.
    */
   public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain) throws java.io.IOException, ServletException {


      HttpServletRequest req = (HttpServletRequest)request;
      HttpServletResponse res = (HttpServletResponse)response;

      // pre login action

      // get username
      String username = req.getParameter("j_username");

      // if user is in revoked list send error
      if ( revokeList.contains(username) ) {
      res.sendError(javax.servlet.http.HttpServletResponse.SC_UNAUTHORIZED);
      return;
      }

      // call next filter in the chain : let j_security_check authenticate
      // user
      chain.doFilter(request, response);

      // post login action
```

```
   }

   /**
    * readConfig() : Reads revoked user list file and creates a revoked
    * user list.
    */
   private void readConfig() {
      if ( filterConfig != null ) {

         // get the revoked user list file and open it.
         BufferedReader in;
         try {
               String filename = filterConfig.getInitParameter("RevokedUsers");
               in = new BufferedReader( new FileReader(filename));
         } catch ( FileNotFoundException fnfe) {
               return;
         }

         // read all the revoked users and add to revokeList.
         String userName;
         try {
               while ( (userName = in.readLine()) != null )
                   revokeList.add(userName);
         } catch (IOException ioe) {
         }

      }

   }


}
```

**Important:** In the previous code sample, the line that begins `public void doFilter(ServletRequest request` was broken into two lines due to the width of the page. The `public void doFilter(ServletRequest request` line and the line after it are one continuous line.

Portion of the `web.xml` file showing the LoginFilter configured and mapped to j_security_check:

```
<filter id="Filter_1">
     <filter-name>LoginFilter</filter-name>
     <filter-class>LoginFilter</filter-class>
           <description>Performs pre-login and post-login operation</description>
     <init-param>
     <param-name>RevokedUsers</param-name>
     <param-value>c:\WebSphere\AppServer\installedApps\
                                    <app-name>\revokedUsers.lst</param-value>
     </init-param>
</filter-id>

<filter-mapping>
  <filter-name>LoginFilter</filter-name>
        <url-pattern>/j_security_check</url-pattern>
</filter-mapping>
```

An example of a revoked user list file:

```
user1
cn=user1,o=ibm,c=us
user99
cn=user99,o=ibm,c=us
```

## Developing form login pages

A Web client or browser can authenticate a user to a Web server using one of the following mechanisms:
- **HTTP basic authentication**: A Web server requests the Web client to authenticate and the Web client passes a user ID and password in the HTTP header.
- **HTTPS client authentication**: This mechanism requires a user (Web client) to possess a public key certificate. The Web client sends the certificate to a Web server that requests the client certificates. This is a strong authentication mechanism and uses the Hypertext Transfer Protocol with Secure Sockets Layer (HTTPS) protocol.
- **Form-based Authentication**: A developer controls the look and feel of the login screens using this authentication mechanism.

The Hypertext Transfer Protocol (HTTP) basic authentication transmits a user password from the Web client to the Web server in simple base64 encoding. Form-based authentication transmits a user password from the browser to the Web server in plain text. Therefore, both HTTP basic authentication and form-based authentication are not very secure unless the HTTPS protocol is used.

The Web application deployment descriptor contains information about which authentication mechanism to use. When form-based authentication is used, the deployment descriptor also contains entries for login and error pages. A login page can be either an HTML page or a JavaServer Pages (JSP) file. This login page displays on the Web client side when a secured resource (servlet, JSP file, HTML page) is accessed from the application. On authentication failure, an error page displays. You can write login and error pages to suit the application needs and control the look and feel of these pages. During assembly of the application, an assembler can set the authentication mechanism for the application and set the login and error pages in the deployment descriptor.

Form login uses the servlet sendRedirect() method, which has several implications for the user. The sendRedirect() method is used twice during form login:
- The sendRedirect() method initially displays the form login page in the Web browser. It later redirects the Web browser back to the originally requested protected page. The sendRedirect(String URL) method tells the Web browser to use the HTTP GET (not the HTTP POST) request to get the page specified in the URL. If HTTP POST is the first request to a protected servlet or JavaServer Pages (JSP) file, and no previous authentication or login occurred, then HTTP POST is not delivered to the requested page. However, HTTP GET is delivered because form login uses the sendRedirect() method, which behaves as an HTTP GET request that tries to display a requested page after a login occurs.
- Using HTTP POST, you might experience a scenario where an unprotected HTML form collects data from users and then posts this data to protected servlets or JSP files for processing, but the users are not logged in for the resource. To avoid this scenario, structure your Web application or permissions so that users are forced to use a form login page before the application performs any HTTP POST actions to protected servlets or JSP files.

1. Create a form login page with the required look and feel including the required elements to perform form-based authentication. For an example, see “Example: Form login” on page 60
2. Create an error page. You can program error pages to retry authentication or display an appropriate error message.
3. Place the login page and error page in the Web archive (WAR) file relative to the top directory. For example, if the login page is configured as `/login.html` in the deployment descriptor, place it in the top directory of the WAR file. An assembler can also perform this step using the assembly tool.

4. Create a form logout page and insert it to the application only if required. This step is required when a Web application requires a form-based authentication mechanism.

See the “Example: Form login” article for sample form login pages.

The WebSphere Application Server Samples Gallery provides a form login sample that demonstrates how to use the WebSphere Application Server login facilities to implement and configure form login procedures. The sample integrates the following technologies to demonstrate the WebSphere Application Server and Java 2 Platform, Enterprise Edition (J2EE) login functionality:

- J2EE form-based login
- J2EE servlet filter with login
- IBM extension: form-based login

The form login sample is part of the Technology Samples package. For more information on how to access the form login sample, see Accessing the Samples (Samples Gallery).

After developing login and error pages, add them to the Web application. Use the assembly tool to configure an authentication mechanism and insert the developed login page and error page in the deployment descriptor of the application.

## Example: Form login

For the authentication to proceed appropriately, the action of the login form must always be `j_security_check`. The following example shows how to code the form into the HTML page:

```
<form method="POST" action="j_security_check">
<input type="text" name="j_username">
<input type="text" name="j_password">
<\form>
```

use the **j_username** input field to get the user name and use the **j_password** input field to get the user password.

On receiving a request from a Web client, the Web server sends the configured form page to the client and preserves the original request. When the Web server receives the completed Form page from the Web client, it extracts the user name and password from the form and authenticates the user. On successful authentication, the Web server redirects the call to the original request. If authentication fails, the Web server redirects the call to the configured error page.

The following example depicts a login page in HTML (`login.html`):

```
<!DOCTYPE HTML PUBLIC "-//W3C/DTD HTML 4.0 Transitional//EN">
<html>
<META HTTP-EQUIV = "Pragma" CONTENT="no-cache">
<title> Security FVT Login Page </title>
<body>
<h2>Form Login</h2>
<FORM METHOD=POST ACTION="j_security_check">
<p>
<font size="2"> <strong> Enter user ID and password: </strong></font>
<BR>
<strong> User ID</strong> <input type="text" size="20" name="j_username">
<strong> Password </strong>  <input type="password" size="20" name="j_password">
<BR>
<BR>
<font size="2">  <strong> And then click this button: </strong></font>
```

```
<input type="submit" name="login" value="Login">
</p>

</form>
</body>
</html>
```

The following example depicts an error page in a JSP file:

```
<!DOCTYPE HTML PUBLIC "-//W3C/DTD HTML 4.0 Transitional//EN">
<html>
<head><title>A Form login authentication failure occurred</head></title>
<body>
<H1><B>A Form login authentication failure occurred</H1></B>
<P>Authentication may fail for one of many reasons. Some possibilities include:
<OL>
<LI>The user-id or password may be entered incorrectly; either misspelled or the
wrong case was used.
<LI>The user-id or password does not exist, has expired, or has been disabled.
</OL>
</P>


</body>
</html>
```

After an assembler configures the Web application to use form-based authentication, the deployment descriptor contains the login configuration as shown:

```
<login-config id="LoginConfig_1">
<auth-method>FORMauth-method>FORM>
<realm-name>Example Form-Based Authentication Area</realm-name>
<form-login-config id="FormLoginConfig_1">
<form-login-page>/login.html</form-login-page>
<form-error-page>/error.jsp</form-error-page>
</form-login-config>
</login-config>
```

A sample Web application archive (WAR) file directory structure showing login and error pages for the previous login configuration:

```
META-INF
     META-INF/MANIFEST.MF
     login.html
     error.jsp
     WEB-INF/
     WEB-INF/classes/
     WEB-INF/classes/aServlet.class
```

### Form logout

*Form logout* is a mechanism to log out without having to close all Web-browser sessions. After logging out the form logout mechanism, access to a protected Web resource requires reauthentication. This feature is not required by J2EE specifications, but is provided as an additional feature in WebSphere security.

Suppose that it is desirable to log out after logging into a Web application and perform some actions. A form logout works in the following manner:
1. The logout-form URI is specified in the Web browser and loads the form.
2. The user clicks **Submit** on the form to log out.
3. The WebSphere security code logs the user out.
4. Upon logout, the user is redirected to a logout exit page.

Form logout does not require any attributes in a deployment descriptor. It is an HTML or JSP file that is included with the Web application. The form-logout page is like most HTML forms except that like the form-login page, it has a special post action. This post action is recognized by the Web container, which dispatches it to a special internal WebSphere form-logout servlet. The post action in the form-logout page must be `ibm_security_logout`.

You can specify a logout-exit page in the logout form and the exit page can represent an HTML or JSP file within the same Web application to which that the user is redirected after logging out. The logout-exit page is specified as a parameter in the form-logout page. If no logout-exit page is specified, a default logout HTML message is returned to the user. Here is a sample form logout HTML form. This form configures the logout-exit page to redirect the user back to the login page after logout.

```
<!DOCTYPE HTML PUBliC "-//W3C/DTD HTML 4.0 Transitional//EN">
<html>
  <META HTTP-EQUIV = "Pragma" CONTENT="no-cache">
  <title>Logout Page </title>
  <body>
  <h2>Sample Form Logout</h2>
      <FORM METHOD=POST ACTION="ibm_security_logout" NAME="logout">
      <p>
      <BR>
      <BR>
      <font size="2"><strong> Click this button to log out: </strong></font>
      <input type="submit" name="logout" value="Logout">
      <INPUT TYPE="HIDDEN" name="logoutExitPage" VALUE="/login.html">
      </p>
      </form>
  </body>
</html>
```

The WebSphere Application Server samples gallery provides a form login sample that demonstrates how to use the WebSphere Application Server login facilities to implement and configure form login procedures. The sample integrates the following technologies to demonstrate the WebSphere Application Server and Java 2 Platform, Enterprise Edition (J2EE) login functionality:
- J2EE form-based login
- J2EE servlet filter with login
- IBM extension: form-based login

The form login sample is part of the Technology Samples package. For more information on how to access the form login sample, see Accessing the Samples (Samples Gallery).

## Developing with programmatic APIs for EJB applications

Programmatic security is used by security-aware applications when declarative security alone is not sufficient to express the security model of the application. The `javax.ejb.EJBContext` interface provides two methods whereby the bean provider can access security information about the enterprise bean caller.

- **IsCallerInRole**(String rolename): Returns true if the bean caller is granted the specified security role (specified by role name). If the caller is not granted the specified role, or if the caller is not authenticated, it returns false. If the specified role is granted **Everyone** access, it always returns true.
- **getCallerPrincipal()**: Returns the java.security.Principal object containing the bean caller name. If the caller is not authenticated, it returns a principal containing UNAUTHENTICATED name.

You can enable a login module to indicate which principal class is returned by these calls.

Refer to for more information.

When the `isCallerInRole()` method is used, declare a security-role-ref element in the deployment descriptor with a role-name subelement containing the role name passed to this method. Since actual roles are created during the assembly stage of the application, you can use a logical role as the role name and provide enough hints to the assembler in the description of the security-role-ref element to link that role to actual role. During assembly, assembler creates a role-link sub element to link the role-name to the actual role. Creation of a security-role-ref element is possible if development tools such as Rational Web Developer is used. You also can create the security-role-ref element during the assembly stage using an assembly tool.

1. Add the required security methods in the EJB module code.
2. Create a security-role-ref element with a role-name field for all the role names used in the `isCallerInRole()` method. If a security-role-ref element is not created during development, make sure it is created during the assembly stage.

A programmatically secured EJB application.

Hard coding security policies in applications is strongly discouraged. The Java 2 Platform, Enterprise Edition (J2EE) security model capabilities of declaratively specifying security policies is encouraged wherever possible. Use these APIs to develop security-aware EJB applications. An example where this implementation is useful is when an EJB application wants to access external resources and wants to control the access to these external resources using its own authorization table (external-resource to user mapping). In this case, use the `getCallerPrincipal()` method to get the caller identity and then the application can consult its own authorization table to perform authorization. The caller identification also can help retrieve the corresponding user information from an external source, such as database or from another enterprise bean. You can use the isCallerInRole() method in a similar way.

After development, a security-role-ref element can be created:

```
<security-role-ref>
<description>Provide hints to assembler for linking this role-name to
actual role here<\description>
<role-name>Mgr<\role-name>
</security-role-ref>
```

During assembly, the assembler creates a role-link element:

```
<security-role-ref>
<description>Hints provided by developer to map role-name to role-link</description>
<role-name>Mgr</role-name>
<role-link>Manager</role-link>
</security-role-ref>
```

You can add programmatic EJB component security methods (isCallerInRole() and getCallerPrincipal()) inside any business methods of an enterprise bean. The following example of programmatic security APIs includes a session bean:

```
public class aSessionBean implements SessionBean {

       .....

       // SessionContext extends EJBContext. If it is entity bean use EntityContext
       javax.ejb.SessionContext context;

       // The following method will be called by the EJB container
       // automatically
       public void setSessionContext(javax.ejb.SessionContext ctx) {
              context = ctx; // save the session bean's context
       }

       ....

       private  void aBusinessMethod()  {
       ....

       // to get  bean's caller using getCallerPrincipal()
       java.security.Principal principal = context.getCallerPrincipal();
       String  callerId= principal.getName();

       // to check if  bean's caller is granted Mgr role
       boolean isMgr = context.isCallerInRole("Mgr");

       // use the above information in any way as needed by the
       //application

       ....
       }

       ....
}
```

After developing an application, use an assembly tool to create roles and to link the actual roles to role names in the security-role-ref elements. For more information, see "Securing enterprise bean applications" on page 116.

## Example: Enterprise bean application code

The following EJB component example illustrates the use of isCallerInRole() and getCallerPrincipal() methods in an EJB module. Using that declarative security is recommended. The following example is one way of using the isCallerInRole() and getCallerPrincipal() methods. The application can use this result in any way that is suitable.

### A remote interface

```
File : Hello.java

package tests;
import java.rmi.RemoteException;
/**
 * Remote interface for Enterprise Bean: Hello
 */
public interface Hello extends javax.ejb.EJBObject {
```

```
      public abstract String getMessage()throws RemoteException;
     public abstract void setMessage(String s)throws RemoteException;
}
```

### A home interface

```
File : HelloHome.java
package tests;
/**
 * Home interface for Enterprise Bean: Hello
 */
public interface HelloHome extends javax.ejb.EJBHome {
 /**
  * Creates a default instance of Session Bean: Hello
  */
 public tests.Hello create() throws javax.ejb.CreateException,
  java.rmi.RemoteException;
}
```

### A bean implementation

```
File : HelloBean.java

package tests;
/**
 * Bean implementation class for Enterprise Bean: Hello
 */
public class HelloBean implements javax.ejb.SessionBean {
 private javax.ejb.SessionContext mySessionCtx;
 /**
  * getSessionContext
  */
 public javax.ejb.SessionContext getSessionContext() {
  return mySessionCtx;
 }
 /**
  * setSessionContext
  */
 public void setSessionContext(javax.ejb.SessionContext ctx) {
  mySessionCtx = ctx;
 }
 /**
  * ejbActivate
  */
 public void ejbActivate() {
 }
 /**
  * ejbCreate
  */
 public void ejbCreate() throws javax.ejb.CreateException {
 }
 /**
  * ejbPassivate
  */
 public void ejbPassivate() {
 }
```

```
/**
 * ejbRemove
 */
public void ejbRemove() {
}

public java.lang.String message;


      //business methods

      // all users can call getMessage()
      public String getMessage() throws java.rmi.RemoteException {
          return message;
      }

      // all users can call setMessage() but only few users can set new message.
      public void setMessage(String s) throws java.rmi.RemoteException {

        // get  bean's caller using getCallerPrincipal()
        java.security.Principal principal = mySessionCtx.getCallerPrincipal();
        java.lang.String  callerId= principal.getName();

        // check if  bean's caller is granted Mgr role
        boolean isMgr = mySessionCtx.isCallerInRole("Mgr");

        // only set supplied message if caller is "bob" or caller is granted Mgr role
        if ( isMgr || callerId.equals("bob") )
            message = s;
        else
            message = "Hello";
      }

}
```

After development of the entity bean, create a security role reference in the deployment descriptor under the session bean, Hello:

```
<security-role-ref>
<description>Only Managers can call setMessage() on this bean (Hello)</description>
<role-name>Mgr</role-name>
</security-role-ref>
```

For an explanation of how to create a `<security-role-ref>` element, see "Securing enterprise bean applications" on page 116. Use the information under Map security-role-ref and role-name to role-link to create the element.

## Programmatic login

*Programmatic login* is a type of form login that supports application presentation site-specific login forms for the purpose of authentication.

When enterprise bean client applications require the user to provide identifying information, the writer of the application must collect that information and authenticate the user. You can broadly classify the work of the programmer in terms of where the actual user authentication is performed:

- In a client program

- In a server program

Users of Web applications can receive prompts for authentication data in many ways. The `<login-config>` element in the Web application deployment descriptor file defines the mechanism used to collect this information. Programmers who want to customize login procedures, rather than relying on general purpose devices like a 401 dialog window in a browser, can use a form-based login to provide an application-specific HTML form for collecting login information.

No authentication occurs unless global security is enabled. If you want to use form-based login for Web applications, you must specify `FORM` in the auth-method tag of the `<login-config>` element in the deployment descriptor of each Web application.

Applications can present site-specific login forms by using the WebSphere Application Server form-login type. The Java 2 Platform, Enterprise Edition (J2EE) specification defines form login as one of the authentication methods for Web applications. WebSphere Application Server provides a form-logout mechanism.

**Java Authentication and Authorization Service programmatic login**

Java Authentication and Authorization Service (JAAS) is a new feature in WebSphere Application Server. It is also mandated by the J2EE 1.3 Specification. JAAS is a collection of strategic authentication APIs that replace the CORBA programmatic login APIs. WebSphere Application Server provides some extensions to JAAS:

Before you begin developing with programmatic login APIs, consider the following points :

- For the pure Java client application or client container application, initialize the client Object Request Broker (ORB) security prior to performing a JAAS login. Do this by executing the following code prior to the JAAS login:

  ```
  ...
  import java.util.Hashtable;
  import javax.naming.Context;
  import javax.naming.InitialContext;
  ...
  // Perform an InitialContext and default lookup prior to logging
  // in to initialize ORB security and for the bootstrap host/port
  // to be determined for SecurityServer lookup. If you do not want
  // to validate the userid/password during the JAAS login, disable
  // the com.ibm.CORBA.validateBasicAuth property in the
  // sas.client.props file.

  Hashtable env = new Hashtable();
  env.put(Context.INITIAL_CONTEXT_FACTORY,
      "com.ibm.websphere.naming.WsnInitialContextFactory");
  env.put(Context.PROVIDER_URL,
       "corbaloc:iiop:myhost.mycompany.com:2809");
  Context initialContext = new InitialContext(env);
  Object obj = initialContext.lookup("");
  ```

  For more information, see “Example: Programmatic logins” on page 76.
- For the pure Java client application or client container application, make sure that the host name and the port number of the target JNDI bootstrap properties are specified properly. See the Developing applications that use CosNaming (CORBA Naming interface) section for details.
- If the application uses custom JAAS login configuration, make sure that the custom JAAS login configuration is properly defined. See the “Configuring application logins for Java Authentication and Authorization Service” on page 242 section for details.

- Some of the JAAS APIs are protected by Java 2 security permissions. If these APIs are used by application code, make sure that these permissions are added to the application `was.policy` file. See “Adding the was.policy file to applications” on page 486 to the application, “Using PolicyTool to edit policy files” on page 471 and “Configuring the was.policy file” on page 482 sections for details. For more details of which APIs are protected by Java 2 Security permissions, check the IBM Developer Kit, Java edition; JAAS and the WebSphere Application Server public APIs Javadoc for more details. The following list indicates the APIs used in the samples code provided in this documentation.
  - javax.security.auth.login.LoginContext constructors are protected by javax.security.auth.AuthPermission ″createLoginContext″.
  - javax.security.auth.Subject.doAs() and com.ibm.websphere.security.auth.WSSubject.doAs() are protected by javax.security.auth.AuthPermission ″doAs″.
  - javax.security.auth.Subject.doAsPrivileged() and com.ibm.websphere.security.auth.WSSubject.doAsPrivileged() are protected by javax.security.auth.AuthPermission ″doAsPrivileged″.
- com.ibm.websphere.security.auth.WSSubject: Due to a design oversight in the JAAS 1.0, javax.security.auth.Subject.getSubject() does not return the Subject associated with the thread of execution inside a java.security.AccessController.doPrivileged() code block. This can present an inconsistent behavior that is problematic and causes undesirable effort. The com.ibm.websphere.security.auth.WSSubject API provides a work around to associate Subject to thread of execution. The com.ibm.websphere.security.auth.WSSubject API extends the JAAS model to J2EE resources for authorization checks. The Subject associated with the thread of execution within com.ibm.websphere.security.auth.WSSubject.doAs() or com.ibm.websphere.security.auth.WSSubject.doAsPrivileged() code block is used for J2EE resources authorization checks.
- UI support for defining new JAAS login configuration: You can configure JAAS login configuration in the administrative console and store it in the WebSphere Configuration API. Applications can define new JAAS login configuration in the administrative console and the data is persisted in the configuration repository (stored in the WebSphere Configuration API). However, WebSphere Application Server still supports the default JAAS login configuration format (plain text file) provided by the JAAS default implementation. But if there are duplication login configurations defined in both the WebSphere Configuration API and the plain text file format, the one in the WebSphere Configuration API takes precedence. There are advantages to defining the login configuration in the WebSphere Configuration API:
  - UI support in defining JAAS login configuration.
  - You can manage the JAAS configuration login configuration centrally.
  - The JAAS configuration login configuration is distributed in a Network Deployment installation.
- JAAS login configurations For WebSphere Application Server: WebSphere Application Server provides JAAS login configurations for application to perform programmatic authentication to the WebSphere Application Server security run time. These JAAS login configurations for WebSphere Application Server perform authentication to the configured authentication mechanism (SWAM or LTPA) and user registry (Local OS, LDAP, or Custom) based on the authentication data supplied. The authenticated Subject from these JAAS login configurations contain the required Principal and Credentials that can be used by WebSphere Application Server security run time to perform authorization checks on J2EE role-based protected resources. Here is the JAAS login configurations provided by WebSphere Application Server:
  - *WSLogin JAAS login configuration*: A generic JAAS login configuration that a Java Client, client container application, servlet, JSP file, enterprise bean, and so on, can use to perform authentication based on a user ID and password, or a token to the WebSphere Application Server security run time. However, this does not honor the CallbackHandler specified in the Client Container deployment descriptor.
  - *ClientContainer JAAS login configuration*: This JAAS login configuration honors the CallbackHandler specified in the client container deployment descriptor. The login module of this login configuration uses the CallbackHandler in the client container deployment descriptor if one is specified, even if the application code specified one CallbackHandler in the LoginContext. This is for client container application.
  - Subject authenticated with the previously mentioned JAAS login configurations contain a com.ibm.websphere.security.auth.WSPrincipal and a com.ibm.websphere.security.auth.WSCredential.

If the authenticated Subject is passed the in com.ibm.websphere.security.auth.WSSubject.doAs() (or the other doAs() methods), the WebSphere Application Server security run time can perform authorization checks on J2EE resources, based on the Subject com.ibm.websphere.security.auth.WSCredential.

- **Customer-defined JAAS login configurations**: You can define other JAAS login configurations. See "Configuring application logins for Java Authentication and Authorization Service" on page 242 section for details. Use these login configurations to perform programmatic authentication to the customer authentication mechanism. However, the subjects from these customer-defined JAAS login configurations might not be used by WebSphere Application Server security run time to perform authorization checks if the subject does not contain the required principal and credentials.

### Finding the root cause login exception from a JAAS login

If you get a LoginException after issuing the LoginContext.login() API, you can find the root cause exception from the configured user registry. In the login modules, the registry exceptions are wrapped by a `com.ibm.websphere.security.auth.WSLoginFailedException`. This exception has a getCause() method that allows you to pull out the exception that was wrapped after issuing the above command.

**Note:** You are not always guaranteed to get an exception of type WSLoginFailedException, but you should note that most of the exceptions generated from the user registry show up here.

The following is a LoginContext.login() API example with associated catch block. WSLoginFailedException has to be casted to `com.ibm.websphere.security.auth.WSLoginFailedException` if you want to issue the getCause() API.

**Note:** The determineCause() example below can be used for processing CustomUserRegistry exception types.

```
try
    {
         lc.login();
    }
    catch (LoginException le)
    {
 // drill down through the exceptions as they might cascade through the runtime
 Throwable root_exception = determineCause(le);

 // now you can use "root_exception" to compare to a particular exception type
 // for example, if you have implemented a CustomUserRegistry type, you would
  //  know what to look for here.
    }


/* Method used to drill down into the WSLoginFailedException to find the
"root cause" exception */

    public Throwable determineCause(Throwable e)
      {
    Throwable root_exception = e, temp_exception = null;

    // keep looping until there are no more embedded WSLoginFailedException or
    // WSSecurityException exceptions
    while (true)
    {
      if (e instanceof com.ibm.websphere.security.auth.WSLoginFailedException)
      {
```

```
        temp_exception = ((com.ibm.websphere.security.auth.WSLoginFailedException)
        e).getCause();
       }
       else if (e instanceof com.ibm.websphere.security.WSSecurityException)
       {
        temp_exception = ((com.ibm.websphere.security.WSSecurityException)
        e).getCause();
       }
       else if (e instanceof javax.naming.NamingException)
         // check for Ldap embedded exception
         {
           temp_exception = ((javax.naming.NamingException)e).getRootCause();
         }
       else if (e instanceof your_custom_exception_here)
       {
         // your custom processing here, if necessary
       }
       else
       {
         // this exception is not one of the types we are looking for,
         // lets return now, this is the root from the WebSphere
         //  Application Server perspective
         return root_exception;
       }
       if (temp_exception != null)
       {
         // we have an exception, let's go back an see if this has another
         // one embedded within it.
         root_exception = temp_exception;
         e = temp_exception;
         continue;
       }
       else
       {
         // we finally have the root exception from this call path, this
         // has to occur at some point
         return root_exception;
       }
     }
   }
```

### Finding the root cause login exception from a Servlet filter

You can also receive the root cause exception from a servlet filter when addressing post-Form Login processing. This is suitable because it shows the user what happened. The following API can be issued to obtain the root cause exception:

```
Throwable t = com.ibm.websphere.security.auth.WSSubject.getRootLoginException();
if (t != null)
         t = determineCause(t);
```

**Note:** Once you have the exception you can run it through the determineCause() example above to get the native registry root cause.

### Enabling root cause login exception propagation to pure Java clients

Currently, the root cause does not get propagated to a pure client for security reasons. However, you might want to propagate the root cause to a pure client in a trusted environment. If you want to enable root cause login exception propagation to a pure client, click **Security > Global Security > Custom Properties** on the WebSphere Application Server administrative console and set the following property:

```
com.ibm.websphere.security.registry.propagateExceptionsToClient=true
```

### Non-prompt programmatic login

WebSphere Application Server provides a non-prompt implementation of the javax.security.auth.callback.CallbackHandler interface, which is called com.ibm.websphere.security.auth.callback.WSCallbackHandlerImpl. Using this interface, an application can push authentication data to the WebSphere LoginModule instance to perform authentication. This capability proves useful for server-side application code to authenticate an identity and to use that identity to invoke downstream J2EE resources.

```
javax.security.auth.login.LoginContext lc = null;

try {
lc = new javax.security.auth.login.LoginContext("WSLogin",
new com.ibm.websphere.security.auth.callback.WSCallbackHandlerImpl("user",
      "securityrealm", "securedpassword"));

// create a LoginContext and specify a CallbackHandler implementation
// CallbackHandler implementation determine how authentication data is collected
// in this case, the authentication data is "push" to the authentication mechanism
//   implemented by the LoginModule.
} catch (javax.security.auth.login.LoginException e) {
System.err.println("ERROR: failed to instantiate a LoginContext and the exception: "
+ e.getMessage());
e.printStackTrace();

// may be javax.security.auth.AuthPermission "createLoginContext" is not granted
//   to the application, or the JAAS login configuration is not defined.
}

if (lc != null)
try {
lc.login();  // perform login
javax.security.auth.Subject s = lc.getSubject();
// get the authenticated subject

// Invoke a J2EE resource using the authenticated subject
com.ibm.websphere.security.auth.WSSubject.doAs(s,
new java.security.PrivilegedAction() {
public Object run() {
try {
bankAccount.deposit(100.00);  // where bankAccount is a protected EJB
} catch (Exception e) {
System.out.println("ERROR: error while accessing EJB resource, exception: "
+ e.getMessage());
e.printStackTrace();
}
return null;
```

```
}
}
);
} catch (javax.security.auth.login.LoginException e) {
System.err.println("ERROR: login failed with exception: " + e.getMessage());
e.printStackTrace();

// login failed, might want to provide relogin logic
}
```

You can use the com.ibm.websphere.security.auth.callback.WSCallbackHandlerImpl callback handler with a pure Java client, a client application container, enterprise bean, JavaServer page (JSP) files, servlet, or other Java 2 Platform, Enterprise Edition (J2EE) resources. See "Example: Programmatic logins" on page 76 for more information about object request broker (ORB) security initialization requirements in a Java pure client.

**User interface prompt programmatic login**

WebSphere Application Server also provides a user interface implementation of the javax.security.auth.callback.CallbackHandler implementation to collect authentication data from user through user interface login prompts. This callack handler, `com.ibm.websphere.security.auth.callback.WSGUICallbackHandlerImpl`, presents a user interface login panel to prompt users for authentication data.

```
javax.security.auth.login.LoginContext lc = null;

try {
lc = new javax.security.auth.login.LoginContext("WSLogin",
new com.ibm.websphere.security.auth.callback.WSGUICallbackHandlerImpl());

// create a LoginContext and specify a CallbackHandler implementation
// CallbackHandler implementation determine how authentication data is collected
// in this case, the authentication date is collected by GUI login prompt
//   and pass to the authentication mechanism implemented by the LoginModule.
} catch (javax.security.auth.login.LoginException e) {
System.err.println("ERROR: failed to instantiate a LoginContext and the exception: "
+ e.getMessage());
e.printStackTrace();

// may be javax.security.auth.AuthPermission "createLoginContext" is not granted
//   to the application, or the JAAS login configuration is not defined.
}

if (lc != null)
try {
lc.login();  // perform login
javax.security.auth.Subject s = lc.getSubject();
// get the authenticated subject

// Invoke a J2EE resources using the authenticated subject
com.ibm.websphere.security.auth.WSSubject.doAs(s,
new java.security.PrivilegedAction() {
public Object run() {
try {
bankAccount.deposit(100.00);  // where bankAccount is a protected enterprise bean
} catch (Exception e) {
```

```
System.out.println("ERROR: error while accessing EJB resource, exception: "
+ e.getMessage());
e.printStackTrace();
}
return null;
}
}
);
} catch (javax.security.auth.login.LoginException e) {
System.err.println("ERROR: login failed with exception: " + e.getMessage());
e.printStackTrace();

// login failed, might want to provide relogin logic
}
```

**Attention:** Do not use the com.ibm.websphere.security.auth.callback.WSGUICallbackHandlerImpl callback handler for server-side resources (like enterprise bean, servlet, JSP file, or any other server side resources). The user interface login prompt blocks the server for user input. This behavior is not desirable for a server process.

**Stdin prompt programmatic login**

WebSphere Application Server also provides a stdin implementation of the javax.security.auth.callback.CallbackHandler interface to collect authentication data from a user through stdin, which is called `com.ibm.websphere.security.auth.callback.WSStdinCallbackHandlerImpl`. This callback handler prompts a user for authentication data.

```
javax.security.auth.login.LoginContext lc = null;

try {
lc = new javax.security.auth.login.LoginContext("WSLogin",
new com.ibm.websphere.security.auth.callback.WSStdinCallbackHandlerImpl());

// create a LoginContext and specify a CallbackHandler implementation
// CallbackHandler implementation determine how authentication data is collected
// in this case, the authentication date is collected by stdin prompt
//   and pass to the authentication mechanism implemented by the LoginModule.
} catch (javax.security.auth.login.LoginException e) {
System.err.println("ERROR: failed to instantiate a LoginContext and the exception:
          " + e.getMessage());
e.printStackTrace();

// may be javax.security.auth.AuthPermission "createLoginContext" is not granted
//   to the application, or the JAAS login configuration is not defined.
}

if (lc != null)
try {
lc.login();  // perform login
javax.security.auth.Subject s = lc.getSubject();
// get the authenticated subject

// Invoke a J2EE resource using the authenticated subject
com.ibm.websphere.security.auth.WSSubject.doAs(s,
new java.security.PrivilegedAction() {
public Object run() {
```

```
try {
bankAccount.deposit(100.00);
// where bankAccount is a protected enterprise bean
} catch (Exception e) {
System.out.println("ERROR: error while accessing EJB resource, exception: "
       + e.getMessage());
e.printStackTrace();
}
return null;
}
}
);
} catch (javax.security.auth.login.LoginException e) {
System.err.println("ERROR: login failed with exception: " + e.getMessage());
e.printStackTrace();

// login failed, might want to provide relogin logic
}
```

Do not use the com.ibm.websphere.security.auth.callback.WSStdinCallbackHandlerImpl callback handler for server side resources (like enterprise beans, servlets, JSP files, and so on). The input from the stdin prompt is not sent to the server environment. Most servers run in the background and do not have a console. However, if the server does have a console, the stdin prompt blocks the server for user input. This behavior is not desirable for a server process.

## Developing programmatic logins with the Java Authentication and Authorization Service

Java Authentication and Authorization Service represents the strategic application programming interfaces (API) for authentication.

JAAS replaces the CORBA programmatic login APIs

WebSphere Application Server provides some extension to JAAS:
- Refer to the Developing applications that use CosNaming (CORBA Naming interface) article for details on how to set up the environment for thin client applications to access remote resources on a server.
- If the application uses custom JAAS login configuration, verify that it is properly defined. See the “Configuring application logins for Java Authentication and Authorization Service” on page 242 article for details.
- Some of the JAAS APIs are protected by Java 2 Security permissions. If these APIs are used by application code, verify that these permissions are added to the application `was.policy` file. See “Adding the was.policy file to applications” on page 486, “Using PolicyTool to edit policy files” on page 471 and “Configuring the was.policy file” on page 482 articles for details. For more details on which APIs are protected by Java 2 Security permissions, check the IBM Application Developer Kit, Java Technology Edition; JAAS and WebSphere Application Server public APIs Javadoc in “Security: Resources for learning” on page 21. Some of the APIs used in the sample code in this documentation and the Java 2 Security permissions required by these APIs follow:
  - javax.security.auth.login.LoginContext constructors are protected by javax.security.auth.AuthPermission "createLoginContext"
  - javax.security.auth.Subject.doAs() and com.ibm.websphere.security.auth.WSSubject.doAs() are protected by javax.security.auth.AuthPermission "doAs"
  - javax.security.auth.Subject.doAsPrivileged() and com.ibm.websphere.security.auth.WSSubject.doAsPrivileged() are protected by javax.security.auth.AuthPermission "doAsPrivileged"

- **Enhanced model to J2EE resources for authorization checks**. Due to a design oversight in JAAS Version 1.0, the `javax.security.auth.Subject.getSubject()` method does not return the Subject associated with the thread of execution inside a `java.security.AccessController.doPrivileged()` code block. This can present an inconsistent behavior, which might have undesirable effects. The `com.ibm.websphere.security.auth.WSSubject` provides a workaround to associate a Subject to a thread of execution. The `com.ibm.websphere.security.auth.WSSubject` extends the JAAS model to J2EE resources for authorization checks. If the Subject associates with the thread of execution within the `com.ibm.websphere.security.auth.WSSubject.doAs()` method or if the `com.ibm.websphere.security.auth.WSSubject.doAsPrivileged()` code block contains product credentials, the Subject is used for J2EE resources authorization checks.
- **User Interface support for defining new JAAS login configuration**. You can configure JAAS login configuration in the administrative console and store it in the WebSphere Common Configuration Model. Applications can define a new JAAS login configuration in the administrative console and the data is persisted in the configuration repository (stored in the WebSphere Common Configuration Model). However, WebSphere Application Server still supports the default JAAS login configuration format (plain text file) provided by the JAAS default implementation. If there are duplication login configurations defined in both the WebSphere Common Configuration and the plain text file format, the one in the WebSphere Common Configuration takes precedence. There are advantages to defining the login configuration in the WebSphere Common Configuration:
  – UI support in defining JAAS login configuration
  – JAAS configuration login configuration can be managed centrally
  – JAAS configuration login configuration is distributed in a Network Deployment installation
- **Application support for programmatic authentication**. WebSphere Application Server provides JAAS login configurations for applications to perform programmatic authentication to the WebSphere security run time. These configurations perform authentication to the WebSphere-configured authentication mechanism (Simple WebSphere Authentication Mechanism (SWAM) or Lightweight Third Party Authentication (LTPA)) and user registry (Local OS, Lightweight Directory Access Protocol (LDAP) or Custom) based on the authentication data supplied. The authenticated Subject from these JAAS login configurations contains the required Principal and Credentials that the WebSphere security run time can use to perform authorization checks on J2EE role-based protected resources. Here are the JAAS login configurations provided by the WebSphere Application Server:
  – **WSLogin JAAS login configuration**. A generic JAAS login configuration can use Java clients, client container applications, servlets, JSP files, and EJB components to perform authentication based on a user ID and password, or a token to the WebSphere security run time. However, this does not honor the CallbackHandler specified in the client container deployment descriptor.
  – **ClientContainer JAAS login configuration**. This JAAS login configuration honors the CallbackHandler specified in the client container deployment descriptor. The login module of this login configuration uses the CallbackHandler in the client container deployment descriptor if one is specified, even if the application code specified one CallbackHandler in the LoginContext. This is for a client container application.

  A Subject authenticated with the previously mentioned JAAS login configurations contains a `com.ibm.websphere.security.auth.WSPrincipal` principal and a `com.ibm.websphere.security.cred.WSCredential` credential. If the authenticated Subject is passed in `com.ibm.websphere.security.auth.WSSubject.doAs()` or the other doAs() methods, the product security run time can perform authorization checks on J2EE resources based on the `Subject` `com.ibm.websphere.security.cred.WSCredential`.
- **Customer-defined JAAS login configurations**. You can define other JAAS login configurations to perform programmatic authentication to your authentication mechanism. See the "Configuring application logins for Java Authentication and Authorization Service" on page 242 article for details. For the product security run time to perform authorization checks, the subjects from these customer-defined JAAS login configurations must contain the required principal and credentials.
- **Naming requirements for programmatic login on a pure Java client**. When programmatic login occurs on a pure Java client and the property `com.ibm.CORBA.validateBasicAuth` equals `true`, it is necessary for the security code to know where the SecurityServer resides. Typically, the default InitialContext is sufficient when a `java.naming.provider.url` property is set as a system property or when the property is set in the `jndi.properties` file. In other cases it is not desirable to have the same

java.naming.provider.url properties set in a system wide scope. In this case, there is a need to specify security specific bootstrap information in the sas.client.props file. The following steps present the order of precedence for determining how to find the SecurityServer in a pure Java client:

1. Use the sas.client.props file and look for the following properties:

   ```
   com.ibm.CORBA.securityServerHost=myhost.mydomain
   com.ibm.CORBA.securityServerPort=mybootstrap port
   ```

   If you specify these properties, you are guaranteed that security looks here for the SecurityServer. The host and port specified can represent any valid WebSphere host and bootstrap port. The SecurityServer resides on all server processes and therefore it is not important which host or port you choose. If specified, the security infrastructure within the client process look up the SecurityServer based on the information in the sas.client.props file.
2. Place the following code in your client application to get a new InitialContext():

   ```
   ...
      import java.util.Hashtable;
      import javax.naming.Context;
      import javax.naming.InitialContext;
      ...

   // Perform an InitialContext and default lookup prior to logging
   // in so that target realm and bootstrap host/port can be
   // determined for SecurityServer lookup.

         Hashtable env = new Hashtable();
         env.put(Context.INITIAL_CONTEXT_FACTORY,    "
                 com.ibm.websphere.naming.WsnInitialContextFactory");
         env.put(Context.PROVIDER_URL,
                 "corbaloc:iiop:myhost.mycompany.com:2809");
         Context initialContext = new InitialContext(env);
         Object obj = initialContext.lookup("");

      // programmatic login code goes here.
   ```

   Complete this step prior to executing any programmatic login. It is in this code that you specify a URL provider for your naming context, but it must point to a valid WebSphere Application Server within the cell that you are authenticating to. This allows thread specific programmatic logins going to different cells to have a single system-wide SecurityServer location.
3. Use the new default InitialContext() method relying on the naming precedence rules. These rules are defined in the article, Example: Getting the default initial context.

See the article, “Example: Programmatic logins.”

## Example: Programmatic logins

The following example illustrates how application programs can perform a programmatic login using Java Authentication and Authorization Service (JAAS):

```
LoginContext lc = null;

 try {
        lc = new LoginContext(″WSLogin″,
                      new WSCallbackHandlerImpl(″userName″, ″realm″, ″password″));
 } catch (LoginException le) {
         System.out.println(″Cannot create LoginContext. ″ + le.getMessage());
         // insert error processing code
 } catch(SecurityException se) {
         System.out.printlin(″Cannot create LoginContext.″ + se.getMessage();
```

```
        // Insert error processing
}

try {
         lc.login();
} catch(LoginExcpetion le) {
         System.out.println("Fails to create Subject. " + le.getMessage());
          // Insert error processing code
```

As shown in the example, the new LoginContext is initialized with the `WSLogin` login configuration and the `WSCallbackHandlerImpl CallbackHandler`. Use the `WSCallbackHandlerImpl` instance on a server-side application where prompting is not desirable. A `WSCallbackHandlerImpl` instance is initialized by the specified user ID, password, and realm information. The present `WSLoginModuleImpl` class implementation that is specified by `WSLogin` can only retrieve authentication information from the specified `CallbackHandler`. You can construct a `LoginContext` with a Subject object, but the Subject is disregarded by the present `WSLoginModuleImpl` implementation. For product client container applications, replace `WSLogin` by `ClientContainer` login configuration, which specifies the `WSClientLoginModuleImpl` implementation that is tailored for client container requirements.

For a pure Java application client, the product provides two other `CallbackHandler` implementations: `WSStdinCallbackHandlerImpl` and `WSGUICallbackHandlerImpl`, which prompt for user ID, password, and realm information on the command line and pop-up panel, respectively. You can choose either of these product `CallbackHandler` implementations depending on the particular application environment. You can develop a new `CallbackHandler` if neither of these implementations fit your particular application requirement.

You also can develop your own `LoginModule` if the default `WSLoginModuleImpl` implementation fails to meet all your requirements. This product provides utility functions that the custom LoginModule can use, which are described in the next section.

In cases where there is no `java.naming.provider.url` set as a system property or in the `jndi.properties` file, a default `InitialContext` does not function if the product server is not at the `localhost:2809` location. In this situation, perform a new `InitialContext` programmatically ahead of the JAAS login. JAAS needs to know where the SecurityServer resides to verify that the user ID or password entered is correct, prior to doing a commit(). By performing a new `InitialContext` in the way specified below, the security code has the information needed to find the SecurityServer location and the target realm.

**Attention:** The first line starting with `env.put` was split into two lines because it extends beyond the width of the printed page.

```
...
    import java.util.Hashtable;
    import javax.naming.Context;
    import javax.naming.InitialContext;
    ...

// Perform an InitialContext and default lookup prior to logging in so that target realm
// and bootstrap host/port can be determined for SecurityServer lookup.

    Hashtable env = new Hashtable();
    env.put(Context.INITIAL_CONTEXT_FACTORY,
             "com.ibm.websphere.naming.WsnInitialContextFactory");
    env.put(Context.PROVIDER_URL, "corbaloc:iiop:myhost.mycompany.com:2809");
    Context initialContext = new InitialContext(env);
    Object obj = initialContext.lookup("");
```

```
LoginContext lc = null;
try {
   lc = new LoginContext("WSLogin",
                new WSCallbackHandlerImpl("userName", "realm", "password"));
} catch (LoginException le) {
     System.out.println("Cannot create LoginContext. " + le.getMessage());
     // insert error processing code
} catch(SecurityException se) {
     System.out.printlin("Cannot create LoginContext." + se.getMessage();
     // Insert error processing
}

try {
      lc.login();
} catch(LoginException le) {
      System.out.println("Fails to create Subject. " + le.getMessage());
        // Insert error processing code
}
```

## Custom login module development for a system login configuration

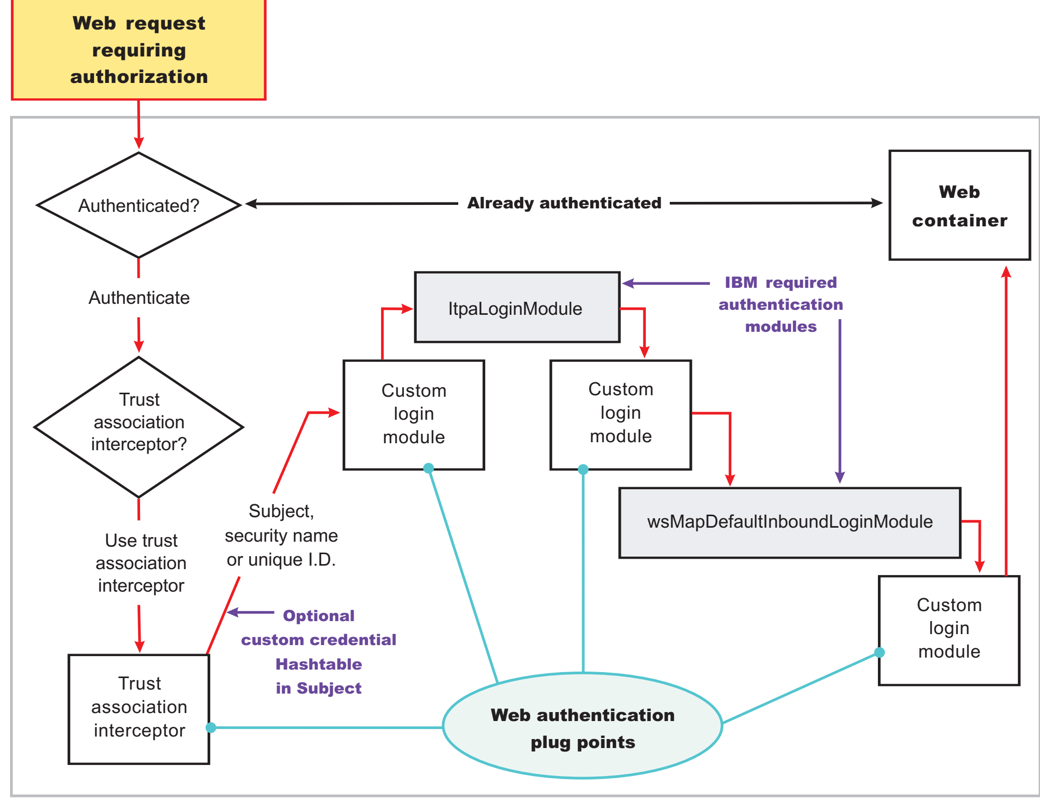
For WebSphere Application Server, there are multiple Java Authentication and Authorization Service (JAAS) plug in points for configuring system logins. WebSphere Application Server uses system login configurations to authenticate incoming requests, outgoing requests, and internal server logins. Application login configurations are called by Java 2 Platform, Enterprise Edition (J2EE) applications for obtaining a Subject based on specific authentication information. This login configuration enables the application to associate the Subject with a specific protected remote action. The Subject is picked up on the outbound request processing. The following list are the main system plug in points. If you write a login module that adds information to the Subject of a system login, these are the main login configurations to plug in:

- WEB_INBOUND
- RMI_OUTBOUND
- RMI_INBOUND
- DEFAULT

**WEB_INBOUND login configuration**

The WEB_INBOUND login configuration authenticates Web requests. Figure 1 shows an example of a configuration using a Trust Association Interceptor (TAI) that creates a Subject with the initial information that is passed into the WEB_INBOUND login configuration. If the trust association interceptor is not configured, the authentication process goes directly to the WEB_INBOUND system login configuration, which consists of all of the login modules combined in Figure 1. Figure 1 shows where you can plug in custom login modules and where the ltpaLoginModule and wsMapDefaultInboundLoginModule are required.

**Figure 1**

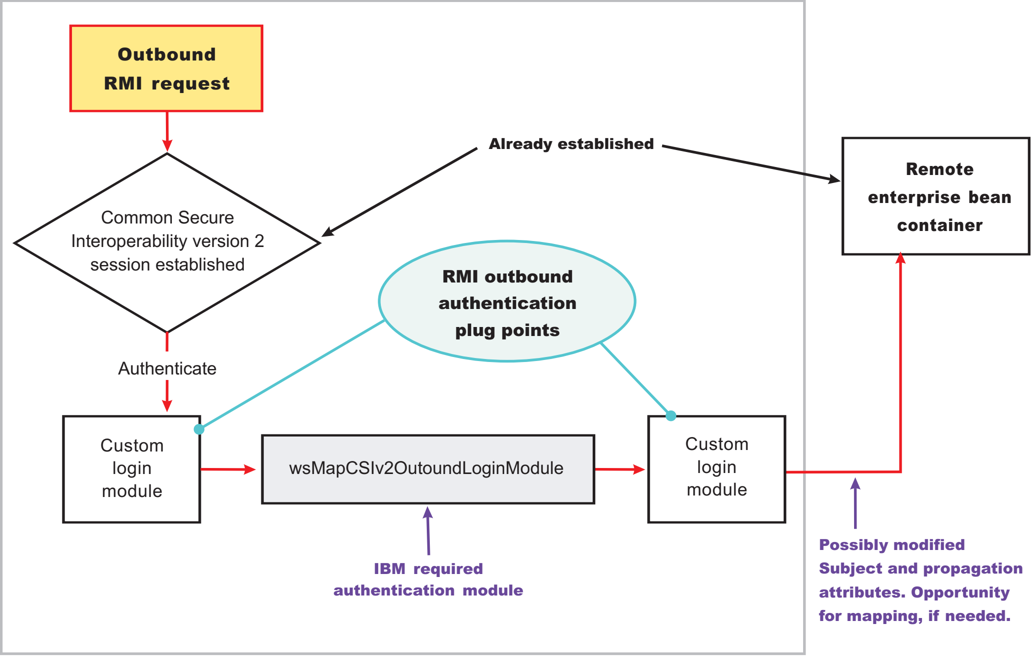



For more detailed information on the WEB_INBOUND configuration including its associated callbacks, see ″RMI_INBOUND, WEB_INBOUND, DEFAULT″ in "System login configuration entry settings for Java Authentication and Authorization Service" on page 248.

### RMI_OUTBOUND login configuration

The RMI_OUTBOUND login configuration is a plug point for handling outbound requests. WebSphere Application Server uses this plug point to create the serialized information that is sent downstream based on the Subject passed in (the invocation Subject) and other security context information such as PropagationTokens. A custom login module can use this plug point to change the identity. For more information, see "Configuring outbound mapping to a different target realm" on page 270. Figure 2 shows where you can plug in custom login modules and shows where the wsMapCSIv2OutboundLoginModule is required.

**Figure 2**

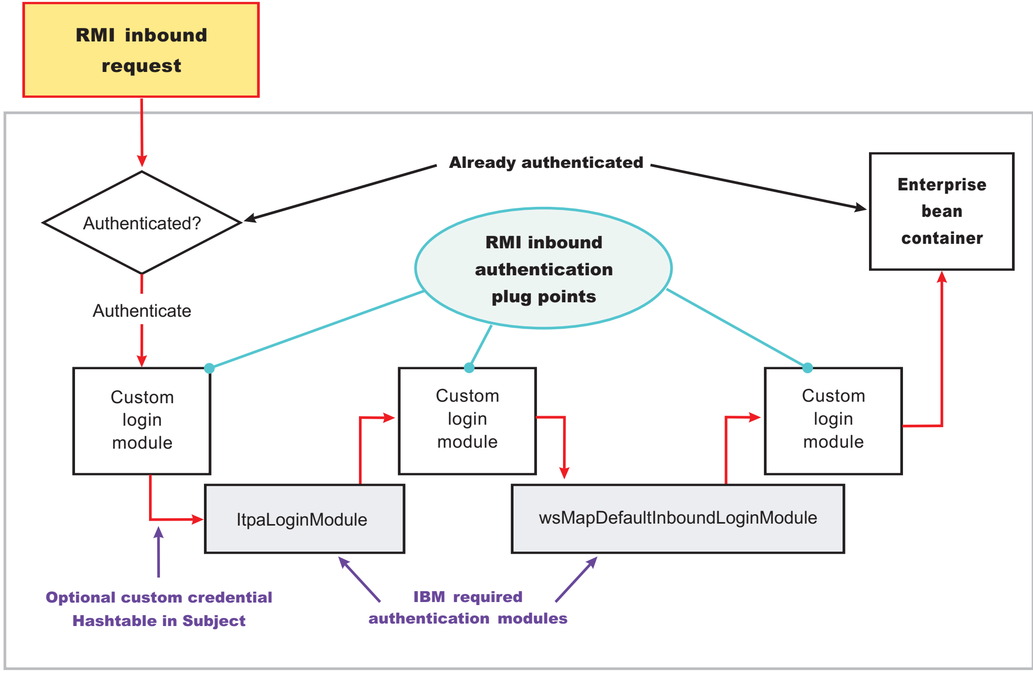



For more information on the RMI_OUTBOUND login configuration including its associated callbacks, see ″RMI_OUTBOUND″ in "System login configuration entry settings for Java Authentication and Authorization Service" on page 248.

## RMI_INBOUND login configuration

The RMI_INBOUND login configuration is a plug point that handles inbound authentication for enterprise bean requests. WebSphere Application Server uses this plug point for either an initial login or a propagation login. For more information about these two login types, see "Security attribute propagation" on page 275. During a propagation login, this plug point is used to de-serialize the information received from an upstream server. A custom login module can use this plug point to change the identity, handle custom tokens, add custom objects into the Subject, and so on. For more information on changing the identity using a Hashtable, which is referenced in figure 3, see "Configuring inbound identity mapping" on page 261. Figure 3 shows where you can plug in custom login modules and shows that the ltpaLoginModule and wsMapDefaultInboundLoginModule are required.

**Figure 3**




For more information on the RMI_INBOUND login configuration including its associated callbacks, see ″RMI_INBOUND, WEB_INBOUND, DEFAULT″ in "System login configuration entry settings for Java Authentication and Authorization Service" on page 248.

## DEFAULT login configuration

The DEFAULT login configuration is a plug point that handles all of the other types of authentication requests, including administrative Simple Object Access Protocol (SOAP) requests and internal authentication of the server ID. Propagation logins typically do not occur at this plug point.

For more information on the DEFAULT login configuration including its associated callbacks, see ″RMI_INBOUND, WEB_INBOUND, DEFAULT″ in "System login configuration entry settings for Java Authentication and Authorization Service" on page 248.

## Writing a login module

When you write a login module that plugs into a WebSphere Application Server application login or system login configuration, read the JAAS programming model located at:http://java.sun.com/products/jaas. The JAAS programming model provides basic information about JAAS. However, before writing a login module for the WebSphere Application Server environment, read the following sections in this article

- Useable callbacks
- Shared state variables
- Initial versus propagation logins
- Sample custom login module

## Useable callbacks

Each login configuration must document the callbacks that are recognized by the login configuration. However, the callbacks are not always passed data. Thus, the login configuration must contain logic to know when specific information is present and how to use the information. For example, if you write a custom login module that can plug into all four of the pre-configured system login configurations mentioned

previously, three sets of callbacks might be presented to authenticate a request. Other callbacks might be present for other reasons, including propagation and making other information available to the login configuration.

Login information can be presented in the following combinations:

**User name (NameCallback) and password (PasswordCallback)**
This information is a typical authentication combination.

**User name only (NameCallback)**
This information used for identity assertion, Trust Association Interceptor (TAI) logins, and certificate logins.

**Token (WSCredTokenCallbackImpl)**
This information is for Lightweight Third Party Authentication (LTPA) token validation.

**Propagation token list (WSTokenHolderCallback)**
This information is used for a propagation login.

The first three combinations are used for typical authentication. However, when the WSTokenHolderCallback is present in addition to one of the first three information combinations, the login is called a *propagation login*. A propagation login means that some security attributes are propagated to this server from another server. The servers can reuse these security attributes if the authentication information validates successfully. In some cases, a WSTokenHolderCallback might not have sufficient attributes for a full login. Thus, check the requiresLogin() method on the WSTokenHolderCallback to determine if a new login is required. You can always ignore the information returned by the requiresLogin() method, but, as a result, you might duplicate information.

The following is a list of the callbacks that might be present in the system login configurations. The list includes a description of their responsibility.

| Callback | Description |
| --- | --- |
| `callbacks[0] = new javax.security.auth.callback.NameCallback ("Username: ");` | This callback handler collects the user name for the login. The result can be the user name for a basic authentication login (user name and password) or a user name for an identity assertion login. |
| `callbacks[1] = new javax.security.auth.callback.PasswordCallback ("Password: ", false);` | This callback handler collects the password for the login. |
| `callbacks[2] = new com.ibm.websphere.security.auth.callback.WSCredTokenCallbackImpl ("Credential Token: ");` | This callback handler collects the Lightweight Third Party Authentication (LTPA) token, or other token type, for the login. It is typically present when a user name and password is not present. |
| `callbacks[3] = new com.ibm.wsspi.security.auth.callback.WSTokenHolderCallback ("Authz Token List: ");` | This callback handler collects the ArrayList of TokenHolder objects that are returned from a call to the WSOpaqueTokenHelper. createTokenHolderListFromOpaqueToken () API using the Common Secure Interoperability version 2 (CSIv2) authorization token as input. |
| `callbacks[4] = new com.ibm.websphere.security.auth.callback.WSServletRequestCallback ("HttpServletRequest: ");` | This callback handler collects the HTTP servlet request object, if present. It enables login modules to get information from the HTTP request for use in the login. This callback handler is presented from the WEB_INBOUND login configuration only. |

| Callback | Description |
|---|---|
| `callbacks[5] = new com.ibm. websphere.security.auth.callback. WSServletResponseCallback ("HttpServletResponse: ");` | This callback handler collects the HTTP servlet response object, if present. It enables login modules to put information into the HTTP response as a result of the login. An example of this situation might be adding the SingleSignonCookie to the response.This callback handler is presented from the WEB_INBOUND login configuration only. |
| `callbacks[6] = new com.ibm. websphere.security.auth.callback. WSAppContextCallback ("ApplicationContextCallback: ");` | This callback handler collects the Web application context used during the login. It consists of a HashMap, which contains the application name and the redirect URL, if present. This callback handler is presented from the WEB_INBOUND login configuration only. |

**Shared state variables**

Shared state variables are used to share information between login modules during the login phase. The following list contains recommendations for using the shared state variables:

- When you have a custom login module, use the shared state variables to communicate to a WebSphere Application Server login module using a documented shared state variable as shown in the following table.
- Try not to update the Subject until the commit phase. If you call the abort() method, you must remove any objects added to the Subject.
- Enable the login module that adds information into the shared state Map during login to remove this information during commit in case the same shared state is used for another login.
- If an abort or logout occurs, clean up the information in the login configuration for the shared state and the Subject.

The com.ibm.wsspi.security.token.AttributeNameConstants.WSCREDENTIAL_PROPERTIES_KEY shared state variable can inform the WebSphere Application Server login configurations about asserted privilege attributes. This variable references the com.ibm.wsspi.security.cred.propertiesObject property. You should associate a java.util.Hashtable with this property. This hashtable contains properties used by WebSphere Application Server for login purposes and ignores the callback information. This hashtable enables a custom login module, which is carried out first in the login configuration, to map user identities or enable WebSphere Application Server to avoid making unnecessary user registry calls if you already have the required information. For more information, see "Configuring inbound identity mapping" on page 261.

If you want to access the objects that WebSphere Application Server creates during a login, refer to the following shared state variables.

**Login module in which variables are set:**

ltpaLoginModule, swamLoginModule, and wsMapDefaultInboundLoginModule

**Shared state variable**
com.ibm.wsspi.security.auth.callback.Constants.WSPRINCIPAL_KEY

**Purpose**
Specifies the com.ibm.websphere.security.auth.WSPrincipal object. See the WebSphere Application Server Javadoc for application programming interface (API) usage. This shared state variable is for read-only purposes. Do not set this variable in the shared state for custom login modules.

**Login module in which variables are set:**

ltpaLoginModule, swamLoginModule, and wsMapDefaultInboundLoginModule

**Shared state variable**
com.ibm.wsspi.security.auth.callback.Constants.WSCREDENTIAL_KEY

**Purpose**
Specifies the com.ibm.websphere.security.cred.WSCredential object. See the WebSphere Application Server Javadoc for API usage. This shared state variable is for read-only purposes. Do not set this variable in the shared state for custom login modules.

**Login module in which variables are set:**

wsMapDefaultInboundLoginModule

**Shared state variable**
com.ibm.wsspi.security.auth.callback.Constants.WSAUTHZTOKEN_KEY

**Purpose**
Specifies the default com.ibm.wsspi.security.token.AuthorizationToken object. Login modules can use this object to set custom attributes plugged in after wsMapDefaultInboundLoginModule. The information set here is propagated downstream and available to the Application. See the WebSphere Application Server Javadoc for API usage.

### Initial versus propagation logins

As mentioned previously, some logins are considered initial logins because of the following reasons:

- It is the first time authentication information is presented to WebSphere Application Server.
- The login information is received from a server that does not propagate security attributes so this information must be gathered from a user registry.

Other logins are considered propagation logins when a WSTokenHolderCallback is present and contains sufficient information from a sending server to recreate all the required objects needed by WebSphere Application Server run time. In cases where there is sufficient information for WebSphere Application Server run time, the information you might add to the Subject is likely exists from the previous login. To verify if your object is present, you can get access to the ArrayList present in the WSTokenHolderCallback, and search through this list looking at each TokenHolder getName() method. This search is used to determine if WebSphere Application Server is deserializing your custom object during this login. Check the class name returned from the getName() method using the String startsWith() method because the run time might add additional information at the end of the name to know which Subject set to add the custom object after de-serialization.

The following code snippet can be used in your login() method to determine when sufficient information is present. For another example, see “Configuring inbound identity mapping” on page 261.

### Sample code

```
// This is a hint provided by WebSphere Application Server that
   // sufficient propagation information does not exist and, therefore,
   // a login is required to provide the sufficient information. In this
   // situation, a Hashtable login might be used.
  boolean requiresLogin = ((com.ibm.wsspi.security.auth.callback.
        WSTokenHolderCallback) callbacks[1]).requiresLogin();

  if (requiresLogin)
  {
   // Check to see if your object exists in the TokenHolder list,
          if not, add it.
```

```
      java.util.ArrayList authzTokenList = ((WSTokenHolderCallback) callbacks[6]).
        getTokenHolderList();boolean found = false;

      if (authzTokenList != null)
           {
              Iterator tokenListIterator = authzTokenList.iterator();

              while (tokenListIterator.hasNext())
           {
      com.ibm.wsspi.security.token.TokenHolder th = (com.ibm.wsspi.security.token.
         TokenHolder) tokenListIterator.next();

      if (th != null && th.getName().startsWith("com.acme.myCustomClass"))
           {
              found=true;
              break;
           }
      }
      if (!found)
      {
      // go ahead and add your custom object.
      }
      }
    }
 else
 {
      // This code indicates that sufficient propagation information is present.
      // User registry calls are not needed by WebSphere Application Server to
      // create a valid Subject. This code might be a no-op in your login module.
 }
```

### Sample custom login module

You can use the following sample to get ideas on how to use some of the callbacks and shared state variables.

```
public customLoginModule()
{
 // Defines your login module variables
 com.ibm.wsspi.security.token.AuthenticationToken customAuthzToken = null;
 com.ibm.wsspi.security.token.AuthenticationToken defaultAuthzToken = null;
 com.ibm.websphere.security.cred.WSCredential credential = null;
 com.ibm.websphere.security.auth.WSPrincipal principal = null;
 private javax.security.auth.Subject _subject;
 private javax.security.auth.callback.CallbackHandler _callbackHandler;
 private java.util.Map _sharedState;
 private java.util.Map _options;

 public void initialize(Subject subject, CallbackHandler callbackHandler,
     Map sharedState, Map options)
 {
  _subject = subject;
  _callbackHandler = callbackHandler;
  _sharedState = sharedState;
  _options = options;
 }
```

```
public boolean login() throws LoginException
{
 boolean succeeded = true;

 // Gets the CALLBACK information
 javax.security.auth.callback.Callback callbacks[] = new javax.security.
       auth.callback.Callback[7];
 callbacks[0] = new javax.security.auth.callback.NameCallback(
       "Username: ");
 callbacks[1] = new javax.security.auth.callback.PasswordCallback(
       "Password: ", false);
 callbacks[2] = new com.ibm.websphere.security.auth.callback.
       WSCredTokenCallbackImpl ("Credential Token: ");
 callbacks[3] = new com.ibm.wsspi.security.auth.callback.
       WSServletRequestCallback ("HttpServletRequest: ");
 callbacks[4] = new com.ibm.wsspi.security.auth.callback.
       WSServletResponseCallback ("HttpServletResponse: ");
 callbacks[5] = new com.ibm.wsspi.security.auth.callback.
       WSAppContextCallback ("ApplicationContextCallback: ");
 callbacks[6] = new com.ibm.wsspi.security.auth.callback.
        WSTokenHolderCallback ("Authz Token List: ");

 try
 {
  callbackHandler.handle(callbacks);
 }
 catch (Exception e)
 {
  // Handles exceptions
  throw new WSLoginFailedException (e.getMessage(), e);
 }

 // Sees which callbacks contain information
 uid = ((NameCallback) callbacks[0]).getName();
 char password[] = ((PasswordCallback) callbacks[1]).getPassword();
 byte[] credToken = ((WSCredTokenCallbackImpl) callbacks[2]).getCredToken();
 javax.servlet.http.HttpServletRequest request = ((WSServletRequestCallback)
       callbacks[3]).getHttpServletRequest();
 javax.servlet.http.HttpServletResponse response = ((WSServletResponseCallback)
       callbacks[4]).getHttpServletResponse();
 java.util.Map appContext = ((WSAppContextCallback)
       callbacks[5]).getContext();
 java.util.List authzTokenList = ((WSTokenHolderCallback)
       callbacks[6]).getTokenHolderList();

 // Gets the SHARED STATE information
 principal = (WSPrincipal) _sharedState.get(com.ibm.wsspi.security.
       auth.callback.Constants.WSPRINCIPAL_KEY);
 credential = (WSCredential) _sharedState.get(com.ibm.wsspi.security.
       auth.callback.Constants.WSCREDENTIAL_KEY);
 defaultAuthzToken = (AuthorizationToken) _sharedState.get(com.ibm.
       wsspi.security.auth.callback.Constants.WSAUTHZTOKEN_KEY);

    // What you tend to do with this information depends upon the scenario
    // that you are trying to accomplish. This example demonstrates how to
```

```
   // access various different information:
   // - Determine if a login is initial versus propagation
   // - Deserialize a custom authorization token (For more information, see
   //   "Security attribute propagation" on page 275
   // - Add a new custom authorization token (For more information, see
   //   "Security attribute propagation" on page 275
   // - Look for a WSCredential and read attributes, if found.
   // - Look for a WSPrincipal and read attributes, if found.
   // - Look for a default AuthorizationToken and add attributes, if found.
   // - Read the header attributes from the HttpServletRequest, if found.
   // - Add an attribute to the HttpServletResponse, if found.
   // - Get the web application name from the appContext, if found.

   // - Determines if a login is initial versus propagation. This is most
   //  useful when login module is first.
boolean requiresLogin = ((WSTokenHolderCallback) callbacks[6]).requiresLogin();

// initial login - asserts privilege attributes based on user identity
if (requiresLogin)
{

 // If you are validating a token from another server, there is an
     // application programming interface  (API) to get the uniqueID from it.
 if (credToken != null && uid == null)
 {
  try
  {
   String uniqueID = WSSecurityPropagationHelper.
             validateLTPAToken(credToken);
   String realm = WSSecurityPropagationHelper.getRealmFromUniqueID
              (uniqueID);
          // Now set it to the UID so you can use that to either map or
          // login with.
   uid = WSSecurityPropagationHelper.getUserFromUniqueID (uniqueID);
  }
  catch (Exception e)
  {
   // handle exception
  }
 }
     // Adds a Hashtable to shared state.
     // Note: You can perform custom mapping on the NameCallback value returned
     // to change the identity based upon your own mapping rules.
 uid = mapUser (uid);

 // Gets the default InitialContext for this server.
 javax.naming.InitialContext ctx = new javax.naming.InitialContext();

 // Gets the local UserRegistry object.
 com.ibm.websphere.security.UserRegistry reg = (com.ibm.websphere.security.
          UserRegistry) ctx.lookup("UserRegistry");

     // Gets the user registry uniqueID based on the uid specified in the
     // NameCallback.
 String uniqueid = reg.getUniqueUserId(uid);
  uid = WSSecurityPropagationHelper.getUserFromUniqueID (uniqueID);
```

```
 // Gets the display name from the user registry based on the uniqueID.
 String securityName = reg.getUserSecurityName(uid);

 // Gets the groups associated with this uniqueID.
 java.util.List groupList = reg.getUniqueGroupIds(uid);

     // Creates the java.util.Hashtable with the information you gathered from
     // the UserRegistry.
 java.util.Hashtable hashtable = new java.util.Hashtable();
 hashtable.put(com.ibm.wsspi.security.token.AttributeNameConstants.
         WSCREDENTIAL_UNIQUEID, uniqueid);
 hashtable.put(com.ibm.wsspi.security.token.AttributeNameConstants.
         WSCREDENTIAL_SECURITYNAME, securityName);
 hashtable.put(com.ibm.wsspi.security.token.AttributeNameConstants.
         WSCREDENTIAL_GROUPS, groupList);

     // Adds a cache key that is used as part of the lookup mechanism for
     // the created Subject. The cache key can be an Object, but should
     // implement the toString() method. Make sure the cacheKey contains
     // enough information to scope it to the user and any additional
     // attributes that you use. If you do not specify this property the
     // Subject is scoped to the WSCREDENTIAL_UNIQUEID returned, by default.
 hashtable.put(com.ibm.wsspi.security.token.AttributeNameConstants.
        WSCREDENTIAL_CACHE_KEY,
   "myCustomAttribute" + uniqueid);

  // Adds the hashtable to the sharedState of the Subject.
  _sharedState.put(com.ibm.wsspi.security.token.AttributeNameConstants.
           WSCREDENTIAL_PROPERTIES_KEY,hashtable);
}
// propagation login - process propagated tokens
else
{
 // - Deserializes a custom authorization token. For more information, see
     //    "Security attribute propagation" on page 275.
     //    This can be done at any login module plug in point (first,
     //     middle, or last).
 if (authzTokenList != null)
 {
  // Iterates through the list looking for your custom token
  for (int i=0; i<authzTokenList.size(); i++)
  {
   TokenHolder tokenHolder = (TokenHolder)authzTokenList.get(i);

   // Looks for the name and version of your custom AuthorizationToken
          // implementation
   if (tokenHolder.getName().equals("com.ibm.websphere.security.token.
              CustomAuthorizationTokenImpl") && tokenHolder.getVersion() == 1)
   {
    // Passes the bytes into your custom AuthorizationToken constructor
             // to deserialize
    customAuthzToken = new
     com.ibm.websphere.security.token.
                CustomAuthorizationTokenImpl(tokenHolder.getBytes());
```

```
    }
   }
  }
       // - Adds a new custom authorization token (For more information,
       //   see "Security attribute propagation" on page 275)
       //    This can be done at any login module plug in point (first, middle,
       //    or last).
  else
  {
   // Gets the PRINCIPAL from the default AuthenticationToken. This must
         // match all of the tokens.
   defaultAuthToken = (com.ibm.wsspi.security.token.AuthenticationToken)
    sharedState.get(com.ibm.wsspi.security.auth.callback.Constants.
              WSAUTHTOKEN_KEY);
   String principal = defaultAuthToken.getPrincipal();

   // Adds a new custom authorization token. This is an initial login.
         // Pass the principal into the constructor
   customAuthzToken = new com.ibm.websphere.security.token.
              CustomAuthorizationTokenImpl(principal);

   // Adds any initial attributes
   if (customAuthzToken != null)
   {
    customAuthzToken.addAttribute("key1", "value1");
    customAuthzToken.addAttribute("key1", "value2");
    customAuthzToken.addAttribute("key2", "value1");
    customAuthzToken.addAttribute("key3", "something different");
   }
  }
 }

 // - Looks for a WSCredential and read attributes, if found.
 //   This is most useful when plugged in as the last login module.
 if (credential != null)
 {
  try
  {
   // Reads some data from the credential
   String securityName = credential.getSecurityName();
   java.util.ArrayList = credential.getGroupIds();
  }
  catch (Exception e)
  {
   // Handles exceptions
   throw new WSLoginFailedException (e.getMessage(), e);
  }
 }

 // - Looks for a WSPrincipal and read attributes, if found.
 //   This is most useful when plugged as the last login module.
 if (principal != null)
 {
  try
  {
   // Reads some data from the principal
```

```
  String principalName = principal.getName();
 }
 catch (Exception e)
 {
  // Handles exceptions
  throw new WSLoginFailedException (e.getMessage(), e);
 }
}

// - Looks for a default AuthorizationToken and add attributes, if found.
//   This is most useful when plugged in as the last login module.
if (defaultAuthzToken != null)
{
 try
 {
  // Reads some data from the defaultAuthzToken
  String[] myCustomValue = defaultAuthzToken.getAttributes ("myKey");
  // Adds some data if not present in the defaultAuthzToken
  if (myCustomValue == null)
   defaultAuthzToken.addAttribute ("myKey", "myCustomData");
 }
 catch (Exception e)
 {
  // Handles exceptions
  throw new WSLoginFailedException (e.getMessage(), e);
 }
}

// - Reads the header attributes from the HttpServletRequest, if found.
//  This can be done at any login module plug in point (first, middle,
   //  or last).
if (request != null)
{
 java.util.Enumeration headerEnum = request.getHeaders();
 while (headerEnum.hasMoreElements())
 {
  System.out.println ("Header element: " + (String)headerEnum.nextElement());
 }
}

// - Adds an attribute to the HttpServletResponse, if found
   //  This can be done at any login module plug in point (first, middle,
   //  or last).
if (response != null)
{
 response.addHeader ("myKey", "myValue");
}

// - Gets the web application name from the appContext, if found
//   This can be done at any login module plug in point (first, middle,
   //   or last).
if (appContext != null)
{
 String appName = (String) appContext.get(com.ibm.wsspi.security.auth.
       callback.Constants.WEB_APP_NAME);
}
```

```
 return succeeded;
}

public boolean commit() throws LoginException
{
 boolean succeeded = true;

 // Add any objects here that you have created and belong in the
 // Subject. Make sure the objects are not already added. If you added
 // any sharedState variables, remove them before you exit. If the abort()
    // method gets called, make sure you cleanup anything added to the
 // Subject here.

 if (customAuthzToken != null)
 {
  // Sets the customAuthzToken token into the Subject
  try
  {
   // Do this in a doPrivileged code block so that application code
         // does not need to add additional permissions
   java.security.AccessController.doPrivileged(new java.security.PrivilegedAction()
   {
    public Object run()
    {
     try
     {
      // Adds the custom authorization token if it is not
                // null and not already in the Subject
                               if ((customAuthzTokenPriv != null) &&
        (!_subject.getPrivateCredentials().contains(customAuthzTokenPriv)))
      {
       _subject.getPrivateCredentials().add(customAuthzTokenPriv);
      }
     }
     catch (Exception e)
     {
      throw new WSLoginFailedException (e.getMessage(), e);
     }

     return null;
    }
   });
  }
  catch (Exception e)
  {
   throw new WSLoginFailedException (e.getMessage(), e);
  }
 }

 return succeeded;
}

public boolean abort() throws LoginException
{
 boolean succeeded = true;
```

```
 // Makes sure to remove all objects that have already been added (both into the
    // Subject and shared state).

 if (customAuthzToken != null)
 {
  // remove the customAuthzToken token from the Subject
  try
  {
   final AuthorizationToken customAuthzTokenPriv = customAuthzToken;
   // Do this in a doPrivileged block so that application code does not need
         // to add additional permissions
   java.security.AccessController.doPrivileged(new java.security.PrivilegedAction()
   {
    public Object run()
    {
     try
     {
      // Removes the custom authorization token if it is not
                // null and not already in the Subject
                    if ((customAuthzTokenPriv != null) &&
                     (_subject.getPrivateCredentials().
                      contains(customAuthzTokenPriv)))
      {
       _subject.getPrivateCredentials().
                   remove(customAuthzTokenPriv);
      }
     }
     catch (Exception e)
     {
      throw new WSLoginFailedException (e.getMessage(), e);
     }

     return null;
    }
   });
  }
  catch (Exception e)
  {
   throw new WSLoginFailedException (e.getMessage(), e);
  }
 }

 return succeeded;
}


public boolean logout() throws LoginException
{
 boolean succeeded = true;

 // Makes sure to remove all objects that have already been added
    // (both into the Subject and shared state).

 if (customAuthzToken != null)
 {
```

```
   // Removes the customAuthzToken token from the Subject
   try
   {
    final AuthorizationToken customAuthzTokenPriv = customAuthzToken;
    // Do this in a doPrivileged code block so that application code does
          // not need to add additional permissions
    java.security.AccessController.doPrivileged(new java.security.
            PrivilegedAction()
    {
     public Object run()
     {
      try
      {
       // Removes the custom authorization token if it is not null and not
                 //  already in the Subject
                     if ((customAuthzTokenPriv != null) && (_subject.
                     getPrivateCredentials().
                     contains(customAuthzTokenPriv)))
       {
        _subject.getPrivateCredentials().remove(customAuthzTokenPriv);
       }
      }
      catch (Exception e)
      {
       throw new WSLoginFailedException (e.getMessage(), e);
      }

      return null;
     }
    });
   }
   catch (Exception e)
   {
    throw new WSLoginFailedException (e.getMessage(), e);
   }
  }

  return succeeded;

 }

}
```

After developing your custom login module for a system login configuration, you can configure the system login using either the administrative console or using the wsadmin utility. To configure the system login using the administrative console, click **Security > JAAS Configuration > System logins**. For more information on using the wsadmin utility for system login configuration, see "Example: Customizing a server-side Java Authentication and Authorization Service authentication and login configuration" on page 94. Also refer to the "Example: Customizing a server-side Java Authentication and Authorization Service authentication and login configuration" on page 94 article for information on system login modules and to determine whether to add additional login modules.

## Example: Customizing a server-side Java Authentication and Authorization Service authentication and login configuration

WebSphere Application Server supports plugging in a custom Java Authentication and Authorization Service (JAAS) login module before or after the WebSphere Application Server system login module. However, WebSphere Application Server does not support the replacement of the WebSphere Application Server system login modules, which are used to create WSCredential and WSPrincipal in the Subject. By using a custom login module, you can either make additional authentication decisions or add information to the Subject to make additional, potentially finer-grained, authorization decisions inside a Java 2 Platform, Enterprise Edition (J2EE) application.

WebSphere Application Server enables you to propagate information downstream that is added to the Subject by a custom login module. For more information, see “Security attribute propagation” on page 275. To determine which login configuration to use for plugging in your custom login modules, see the descriptions of the login configurations located in the “System login configuration entry settings for Java Authentication and Authorization Service” on page 248 article.

WebSphere Application Server supports the modification of the system login configuration through the administrative console and by using the wsadmin scripting utility. To configure the system login configuration using the administrative console, click **Security**. Under Authentication, click **JAAS Configuration > System logins**.

Refer to the following code sample to configure a system login configuration using the wsadmin tool. The following sample JACL script adds a custom login module into the Lightweight Third-party Authentication (LTPA) Web system login configuration:

**Attention:** Lines 32, 33, and 34 in the following code sample were split onto two lines because of the width of the printed page.

```
1. ########################################
2. #
3. # Open security.xml
4. #
5. ########################################
6.
7.
8. set sec [$AdminConfig getid /Cell:hillside/Security:/]
9.
10.
11. ########################################
12. #
13. # Locate systemLoginConfig
14. #
15. ########################################
16.
17.
18. set slc [$AdminConfig showAttribute $sec systemLoginConfig]
19.
20. set entries [lindex [$AdminConfig showAttribute $slc entries] 0]
21.
22.
23. ########################################
24. #
25. # Append a new LoginModule to LTPA_WEB
26. #
```

```
27. ########################################
28.
29. foreach entry $entries {
30. set alias [$AdminConfig showAttribute $entry alias]
31. if {$alias == "LTPA_WEB"} {
32.    set newJAASLoginModuleId [$AdminConfig create JAASLoginModule
        $entry {{moduleClassName
        "com.ibm.ws.security.common.auth.module.proxy.WSLoginModuleProxy"}}]
33.    set newPropertyId [$AdminConfig create Property
        $newJAASLoginModuleId {{name delegate}{value
        "com.ABC.security.auth.CustomLoginModule"}}]
34.    $AdminConfig modify $newJAASLoginModuleId
        {{authenticationStrategy REQUIRED}}
35.    break
36.    }
37. }
38.
39.
40. ########################################
41. #
42. # save the change
43. #
44. ########################################
45.
46. $AdminConfig save
47.
```

**Attention:** The wsadmin scripting utility inserts a new object to the end of the list. To insert the custom LoginModule before the AuthenLoginModule, delete the AuthenLoginModule and then recreate it after inserting the custom LoginModule. Save the sample script into a file, sample.jacl, executing the sample script using the following command:

```
Wsadmin -f sample.jacl
```

You can use the following sample JACL script to remove the current LTPA_WEB login configuration and all the LoginModules:

```
48. ########################################
49. #
50. # Open security.xml
51. #
52. ########################################
53.
54.
55. set sec [$AdminConfig getid /Cell:hillside/Security:/]
56.
57.
58. ########################################
59. #
60. # Locate systemLoginConfig
61. #
62. ########################################
63.
64.
65. set slc [$AdminConfig showAttribute $sec systemLoginConfig]
66.
```

```
67. set entries [lindex [$AdminConfig showAttribute $slc entries] 0]
68.
69.
70. ########################################
71. #
72. # Remove the LTPA_WEB login configuration
73. #
74. ########################################
75.
76. foreach entry $entries {
77.    set alias [$AdminConfig showAttribute $entry alias]
78.    if {$alias == "LTPA_WEB"} {
79.       $AdminConfig remove $entry
80.       break
81.    }
82. }
83.
84.
85. ########################################
86. #
87. # save the change
88. #
89. ########################################
90.
91. $AdminConfig save
```

You can use the following sample JACL script to recover the original LTPA_WEB configuration:

**Attention:** Lines 122, 124, and 126 in the following code sample were split onto two or more lines because of the width of the printed page. The two lines of code for line 122 are normally one continuous line. The three lines of code for line 124 are normally one continuous line. Also, the three lines of code for line 126 are normally one continuous line.

```
92. ########################################
93. #
94. # Open security.xml
95. #
96. ########################################
97.
98.
99. set sec [$AdminConfig getid /Cell:hillside/Security:/]
100.
101.
102. ########################################
103. #
104. # Locate systemLoginConfig
105. #
106. ########################################
107.
108.
109. set slc [$AdminConfig showAttribute $sec systemLoginConfig]
110.
111. set entries [lindex [$AdminConfig showAttribute $slc entries] 0]
112.
113.
114.
```

```
115. ########################################
116. #
117. # Recreate the LTPA_WEB login configuration
118. #
119. ########################################
120.
121.
122. set newJAASConfigurationEntryId [$AdminConfig create JAASConfigurationEntry
     $slc {{alias LTPA_WEB}}]
123.
124. set newJAASLoginModuleId [$AdminConfig create JAASLoginModule
     $newJAASConfigurationEntryId
     {{moduleClassName
     "com.ibm.ws.security.common.auth.module.proxy.WSLoginModuleProxy"}}]
125.
126. set newPropertyId [$AdminConfig create Property
     $newJAASLoginModuleId {{name delegate}
     {value "com.ibm.ws.security.web.AuthenLoginModule"}}]
127.
128. $AdminConfig modify $newJAASLoginModuleId {{authenticationStrategy REQUIRED}}
129.
130.
131. ########################################
132. #
133. # save the change
134. #
135. ########################################
136.
137. $AdminConfig save
```

The WebSphere Application Server Version ltpaLoginModule and AuthenLoginModule use the shared state to save state information so that custom LoginModules can modify the information. The ltpaLoginModule initializes the callback array in the login() method using the following code. The callback array is created by ltpaLoginModule only if an array is not defined in the shared state area. In the following code sample, the error handling code was removed to make the sample concise. If you insert a custom LoginModule before the ltpaLoginModule, custom LoginModule might follow the same style to save the callback into the shared state.

**Attention:** In the following code sample, several lines of code have been split onto two lines because of the width of the printed page. Each of these split lines are one continuous line.

```
138.         Callback callbacks[] = null;
139.         if (!sharedState.containsKey(
                 com.ibm.wsspi.security.auth.callback.Constants.
                 CALLBACK_KEY)) {
140.             callbacks = new Callback[3];
141.             callbacks[0] = new NameCallback("Username: ");
142.             callbacks[1] = new PasswordCallback("Password: ", false);
143.             callbacks[2] = new com.ibm.websphere.security.auth.callback.
                 WSCredTokenCallbackImpl( "Credential Token: ");
144.             try {
145.                 callbackHandler.handle(callbacks);
146.             } catch (java.io.IOException e) {
147.                 . . .
148.            } catch (UnsupportedCallbackException uce) {
149.                 . . .
```

```
150.            }
151.            sharedState.put(
                com.ibm.wsspi.security.auth.callback.Constants.CALLBACK_KEY,
                callbacks);
152.         } else {
153.             callbacks = (Callback [])
                 sharedState.get( com.ibm.wsspi.security.auth.callback.
                 Constants.CALLBACK_KEY);
154.         }
```

ltpaLoginModule and AuthenLoginModule generate both a WSPrincipal and a WSCredential object to represent the authenticated user identity and security credentials. The WSPrincipal and WSCredential objects also are saved in the shared state. A JAAS login uses a two-phase commit protocol. First, the login methods in login modules, which are configured in the login configuration, are called. Then, their commit methods are called. A custom LoginModule, which is inserted after the ltpaLoginModule and the AuthenLoginModule, can modify the WSPrincipal and WSCredential objects before they are committed. The WSCredential and WSPrincipal objects must exist in the Subject after the login is completed. Without these objects in the Subject, WebSphere Application Server run-time code rejects the Subject when it is used to make any security decisions.

AuthenLoginModule uses the following code to initialize the callback array:

**Attention:** In the following code sample, several lines of code have been split onto two lines because of the width of the printed page. Each of these split lines are one continuous line.

```
155.         Callback callbacks[] = null;
156.         if (!sharedState.containsKey(
                 com.ibm.wsspi.security.auth.callback.Constants.
                 CALLBACK_KEY)) {
157.             callbacks = new Callback[6];
158.             callbacks[0] = new NameCallback("Username: ");
159.             callbacks[1] = new PasswordCallback("Password: ", false);
160.             callbacks[2] =
                 new com.ibm.websphere.security.auth.callback.WSCredTokenCallbackImpl(
                 "Credential Token: ");
161.             callbacks[3] =
                 new com.ibm.wsspi.security.auth.callback.WSServletRequestCallback(
                 "HttpServletRequest: ");
162.             callbacks[4] =
                 new com.ibm.wsspi.security.auth.callback.WSServletResponseCallback(
                 "HttpServletResponse: ");
163.             callbacks[5] =
                 new com.ibm.wsspi.security.auth.callback.WSAppContextCallback(
                 "ApplicationContextCallback: ");
164.             try {
165.                 callbackHandler.handle(callbacks);
166.             } catch (java.io.IOException e) {
167.               . . .
168.             } catch (UnsupportedCallbackException uce {
169.               . . .
170.             }
171.             sharedState.put( com.ibm.wsspi.security.auth.callback.
                 Constants.CALLBACK_KEY, callbacks);
172.         } else {
173.             callbacks = (Callback []) sharedState.get(
                 com.ibm.wsspi.security.auth.callback.
```

```
                  Constants.CALLBACK_KEY);
174.          }
```

Three more objects, which contain callback information for the login, are passed from the Web container to the AuthenLoginModule: a `java.util.Map`, a `HttpServletRequest`, and a `HttpServletResponse` object. These objects represent the Web application context. The WebSphere Application Server Version 5.1 application context, `java.util.Map` object, contains the application name and the error page URL. You can obtain the application context, `java.util.Map` object, by calling the `getContext()` method on the WSAppContextCallback object. The `java.util.Map` object is created with the following deployment descriptor information.

**Attention:** In the following code sample, several lines of code have been split onto two lines because of the width of the printed page. Each of these split lines are one continuous line.

```
175.          HashMap appContext = new HashMap(2);
176.          appContext.put(
              com.ibm.wsspi.security.auth.callback.Constants.WEB_APP_NAME,
              web_application_name);
177.          appContext.put(
              com.ibm.wsspi.security.auth.callback.Constants.REDIRECT_URL,
              errorPage);
```

The application name and the `HttpServletRequest` object might be read by the custom LoginModule to perform mapping functions. The error page of the form-based login might be modified by a custom LoginModule. In addition to the JAAS framework, WebSphere Application Server supports the Trust Association Interface (TAI).

Other credential types and information can be added to the caller Subject during the authentication process using a custom LoginModule. The third-party credentials in the caller Subject are managed by WebSphere Application Server as part of the security context. The caller Subject is bound to the thread of execution during the request processing. When a Web or EJB module is configured to use the caller identity, the user identity is propagated to the downstream service in an EJB request. WSCredential and any third-party credentials in the caller Subject are not propagated downstream. Instead, some of the information can be regenerated at the target server based on the propagated identity. Add third-party credentials to the caller Subject at the authentication stage. The caller Subject, which is returned from the WSSubject.getCallerSubject() method, is read-only and thus cannot be modified. For more information on the WSSubject, see “Example: Getting the Caller Subject from the Thread.”

## Example: Getting the Caller Subject from the Thread

The Caller subject (or ″received subject″) contains the user authentication information used in the call for this request. This subject is returned after issuing the WSSubject.getCallerSubject() API to prevent replacing existing objects. The subject is marked read-only. This API can be used to get access to the WSCredential (documented in the Javadoc information) so that you can put or set data in the hashmap within the credential.

Most data within the subject is not propagated downstream to another server. Only the credential token within the WSCredential is propagated downstream (and a new caller subject generated).

```
try
{
 javax.security.auth.Subject caller_subject;
 com.ibm.websphere.security.cred.WSCredential caller_cred;

 caller_subject = com.ibm.websphere.security.auth.WSSubject.getCallerSubject();
```

```
 if (caller_subject != null)
 {
  caller_cred = caller_subject.getPublicCredentials
     (com.ibm.websphere.security.cred.WSCredential.class).iterator().next();
  String CALLERDATA = (String) caller_cred.get ("MYKEY");
  System.out.println("My data from the Caller credential is:  " + CALLERDATA);
 }
}
catch (WSSecurityException e)
{
 // log error
}
catch (Exception e)
{
 // log error
}
```

**Requirement:** You need the following Java 2 Security permissions to execute this API: `permission javax.security.auth.AuthPermission "wssecurity.getCallerSubject;"`.

## Example: Getting the RunAs Subject from the Thread

The RunAs subject (or invocation subject) contains the user authentication information for the RunAs mode set in the application deployment descriptor for this method.

The RunAs subject (or invocation subject) contains the user authentication information for the RunAs mode set in the application deployment descriptor for this method. This subject is marked read-only when returned from theWSSubject.getRunAsSubject() application programming interface (API) to prevent replacing existing objects. You can use this API to get access to the WSCredential (documented in the Javadoc information) so that you can put or set data in the hashmap within the credential.

**Note:** Most data within the Subject is not propagated downstream to another server. Only the credential token within the WSCredential is propagated downstream and a new Caller subject is generated.

```
try
{
 javax.security.auth.Subject  runas_subject;
 com.ibm.websphere.security.cred.WSCredential  runas_cred;

 runas_subject  =  com.ibm.websphere.security.auth.WSSubject.getRunAsSubject();

 if  (runas_subject  !=  null)
 {
  runas_cred  =  runas_subject.getPublicCredentials(
                 com.ibm.websphere.security.cred.WSCredential.class).iterator().next();
  String  RUNASDATA  =  (String)  runas_cred.get  ("MYKEY");
  System.out.println("My  data  from  the  RunAs  credential  is:   "  +  RUNASDATA  );
 }
}
catch  (WSSecurityException  e)
{
 //  log  error
}
catch  (Exception  e)
```

```
{
 // log error
}
```

**Requirements:** You need the following Java 2 Security permissions to run this API: `permission javax.security.auth.AuthPermission ″wssecurity.getRunAsSubject;″`.

## Example: Overriding the RunAs Subject on the Thread

To extend the function provided by the Java Authentication and Authorization Service (JAAS) application programming interfaces (APIs), you can set the RunAs subject (or invocation subject) with a different valid entry that is used for outbound requests on this execution thread.

Gives flexibility for associating the Subject with all remote calls on this thread whether using a `WSSubject.doAs()` to associate the subject with the remote action or not. For example:

```
try
{
javax.security.auth.Subject runas_subject, caller_subject;

 runas_subject = com.ibm.websphere.security.auth.WSSubject.getRunAsSubject();
 caller_subject = com.ibm.websphere.security.auth.WSSubject.getCallerSubject();

 // set a new RunAs subject for the thread, overriding the one declaratively set
 com.ibm.websphere.security.auth.WSSubject.setRunAsSubject(caller_subject);

 // do some remote calls

 // restore back to the previous runAsSubject
 com.ibm.websphere.security.auth.WSSubject.setRunAsSubject(runas_subject);
}
catch (WSSecurityException e)
{
 // log error
}
catch (Exception e)
{
 // log error
}
```

You need the following Java 2 Security permissions to run these APIs:

```
permission javax.security.auth.AuthPermission ″wssecurity.getRunAsSubject″;
permission javax.security.auth.AuthPermission ″wssecurity.getCallerSubject″;
permission javax.security.auth.AuthPermission ″wssecurity.setRunAsSubject″;
```

## Example: User revocation from a cache

In WebSphere Application Server, Version 5.0.2 and later, revocation of a user from the security cache using an MBean interface is supported. The following Java Command Language (JACL) revokes a user when given the realm and user ID, and cycles through all security administration MBean instances returned for the entire cell when run from the Deployment Manager WSADMIN. The command also purges the user from the cache during each process.

**Note:** This procedure can be called from another JACL script.

**Attention:** In some of the following lines of code, the lines have been split onto two or more lines.

```
proc revokeUser {realm userid} {
 global AdminControl AdminConfig

        if {[catch {$AdminControl queryNames WebSphere:type=SecurityAdmin,*}
      result]} {
  puts stdout "\$AdminControl queryNames WebSphere:type=SecurityAdmin,*
       caught an exception $result\n"
  return
 } else {
  if {$result != {}} {
                 foreach secBean $result {
                         if {$secBean != {} || $secBean != "null"} {
                          if {[catch {$AdminControl invoke $secBean
                  purgeUserFromAuthCache "$realm $userid"} result]} {
                           puts stdout "\$AdminControl invoke $secBean
                     purgeUserFromAuthCache $realm $userid caught an
                     exception $result\n"
                           return
                          } else {
                           puts stdout "\nUser $userid has been purged from the
                    cache of process $secBean\n"
                          }
                         } else {
                          puts stdout "unable to get securityAdmin Mbean, user
                  $userid not revoked"
                      }
                        }
 } else {
  puts stdout "Security Mbean was not found\n"
  return
 }
        }
        return true
}
```

## Developing your own J2C principal mapping module

You can develop your own J2C mapping module if your application requires more sophisticated mapping functions. The mapping LoginModule that you might have developed on WebSphere Application Server Version 5 is still supported in WebSphere Application Server Version 6. The Version 5 LoginModules can be used in the connection factory mapping configuration (that is, they can be defined on the resource). They also can also be used in the resource manager connection factory reference mapping configuration. A Release 5 mapping LoginModule is not able to take advantage of the custom mapping properties.

If you want to develop a new mapping LoginModule in Version 6, use the programming interface described in the following sections.

Migrate your Version 5 mapping LoginModule to use the new programming model to take advantage of the new custom properties as well as the mapping configuration isolation at application scope. Note that mapping LoginModules developed using the WebSphere Application Server Release 6 cannot be used at the deprecated resource connection factory mapping configuration.

**Resource Reference Mapping LoginModule invocation**

A com.ibm.wsspi.security.auth.callback.WSMappingCallbackHandler class, which implements the javax.security.auth.callback.CallbackHandler interface, is a new WebSphere Application Service Provider Programming Interface (SPI) in WebSphere Application Server Version 6.

Application code uses the com.ibm.wsspi.security.auth.callback.WSMappingCallbackHandlerFactory helper class to retrieve a CallbackHandler object:

```
package  com.ibm.wsspi.security.auth.callback;

public class WSMappingCallbackHandlerFactory {
 private WSMappingCallbackHandlerFactory;
 public static CallbackHandler getMappingCallbackHandler(
ManagedConnectionFactory mcf,
HashMap mappingProperties);
}
```

The WSMappingCallbackHandler class implements the CallbackHandler interface:

```
package com.ibm.wsspi.security.auth.callback;

public class WSMappingCallbackHandler implements CallbackHandler {
 public WSMappingCallbackHandler(ManagedConnectionFactory mcf,
HashMap mappingProperties);
 public void handle(Callback[] callbacks) throws IOException,
     UnsupportedCallbackException;
}
```

WSMappingCallbackHandler can handle two new callback types defined in Release 6:

```
com.ibm.wsspi.security.auth.callback.WSManagedConnectionFactoryCallback
com.ibm.wsspi.security.auth.callback.WSMappingPropertiesCallback
```

The two Callback types should be used by new LoginModules that are used at the resource manager connection factory reference mapping configuration. The WSManagedConnectionFactoryCallback provides a ManagedConnectionFactory instance that should be set in the PasswordCredential. It allows a ManagedConnectionFactory instance to determine whether a PasswordCredential instance is used for sign-on to the target EIS instance. The WSMappingPropertiesCallback provides a HashMap that contains custom mapping properties. The property name ″com.ibm.mapping.authDataAlias″ is reserved for setting the authentication data alias.

The WebSphere Application Server Release 6 WSMappingCallbackHandle continues to support the two WebSphere Application Server Release 5 Callback types that can be used by older mapping LoginModules. The two Callbacks defined below can only be used by LoginModules that are used by login configuration at the connection factory. For backward compatibility, WebSphere Application Server Release 6 passes the authentication data alias, if defined in the list of custom properties under the “com.ibm.mapping.authDataAlias” property name, via the WSAuthDataAliasCallback to Release 5 LoginModules:

```
com.ibm.ws.security.auth.j2c.WSManagedConnectionFactoryCallback
com.ibm.ws.security.auth.j2c.WSAuthDataAliasCallback
```

### Connection Factory Mapping LoginModule Invocation

The WSPrincipalMappingCallbackHandler class handles two Callback types: WSManagedConnectionFactoryCallback and WSMappingPropertiesCallback:

```
com.ibm.wsspi.security.auth.callback.WSManagedConnectionFactoryCallback
com.ibm.wsspi.security.auth.callback.WSMappingPropertiesCallback
```

The WSPrincipalMappingCallbackHandler and the two Callbacks are deprecated in WebSphere Application Server Release 6 and should not be used by new development work.

### Mapping LoginModule Resource Reference Mapping Properties

You can pass arbitrary custom properties to your mapping LoginModule. The following example shows how the WebSphere Application Server default mapping LoginModule looks for the authentication data alias property.

```
 try {
     wspm_callbackHandler.handle(callbacks);
            String userID = null;
            String password = null;
            String alias = null;
            wspm_properties = ((WSMappingPropertiesCallback)callbacks[1]).getProperties();

            if (wspm_properties != null) {
                alias = (String) wspm_properties.get(com.ibm.wsspi.security.auth.callback.Constants.MAPPING_ALIAS);
                if (alias != null) {
                    alias = alias.trim();
                }
            }
            } catch (UnsupportedCallbackException unsupportedcallbackexception) {
. . . // error handling
```

The WebSphere Application Server Version 6 default mapping LoginModule requires one mapping property to define the authentication data alias. The property name, MAPPING_ALIAS, is defined in the Constants.class in the com.ibm.wsspi.security.auth.callback package.

MAPPING_ALIAS = "com.ibm.mapping.authDataAlias"

When you specify the **Use default method** > **Select authentication data entry authentication** method on the **Map resource references to resources** panel, the administrative console automatically creates a MAPPING_ALIAS entry with the selected authentication data alias value in the mapping properties. If you choose to create your own custom login configuration and then use the default mapping LoginModule, you'll have to set this property manually on the mapping properties for the resource factory reference.

In a custom login module, you can use the WSSubject.getRunAsSubject() method to retrieve the subject that represents the identity of the current running thread. The identity of the current running thread is known as the*RunAs* identity. The *RunAs* subject typically contains a WSPrincipal in the principal set and a WSCredential in the public credential set. The subject instance that is created by your mapping module contains a Principal instance in the principals set and a PasswordCredential or an org.ietf.jgss.GSSCredential instance in the set of private credentials.

The GenericCredential interface that was defined in Java Cryptography Architecture (JCA) Spec Version 1.0 has been removed in the JCA Version 1.5 spec. The GenericCredentail interface is supported by WebSphere Application Server Version 6 to support older resource adapters that might have been programmed to the GenericCredential interface.

## Developing custom user registries

WebSphere Application Server security supports the use of custom registries in addition to Local OS and Lightweight Directory Access Protocol (LDAP) registries for authentication and authorization purposes. A custom user registry is a customer implemented user registry which implements the UserRegistry Java interface as provided by WebSphere Application Server. A custom implemented user registry can support virtually any type or notion of an accounts repository from a relational database, flat file, and so on. The custom user registry provides considerable flexibility in adapting WebSphere Application Server security to various environments where some notion of a user registry, other than LDAP or LocalOS, already exist in the operational environment.

Implementing a custom user registry is a software development effort. Use the methods defined in the UserRegistry interface to make calls to the desired registry to obtain user and group information. The

interface defines a very general set of methods, for encapsulating a wide variety of registries. You can configure a custom user registry as the active user registry when configuring WebSphere Application Server global security.

Make sure that your implementation of the custom registry does not depend on any WebSphere Application Server components such as data sources, enterprise beans, and so on. Do not have this dependency because security is initialized and enabled prior to most of the other WebSphere Application Server components during startup. If your previous implementation used these components, make a change that will eliminate the dependency. For example, if your previous implementation used data sources to connect to a database, use Java DataBase Connectivity (JDBC) to connect to the database.

For backward compatibility, the WebSphere Application Server Version 4 custom registry is also supported. Refer to the “Migrating custom user registries” on page 39 for more information on migrating. If your previous implementation uses data sources to connect to a database, change the implementation to use JDBC connections. However, it is recommended that you use the new interface to implement your custom registry.

1. If not familiar with the custom user registry concept, refer to the article, “Custom user registries” on page 211. This section explains each of the methods in the interface in detail and the changes for these methods from the version 4 release.
2. Implement all the methods in the interface except for the `CreateCredential` method, which is implemented by WebSphere Application Server. “FileRegistrySample.java file” on page 220 is provided for reference.
3. Build your implementation. You need the `%install_root%/lib/sas.jar` and `%install_root%/lib/wssec.jar` files in your class path. For example: `%install_root%\java\bin\javac -classpath %install_root%\lib\wssec.jar;%install_root%\lib\sas.jar yourImplementationFile.java`.
4. Copy the class files generated in the previous step to the product class path. The preferred location is the `%install_root%/lib/ext` directory. This should be copied to all the product processes (cell, all NodeAgents) class path.
5. Follow the steps in “Configuring custom user registries” on page 213 to configure your implementation using the administrative console. This step is required to implement custom user registries in Version 5.x or later.

If you enabling security, make sure you complete the remaining steps. Once this is done, make sure you save and synchronize the configuration and restart all the servers. Try accessing some J2EE resources to verify that the custom registry implementation is successful.

## Example: Custom user registries

A *custom user registry* is a customer-implemented user registry that implements the `UserRegistry` Java interface as provided by WebSphere Application Server. A custom-implemented user registry can support virtually any type or form of an accounts repository from a relational database, flat file, and so on. The custom user registry provides considerable flexibility in adapting WebSphere Application Server security to various environments where some form of a user registry, other than Lightweight Directory Access Protocol (LDAP) or Local OS, already exist in the operational environment.

Implementing a custom user registry is a software development effort. You must use the methods defined in the UserRegistry interface to make calls to the desired registry for obtaining user and group information. The interface defines a very general set of methods, so it can encapsulate a wide variety of registries. You can configure a custom user registry as the active user registry when configuring the product global security.

If you are using the WebSphere Application Server Version 4.x CustomRegistry interface, you can plug in your registry without any changes. However, using the new interface to implement your custom registry is recommended.

To view a sample custom registry, refer to the following files:
- "FileRegistrySample.java file" on page 220
- "users.props file" on page 239
- "groups.props file" on page 239

## UserRegistry interface methods

Implementing this interface enables WebSphere Application Server security to use custom registries. This capability should extend the `java.rmi` file. With a remote registry, you can complete this process remotely.

Implementation of this interface must provide implementations for:
- initialize(java.util.Properties)
- checkPassword(String,String)
- mapCertificate(X509Certificate[])
- getRealm
- getUsers(String,int)
- getUserDisplayName(String)
- getUniqueUserId(String)
- getUserSecurityName(String)
- isValidUser(String)
- getGroups(String,int)
- getGroupDisplayName(String)
- getUniqueGroupId(String)
- getUniqueGroupIds(String)
- getGroupSecurityName(String)
- isValidGroup(String)
- getGroupsForUser(String)
- getUsersForGroup(String,int)
- createCredential(String)

```
public void initialize(java.util.Properties props)
      throws CustomRegistryException,
             RemoteException;
```

This method is called to initialize the UserRegistry method. All the properties defined in the Custom User Registry panel propagate to this method.

For the sample, the initialize method retrieves the names of the registry files containing the user and group information.

This method is called during server bring up to initialize the registry. This method is also called when validation is performed by the administrative console, when security is on. This method remains the same as in version 4.x.

```
public String checkPassword(String userSecurityName, String password)
      throws PasswordCheckFailedException,
             CustomRegistryException,
             RemoteException;
```

The `checkPassword` method is called to authenticate users when they log in using a name (or user ID) and a password. This method returns a string which, in most cases, is the user being authenticated. Then, a credential is created for the user for authorization purposes. This user name is also returned for the enterprise bean call, `getCallerPrincipal()`, and the servlet calls, `getUserPrincipal()` and `getRemoteUser()`. See the `getUserDisplayName` method for more information if you have display names in your registry. In some situations, if you return a user other than the one who is logged in, verify that the user is valid in the registry.

For the sample, the `mapCertificate` method gets the distinguished name (DN) from the certificate chain and makes sure it is a valid user in the registry before returning the user. For the sample, the checkPassword method checks the name and password combination in the registry and (if they match) returns the user being authenticated.

This method is called for various scenarios. It is called by the administrative console to validate the user information once the registry is initialized. It is also called when you access protected resources in the product for authenticating the user and before proceeding with the authorization. This method is the same as in version 4.x.

```
public String mapCertificate(X509Certificate[] cert)
      throws CertificateMapNotSupportedException,
             CertificateMapFailedException,
             CustomRegistryException,
             RemoteException;
```

The `mapCertificate` method is called to obtain a user name from an X.509 certificate chain supplied by the browser. The complete certificate chain is passed to this method and the implementation can validate the chain if needed and get the user information. A credential is created for this user for authorization purposes. If browser certificates are not supported in your configuration, you can throw the exception, CertificateMapNotSupportedException. The consequence of not supporting certificates is authentication failure if the challenge type is certificates, even if valid certificates are in the browser.

This method is called when certificates are provided for authentication. For Web applications, when the authentication constraints are set to CLIENT-CERT in the `web.xml` file of the application, this method is called to map a certificate to a valid user in the registry. For Java clients, this method is called to map the client certificates in the transport layer, when using the transport layer authentication. Also, when the Identity Assertion Token (when using the CSIv2 authentication protocol) is set to contain certificates, this method is called to map the certificates to a valid user.

In WebSphere Application Server Version 4.x, the input parameter was the X509Certificate. In WebSphere Application Server Version 5.x and later, this parameter changes to accept an array of X509Certificate certificates (such as a certificate chain). In version 4.x, this parameter was called only for Web applications, but in version 5.x and lateryou can call this method for both Web and Java clients.

```
public String getRealm()
      throws CustomRegistryException,
             RemoteException;
```

The getRealm method is called to get the name of the security realm. The name of the realm identifies the security domain for which the registry authenticates users. If this method returns a null value, a default name of `customRealm` is used.

For the sample, the getRealm method returns the string, `customRealm`. One of the calls to this method is when the registry information is validated. This method is the same as in version 4.x.

```
public Result getUsers(String pattern, int limit)
      throws CustomRegistryException,
             RemoteException;
```

The `getUsers` method returns the list of users from the registry. The names of users depend on the pattern parameter. The number of users are limited by the limit parameter. In a registry that has many users, getting all the users is not practical. So the limit parameter is introduced to limit the number of users retrieved from the registry. A limit of 0 indicates to return all the users that match the pattern and might cause problems for large registries. Use this limit with care.

The custom registry implementations are expected to support at least the wildcard search (*). For example, a pattern of (*) returns all the users and a pattern of (b*) returns the users starting with *b*.

The return parameter is an object of type `com.ibm.websphere.security.Result`. This object contains two attributes, a `java.util.List` and a `java.lang.boolean`. The list contains the users returned and the Boolean flag indicates if there are more users available in the registry for the search pattern. This Boolean flag is used to indicate to the client whether more users are available in the registry.

In the sample, the `getUsers` retrieves the required number of users from the registry and sets them as a list in the result object. To find out if there are more users than requested, the sample gets one more user than requested and if it finds the additional user, it sets the Boolean flag to `true`. For pattern matching, the match method in the `RegExpSample` class is used, which supports wildcard characters such as the asterisk (*) and question mark (?).

This method is called by the administrative console to add users to roles in the various map users to roles panels. The administrative console uses the Boolean set in the result object to indicate that more entries matching the pattern are available in the registry.

In WebSphere Application Server Version 4.x, this method specifies to take only the pattern parameter. The return is a list. In WebSphere Application Server Version 5.x or later, this method is changed to take one additional parameter, the limit. Ideally, your implementation should change to take the limit value and limit the users returned. The return is changed to return a result object, which consists of the list (as in version 4) and a flag indicating if more entries exist. So, when the list returns, use the `Result.setList(List)` to set the List in the result object. If there are more entries than requested in the limit parameter, set the Boolean attribute to `true` in the result object, using `Result.setHasMore()` method. The default for the Boolean attribute in the result object is `false`.

```
public String getUserDisplayName(String userSecurityName)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;
```

The `getUserDisplayName` method returns a display name for a user, if one exists. The display name is an optional string that describes the user that you can set in some registries. This is a descriptive name for the user and does not have to be unique in the registry.

For example in Windows systems, you can display the full name of the user.

If you do not need display names in your registry, return null or an empty string for this method.

**Note:** In WebSphere Application Server Version 4.x, if display names existed for any user these names were useful for the EJB method call `getCallerPrincipal()` and the servlet calls `getUserPrincipal()` and `getRemoteUser()`. If the display names were not the same as the security name for any user, the display names are returned for the previously mentioned enterprise beans and servlet methods. Returning display names for these methods might become problematic is some situations because the display names might not be unique in the registry. Avoid this problem by changing the default behavior to return the user's security name instead of the user's display name in this version of the product. However, if you want to have the same behavior as in Version 4, set the property `WAS_UseDisplayName` to `true` in the **Custom Registry Properties** panel in the administrative console. For more information on how to set properties for the custom registry, see the section on *Setting Properties for Custom Registries*.

In the sample, this method returns the display name of the user whose name matches the user name provided. If the display name does not exist this returns an empty string.

This method can be called by the product to present the display names in the administrative console, or using the command line using the `wsadmin` tool. Use this method only for displaying. This method is the same as in Version 4.0.

```
public String getUniqueUserId(String userSecurityName)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;
```

This method returns the unique ID of the user given the security name.

In the sample, this method returns the uniqueId of the user whose name matches the supplied name. This method is called when forming a credential for a user and also when creating the authorization table for the application.

```
public String getUserSecurityName(String uniqueUserId)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;
```

This method returns the security name of a user given the unique ID. In the sample, this method returns the security name of the user whose unique ID matches the supplied ID.

This method is called to make sure a valid user exists for a given uniqueUserId. This method is called to get the security name of the user when the uniqueUserId is obtained from a token.

```
public boolean isValidUser(String userSecurityName)
      throws CustomRegistryException,
             RemoteException;
```

This method indicates whether the given user is a valid user in the registry.

In the Sample, this method returns `true` if the user is found in the registry, otherwise this method returns `false`. This method is primarily called in situations where knowing if the user exists in the directory prevents problems later. For example, in the mapCertificate call, once the name is obtained from the certificate if the user is found to be an invalid user in the registry, you can avoid trying to create the credential for the user.

```
 public Result getGroups(String pattern, int limit)
      throws CustomRegistryException,
             RemoteException;
```

The `getGroups` method returns the list of groups from the registry. The names of groups depend on the pattern parameter. The number of groups is limited by the limit parameter. In a registry that has many groups, getting all the groups is not practical. So, the limit parameter is introduced to limit the number of groups retrieved from the registry. A limit of 0 implies to return all the groups that match the pattern and can cause problems for large registries. Use this limit with care. The custom registry implementations are expected to support at least the wildcard search (*). For example, a pattern of (*) returns all the users and a pattern of (b*) returns the users starting with *b*.

The return parameter is an object of type `com.ibm.websphere.security.Result`. This object contains two attributes, a `java.util.List` and a `java.lang.boolean`. The list contains the groups returned and the Boolean flag indicates whether there are more groups available in the registry for the pattern searched. This Boolean flag is used to indicate to the client if more groups are available in the registry.

In the sample, the getUsers retrieves the required number of groups from the registry and sets them as a list in the result object. To find out if there are more groups than requested, the sample gets one more user than requested and if it finds the additional user, it sets the Boolean flag to true. For pattern matching, the match method in the RegExpSample class is used. It supports wildcards like **, ?*.

This method is called by the administrative console to add groups to roles in the various map groups to roles panels. The administrative console will use the boolean set in the Result object to indicate that more entries matching the pattern are available in the registry.

In WebSphere Application Server Version 4, this method is used to take the pattern parameter only and returns a list. In WebSphere Application Server Version 5.x or later, this method is changed to take one additional parameter, the limit. Change to take the limit value and limit the users returned. The return is changed to return a result object, which consists of the list (as in version 4) and a flag indicating whether more entries exist. Use the Result.setList(List) to set the list in the result object. If there are more entries than requested in the limit parameter, set the Boolean attribute to true in the result object using Result.setHasMore(). The default for the Boolean attribute in the result object is false.

```
public String getGroupDisplayName(String groupSecurityName)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;
```

The getGroupDisplayName method returns a display name for a group if one exists. The display name is an optional string describing the group that you can set in some registries. This name is a descriptive name for the group and does not have to be unique in the registry. If you do not need to have display names for groups in your registry, return null or an empty string for this method.

In the sample, this method returns the display name of the group whose name matches the group name provided. If the display name does not exist, this method returns an empty string.

The product can call this method to present the display names in the administrative console or through command line using the wsadmin tool. This method is only used for displaying.

```
public String getUniqueGroupId(String groupSecurityName)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;
```

This method returns the unique ID of the group given the security name.

In the sample, this method returns the unique ID of the group whose name matches the supplied name. This method is called when creating the authorization table for the application.

```
public List getUniqueGroupIds(String uniqueUserId)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;
```

This method returns the unique IDs of all the groups to which a user belongs.

In the sample, this method returns the unique ID of all the groups that contain this uniqueUserID. This method is called when creating the credential for the user. As part of creating the credential, all the groupUniqueIds in which the user belongs are collected and put in the credential for authorization purposes when groups are given access to a resource.

```
public String getGroupSecurityName(String uniqueGroupId)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;
```

This method returns the security name of a group given its unique ID.

In the sample, this method returns the security name of the group whose unique ID matches the supplied ID. This method verifies that a valid group exists for a given uniqueGroupId.

```
public boolean isValidGroup(String groupSecurityName)
      throws CustomRegistryException,
             RemoteException;
```

This method indicates if the given group is a valid group in the registry.

In the sample, this method returns `true` if the group is found in the registry, otherwise the method returns `false`. This method can be used in situations where knowing whether the group exists in the directory might prevent problems later.

```
public List getGroupsForUser(String userSecurityName)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;
```

This method returns all the groups to which a user belongs whose name matches the supplied name. This method is similar to the `getUniqueGroupIds` method with the exception that the security names are used instead of the unique IDs.

In the sample, this method returns all the group security names that contain the userSecurityName.

This method is called by the administrative console or the scripting tool to verify that the users entered for the `RunAs` roles are already part of that role in the users and groups to role mapping. This check is required to ensure that a user cannot be added to a `RunAs` role unless that user is assigned to the role in the users and groups to role mapping either directly or indirectly (through a group that contains this user). Since a group in which the user belongs can be part of the role in the users and groups to role mapping, this method is called to check if any of the groups that this user belongs to mapped to that role.

```
public Result getUsersForGroup(String groupSecurityName, int limit)
      throws NotImplementedException,
             EntryNotFoundException,
             CustomRegistryException,
             RemoteException;
```

This method retrieves users from the specified group. The number of users returned is limited by the limit parameter. A limit of **0** indicates to return all the users in that group. This method is not directly called by the WebSphere Application Server security component. However, this can be called by other components. For example, this method issued by the process choreographer when staff assignments are modeled using groups. In rare situations, if you are working with a registry where getting all the users from any of your groups is not practical (for example, if there are a large number of users), you can throw the `NotImplementedException` exception for the particular groups. In this case, verify that if the process choreographer is installed (or if it is installed later) the staff assignments are not modeled using these particular groups. If there is no concern about returning the users from groups in the registry, it is recommended that you do not throw the `NotImplemented` exception when implementing this method.

The return parameter is an object of type `com.ibm.websphere.security.Result`. This object contains two attributes, `java.util.List` and `java.lang.boolean`. The list contains the users returned and the Boolean flag, which indicates whether there are more users available in the registry for the search pattern. This Boolean flag indicates to the client whether users are available in the registry.

In the example, this method gets one user more than the requested number of users for a group if the limit parameter is not set to **0**. If it succeeds in getting one more user, it sets the Boolean flag to `true`.

In WebSphere Application Server Version 4, this getUsers method was mandatory for the product. For WebSphere Application Server Version 5.x or later, this method can throw the exception `NotImplementedException` exception in situations where it is not practical to get the requested set of users. However, this exception should be thrown in rare situations, as other components can be affected. In version 4, this method accepted only the pattern parameter and the returned a list. In version 5, this method accepts one additional parameter, the limit. Change your implementation to take the limit value and limit the users returned. The return changes to return a result object, which consists of the list (as in version 4) and a flag indicating whether more entries exist. When the list is returned, use the `Result.setList(List)` method to set the list in the Result object. If there are more entries than requested in the limit parameter, set the Boolean attribute to `true` in the result object using `Result.setHasMore()`. The default for the Boolean attribute in the Result object is `false`.

**Attention:** The first two lines of the following code sample is one continuous line.

```
public com.ibm.websphere.security.cred.WSCredential
        createCredential(String userSecurityName)
     throws NotImplementedException,
            EntryNotFoundException,
            CustomRegistryException,
            RemoteException;
```

In this release of the WebSphere Application Server, this method is not called. You can return *null*. In the example, a *null* is returned.

## Trust association interceptor support for Subject creation

The new Trust Association Interceptor (TAI) interface, com.ibm.wsspi.security.tai.TrustAssociationInterceptor, supports several new features and is different from the existing com.ibm.websphere.security.TrustAssociationInterceptor interface. Although the existing interface is still supported, it is being deprecated in a future release.

The new TAI interface supports a multi-phase, negotiated authentication process. For example, some systems require a challenge response protocol back to the client. The two key methods in this new interface are:

**Key method name**

public boolean isTargetInterceptor (HttpServletRequest req)

The isTargetInterceptor method determines whether the request originated with the proxy server associated with the interceptor. The implementation code must examine the incoming request object and determine if the proxy server forwarding the request is a valid proxy server for this interceptor. The result of this method determines whether the interceptor processes the request.

**Method result**

A `true` value tells WebSphere Application Server to have the TAI handle the request.

A `false` value, tells WebSphere Application Server to ignore the TAI.

The negotiateValidateandEstablishTrust method determines whether to trust the proxy server from which the request originated. The implementation code must authenticate the proxy server. The authentication mechanism is proxy-server specific. For example, in the product implementation for the WebSEAL server, this method retrieves the basic authentication information from the HTTP header and validates the information against the user registry used by WebSphere Application Server. If the credentials are invalid, the code throws the WebTrustAssociationException, which indicates that the proxy server is not trusted and the request is denied. If the credentials are valid, the code returns a TAIResult, which indicates the status of the request processing along with the client identity (Subject and principal name) to be used for authorizing the Web resource.

**Key method name**
public TAIResult negotiateValidateandEstablishTrust (HttpServletRequest req, HttpServletResponse res)

**Method result**
Returns a TAIResult, which indicates the status of the request processing. The request object can be queried and the response object can be modified.

The TAIResult class has three static methods for creating a TAIResult. The TAIResult create methods take an int type as the first parameter. WebSphere Application Server expects the result to be a valid HTTP request return code and is interpreted in one of the following ways:

- If the value is `HttpServletResponse.SC_OK`, this response tells WebSphere Application Server that the TAI has completed its negotiation. The response also tells WebSphere Application Server use the information in the TAIResult to create a user identity.
- Other values tell WebSphere Application Server to return the TAI output, which is placed into the HttpServletResponse, to the Web client. Typically, the Web client provides additional information and then places another call to the TAI.

The created TAIResults have the following meanings:

| TAIResult | Explanation |
|---|---|
| public static TAIResult create(int status); | Indicates a status to WebSphere Application Server. The status should not be SC_OK because the identity information is provided. |
| public static TAIResult create(int status, String principal); | Indicates a status to WebSphere Application Server and provides the user ID or the unique ID for this user. WebSphere Application Server creates credentials by querying the user registry. |
| public static TAIResult create(int status, String principal, Subject subject); | Indicates a status to WebSphere Application Server, the user ID or the unique ID for the user, and a custom Subject. If the Subject contains a Hashtable, the principal is ignored. The contents of the Subject becomes part of the eventual user Subject. |

All of the following examples are within the negotiateValidateandEstablishTrust() method of a TAI.

The following code sample indicates that additional negotiation is required:

```
// Modify the HttpServletResponse object
 //  The response code is meaningful only on the client
 return TAIResult.create(HttpServletResponse.SC_CONTINUE);
```

The following code sample indicates that the TAI has determined the user identity. WebSphere Application Server receives the user ID only and then it queries the user registry for additional information:

```
// modify the HttpServletResponse object
 return TAIResult.create(HttpServletResponse.SC_OK, userid);
```

The following code sample indicates that the TAI had determined the user identity. WebSphere Application Server receives the complete user information that is contained in the Hashtable. For more information on the Hashtable, see “Configuring inbound identity mapping” on page 261. In this code sample, the Hashtable is placed in the public credential portion of the Subject:

```
// create Subject and place Hashtable in it
 Subject subject = new Subject;
 subject.getPublicCredentials().add(hashtable);
 //the response code is meaningful only the client
 return TAIResult.create(HttpServletResponse.SC_OK, "ignored", subject);
```

The following code sample indicates that there is an authentication failure. WebSphere Application Server fails the authentication request:

```
//log error message
 // ....
 throw new WebTrustAssociationFailedException("TAI failed for this reason");
```

There are a few additional methods on the TrustAssociationInterceptor interface that are discussed in the Java documentation. These methods are used for initialization, shut down, and for identifying the TAI to WebSphere Application Server.

# Chapter 9. Assembling secured applications

There are several assembly tools that are graphical user interfaces for assembling enterprise (J2EE) applications. You can use these tools to assemble an application and secure EJB and Web modules in that application. An EJB module consists of one or more beans. You can enforce security at the EJB method level. A Web module consists of one or more Web resources (an HTML page, a JSP file or a servlet). You can also enforce security for each Web resource. You can use an assembly tool to secure an EJB module (Java archive (JAR) file) or a Web module (Web archive (WAR) file) or an application (enterprise archive (EAR) file). You can create an application, an EJB module, or a Web Module and secure them using an assembly tool or development tools like the IBM Rational Application Developer.

1. Secure EJB applications using an assembly tool. For more information, see “Securing enterprise bean applications” on page 116.
2. Secure Web applications using an assembly tool. For more information, see “Securing Web applications using an assembly tool” on page 118.
3. Add users and groups to roles while assembling secured application using an assembly tool. For more information, see “Adding users and groups to roles using an assembly tool” on page 124.
4. Map users to RunAs roles using an assembly tool. For more information, see “Mapping users to RunAs roles using an assembly tool” on page 125.
5. “Adding the was.policy file to applications” on page 486.
6. Assemble the application components that you just secured using an assembly tool. For more information, see Assembling applications.

After securing an application, the resulting `.ear` file contains security information in its deployment descriptor. The EJB module security information is stored in the `ejb-jar.xml` file and the Web module security information is stored in the `web.xml` file. The `application.xml` file of the application EAR file contains all the roles used in the application. The user and group to roles mapping is stored in the `ibm-application-bnd.xmi` file of the application EAR file.

The `was.policy` file of the application EAR contains the permissions granted for the application to access system resources.

This task is required to secure EJB modules and Web modules in an application. This task is also required for applications to run properly when Java 2 security is enabled. If the `was.policy` file is not created and it does not contain required permissions, the application might not be able to access system resources.

After securing an application, you can install an application using the administrative console. When you install a secured application, refer to the Chapter 10, “Deploying secured applications,” on page 127 article to complete this task.

## Enterprise bean component security

An EJB module consists of one or more beans. You can use development tools such as Rational Web Developer to develop an EJB module. You can also enforce security at the EJB method level.

You can assign a set of EJB methods to a set of one or more roles. When an EJB method is secured by associating a set of roles, grant at least one role in that set so that you can access that method. To exclude a set of EJB methods from being accessed by anyone mark them **excluded**. You can give everyone access to a set of enterprise beans method by clearing those methods. You can run enterprise beans as a different identity (runAs identity) before invoking other enterprise beans.

© Copyright IBM Corp. 2004

## Securing enterprise bean applications

You can protect enterprise bean methods by assigning security roles to them. Before you assign security roles, you need to know which EJB methods need protecting and how to protect them.

1. In an assembly tool, import your EJB JAR file or an application archive (EAR) file that contains one or more Web modules. For more information, see the Importing EJB files article or the Importing enterprise applications article.
2. In the Project Explorer , click the **EJB Projects** directory and click the name of your application.
3. Right-click the Deployment descriptor and select **Open with > Deployment Descriptor Editor**. If you selected an EJB `.jar` file, an EJB deployment descriptor editor opens. If you selected an application `.ear` file, an application deployment descriptor editor opens. To see online information about the editor, press **F1** and click the editor name.
4. Create security roles. You can create security roles at the application level or at the EJB module level. If you create a security role at the EJB module level, the role displays in the application level. If a security role is created at the application level, the role does not appear in all the EJB modules. You can copy and paste one or more EJB module security roles that you create at application level:
   - Create a role at an EJB module level. In an EJB deployment descriptor editor, select the **Assembly** tab. Under **Security Roles**, click **Add**. In the Add Security Role wizard, name and describe the security role; then click **Finish**.
   - Create a role at the application level. In an application deployment descriptor editor, select the **Security** tab. Under the list of security roles, click **Add**. In the Add Security Role wizard, name and describe the security role; then click **Finish**.
5. Create method permissions. Method permissions map one or more methods to a set of roles. An enterprise bean has four types of methods: Home methods, Remote methods, LocalHome methods and Local methods. You can add permissions to enterprise beans on the method level. You cannot add a method permission to an enterprise bean unless you already have one or more security roles defined. For Version 2.0 EJB projects, there is an unchecked option that specifies that the selected methods from the selected beans do not require authorization to execute. To add a method permission to an enterprise bean:
   a. On the **Assembly** tab of an EJB deployment descriptor editor, under **Method Permissions**, click **Add**. The Add Method Permission wizard opens.
   b. Select a security role from the list of roles found and click **Next**.
   c. Select one or more enterprise beans from the list of beans found. You can click **Select All** or **Deselect All** to select or deselect all of the enterprise beans in the list. Click **Next**.
   d. Select the methods that you want to bind to your security role. The Method Elements page lists all methods associated with the enterprise bean(s). You can click **Apply to All** or **Deselect All** to quickly select or clear multiple methods. It selects only the `*` method for each bean. Creating a method permission for the exact method signature overrides the default (`*`) method permission setting. The `*` method represents all methods within the bean. There are `*` for each interface as well. By not selecting all of the individual methods in the tree, you can set other permissions on the remaining methods.
   e. Click **Finish**.

   After the method permission is created, you can see the new method permission in the tree. Expand the tree to see the bean and methods defined in the method permission.
6. Exclude user access to methods. Users cannot access excluded methods. Any method in the enterprise beans that is not assigned to a role or is not excluded, is deselected during the application installation by the deployer.
   a. On the **Assembly** tab of an EJB deployment descriptor editor, under **Excludes List**, click **Add**. The Exclude List wizard opens.
   b. Select one or more enterprise beans from the list of beans found and click **Next**.
   c. Select one or more of the method elements for the security identity and click **Finish**.

7. Map the security-role-ref and role-name to the role-link. When developing enterprise beans, you can create the security-role-ref element. The security-role-ref element contains only the role-name field. The role-name field determines if the caller is in a specified `role(isCallerInRole())` and contains the name of the role that is referenced in the code. Since you create security roles during the assembly stage, the developer uses a *logical rolename* in the **role-name** field and provides enough information in the **description** field for the assembler to map the actual role (role-link). The security-role-ref element is located at the EJB level. Enterprise beans can have zero or more security-role-ref elements.
   a. On the **Reference** tab of an EJB deployment descriptor editor, under the list of references, click **Add**. The Add Reference wizard opens.
   b. Select **Security role reference** and click **Next**.
   c. Name the security role reference, select a security role to link the reference to, describe the security role reference, and click **Finish**.
   d. Map every role-name used during development to the role (role-link) using the previous steps.
8. Specify the RunAs Identity for enterprise beans components. The RunAs Identity of the enterprise bean is used to invoke the next enterprise beans in the chain of EJB invocations. When the next enterprise beans are invoked, the `RunAsIdentity` passes to the next enterprise beans for performing an authorization check on the next enterprise bean. If the RunAs Identity is not specified, the client identity is propagated to the next enterprise bean. The RunAs Identity can represent each of the enterprise beans or can represent each method in the enterprise beans.
   a. On the **Access** tab of an EJB deployment descriptor editor, next to the **Security Identity (Bean Level)** field, click **Add**. The Add Security Identity wizard opens.
   b. Select the appropriate run as mode, describe the security identity, and click **Next**. Select the **Use identity of caller** mode to instruct the security service to not make changes to the credential settings for the principal. Select the **Use identity assigned to specific role (below)** mode to use a principal that has been assigned to the specified security role for running the bean methods. This association is part of the application binding in which the role is associated with the user ID and password of a user who is granted that role. If you select the **Use identity assigned to specific role (below)** mode , you must specify a role name and role description.
   c. Select one or more enterprise beans from the list of beans found and click **Next**. If **Next** is unavailable, click **Finish**.
   d. Optional: On the Method Elements page, select one or more of the method elements for the security identity and click **Finish**.
9. Close the deployment descriptor editor and, when prompted, click **Yes** to save the changes.

After securing an EJB application, the resulting `.jar` file contains security information in its deployment descriptor. The security information of the EJB modules is stored in the `ejb-jar.xml` file.

After securing an EJB application using an assembly tool, you can install the EJB application using the administrative console. During the installation of a secured EJB application, follow the steps in the Chapter 10, “Deploying secured applications,” on page 127 article to complete the task of securing the EJB application.

## Web component security

A Web module consists of servlets, JSP files, server-side utility classes, static Web content (HTML, images, sound files, cascading style sheets (CSS)), and client-side classes (applets). You can use development tools such as Rational Application Developer to develop a Web module and enforce security at the method level of each Web resource.

You can identify a Web resource by its URI pattern. A Web resource method can be any HTTP method (GET, POST, DELETE, PUT, for example). You can group a set of URI patterns and a set of HTTP methods together and assign this grouping a set of roles. When a Web resource method is secured by associating a set of roles, grant a user at least one role in that set to access that method. You can exclude

anyone from accessing a set of Web resources by assigning an empty set of roles. A servlet or a JSP file can run as different identities (RunAs identity) before invoking another enterprise bean component. All the secured Web resources require the user to log in by using a configured login mechanism. There are three types of Web login authentication mechanisms: basic authentication, form-based authentication and client certificate-based authentication.

For more detailed information on Web security see the product architectural overview article.

## Securing Web applications using an assembly tool

There are three types of Web login authentication mechanisms that you can configure on a Web application: basic authentication, form-based authentication and client certificate-based authentication. Protect Web resources in a Web application by assigning security roles to those resources.

To secure Web applications, determine the Web resources that need protecting and determine how to protect them.

1. In an assembly tool, import your Web archive (WAR) file or an application archive (EAR) file that contains one or more Web modules. For more information, see the Importing WAR files article or the Importing enterprise applications.
2. In the Project Explorer, locate your Web application.
3. Right-click the deployment descriptor and select **Open With > Deployment Descriptor Editor**. The Deployment Descriptor window opens. To see online information about the editor, press F1 and click the editor name. If you selected Web archive (WAR) file, a Web deployment descriptor editor opens. If you selected an enterprise application (EAR) file, an application deployment descriptor editor opens.
4. Create security roles either at the application level or at Web module level. If a security role is created at the Web module level, the role also displays in the application level. If a security role is created at the application level, the role does not display in all the Web modules. You can copy and paste a security role at the application level to one or more Web module security roles.
   - Create a role at a Web-module level. In a Web deployment descriptor editor, select the **Security** tab. Under **Security Roles**, click **Add.**. Enter the security role name, describe the security role, and click **Finish**.
   - Create a role at the application level. In an application deployment descriptor editor, select the **Security** tab. Under the list of security roles, click **Add**. In the Add Security Role wizard, name and describe the security role; then click **Finish**.
5. Create security constraints. Security constraints are a mapping of one or more Web resources to a set of roles.
   a. On the **Security** tab of a Web deployment descriptor editor, click **Security Constraints**. On the Security Constraints tab that opens, you can do the following:
      - Add or remove security constraints for specific security roles.
      - Add or remove Web resources and their HTTP methods.
      - Define which security roles are authorized to access the Web resources.
      - Specify None, Integral, or Confidential constraints on user data. *None* means that the application requires no transport guarantees. *Integral* means that data cannot be changes in transit between client and server. And *Confidential* means that data content cannot be observed while it is in transit. Integral and Confidential usually require the use of SSL.

   b. Under **Security Constraints**, click **Add**.
   c. Under **Constraint name**, specify a display name for the security constraint and click **Next**.
   d. Type a name and description for the Web resource collection.
   e. Select one or more HTTP methods. The HTTP method options are: GET, PUT, HEAD, TRACE, POST, DELETE, and OPTIONS.
   f. Beside the **Patterns** field, click **Add**.

g. Specify a URL Pattern. For example, type - `/*, *.jsp, /hello`. Consult the Servlet specification Version 2.4 for instructions on mapping URL patterns to servlets. Security run time uses the exact match first to map the incoming URL with URL patterns. If the exact match is not present, the security run time uses the longest match. The wild card (`*.,*.jsp`) URL pattern matching is used last.

h. Click **Finish**.

i. Repeat these steps to create multiple security constraints.

6. Map security-role-ref and role-name elements to the role-link element. During the development of a Web application, you can create the security-role-ref element. The security-role-ref element contains only the role-name field at this stage. The role-name field contains the name of the role that is referenced in the servlet or JSP code to determine if the caller is in a specified role (isUserInRole()). Since security roles are created during the assembly stage, the developer uses a logical role name in the **role-name** field and provides enough description in the **description** field for the assembler to map the role actual (role-link). The Security-role-ref element is at the servlet level. A servlet or JSP file can have zero or more security-role-ref elements.

a. Go to the **References** tab of a Web deployment descriptor editor. On the **References** tab, you can add or remove the name of an enterprise bean reference to the deployment descriptor. There are 5 types of references you can define on this tab:
   - EJB reference
   - Service reference
   - Resource reference
   - Message destination reference
   - Security role reference
   - Resource environment reference

b. Under the list of EJB references, click **Add**.

c. Specify a name and a type for the reference in the **Name** and **Ref Type** fields.

d. Select either **Enterprise Beans in the workplace** or **Enterprise Beans not in the workplace**.

e. Optional: If you select **Enterprise Beans not in the workplace**, select the type of enterprise bean in the **Type** field. You can specify either an entity bean or a session bean.

f. Optional: Click **Browse** to specify values for the local home and local interface in the **Local home** and **Local** fields before you click **Next**.

g. Map every role-name used during development to the role (role-link) using the previous steps. Every role name used during development maps to the actual role.

7. Specify the RunAs identity for servlets and JSP files. The RunAs identity of a servlet is used to invoke enterprise beans from within the servlet code. When enterprise beans are invoked, the RunAs identity is passed to the enterprise bean for performing an authorization check on the enterprise beans. If the RunAs identity is not specified, the client identity is propagated to the enterprise beans. The RunAs identity is assigned at the servlet level.

a. On the **Servlets** tab of a Web deployment descriptor editor, under **Servlets and JSPs**, click **Add**. The Add Servlet or JSP wizard opens.

b. Specify the servlet or JavaServer page (JSP) settings including the name, initialization parameters, and URL mappings and click **Next**.

c. Specify the class file destination.

d. Click **Next** to specify additional settings or click **Finish**.

e. Under **Run As** on the **Servlets** tab, select the security role and describe the role.

f. Specify a RunAs identity for each servlet and JSP file used by your Web application.

8. Configure the login mechanism for the Web module. This configured login mechanism applies to all the servlets, JavaServer page (JSP) files and HTML resources in the Web module.

a. On the **Pages** tab of a Web deployment descriptor editor, under **Login**, select the required authentication method. Available method values include: Unspecified, Basic, Digest, Form, and Client-Cert.[

b. Specify a realm name.
c. If you select the Form authentication method, select a login page and an error page URLs (for example: /login.jsp and /error.jsp). The specified login and error pages are present in the .war file.
d. Install the client certificate on the browser or Web client and place the client certificate in the server trust keyring file, if ClientCert is selected.

9. Close the deployment descriptor editor and, when prompted, click **Yes** to save the changes.

After securing a Web application, the resulting WAR file contains security information in its deployment descriptor. The Web module security information is stored in the web.xml file. When you work in the Web deployment descriptor editor, you also can edit other deployment descriptors in the Web project, including information on bindings and IBM extensions in the ibm-web-bnd.xmi and ibm-web-ext.xmi files.

After using an assembly tool to secure a Web application, you can install the Web application using the administrative console. During the Web application installation, complete the steps in the Chapter 10, “Deploying secured applications,” on page 127 article to finish securing the Web application.

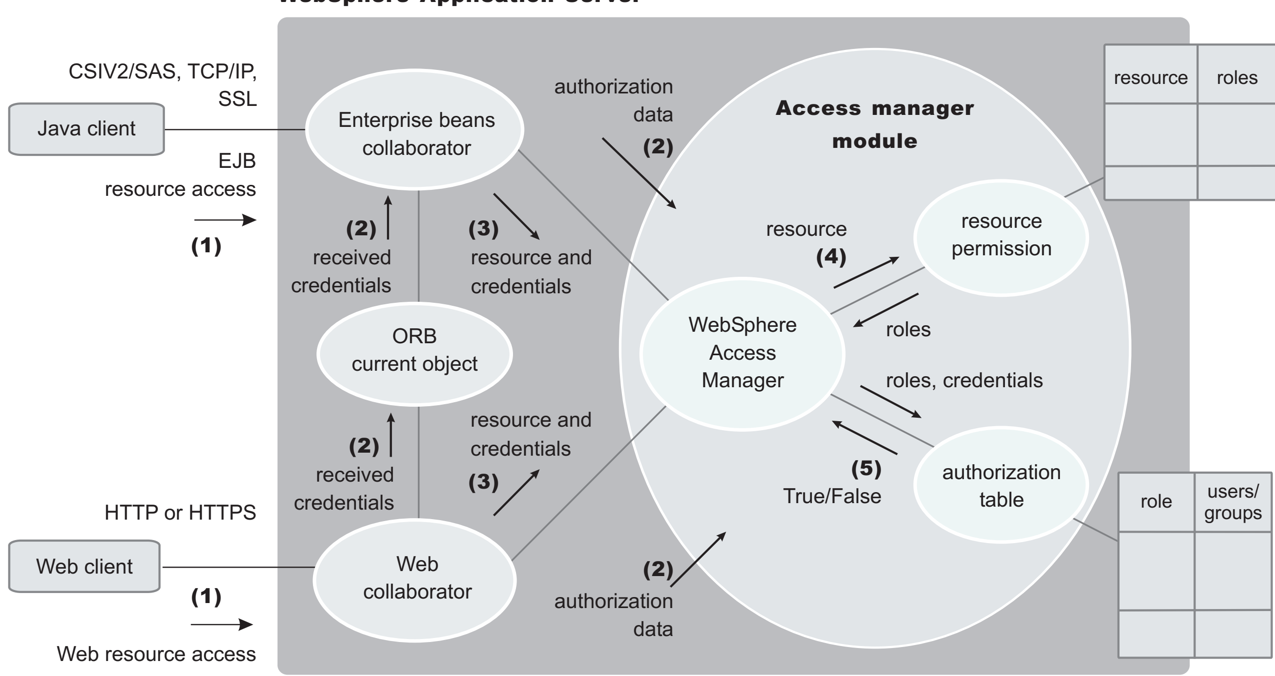
## Role-based authorization

Use authorization information to determine whether a caller has the necessary privileges to request a service.

The following figure illustrates the process used during authorization. Web resource access from a Web client is handled by a Web collaborator. The EJB resource access from a Java client (can be enterprise beans or a servlet) is handled by an EJB Collaborator. The EJB collaborator and the Web collaborator extract the client credentials from the object request broker (ORB) current object. The client credentials are set during the authentication process as received credentials in the ORB Current. The resource and the received credentials are presented to WSAccessManager to check whether access is permitted to the client for accessing the requested resource.

The access manager module contains two main modules:
- Resource permission module helps determine the required roles for a given resource. It uses a resource to roles mapping table that is built by the security run time during application startup. To build the resource-to-role mapping table, the security run time reads the deployment descriptor of the enterprise beans or the Web module (ejb-jar.xml or web.xml)
- Authorization table module consults a role to user or group table to determine whether a client is granted one of the required roles. The role to user or group mapping table, also known as the *authorization table*, is created by the security run time during application startup.
  – To build the authorization table, the security run time reads the application binding file (ibm-application-bnd.xmi file).

**Authentication**




Use authorization information to determine whether a caller has the necessary privilege to request a service. You can store authorization information many ways. For example, with each resource, you can store an *access-control list*, which contains a list of users and user privileges. Another way to store the information is to associate a list of resources and the corresponding privileges with each user. This list is called a *capability list*.

WebSphere Application Server uses the Java 2 Enterprise Edition (J2EE) authorization model. In this model, authorization information is organized as follows:

- During the assembly of an application, permission to invoke methods is granted to one or more roles. A role is a set of permissions; for example, in a banking application, roles can include teller, supervisor, clerk, and other industry-related positions. The teller role is associated with permissions to run methods related to managing the money in an account, such as the withdraw and deposit methods. The teller role is not granted permission to close accounts; this permission is given to the supervisor role. The application assembler defines a list of method permissions for each role; this list is stored in the deployment descriptor for the application.

There are two *special subjects* that are not defined by the J2EE model, but are worth understanding: AllAuthenticatedUsers and Everyone. A special subject is a product-defined entity independent of the user registry. It is used to generically represent a class of users or groups in the registry.

- AllAuthenticatedUsers is a special subject that permits all authenticated users to access protected methods. As long as the user can authenticate successfully, the user is permitted access to the protected resource.
- Everyone is a special subject that permits unrestricted access to a protected resource. Users do not have to authenticate to get access; this special subject provides access to protected methods as if the resources are unprotected.

During the deployment of an application, real users or groups of users are assigned to the roles. When a user is assigned to a role, the user gets all the method permissions that are granted to that role.

The application deployer does not need to understand the individual methods. By assigning roles to methods, the application assembler simplifies the job of the application deployer. Instead of working with a set of methods, the deployer works with the roles, which represent semantic groupings of the methods.

Users can be assigned to more than one role; the permissions granted to the user are the union of the permissions granted to each role. Additionally, if the authentication mechanism supports the grouping of users, these groups can be assigned to roles. Assigning a group to a role has the same effect as assigning each individual user to the role.

A best practice during deployment is to assign groups, rather than individual users to roles for the following reasons:
- Improves performance during the authorization check. Typically far fewer groups exist than users.
- Provides greater flexibility, by using group membership to control resource access.
- Supports the addition and deletion of users from groups outside of the product environment. This action is preferred to adding and removing them to WebSphere Application Server roles. Stop and restart the enterprise application for these changes to take effect. This action can be very disruptive in a production environment.

At run time, WebSphere Application Server authorizes incoming requests based on the user's identification information and the mapping of the user to roles. If the user belongs to any role that has permission to execute a method, the request is authorized. If the user does not belong to any role that has permission, the request is denied.

The J2EE approach represents a declarative approach to authorization, but it also recognizes that you cannot deal with all situations declaratively. For these situations, methods are provided for determining user and role information programmatically. For Enterprise JavaBeans, the following two methods are supported by WebSphere Application Server:
- **getCallerPrincipal**: This method retrieves the user identification information.
- **isCallerInRole**: This method checks the user identification information against a specific role.

For servlets, the following methods are supported by WebSphere Application Server:
- getRemoteUser
- isUserInRole
- getUserPrincipal

These methods correspond in purpose to the enterprise bean methods.

For more information on the J2EE security authorization model see the following Web site: http://java.sun.com

## Admin roles

The J2EE role-based authorization concept has been extended to protect the WebSphere Application Server administrative subsystem. A number of administrative roles have been defined to provide degrees of authority needed to perform certain WebSphere administrative functions from either the Web-based administrative console or the system management scripting interface. The authorization policy is only enforced when global security is enabled. The following table describes the admin roles:

*Admin roles*

| Role | Description |
| --- | --- |
| monitor | Least privileged that basically allows a user to view the WebSphere Application Server configuration and current state. |
| configurator | Monitor privilege plus the ability to change the WebSphere Application Server configuration. |

*Admin roles*

| | |
|---|---|
| operator | Monitor privilege plus the ability to change runtime state, such as starting or stopping services for example. |
| administrator | Operator and configurator privilege, plus additional privileges granted solely to the administrator role. Examples include:<br>• Modifying the server user ID and password<br>• Mapping users and groups to the administrator role |

The identity specified when enabling global security is automatically mapped to the administrator role. Users, groups, can be added or removed from the admin roles from the WebSphere Application Server administrative console at anytime. However, a server restart is required for the changes to take effect. A best practice is to map a group or groups, rather than specific users, to admin roles because it is more flexible and easier to administer in the long run. By mapping a group to an admin role, adding or removing users to or from the group occurs outside of WebSphere Application Server and does not require a server restart for the change to take effect.

In addition to mapping user or groups, a special-subject can also be mapped to the admin roles. A special-subject is a generalization of a particular class of users. The AllAuthenticated special subject means that the access check of the admin role ensures that the user making the request has at least been authenticated. The Everyone special subject means that anyone, authenticated or not, can perform the action, as if security was not enabled.

## Naming roles

The J2EE role-based authorization concept has been extended to protect the WebSphere CosNaming service.

CosNaming security offers increased granularity of security control over CosNaming functions. CosNaming functions are available on CosNaming servers such as the WebSphere Application Server. They affect the content of the WebSphere Name Space. There are generally two ways in which client programs will result in CosNaming calls. The first is through the JNDI interfaces. The second is CORBA clients invoking CosNaming methods directly.

Four security roles are introduced: **CosNamingRead, CosNamingWrite, CosNamingCreate, and CosNamingDelete**. However, the roles now have authority level from low to high as follows:
- **CosNamingRead**. Users who have been assigned the CosNamingRead role will be allowed to do queries of the WebSphere Name Space, such as through the JNDI ″lookup″ method. The special-subject Everyone is the default policy for this role.
- **CosNamingWrite**. Users who have been assigned the CosNamingWrite role will be allowed to do write operations such as JNDI ″bind″, ″rebind″, or ″unbind″, plus CosNamingRead operations. The special-subject AllAuthenticated is the default policy for this role.
- **CosNamingCreate**. Users who have been assigned the CosNamingCreate role will be allowed to create new objects in the Name Space through such operations as JNDI ″createSubcontext″, plus CosNamingWrite operations. The special-subject AllAuthenticated is the default policy for this role.
- **CosNamingDelete**. And finally users who have been assigned CosNamingDelete role will be able to destroy objects in the Name Space, for example using the JNDI ″destroySubcontext″ method, as well as CosNamingCreate operations. The special-subject AllAuthenticated is the default policy for this role.

Users, groups, or the special subjects AllAuthenticated and Everyone can be added or removed to or from the naming roles from the WebSphere Application Server administrative console at anytime. However, you must restart the server for the changes to take effect. A best practice is to map groups or one of the special-subjects, rather than specific users, to Naming roles because it is more flexible and easier to

administer in the long run. By mapping a group to an naming role, adding or removing users to or from the group occurs outside of WebSphere Application Server and does not require a server restart for the change to take effect.

If a user is assigned a particular naming role and that user is a member of a group that has been assigned a different naming role, the user will be granted the most permissive access between the role he was assigned and the role his group was assigned. For example, assume that user MyUser has been assigned the CosNamingRead role. Also, assume that group MyGroup has been assigned the CosNamingCreate role. If MyUser is a member of MyGroup, MyUser will be assigned the CosNamingCreate role because he is a member of MyGroup. If MyUser were not a member of MyGroup, he would be assigned the CosNamingRead role.

The CosNaming authorization policy is only enforced when global security is enabled. When global security is enabled, attempts to do CosNaming operations without the proper role assignment will result in a org.omg.CORBA.NO_PERMISSION exception from the CosNaming Server.

In WebSphere Application Server version 4.0.2, each CosNaming function is assigned to only one role. Therefore, users who have been assigned CosNamingCreate role will not be able to query the Name Space unless they have also been assigned CosNamingRead. In most cases a creator would need to be assigned three roles: **CosNamingRead, CosNamingWrite, and CosNamingCreate**. This has been changed in the release. The **CosNamingRead** and **CosNamingWrite** roles assignment for the creator example in above have been included in **CosNamingCreate** role. In most of the cases, WebSphere Application Server administrators do not have to change the roles assignment for every user or group when they move to this release from previous one.

Although the ability exist to greatly restrict access to the Name space by changing the default policy, doing so may result in unexpected org.omg.CORBA.NO_PERMISSION exceptions at run time. Typically, J2EE applications access the Name space and the identity they use is that of the user that authenticated to WebSphere Application Server when they access the J2EE application. Unless the J2EE application provider clearly communicates the expected Naming roles, care should be taken when changing the default naming authorization policy.

## Adding users and groups to roles using an assembly tool

Before you perform this task, you should have already completed the steps in the "Securing Web applications using an assembly tool" on page 118 and "Securing enterprise bean applications" on page 116 articles where you created new roles and assigned those roles to EJB and Web resources. Complete these steps during application installation. This is because the environment (user registry) under which the application is running is not known until deployment.

If you already know the environment in which the application is running and the user registry that is used, then you can use an assembly tool to assign users and groups to roles. It is recommended that you use the administrative console to assign users and groups to roles.

1. In the Project Explorer view of an assembly tool, right-click an enterprise application project (EAR file) and click **Open With** > **Deployment Descriptor Editor**. An application deployment descriptor editor opens on the EAR file. To access information about the editor, press F1 and click **Application deployment descriptor editor**.
2. Click the **Security** tab and, under the main pane, click **Add**.
3. In the Add Security Role wizard, name and describe the security role. Then click **Finish**.
4. Under **WebSphere Bindings**, select the user or group extension properties for the security role. Available values include: Everyone, All authenticated users, and Users/Groups.
5. If you selected Users/Groups, click **Add** beside the **Users** or **Groups** panes. In the wizard that opens, specify a user or group name and click **Finish**. Repeat this step until you have added all users and groups to which the security role applies.

6. Close the application deployment descriptor editor and, when prompted, click **Yes** to save the changes.

The `ibm-application-bnd.xmi` file in the application contains the users and groups to roles mapping table (*authorization table*).

After securing an application, install the application using the administrative console.

## Mapping users to RunAs roles using an assembly tool

RunAs roles are used for delegation. A servlet or enterprise bean component uses the RunAs role to invoke another enterprise bean by impersonating that role.

Before you perform this task:
- Secure the Web application and enterprise bean applications, including creating and assigning new roles to enterprise bean and Web resources. For more information, see "Securing Web applications using an assembly tool" on page 118 and "Securing enterprise bean applications" on page 116.
- Assign users and groups to roles. For more information, see "Adding users and groups to roles using an assembly tool" on page 124. Complete this step during the installation of the application. The environment or user registry under which the application is going to run is not known until deployment. If you already know the environment in which the application is going to run and you know the user registry, then you can use an assembly tool to assign users to RunAs roles.

You must define RunAs roles when a servlet or an enterprise bean in an application is configured with RunAs settings.
1. In the Project Explorer view of an assembly tool, right-click an enterprise application project (EAR file) and click **Open With** > **Deployment Descriptor Editor**. An application deployment descriptor editor opens on the EAR file. To access information about the editor, press F1 and click **Application deployment descriptor editor**.
2. On the **Security** tab, under **Security Role Run As Bindings**, click **Add**.
3. Click **Add** under **RunAs Bindings**.
4. In the Security Role wizard, select one or more roles and click **Finish**.
5. Repeat steps 3 through 5 for all the RunAs roles in the application.
6. Close the application deployment descriptor editor and, when prompted, click **Yes** to save the changes.

The `ibm-application-bnd.xmi` file in the application contains the user to RunAs role mapping table.

After securing an application, you can install the application using the administrative console. You can change the RunAs role mappings of an installed application. For more information, see "RunAs roles to users mapping" on page 136.

# Chapter 10. Deploying secured applications

Before you perform this task, verify that you have already designed, developed and assembled an application with all the relevant security configurations. For more information on these tasks refer to the Chapter 8, “Developing secured applications,” on page 51 and Chapter 9, “Assembling secured applications,” on page 115 articles. In this context, deploying and installing an application are considered the same task.

Deploying applications that have security constraints (secured applications) is not much different than deploying applications any security constraints. The only difference is that you might need to assign users and groups to roles for a secured application, which requires that you have the correct active registry. To deploy a newly secured application click **Applications > Install New Application** in the navigation panel on the left and follow the prompts. If you are installing a secured application, roles would have been defined in the application. If delegation was required in the application, RunAs roles also are defined.

One of the steps required to deploy secured applications is to assign users and groups to roles defined in the application. This task is completed as part of the step titled *Map security roles to users and groups*. This assignment might have already been done through an assembly tool. In that case you can confirm the mapping by going through this step. You can add new users and groups and modify existing information during this step.

If the applications support delegation, then a RunAs role is already defined in the application. If the delegation policy is set to **Specified Identity** (during assembly) the intermediary invokes a method using an identity setup during deployment. Use the RunAs role to specify the identity under which the downstream invocations are made. For example, if the RunAs role is assigned user ″bob″ and the client ″alice″ is invoking a servlet, with delegation set, which in turn calls the enterprise beans, then the method on the enterprise beans is invoked with ″bob″ as the identity. As part of the deployment process one of the steps is to assign or modify users to the RunAs roles. This step is titled ″Map RunAs roles to users″. Use this step to assign new users or modify existing users to RunAs roles when the delegation policy is set to Specified Identity.

These steps are common for both installing an application and modifying an existing application. If the application contains roles, you see the ″Map security roles to users and groups″ link during application installation and also during managing applications, as a link in the Additional properties section.

1. Click **Applications > Install New Application**. Complete the steps (non-security related) required prior to the step titled **Map security roles to users and groups**.
2. Assign users and groups to roles. For more information, see “Assigning users and groups to roles” on page 128.
3. Map users to RunAs roles if RunAs roles exist in the application. For more information, see “Assigning users to RunAs roles” on page 134.
4. Click **Correct use of System Identity** to specify RunAs roles if needed. Complete this action if the application has delegation set to use System Identity (applicable to enterprise beans only). System Identity uses the WebSphere Application Server security server ID to invoke downstream methods and should be used with caution as this ID has more privileges than other identities in terms of accessing WebSphere Application Server internal methods. This task is provided to make sure that the deployer is aware that the methods listed in the panel have System Identity set up for delegation and to correct them if necessary. If no changes are necessary, skip this task.
5. Complete the remaining (non-security related) steps to finish installing and deploying the application.

Once a secured application is deployed, verify that you can access the resources in the application with the correct credentials. For example, if your application has a protected Web module, make sure only the users that you assigned to the roles are able to use the application.

© Copyright IBM Corp. 2004

## Assigning users and groups to roles

Before you perform this task:
- Secure the Web applications and EJB applications where new roles were created and assigned to Web and EJB resources.
- Create all the roles in your application.
- Verify that you have properly configured the user registry that contains the users that you want to assign. It is preferable to have security turned on with the user registry of your choice before beginning this process.
- Make sure that if you change anything in the security configuration (for example, enable security or change the user registry) you save the configuration and restart the server before the changes become effective.

Since the default active registry is LocalOS, it is not necessary, although it is recommended, that you enable security if you want to use the LocalOS registry to assign users and groups to roles. You can enable security once the users and groups are assigned in this case. The advantage of enabling security with the appropriate registry before proceeding with this task is that you can validate the security setup (which includes checking the user registry configuration) and avoid any problems using the registry.

These steps are common for both installing an application and modifying an existing application. If the application contains roles, you see the Map security roles to users/groups link during application installation and also during application management, as a link in the Additional properties section at the bottom.

1. Access the administrative console by typing `http://localhost:9060/ibm/console` in a Web browser.
2. Click **Applications > Enterprise applications >***application_name*.
3. Under Additional properties, click **Map security roles to users/groups**. A list of all the roles that belong to this application displays. If the roles already had users or special subjects (All Authenticated, Everyone) assigned, they display here.
4. To assign the special subjects, select either the **Everyone** or the **All Authenticated** check box for the appropriate roles.
5. Click **Apply** to save any changes and then continue working with user or group roles.
6. To assign users or groups, select the role. You can select multiple roles at the same time, if the same users or groups are assigned to all the roles.
7. Click **Look up users** or **Look up groups**.
8. Get the appropriate users and groups from the registry by completing the **limit** (number of items) and the **Search String** fields and clicking **Search**. The **limit** field limits the number of users that are obtained and displayed from the registry. The pattern is a searchable pattern matching one or more users and groups. For example, `user*` lists users like user1, user2. A pattern of asterisk (*) indicates all users or groups.

   Use the limit and the search strings cautiously so as not to overwhelm the registry. When using large registries (like Lightweight Directory Access Protocol (LDAP)) where information on thousands of users and groups resides, a search for a large number of users or groups can make the system very slow and can make it fail. When there are more entries than requests for entries, a message displays on top of the panel. You can refine your search until you have the required list.
9. Select the users and groups to include as members of these roles from the **Available** field and click **>>** to add them to the roles.
10. To remove existing users and groups, select them from the **Selected** field and click **<<**. When removing existing users and groups from roles use caution if those same roles are used as RunAs roles.

    For example, if user1 is assigned to RunAs role, role1, and you try to remove user1 from role1, the administrative console validation does not delete the user since a user can only be a part of a RunAs role if the user is already in a role (User1 should be in role1 in this case) either directly or indirectly

through a group. For more information on the validation checks that are performed between RunAs role mapping and user and group mapping to roles, see the "Assigning users to RunAs roles" on page 134 section.

11. Click **OK**. If there are any validation problems between the role assignments and the RunAs role assignments the changes are not committed and an error message indicating the problem displays at the top of the panel. If there is a problem, make sure that the user in the RunAs role is also a member of the regular role. If the regular role contains a group which contains the user in the RunAs role, make sure that the group is assigned to the role using the administrative console. Follow steps 4 and 5. Avoid using the Application Server Toolkit or any other manual process where the complete name of the group, host name, group name, or distinguished name (DN) is not used.

The user and group information is added to the binding file in the application. This information is used later for authorization purposes.

This task is required to assign users and groups to roles, which enables the correct users and groups to access a secured application. If you are installing an application, complete your installation. Once the application is installed and running you can access your resources according to the user and group mapping you did in this task. If you are managing applications and have modified the users and groups to role mapping, make sure you save, stop and restart the application so that the changes become effective. Try accessing the J2EE resources in the application to verify that the changes are effective.

## Security role to user and group mappings

Use this page to map security roles to users. You can map roles to specific users, to specific groups, or to different categories.

To view this administrative console page, click **Application** > **Install New Application**. While running the Application Installation Wizard, prompts appear to help you map security roles to users or groups. To change role to user or group mappings for deployed applications, click **Application** > **Enterprise Application** > *deployed_application* > **Map security roles to users/groups**.

### Users

Specifies the users for role mapping. Verify that the users are defined in your chosen user registry.

To change the roles to users mapping, click **Manage Application >***application* **> Map security roles to users**.

| | |
|---|---|
| **Data type:** | String |

### Groups

Specifies the groups for role mapping. Verify that the groups are defined in your chosen user registry.

To change the roles to users mapping, click **Manage Application >***application* **> Map security roles to groups**.

| | |
|---|---|
| **Data type:** | String |

### Roles

Specifies the roles to which you want to map users and groups. Role privileges give users and groups permission to run as specified.

Select the check boxes to choose a role or a set of roles. Click **Look-up Users** to map users to the roles that you have selected. Click **Look-up Groups** to map groups to the selected roles. Use the check boxes to map roles to **EVERYONE** or **ALL AUTHENTICATED** special subject.

| | |
|---|---|
| **Data type:** | String |

### Everyone

Specifies to map roles to everyone. Mapping a role to everyone means that anyone can access resources protected by this role, and essentially, there is no security.

| | |
|---|---|
| **Data type:** | Boolean |

### All Authenticated

Specifies to authenticate all users. Roles are mapped to all authenticated users, and all authenticated users in the selected user registry are granted access to the role.

| | |
|---|---|
| **Data type:** | Boolean |

## Security role to user and group selections

Use this page to select users and groups for security roles.

To view this administrative console page, click **Application > Install New Application**.

While using the Install New Application Wizard, prompts appear to help you map security roles to users. You also can configure security roles to user mappings of deployed applications. Different roles can have different security authorizations. Mapping users or groups to a role authorizes those users or groups to access applications defined by the role. Users, groups and roles are defined when an application is installed or configured.

You also can select role to user and group mappings while you are deploying applications. After deployment in **Additional Properties**, click **Map Security roles to users** to change user and group mappings to a role.

### Look up users

Specifies whether the server looks up selected users.

Choose the role by selecting the check box beside the role and clicking **Lookup users**. Complete the **Limit** and the **Pattern** fields. The **Limit** field contains the number of entries that the search function returns. The **Pattern** field contains the search pattern used for searching entries. For example, `bob*` searches all users or groups starting with `bob`. A limit of zero returns all the entries that match the pattern. Use this value only when a small number of users or groups match this pattern in the registry. If the registry contains more entries that match the pattern than requested, a message appears in the console to indicate that there are more entries in the registry. You can either increase the limit or refine the search pattern to get all the entries.

### Look up groups

Specifies whether the server looks up selected groups.

Choose the role by selecting the check box beside the role and clicking **Lookup groups**. Complete the **Limit** and the **Pattern** fields. The **Limit** field contains the number of entries that the search function returns. The **Pattern** field contains the search pattern used for searching entries. For example, `bob*` searches all users or groups starting with `bob`. A limit of zero returns all the entries that match the pattern. Use this value only when a small number of users or groups match this pattern in the registry. If the registry contains more entries that match the pattern than requested, a message appears in the console to indicate that there are more entries in the registry. You can either increase the limit or refine the search pattern to get all the entries.

### Role

Specifies user roles.

A number of administrative roles are defined to provide degrees of authority needed to perform certain WebSphere administrative functions from either the Web-based administrative console or the system management scripting interface. The authorization policy is only enforced when global security is enabled. The following roles are valid:

- **Monitor**--least privileged that basically allows a user to view the server configuration and current state
- **Configurator**--monitor privilege plus the ability to change the server configuration
- **Operator**--monitor privilege plus the ability to change the run time state, such as starting or stopping services
- **Administrator**--operator plus configurator privilege

| | |
|---|---|
| **Range** | Monitor, Configurator, Operator, Administrator |

### Everyone

Specifies to authenticate everyone.

| | |
|---|---|
| **Range** | Monitor, Configurator, Operator, Administrator |

### All authenticated

| | |
|---|---|
| **Range** | Monitor, Configurator, Operator, Administrator |

### Mapped users

### Mapped groups

## Look up users and groups settings

Use this page to select users and groups for security roles.

To view this administrative console page, click **Applications** > **Enterprise Applications** > *application_name* > **Map security roles to users/groups** > **Look up users or groups** button.

Different roles can have different security authorizations. Mapping users or groups to a role authorizes those users or groups to access applications defined by the role. Users, groups and roles are defined when an application is installed or configured. Use the Search field to display users in the Available Users list. Click the arrows to add users from the Available Users list to the Selected Users list.

### Limit

Specifies the maximum number of users/groups that can be returned when assigning users/groups to roles.

A value of 0 implies a return of all users/groups that match the pattern. You can either increase the limit or refine the search pattern to get all the entries.

| | |
|---|---|
| **Data type** | Integer |
| **Units** | User name |
| **Default** | 20 |
| **Range** | 0 or more |

### Pattern

Indicates the search pattern used to search for the entries.

The pattern field should contain the search pattern that should be used to search for the entries. For example, bob* will search all users or groups starting with bob. A limit of 0 gets all the entries that match the pattern and should be used only when a small number users/groups match that pattern in the registry. If the registry contains more entries that match the pattern than requested for, a message shows in the console to indicate that there are more entries in the registry.

| | |
|---|---|
| **Data type** | String |
| **Units** | Number of users |
| **Default** | 20 |
| **Range** | A-Z with * |

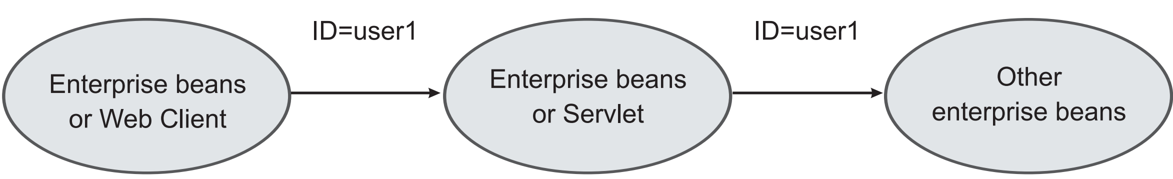
## Delegations

*Delegation* is a process security identity propagation from a caller to a called object. As per the J2EE specification, a servlet and enterprise beans can propagate either the client (remote user) identity when invoking enterprise beans or they can use another specified identity as indicated in the corresponding deployment descriptor.

The IBM extension supports Enterprise JavaBeans (EJB) to propagate to the server ID when invoking other entity beans. There are three types of delegations:
- Delegate (RunAs) Client Identity
- Delegate (RunAs) Specified Identity
- Delegate (RunAs) System Identity

**Delegate (RunAs) Client Identity**

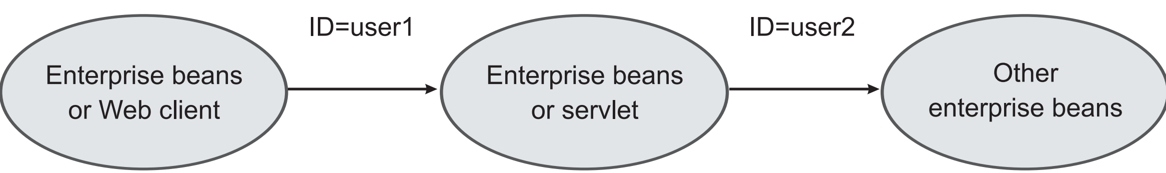
**Delegate Client Identity**




**RunAs client ID**

**Delegate (RunAs) Specified Identity**

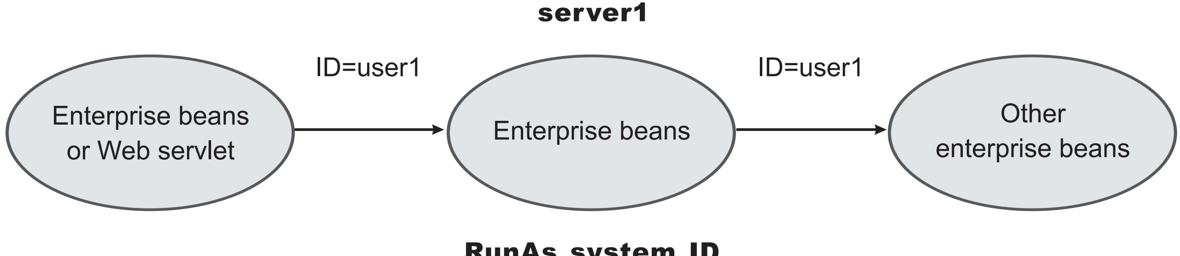
**Delegate Specified Identity**




**Run As specified role mapped to user2**

**Delegate (RunAs) System Identity**

## Delegate System Identity




The EJB specification only supports delegation (RunAs) at the EJB level. But an IBM extension allows EJB method level RunAs specification. Method EJB method level runAs specification allows one to specify a different RunAs role for different methods within the same enterprise beans.

The RunAs specification is detailed in the deployment descriptor (the `ejb-jar.xml` file in the EJB module and the `web.xml` file in the Web module). The IBM extension to the RunAs specification is included in the `ibm-ejb-jar-ext.xmi` file.

There is also an IBM specific binding file for each application that contains a mapping from the RunAs role to the user. This file is specified in the `ibm-application-bnd.xmi` file.

These specifications are read by the run time during application startup. The following figure illustrates the delegation mechanism as implemented in the WebSphere Application Server security model.

**Delegation**

WebSphere Application Server

CSIV2/SAS, TCP/IP, SSL
Java client
EJB resource access
(1)
Enterprise beans collaborator
(2) Resource
Delegation process
Resource (3)
Resource to Run As role
Run As roles
Delegate module
Invoke credentials (5)
ORB current
(6)
Enterprise beans
(7)
Servlet
(6)
(7)
Run As roles
(4) Credentials
Run As role to credentials
Resource (2)
HTTP or HTTPS
Web client
(1)
Web resource access
Web authenticator

| Resource | Run As roles |
| --- | --- |
| | |
| | |

| Run As role | users and passwords |
| --- | --- |
| | |
| | |

### Delegation Process

There are two tables that help in the delegation process:
- Resource to RunAs role mapping table
- RunAs role to user ID and password mapping table

Use the Resource to RunAs role mapping table to get the role that is used by a servlet or by enterprise beans to propagate to the next enterprise beans call.

Use the RunAsRole to User ID and Password mapping table to get the user ID that belongs to the RunAs role and its password.

Delegation is performed after successful authentication and authorization. During this process, the delegation module consults the Resource to RunAs role mapping table to get the RunAs role (3). The delegation module consults the RunAs role to user ID and password mapping table to get the user that belongs to the RunAs role (4). The user ID and password is used to create a new credential using the authentication module, which is not shown in figure.The resulting credential is stored in the ORB Current as an invocation credential (5). Servlet and enterprise beans when invoking other enterprise beans pick up the invocation credential from the ORB Current (6) and call the next enterprise beans (7).

## Assigning users to RunAs roles

Before you perform this task,
- Secure the Web applications and EJB applications where new RunAs roles were created and assigned to Web and EJB resources.
- Create all the RunAs roles in your application. The user in the RunAs role can only be entered if that user or a group to which that user belongs is already part of the regular role.
- Assign users and groups to security roles. Refer to “Assigning users and groups to roles” on page 128 for more information.
- Verify that the user registry requirements are met. These requirements are the same as those discussed in the same as in the case of “Assigning users and groups to roles” on page 128 task. For example, if role1 is a role that is also used as a RunAs role, then the user, user1, can be added to the RunAs role. role1, if user1 or a group that user1 belongs to, already is assigned to role1. The administrative console checks this logic when **Apply** or **OK** is clicked. If the check fails, the change is not made and an error message displays at the top of the panel.

If the special subjects ″Everyone″ or ″All Authenticated″ are assigned to a role, then no check takes place for that role.

The checking is done every time **Apply** in this panel is clicked or when **OK** is clicked in the **Map security roles to users/groups** panel. The check verifies that all the users in all the RunAs roles do exist directly or indirectly (through a group) in those roles in the **Map security roles to users/groups** panel. If a role is assigned both a user and a group to which that user belongs, then either the user or the group (not both) can be deleted from **Map security roles to users/groups** panel.

If the RunAs role user belongs to a group and if that group is assigned to that role, make sure that the assignment of this group to the role is done through administrative console and not through an assembly tool or any other method. When using the administrative console, the full name of the group is used (for example, `hostname\groupName` in windows systems, and distinguished names (DN) in Lightweight Directory Access Protocol (LDAP)). During the check, all the groups to which the RunAs role user belongs are obtained from the registry. Since the list of groups obtained from the registry are the full names of the groups, the check works correctly. If the short name of a group is entered using an assembly tool (for example, `group1` instead of `CN=group1, o=myCompany.com`) then this check fails.

These steps are common to both installing an application and modifying an existing application. If the application contains RunAs roles, you see the **Map RunAs roles to users** link during application installation and also during managing applications as a link in the **Additional properties** section at the bottom.

1. Click **Applications > Enterprise Applications >***application_name*.
2. Under Additional properties, click **Map RunAs roles to users**. A list of all the RunAs roles that belong to this application displays. If the roles already had users assigned, they display here.

3. To assign a user, select the role. You can select multiple roles at the same time if the same user is assigned to all the roles.
4. Enter the user's name and password in the designated fields. The user name entered can be either the short name (preferred) or the full name (as seen when getting users and groups from the registry).
5. Click **Apply**. The user is authenticated using the active user registry. If authentication is successful, a check is made to verify that this user or group is mapped to the role in the **Map security roles to users and groups** panel. If authentication fails, verify that the user and password are correct and that the active registry configuration is correct.
6. To remove a user from a RunAs role, select the roles and click **Remove**.

The RunAs role user is added to the binding file in the application. This file is used for delegation purposes when accessing J2EE resources. This step is required to assign users to RunAs roles so that during delegation the appropriate user is used to invoke the EJB methods.

If you are installing the application, complete installation. Once the application is installed and running you can access your resources according to the RunAS role mapping. Save the configuration.

If you are managing applications and have modified the RunAs roles to users mapping, make sure you save, stop and restart the application so that the changes become effective. Try accessing your J2EE resources to verify that the new changes are in effect.

## Unprotected EJB 2.0 methods protection settings

Use this page to verify that unprotected EJB 2.0 methods have the correct level of protection before you map users to roles.

To view this administrative console page, click **Application** > **Install New Application**. While running the Install New Application Wizard, prompts appear to help you map security roles to users.

### Exclude

Specifies that the method is completely protected.

| | |
|---|---|
| **Data type:** | Check box |
| **Default:** | Cleared |

### Uncheck

Specifies that everyone can access the security method.

| | |
|---|---|
| **Data type:** | Check box |
| **Default:** | Uncheck |

### Specify role

Specifies the EJB level of protection based on the security role.

The roles listed in this menu are obtained from the application scope. If the selected role is not in the module, then it is added to the modules or Java archive (JAR) files.

| | |
|---|---|
| **Data type:** | String |
| **Units:** | Role |

### Module name

Specifies the name of the module.

If a module name appears in this list, then the module contains unprotected EJB methods.

| | |
|---|---|
| **Data type:** | String |
| **Units:** | Module name |

### Protection

Specifies the level of protection assigned to a particular module name.

| | |
|---|---|
| **Data type:** | String |
| **Default:** | Cleared |

## EJB 2.1 method protection level settings

Use this page to verify that all unprotected EJB 2.1 methods have the correct level of protection before you map users to roles.

To view this administrative console page, click **Applications > Install New Application**. While running the Install New Application Wizard, prompts appear to help you determine that all unprotected EJB 2.1 methods have the correct level of protection.

### EJB Module

Specifies the enterprise bean module name.

| | |
|---|---|
| **Data Type:** | String |
| **Units:** | EJB module name |

### Module URI

Specifies the Java archive (JAR) file name.

| | |
|---|---|
| **Data Type:** | String |
| **Units:** | JAR file name |

### Method protection

Specifies the level of protection assigned to the EJB module.

A selected box means to *Deny All* and that the method is completely protected.

| | |
|---|---|
| **Data Type:** | Check box |
| **Default:** | Cleared |
| **Range:** | Yes or No |

## RunAs roles to users mapping

Use this page to map RunAs roles to users. You can change the RunAs settings after an application deploys.

To view this administrative console page, click **Applications > Install New Application**. While running the application installation wizard, prompts appear to help you map RunAs roles to users. You can change the RunAs roles to users mappings for deployed applications by completing the following steps:

1. Click **Applications > Enterprise Applications >***application_name*.
2. Under Additional properties, click **Map RunAs roles to users**.

The enterprise beans you are installing contain predefined RunAs roles. RunAs roles are used by enterprise beans that need to run as a particular role for recognition while interacting with another enterprise bean.

### User name

Specifies a user name for the RunAs role user.

This user already maps to the role specified in the Mapping users and groups to roles panel. You can map the user to its appropriate role by either mapping the user to that role directly or mapping a group that contains the user to that role.

| **Data type:** | String |
|---|---|

### Password

Specifies the password for the RunAs user.

| **Data type:** | String |
|---|---|

***Confirm password:***

Specifies the confirmed password of the administrative user.

| **Data type** | String |
|---|---|

*Role:*

Specifies administrative user roles.

A number of administrative roles have been defined to provide degrees of authority needed to perform certain WebSphere administrative functions from either the web based administrative console or the system management scripting interface. The authorization policy is only enforced when global security is enabled. The following roles are valid:

- **Monitor**--least privileged that basically allows a user to view the WebSphere configuration and current state
- **Configurator**--monitor privilege plus the ability to change the WebSphere configuration
- **Operator**--monitor privilege plus the ability to change runtime state, such as starting or stopping services for example
- **Administrator**--operator plus configurator privilege

## Updating and redeploying secured applications

Before you perform this task, secure Web applications, secure EJB applications, and deploy them in WebSphere Application Server. This section addresses the way to update existing applications.

1. Use the administrative console to modify the existing users and groups mapping to roles. For information on the required steps, see “Assigning users and groups to roles” on page 128.
2. Use the administrative console to modify the users for the RunAs roles. For information on the required steps, see “Assigning users to RunAs roles” on page 134.
3. Complete the changes and save them.
4. Stop and restart the application for the changes to become effective.
5. Use the an assembly tool. For more information, see Assembling applications.
6. Use an assembly tool to modify roles, method permissions, auth-constraints, data-constraints and so on. For more information, see Assembling applications.

7. Save the Enterprise Archive (EAR) file, uninstall the old application, deploy the modified application and start the application to make the changes effective.

The applications are modified and redeployed. This step is required to modify existing secured applications.

If information about roles is modified make sure you update the user and group information using the administrative console. Once the secured applications are modified and either restarted or redeployed, make sure that the changes are effective by accessing the resources in the application.

# Chapter 11. Testing security

After configuring global security and restarting all of your servers in a secure mode, it is best to validate that security is properly enabled.

There are a few techniques that you can use to test the various security login types. For example, you can test the Web-based BasicAuth login, Web-based form login, and the Java client BasicAuth login.

There are basic tests that show that the fundamental security components are working properly. Complete the following steps to validate your security configuration:

1. Test the Web-based BasicAuth with *Snoop*, by accessing the following URL: `http://hostname.domain:9080/snoop`. A login panel appears. If a login panel does not appear, then a problem exists. If the panel appears, type in any valid user ID and password in your configured user registry.

   **Note:** In a Network Deployment environment, the Snoop servlet is only available in the domain if you included the **DefaultApplication** option when adding the application server to the cell. The **-includeapps** option for the **addNode** command migrates the **DefaultApplication** option to the cell. Otherwise, skip this step.
2. Test the Web-based form login by bringing up the administrative console: `http://hostname.domain:9060/ibm/console`. A form-based login page appears. If a login page does not appear, try accessing the administrative console by typing `https://myhost.domain:9043/ibm/console`.

   Type in the administrative user ID and password used for configuring your user registry when configuring security.

   When the authentication mechanism is set as Lightweight Third Party Authentication (LTPA), represent the host name as a fully qualified host name (that is, `myhost.mycompany.com:9060` rather than just `myhost:9060`).
3. Test Java Client BasicAuth with *dumpNameSpace* by executing the *install_root*`\bin\dumpNameSpace.bat` file. A login panel appears. If a login panel does not appear, there is a problem. Type in any valid user ID and password in your configured user registry.
4. Thoroughly test all of your applications in secure mode.
5. After enabling security, verify that your system comes up in secure mode.
6. If all tests pass, proceed with more rigorous testing of your secured applications. If you have any problems, review the output logs in the WebSphere Application Server `/logs/`*nodeagent* or WebSphere Application Server `/logs/`*server_name* directories, respectively. Then check the security troubleshooting article to see if it references any common problems.

The results of these tests, if successful, indicate that security is fully enabled and working properly.

© Copyright IBM Corp. 2004

# Chapter 12. Administering security

Administering secure applications requires access to the WebSphere Application Server administrative console. Log in with a valid user ID and password that have administrative access. To administer security, complete these steps:

1. Configure global security. For more information, see "Configuring global security" on page 142.
2. Assign users to administrator roles. For more information, see "Assigning users to administrator roles" on page 153.
3. Assign users to naming roles. For more information, see "Assigning users to naming roles" on page 156.
4. Configure authentication mechanisms. For more information, see "Configuring authentication mechanisms" on page 160.
5. Configure Lightweight Third Party Authentication. For more information, see "Configuring Lightweight Third Party Authentication" on page 161.
6. Configure trust association interceptors. For more information, see "Configuring trust association interceptors" on page 169.
7. Configure single signon. For more information, see "Configuring single signon" on page 172.
8. Configure user registries. For more information, see "Configuring user registries" on page 190.
   a. Configure local operating system user registries. For more information, see "Configuring local operating system user registries" on page 195.
   b. Configure Lightweight Directory Access Protocol user registries. For more information, see "Configuring Lightweight Directory Access Protocol user registries" on page 198.
   c. Configure custom user registries. For more information, see "Configuring custom user registries" on page 213.
9. Configure Java Authentication and Authorization Service login. For more information, see "Configuring application logins for Java Authentication and Authorization Service" on page 242.
10. Configure an authorization provider. For more information, see "Configuring a JACC provider" on page 346. To configure the Tivoli Access Manager Java Authorization Contract for Containers (JACC) provider, see either "Configuring the JACC provider for Tivoli Access Manager using the wsadmin utility" on page 356 or "Configuring the JACC provider for Tivoli Access Manager using the administrative console" on page 358.
11. Configure the Common Secure Interoperability Version 2 and Security Authentication Service authentication protocols. For more information, see "Configuring Common Secure Interoperability Version 2 and Security Authentication Service authentication protocols" on page 386.
12. Configure Secure Sockets Layer. For more information, see "Configuring Secure Sockets Layer" on page 417.
13. Configure Java 2 Security Manager. For more information, see "Configuring Java 2 security" on page 469.
14. **Optional:** Configure security attribute propagation. For more information, see "Security attribute propagation" on page 275.

## Global security

Global security applies to all applications running in the environment and determines whether security is used at all, the type of registry against which authentication takes place, and other values, many of which act as defaults.

The term *global security* represents the security configuration that is effective for the entire security domain. A *security domain* consists of all servers configured with the same user registry *realm* name. In some cases, the realm can be the machine name of a Local OS user registry. In this case, all application servers must reside on the same physical machine. In other cases, the realm can be the machine name of

© Copyright IBM Corp. 2004

an Lightweight Directory Access Protocol (LDAP) user registry. Since LDAP is a distributed user registry, a multiple node configuration is supported, such as the case for a Network Deployment environment. The basic requirement for a security domain is that the access ID returned by the registry from one server within the security domain is the same access ID as that returned from the registry on any other server within the same security domain. The *access ID* is the unique identification of a user and is used during authorization to determine if access is permitted to the resource.

Configuration of global security for a security domain consists of configuring the common user registry, the authentication mechanism, and other security information that defines the behavior of a security domain. The other security information that you can configure includes Java 2 Security Manager, Java Authentication and Authorization Service (JAAS), Java 2 Connector authentication data entries, Common Secure Interoperability Version 2 (CSIv2)/Security Authentication Service (SAS) authentication protocol (Remote Method Invocation over the Internet Inter-ORB Protocol (RMI/IIOP) security), and other miscellaneous attributes. The global security configuration usually applies to every server within the security domain.

## Configuring global security

It is helpful to understand security from an infrastructure standpoint so that you know the advantages of different authentication mechanisms, user registries, authentication protocols, and so on. Picking the right security components to meet your needs is a part of configuring global security. The following sections help you make these decisions. Read the following articles before continuing with the security configuration.

- “Global security” on page 141
- Introduction: Security

After you understand the security components, you can proceed to configure global security in WebSphere Application Server.

1. Start the WebSphere Application Server administrative console by clicking `http://yourhost.domain:9060/ibm/console` after starting the WebSphere Application Server. If security is currently disabled, log in with any user ID. If security is currently enabled, log in with a predefined administrative ID and password (this is typically the server user ID specified when you configured the user registry).
2. Click **Security > Global security** and configure the authentication mechanism, user registry, and so on. The configuration order is not important. However, when you select the **Enable global security** flag in the **Global security** panel, verify that all these tasks are completed. When you click **Apply** or **OK** and the **Enable global security** option is set, a verification occurs to see if the administrative user ID and password can be authenticated to the configured user registry. If you do not configure these, the validation fails.
3. Configure a user registry. For more information, see “Configuring user registries” on page 190. You can configure a local OS, LDAP, or custom user registry through the links under User registry on the Global security panel.

   One of the details common to all user registries is the **server user ID**. This ID is a member of the chosen user registry, but also has special privileges in WebSphere Application Server. The privileges for this ID and the privileges associated with the administrative role ID are the same. The server user ID can access all protected administrative methods. On Windows systems, the ID must not be the same name as the machine name of your system, since the registry sometimes returns machine-specific information when querying a user of the same name. In LDAP user registries, verify that the server user ID is a member of the registry and not just the LDAP administrative role ID. The entry must be searchable.

   The server user ID does **not** run WebSphere Application Server processes. Rather, the process ID runs the WebSphere Application Server processes.The process ID runs the WebSphere Application Server processes.

The process ID is determined by the way the process starts. For example, if you use a command line to start processes, the user ID that is logged into the system is the process ID. If running as a service, the user ID that is logged into the system is the user ID running the service. If you choose the LocalOS registry, the process ID requires special privileges to call the operating system APIs. Specifically, the process ID must have the **Act as Part of Operating System** privileges on Windows systems or **root** privileges on a UNIX system.

4. Configure the authentication mechanism. To get details about configuring authentication mechanisms, read the "Configuring authentication mechanisms" on page 160 article. There are two authentication mechanisms to choose from in the Global Security panel: Simple WebSphere Authentication Mechanism (SWAM) and Lightweight Third-Party Authentication (LTPA). However, only LTPA requires any additional configuration parameters. Use the SWAM option for single server requirements. Use the LTPA option for multi-server distributed requirements. SWAM credentials are not forwardable to other machines and for that reason do not expire. Credentials for LTPA are forwardable to other machines and for security reasons do expire. This expiration time is configurable. If you choose to go with LTPA, then "Configuring single signon" on page 172 support. This support permits browsers to visit different product servers without having to authenticate multiple times.
5. Configure the authentication protocol for special security requirements from Java clients, if needed. This task entails choosing a protocol, either Common Secure Interoperability Version 2 (CSIv2) or Security Authentication Service (SAS). The SAS protocol is still provided as a backwards compatibility to previous product releases, but is being deprecated. For details on configuring CSIv2 or SAS, see the article, "Configuring Common Secure Interoperability Version 2 and Security Authentication Service authentication protocols" on page 386.
6. Modify the default Secure Sockets Layer (SSL) keystore and truststore files that are packaged with the product. This action protects the integrity of the messages sent across the Internet. The product provides a single location where you can specify SSL configurations that the various WebSphere Application Server features that use SSL can utilize, including the LDAP user registry, Web container and the authentication protocol (CSIv2 and SAS). Create a new keystore and truststore, by referring to the "Creating a keystore file" on page 446 and "Creating truststore files" on page 450 articles. You can create different keystore files and truststore files for different uses or you can create just one set for everything that the server uses Secure Sockets Layer (SSL) for. After you create these new keystore and truststore files, specify them in the **SSL Configuration Repertoires**. To get to the **SSL Configuration Repertoires**, click **Security > SSL**. See the article, "Configuring Secure Sockets Layer" on page 417 for more information. To get to the SSL Configuration Repertoire, click **Security > SSL**. You can either edit the DefaultSSLConfig file or create a new SSL configuration with a new alias name. If you create a new alias name for your new keystore and truststore files, change every location that references the DefaultSSLConfig SSL configuration alias. The following list specifies the locations of where the SSL configuration repertoire aliases are used in the WebSphere Application Server configuration.

   For any transports that use the new network input/output channel chains, including HTTP and Java Message Service (JMS), you can modify the SSL configuration repertoire aliases in the following locations for each server:

   - Click **Server > Application server >** *server_name*. Under Communications, click **Ports**. Locate a transport chain where SSL is enabled and click **View associated transports**. Click *transport_channel_name*. Under Transport Channels, click **SSL Inbound Channel (SSL_2)**.

   For the Object Request Broker (ORB) SSL transports, you can modify the SSL configuration repertoire aliases in the following locations. These configurations are for the server-level for WebSphere Application Server and WebSphere Application Server Express and the cell level for WebSphere Application Server Network Deployment.

   - Click **Security > Global security**. Under Authentication, click **Authentication protocol > CSIv2 Inbound Transport**.
   - Click **Security > Global security**. Under Authentication, click **Authentication protocol > CSIv2 Outbound Transport**.
   - Click **Security > Global security**. Under Authentication, click **Authentication protocol > SAS Inbound Transport**.

- Click **Security > Global security**. Under Authentication, click **Authentication protocol > SAS Outbound Transport**.

For the Simple Object Access Protocol (SOAP) Java Management Extensions (JMX) administrative transports, you can modify the SSL configurations repertoire aliases by clicking **Servers > Application servers >***server_name*. Under Server infrastructure, click **Administration > Administration services**. Under Additional properties, click **JMX connectors > SOAPConnector**. Under Additional properties, click **Custom properties**. If you want to point the sslConfig property to a new alias, click **sslConfig** and select an alias in the Value field.

For the Lightweight Directory Access Protocol (LDAP) SSL transport, you can modify the SSL configuration repertoire aliases by clicking **Security > Global security**. Under User registries, click **LDAP**.

7. Click **Security** > **Global security** to configure the rest of the security settings and enable security. This panel performs a final validation of the security configuration. When you click **OK** or **Apply** from this panel, the security validation routine is performed and any problems are reported at the top of the page. See the "Global security settings" on page 145 article for detailed information about these fields. When you complete all of the fields, click **OK** or **Apply** to accept the selected settings. Click **Save** to persist these settings out to a file. If you see any informational messages in red text color, then a problem has occurred with the security validation. Typically, the message indicates the problem; therefore, review your configuration to verify that the user registry settings are accurate and the correct registry is selected. In some cases the LTPA configuration might not be fully specified.
8. Store the configuration for the server to use after it restarts. Complete this action if you have clicked **OK** or **Apply** on the **Security > Global security** panel, and there are no validation problems. To save the configuration, click **Save** in the menu bar at the top. This action writes the settings out to the configuration repository. If you do not click **Apply** or **OK** in the **Global security** panel before clicking **Save** on the main menu, your changes are not written to the repository.
9. Start the WebSphere Application Server administrative console by typing `http://yourhost.domain:9060/ibm/console` after the WebSphere Application Server deployment manager has been started. If security is currently disabled, log in with any user ID. If security is currently enabled, log in with a predefined administrative ID and password, which is typically the server user ID specified when you configure the user registry.

## Enabling global security

You can decide whether to enable IBM WebSphere Application Server security. You must enable security for all other security settings to function.

**Note:** WebSphere Application Server uses cryptography to protect sensitive data and ensure confidentiallity and integrity of communications between WebSphere Application Server and other components in the network. Cryptography is also used by Web Services security when certain security constraints have been configured for the Web Services application.

WebSphere uses JSSE and JCE libraries in the JDK to enforce this cryptography. The JDK provides strong but limited jurisdiction policy files. Unrestricted policy files provide the ability to perform full strength cryptography and improve performance.

IBM's SDKs ship with strong but limited jurisdiction policy files. For Windows, Linux, HPUX Solaris, and AIX platforms, the unrestricted policy files can be downloaded from https://www6.software.ibm.com/dl/jcesdk/jcesdk-p (Use the link near the bottom of the page.) The ZIP file should be unpacked and the two JAR files placed in the `JRE's jre/lib/security/` directory.

1. Select the unrestricted JCE policy files for SDK 1.4.2.
2. Extract the unlimited jurisdiction policy files that are packaged in the ZIP file that contains two files:
   a. `US_export_policy.jar`

b. `local_policy.jar`
3. Back up the original version of the policy files.
4. Replace the policy files in the `$JAVA_HOME/jre/lib/security` directory with the two files above.

## Global security settings

Use this page to configure security. When you enable security, you are enabling security settings on a global level.

To view this administrative console page, click **Security > Global security**.

When security is disabled, WebSphere Application Server performance is increased between 10-20%. Therefore, consider disabling security when it is not needed.

If you are configuring security for the first time, complete the steps in the ″Configuring server security″ article in the documentation to avoid problems. When security is configured, validate any changes to the registry or authentication mechanism panels. Click **Apply** to validate the user registry settings. An attempt is made to authenticate the server ID to the configured user registry. Validating the user registry settings after enabling global security can avoid problems when you restart the server for the first time.

***Enable global security:***

Specifies whether to enable global security for this WebSphere Application Server domain.

This flag is commonly referred to as the *global security flag* in WebSphere Application Server information. When enabling security, set the authentication mechanism configuration and specify a valid user ID and password in the selected user registry configuration.

| | |
|---|---|
| **Default:** | Disable |

***Enforce Java 2 Security:***

Specifies whether to enable or disable Java 2 security permission checking. By default, Java 2 security is disabled. However, enabling global security automatically enables Java 2 security. You can choose to disable Java 2 security, even when global security is enabled.

When the **Enforce Java 2 security** option is enabled and if an application requires more Java 2 security permissions than are granted in the default policy, then the application might fail to run properly until the required permissions are granted in either the `app.policy` file or the `was.policy` file of the application. AccessControl exceptions are generated by applications that do have all the required permissions. Consult the WebSphere Application Server documentation and review the Java 2 Security and Dynamic Policy sections if you are unfamiliar with Java 2 security.

If your server does not restart after you enable global security, you can disable security. Go to your `$`*`install_root`*`\bin` directory and execute the `wsadmin -conntype NONE` command. At the `wsadmin>` prompt, enter `securityoff` and then type `exit` to return to a command prompt. Restart the server with security disabled to check any incorrect settings through the administrative console.

| | |
|---|---|
| **Default:** | Disabled |

***Enforce fine-grained JCA security:***

Enable this option to restrict application access to sensitive Java Connector Architecture (JCA) mapping authentication data.

Consider enabling this option when both of the following conditions are true:

- Java 2 Security is enforced.
- The application code is granted the accessRuntimeClasses WebSphereRuntimePermission in the `was.policy` file found within the application enterprise archive (EAR) file. For example, the application code is granted the permission when the following line is found in your `was.policy` file:

```
permission com.ibm.websphere.security.WebSphereRuntimePermission "accessRuntimeClasses";
```

The **Enforce fine-grained JCA security** option adds fine-grained Java 2 Security permission checking to the default principal mapping of the WSPrincipalMappingLoginModule implementation. You must grant explicit permission to Java 2 Platform, Enterprise Edition (J2EE) applications that use the WSPrincipalMappingLoginModule implementation directly in the Java Authentication and Authorization Service (JAAS) login when Java 2 Security and the **Enforce fine-grained JCA security** option is enabled.

| | |
|---|---|
| **Default:** | Disabled |

***Use domain-qualified user IDs:***

Specifies that user names returned by methods are qualified with the security domain in which they reside.

| | |
|---|---|
| **Default:** | Disabled |

***Cache timeout:***

Specifies the timeout value in seconds for security cache. This value is a relative timeout.

If WebSphere Application Server security is enabled, the security cache timeout can influence performance. The timeout setting specifies how often to refresh the security-related caches. Security information pertaining to beans, permissions, and credentials is cached. When the cache timeout expires, all cached information becomes invalid. Subsequent requests for the information result in a database lookup. Sometimes, acquiring the information requires invoking a Lightweight Directory Access Protocol (LDAP)-bind or native authentication. Both invocations are relatively costly operations for performance. Determine the best trade off for the application, by looking at usage patterns and security needs for the site.

In a 20-minute performance test, setting the cache timeout so that a timeout does not occur yields a 40% performance improvement.

| | |
|---|---|
| **Data type:** | Integer |
| **Units:** | Seconds |
| **Default:** | 600 |
| **Range:** | Greater than 30 seconds |

***Issue permission warning:***

Specifies that during application deployment and application start, the security run time issues a warning if applications are granted any custom permissions. Custom permissions are permissions defined by the user applications, not Java API permissions. Java API permissions are permissions in `package java.*` and `javax.*`.

WebSphere Application Server provides support for policy file management. A number of policy files are available in this product, some of them are static and some of them are dynamic. Dynamic policy is a template of permissions for a particular type of resource. There is no code base defined or relative code base used in the dynamic policy template. The real code base is dynamically created from the configuration and run-time data. The `filter.policy` file contains a list of permissions that an application

should not have according to the J2EE 1.3 specification. For more information on permissions, see the ″Java 2 security policy files″ article in the documentation.

**Default:** Disabled

***Active protocol:***

Specifies the active authentication protocol for Remote Method Invocation over the Internet Inter-ORB Protocol (RMI IIOP) requests when security is enabled.

In previous releases the Security Authentication Service (SAS) protocol was the only available protocol.

An Object Management Group (OMG) protocol called Common Secure Interoperability Version 2 (CSIv2) supports increased vendor interoperability and additional features. If all of the servers in your security domain are Version 5 servers, specify `CSI` as your protocol.

If some servers are version 3.x or version 4.x servers, specify `CSI and SAS`.

**Default:** BOTH
**Range:** CSI and SAS, CSI
**Range:**

***Active authentication mechanism:***

Specifies the active authentication mechanism when security is enabled.

WebSphere Application Server and WebSphere Application Server Express, Version 6 support the following authentication mechanisms: Simple WebSphere Authentication Mechanism (SWAM) and Lightweight Third Party Authentication (LTPA).

**Default:** SWAM (WebSphere Application Server)
**Default:**
**Range:** SWAM, LTPA

**Default:**
**Range:**

***Active User Registry:***

Specifies the active user registry, when security is enabled. LDAP or a custom user registry is required when running as a UNIX nonroot user or running in a multi-node environment.

You can configure settings for one of the following user registries:

- Local OS

  When you enable global security on a UNIX platform and the user registry is the local OS, you must run the server as root. The local OS user registry is not supported for nonroot users on a UNIX platform.
- LDAP user registry

  The LDAP user registry settings are used when users and groups reside in an external LDAP directory. When security is enabled and any of these properties change, go to the Global Security panel and click **Apply** or **OK** to validate the changes.
- Custom user registry

**Default:** Local OS (single, stand-alone server or sysplex and root administrator only)

**Range:** Local OS (single, stand-alone server or sysplex and root administrator only), LDAP user registry, Custom user registry

***Use the Federal Information Processing Standard (FIPS):***

Enables the use of Federal Information Processing Standard (FIPS)-approved cryptographic algorithms.

When **Use the Federal Information Processing Standard (FIPS)** is enabled, the Lightweight Third Party Authentication (LTPA) implementation uses IBMJCEFIPS. IBMJCEFIPS supports the Federal Information Processing Standard (FIPS)-approved cryptographic algorithms for DES, Triple DES, and AES. Although the LTPA keys are backwards compatible with prior releases of WebSphere Application Server, the LTPA token is not compatible with prior releases.

WebSphere Application Server provides a FIPS-approved Java Secure Socket Extension (JSSE) provider called IBMJSSEFIPS. A FIPS-approved JSSE requires the Transport Layer Security (TLS) protocol because it is not compatible with the Secure Sockets Layer (SSL) protocol. If you select the **Use the Federal Information Processing Standard (FIPS)** option prior to specifying a FIPS-approved JSSE provider and a TLS protocol, the following error message displays at the top of the **Global security** panel:

```
The security policy is set to use only FIPS approved cryptographic algorithms.
However at least one SSL configuration may not be using a FIPS approved JSSE provider.
FIPS approved cryptographic algorithms may not be used in those cases.
```

To correct this problem, configure your JSSE provider and security protocol on the **SSL configuration repertoires** panel by completing one of the following tasks:
- Clicking **Security > SSL** and modifying an existing configuration
- Clicking **New** and creating a new configuration

**Note:**
- The IBMJSSEFIPS provider is not supported on the HP-UX platform.
- In WebSphere Application Server Version 6, the HTTP transport does not support the FIPS-approved providers because it uses the new IBMJSSE2 provider to support channel framework. The channel framework supports asynchronous communication that might enhance performance. The IBMJSSE2 provider is specified for the Secure Sockets Layer (SSL) channel because it must use the IBMJCE provider for encryption. Currently, an IBMJCEFIPS provider that works with the IBMJSSE2 provider, is not available.

**Default:** Disabled

***Custom Properties:*** For an existing configuration, there are a number of profiles that you must modify. To modify the profiles, go into the administrative console and click **Security > Global security**. Under Additional Properties, click **Custom properties**.

## Configuring server security

You can customize security to some extent at the application server level. You can disable user security on an application server (administrative security remains enabled when global security is enabled). You can also modify Java 2 Security Manager, CSIv2 or Secure Authentication Service (SAS), and some of the other security attributes that are found on the global security (also called *cell-level* security) panel. You

cannot configure a different authentication mechanism or user registry on an individual server basis. This feature is limited to cell-level configuration only. Also, when global security is disabled, you cannot enable application server security.

By default, server security inherits all of the values that are configured in global security (cell-level security). To override the security configuration at the server level, click **Servers > Application Servers >** *server name*. Under Security, click **Server Security > Additional properties** and click any of the following panels:

- **CSIv2 Inbound Authentication**
- **CSIv2 Inbound Transport**
- **CSIv2 Outbound Authentication**
- **CSIv2 Outbound Transport**
- **SAS Inbound Transport**
- **SAS Outbound Transport**
- **Server-level Security**

After modifying the configuration in any of these panels and clicking **OK** or **Apply**, the security configuration for that panel or set of panels now overrides cell-level security. Other panels that are not overridden continue to be inherited at the cell-level. However, you can always revert back to the cell-level configuration at any time. On the Server Security panel, click to revert back to the global security configuration on these panels:

- **Use cell security**
- **Use cell CSI**
- **Use cell SAS**

1. Start the administrative console for the deployment manager. To get to the administrative console, go to `http://host.domain:9060/ibm/console`. If security is disabled, you can enter any ID. If security is enabled, you must enter a valid user ID and password, which is either the administrative ID (configured for the user registry) or a user ID entered as an administrative user. To add a user ID as an administrative user, click **System Administration > Console settings > Console Users**.
2. Configure global security if you have not already done so. Go to the “Configuring global security” on page 142 article for detailed steps. After global security is configured, configure server-level security.
3. To configure server-level security, click **Servers > Application Servers >** *server name*. Under Security, click **Server Security**. The status of the security level that is in use for this application server is displayed.

   By default, you can see that global security, CSI, and SAS have not been overridden at the server level. CSI and SAS are authentication protocols for RMI/IIOP requests. The Server Level Security panel lists attributes that are on the Global Security panel and can be overridden at the server level. Not all of the attributes on the Global Security panel can be overridden at the server level, including Active Authentication Mechanism and Active User Registry.
4. To disable security for this application server, go to the Server Level Security panel, clear the **Enabled** flag and click **OK** or **Apply**. Click **Save**. By modifying the Server Level Security panel, you can see that this flag overrides the cell-level security.
5. To configure CSI at the server level, you can change any panel that starts with CSI. By doing so, all panels that start with CSI will override the CSI settings specified at the cell level. This change includes all authentication and transport panels for CSI. See the “Configuring Common Secure Interoperability Version 2 and Security Authentication Service authentication protocols” on page 386 article for more detailed steps regarding configuring CSI authentication protocol.

Typically server-level security is used to disable user security for a specific application server. However, this can also be used to disable (or enable) the Java 2 Security Manager, and configure the authentication requirements for RMI/IIOP requests both incoming and outgoing from this application server.

After you modify the configuration for a particular application server, you must restart the application server for the changes to become effective. To restart the application server, go to **Servers > Application Servers** and click the server name that you recently modified. Then, click the **Stop** button and then the **Start** button.

If you disabled security for the application server, you can typically test a URL that is protected when security is enabled.

## Server-level security settings

Use this page to enable server level security and specify other server level security configurations.

To view this administrative console page, click **Servers** > **Application Servers** > *server_name*. Under Security, click **Server Security**. Under Additional properties, click **Server Level Security**.

### Enable global security

Use this flag to disable or enable security again for this application server while global security is enabled. Server security is enabled by default when global security is enabled. You cannot enable security on an application server while global security is disabled. Administrative (administrative console and wsadmin) and naming security remain enabled while global security is enabled, regardless of the status of this flag.

| | |
|---|---|
| **Default** | Disable |

### Enforce Java 2 security

Specifies that the server enforces Java 2 Security permission checking at the server level. When cleared, the Java 2 server-level security manager is not installed and all of the Java 2 Security permission checking is disabled at the server level.

If your application policy file is not set up correctly, see the documentation on configuring an application policy in a `was.policy` file.

| | |
|---|---|
| **Default** | Disabled |

### Enforce fine-grained JCA security

Enable this option to restrict application access to sensitive Java Connector Architecture (JCA) mapping authentication data.

| | |
|---|---|
| **Default** | Disabled |

### Use domain qualified user IDs

Specifies whether user IDs returned by getUserPrincipal()-like calls are qualified with the server level security domain within which they reside.

| | |
|---|---|
| **Default** | Disabled |

### Cache timeout

Specifies the timeout value for server level security cache in seconds.

| | |
|---|---|
| **Data type** | Integer |
| **Units** | Seconds |
| **Default** | 600 |
| **Range** | Greater than 30 seconds. Avoid setting cache timeout value to 30 seconds or less. |

### Issue permission warning

Specifies whether a warning is issued during application installation when an application requires a Java 2 permission that is normally not granted to an application.

WebSphere Application Server provides support for policy file management. A number of policy files are included in WebSphere Application Server. Some of these policy files are static and some of them are dynamic. Dynamic policy is a template of permissions for a particular type of resource. In dynamic policy files, the code bases are evaluated at run time using configuration data. You can add or remove permissions, as needed, for each code base. However, do not add, remove, or modify the existing code bases. The real code base is dynamically created from the configuration and run-time data. The `filter.policy` file contains a list of permissions that an application does not have, according to the J2EE 1.3 Specification. For more information on permissions, see the documentation on the Java 2 security policy files.

| | |
|---|---|
| **Default** | Enabled |

### Active protocol

Specifies the active server level security authentication protocol when server level security is enabled.

You can use an Object Management Group (OMG) protocol called Common Secure Interoperability Version 2 (CSIv2) for more vendor interoperability and additional features. If all of the servers in your entire security domain are Version 5.0 servers, it is best to specify **CSI** as your protocol.

If some servers are Version 3.x or Version 4.x servers, it is best to specify **CSI and SAS**. However, by specifying **CSI and SAS**, you now have two interceptors invoking each request.

| | |
|---|---|
| **Data type** | String |
| **Default** | CSI and SAS |
| **Range** | CSI, CSI and SAS |

## Administrative console and naming service authorization

WebSphere Application Server extends the Java 2 Platform, Enterprise Edition (J2EE) security role-based access control to protect the product administrative and naming subsystems.

#### Administrative console

Four administrative roles are defined to provide degrees of authority needed to perform certain WebSphere Application Server administrative functions from either the administrative console or the system management scripting interface. The authorization policy is only enforced when global security is enabled. The four administrative security roles are defined in the following table:

*administrative roles*

| Role | Description |
|---|---|
| monitor | Least privileged where a user can view the WebSphere Application Server configuration and current state. |
| configurator | Monitor privilege plus the ability to change the WebSphere Application Server configuration. |
| operator | Monitor privilege plus the ability to change the run-time state, such as starting or stopping services. |
| administrator | Operator plus configuration privilege and the permission required to access sensitive data including the server password, LTPA password, LTPA, keys, and so on. |

When global security is enabled, the administrative subsystem role-based access control is enforced. The administrative subsystem includes security server, user registry, and all the Java Management Extensions (JMX) MBeans. When security is enabled, both the administrative console and the administrative scripting tool require users to provide the required authentication data. Moreover, the administrative console is designed so the control functions that display on the pages are adjusted according to the security roles that a user has. For example, a user who has only the monitor role can see only the non-sensitive configuration data. A user with the operator role can change the system state.

The server identity specified when enabling global security is automatically mapped to the administrative role. You can add or remove users and groups to or from the administrative roles from the WebSphere Application Server administrative console. However, a server restart is required for the changes to take effect. A best practice is to map a group, rather than specific users, to administrative roles because it is more flexible and easier to administer. By mapping a group to an administrative role, adding or removing users to or from the group occurs outside of WebSphere Application Server and does not require a server restart for the change to take effect.

When global security is enabled, WebSphere Application Servers run under the server identity that is defined under the active user registry configuration. Although it is not shown on the administrative console and in other tools, a special Server subject is mapped to the administrator role. The WebSphere Application Server run-time code, which runs under the server identity, requires authorization to runtime operations. If no other user is assigned administrative roles, you can log into the administrative console or to the wsadmin scripting tool using the server identity to perform administrative operations and to assign other users or groups to administrative roles. Because the server identity is assigned to the administrative role by default, the administrative security policy requires the administrative role to perform the following operations:
- Change server ID and server password
- Enable or disable WebSphere Application Server global security
- Enforce or disable Java 2 Security
- Change the LTPA password or generate keys
- Assign users and groups to administrative roles

When enabling security, you can assign one or more users and groups to administrative roles. For more information, see Assigning users to naming roles. However, before assigning users to naming roles, configure the active user registry. User and group validation depends on the active user registry. For more information, see Configuring user registries.

**Naming service authorization**

CosNaming security offers increased granularity of security control over CosNaming functions. CosNaming functions are available on CosNaming servers such as the WebSphere Application Server. They affect the content of the WebSphere Application Server name space. There are generally two ways in which client programs result in CosNaming calls. The first is through the JNDI interfaces. The second is with CORBA clients invoking CosNaming methods directly.

Four security roles are introduced :
- CosNamingRead
- CosNamingWrite
- CosNamingCreate
- CosNamingDelete

The names of the four roles are the same with WebSphere Application Server Advanced Edition Version 4.0.2. The roles now have authority levels from low to high:

**CosNamingRead**
: Users can query of the WebSphere Application Server name space, using, for example, the JNDI lookup method. The special-subject Everyone is the default policy for this role.

**CosNamingWrite**
: Users can perform write operations such as JNDI **bind**, **rebind**, or **unbind**, and CosNamingRead operations. The special-subject AllAuthenticated is the default policy for this role.

**CosNamingCreate**
: Users can create new objects in the name space through such operations as JNDI createSubcontext and CosNamingWrite operations. The special subject AllAuthenticated is the default policy for this role.

**CosNamingDelete**
: Users can destroy objects in the name space, for example using the JNDI destroySubcontext method and CosNamingCreate operations. The special-subject AllAuthenticated is the default policy for this role.

Additionally, a Server special-subject is assigned to all the four CosNaming roles by default. The Server special-subject provides a WebSphere Application Server server process, which runs under the server identity, access to all the CosNaming operations. Note that the Server special-subject does not display and cannot be modified through the administrative console or other administrative tools.

No special configuration is required to enable the server identity (as specified) when enabling global security for administrative use because the server identity is automatically mapped to the administrator role (enabled with PQ81586) .

Users, groups, or the special subjects AllAuthenticated and Everyone can be added or removed to or from the naming roles from the WebSphere Application Server administrative console at any time. However, a server restart is required for the changes to take effect. A best practice is to map groups or one of the special-subjects, rather than specific users, to naming roles because it is more flexible and easier to administer in the long run. By mapping a group to a naming role, adding or removing users to or from the group occurs outside of WebSphere Application Server and does not require a server restart for the change to take effect.

The CosNaming authorization policy is only enforced when global security is enabled. When global security is enabled, attempts to do CosNaming operations without the proper role assignment result in an `org.omg.CORBA.NO_PERMISSION` exception from the CosNaming Server.

In WebSphere Application Server Version 4.0.2, each CosNaming function is assigned to only one role. Therefore, users who are assigned the CosNamingCreate role cannot query the name space unless they have also been assigned CosNamingRead. And in most cases a creator needs to be assigned three roles: CosNamingRead, CosNamingWrite, and CosNamingCreate. The CosNamingRead and CosNamingWrite roles assignment for the creator example are included in the CosNamingCreate role. In most of the cases, WebSphere Application Server administrators do not have to change the roles assignment for every user or group when they move to this release from a previous one.

Although the ability exists to greatly restrict access to the name space by changing the default policy, unexpected `org.omg.CORBA.NO_PERMISSION` exceptions can occur at run time. Typically, J2EE applications access the name space and the identity they use is that of the user that authenticated to WebSphere Application Server when they access the J2EE application. Unless the J2EE application provider clearly communicates the expected Naming roles, use caution when changing the default naming authorization policy.

## Assigning users to administrator roles

The following steps are needed to assign users to administrative roles.

In the administrative console, click **System Administration > Console settings** . Click either **Console Users** or **Console Groups**.

1. To add a user or a group, click **Add** on the **Console users** or **Console groups** panel.

2. To add a new administrator user, enter a user identity in the User field, highlight **Administrator**, and click **OK**. If there is no validation error, the specified user is displayed with the assigned security role.
3. To add a new administrative group, either enter a group name in the **Specify group** field or select EVERYONE or ALL AUTHENTICATED from the Select from special subject menu, and click **OK**. If no validation error exists, the specified group or special subject displays with the assigned security role.
4. To remove a user or group assignment, click **Remove** on the Console Users or the Console Groups panel. On the Console Users or the Console Groups panel, select the check box of the user or group to remove and click **OK**.
5. To manage the set of users or groups to display, expand the **filter** folder on the right panel and modify the filter. For example, setting the filter to `user*` only displays users with the `user` prefix.
6. After the modifications are complete, click **Save** to save the mappings.
7. Restart the server for changes to take effect.

The task of assigning users and groups to administrative roles is performed to identify users for performing WebSphere Application Server administrative functions. Administrator roles are used to control access to WebSphere Application Server administrative functions. There are four roles: administrator, configurator, operator and monitor.

**Administrator role**
: Users and groups assigned to the administrator role can perform all administrative operations and can set up both J2EE role-based and Java 2 security policy.

**Configurator role**
: Users assigned to the configurator role can perform all of the day-to-day configuration tasks including installing and uninstalling applications, assigning users and groups to role mapping for applications, setting run-as configurations, setting up Java 2 security permissions for applications, and customizing Common Secure Interoperability Version 2 (CSIv2), Security Authentication Service (SAS), and Secure Sockets Layer (SSL) configurations.

**Operator role**
: Users assigned to the operator role can view the WebSphere Application Server configuration and its current state, but also can change the run-time state such as stopping and starting services.

**Monitor role**
: Users assigned the monitor state can view the WebSphere Application server configuration and its current state only.

Before you assign users to administrative roles (administrator, configurator, operator, and monitor), you must set up your user registry, which can be LDAP, local OS, or a custom registry. You can set up your user registries without enabling security.

Once you assign users to administrative roles, you must restart the server for the new roles to take effect. However, the administrative resources are not protected until you enable security.

## Console groups and CORBA naming service groups

Use the Console Groups page to give groups specific authority to administer the WebSphere Application Server using tools such as the administrative console or wsadmin scripting. The authority requirements are only effective when global security is enabled. Use the CORBA naming service groups page to manage CORBA Naming Service groups settings.

To view the Console Groups administrative console page, click **System Administration > Console Groups**.

To view the CORBA naming service groups administrative console page, click **Environment > Naming > CORBA Naming Service Groups**.

## Group (Console groups)

Specifies groups.

The ALL_AUTHENTICATED and the EVERYONE groups can have the following role privileges: Administrator, Configurator, Operator, and Monitor.

| | |
|---|---|
| **Data type:** | String |
| **Range:** | ALL_AUTHENTICATED, EVERYONE |

## Group (CORBA naming service groups)

Identifies CORBA naming service groups.

The ALL_AUTHENTICATED group has the following role privileges: CosNamingRead, CosNamingWrite, CosNamingCreate, and CosNamingDelete. The EVERYONE group indicates that the users in this group have CosNamingRead privileges only.

| | |
|---|---|
| **Data type:** | String |
| **Range:** | ALL_AUTHENTICATED, EVERYONE |

## Role (Console group)

Specifies user roles.

The following administrative roles provide different degrees of authority needed to perform certain WebSphere Application Server administrative functions:

**Administrator**
: The administrator role has operator permissions, configurator permissions, and the permission required to access sensitive data including server password, LTPA password and keys, and so on.

**Configurator**
: The configurator role has monitor permissions and can change the WebSphere Application Server configuration.

**Operator**
: The operator role has monitor permissions and can change the run-time state. For example, the operator can start or stop services.

**Monitor**
: The monitor role has the least permissions. This role primarily confines the user to viewing the WebSphere Application Server configuration and current state.

| | |
|---|---|
| **Data type:** | String |
| **Range:** | Administrator, Configurator, Operator, and Monitor |

## Role (CORBA naming service users)

Identifies naming service group roles.

A number of naming roles are defined to provide degrees of authority needed to perform certain WebSphere naming service functions. The authorization policy is only enforced when global security is enabled.

Four name space security roles are available: CosNamingRead, CosNamingWrite, CosNamingCreate, and CosNamingDelete. The names of the four roles are the same with WebSphere Advanced Edition, Version 4.0.2. However, the roles now have authority levels from low to high:

**CosNamingRead**
: Users can query the WebSphere name space using, for example, the Java Naming and Directory Interface (JNDI) lookup method. The special-subject EVERYONE is the default policy for this role.

**CosNamingWrite**
: Users can perform write operations such as JNDI bind, rebind, or unbind, and CosNamingRead operations. The special-subject ALL_AUTHENTICATED is the default policy for this role.

**CosNamingCreate**
: Users can create new objects in the name space through operations such as JNDI createSubcontext and CosNamingWrite operations. The special-subject ALL_AUTHENTICATED is the default policy for this role.

**CosNamingDelete**
: Users can destroy objects in the name space, for example using the JNDI destroySubcontext method and CosNamingCreate operations. The special-subject ALL_AUTHENTICATED is the default policy for this role.

| | |
|---|---|
| **Data type:** | String |
| **Range:** | CosNamingRead, CosNamingWrite, CosNamingCreate, and CosNamingDelete |

## Assigning users to naming roles

The following steps are needed to assign users to naming roles. In the administrative console, expand **Environment > Naming**, and click **CORBA Naming Service Users** or **CORBA Naming Service Groups**.

1. Click **Add** on the **CORBA Naming Service Users** or **CORBA Naming Service Groups** panel.
2. To add a new naming service user, enter a user identity in the **User** field, highlight one or more naming roles, and click **OK**. If no validation errors occur, the specified user is displayed with the assigned security role.
3. To add a new naming service group, either select **Specify group** and enter a group name or select **Select from special subject** and then select either **EVERYONE** or **ALL AUTHENTICATED**. Click **OK**. If no validation errors occur, the specified group or special subject is displayed with the assigned security role.
4. To remove a user or group assignment, go to the **CORBA Naming Service Users** or **CORBA Naming Service Groups** panel. Select the check box next to the user or group that you want to remove and click **Remove**.
5. To manage the set of users or groups to display, expand the **Filter** folder on the right panel, and modify the filter text box. For example, setting the filter to `user*` displays only users with the `user` prefix.
6. After modifications are complete, click **Save** to save the mappings. Restart the server for the changes to take effect.

The default naming security policy is to grant all users read access to the CosNaming space and to grant any valid user the privilege to modify the contents of the CosNaming space. You can perform the previously mentioned steps to restrict user access to the CosNaming space. However, use caution when changing the naming security policy. Unless a Java 2 Platform, Enterprise Edition (J2EE) application has clearly specified its naming space access requirements, changing the default policy can result in unexpected org.omg.CORBA.NO_PERMISSION exceptions at run time.

## Console users settings and CORBA naming service user settings

Use the Console users settings page to give users specific authority to administer WebSphere Application Server using tools such as the administrative console or wsadmin scripting. The authority requirements are only effective when global security is enabled. Use the common object request broker architecture (CORBA) naming service users settings page to manage CORBA naming service users settings.

To view the Console users administrative console page, click **System Administration > Console Users**.

To view the CORBA naming service users administrative console page, click **Environment > Naming > CORBA Naming Service users**.

## User (Console users)

Specifies users.

The users entered must exist in the configured active user registry.

| | |
|---|---|
| **Data type:** | String |

## User (CORBA naming service users)

Specifies CORBA naming service users.

The users entered must exist in the configured active user registry.

| | |
|---|---|
| **Data type:** | String |

## Role (Console users)

Specifies user roles.

The following administrative roles provide different degrees of authority needed to perform certain WebSphere Application Server administrative functions:

**Administrator**
The administrator role has operator permissions, configurator permissions, and the permission required to access sensitive data including server password, Lightweight Third Party Authentication (LTPA) password and keys, and so on.

**Configurator**
The configurator role has monitor permissions and can change the WebSphere Application Server configuration.

**Operator**
The operator role has monitor permissions and can change the run-time state. For example, the operator can start or stop services.

**Monitor**
The monitor role has the least permissions. This role primarily confines the user to viewing the WebSphere Application Server configuration and current state.

| | |
|---|---|
| **Data type:** | String |
| **Range:** | Administrator, Configurator, Operator, and Monitor |

## Role (CORBA naming service users)

Specifies naming service user roles.

A number of naming roles are defined to provide degrees of authority needed to perform certain WebSphere naming service functions. The authorization policy is only enforced when global security is enabled. The following roles are valid: CosNamingRead, CosNamingWrite, CosNamingCreate, and CosNamingDelete.

The names of the four roles are the same with WebSphere Application Server, Advanced Edition Version 4.0.2. However, the roles now have authority levels from low to high:

**CosNamingRead**
Users can query the WebSphere name space using, for example, the Java Naming and Directory Interface (JNDI) lookup method. The special-subject EVERYONE is the default policy for this role.

**CosNamingWrite**
: Users can perform write operations such as JNDI bind, rebind, or unbind, plus CosNamingRead operations. The special-subject ALL AUTHENTICATED is the default policy for this role.

**CosNamingCreate**
: Users can create new objects in the name space through operations such as JNDI createSubcontext and CosNamingWrite operations. The special-subject ALL AUTHENTICATED is the default policy for this role.

**CosNamingDelete**
: Users can destroy objects in the name space, for example using the JNDI destroySubcontext method and CosNamingCreate operations. The special-subject ALL AUTHENTICATED is the default policy for this role.

| | |
|---|---|
| **Data type:** | String |
| **Range:** | CosNamingRead, CosNamingWrite, CosNamingCreate and CosNamingDelete |

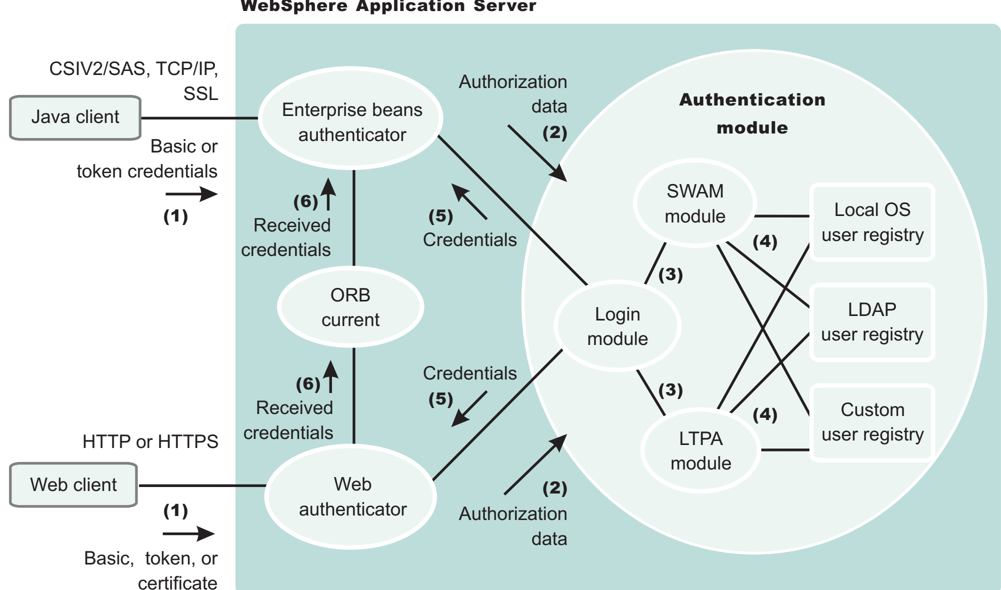
## Authentication mechanisms

An *authentication mechanism* defines rules about security information (for example, whether a credential is forwardable to another Java process), and the format of how security information is stored in both credentials and tokens.

Authentication is the process of establishing whether a client is valid in a particular context. A client can be either an end user, a machine, or an application.

An authentication mechanism in WebSphere Application Server typically collaborates closely with a *user registry*. The user registry is the user and groups account repository that the authentication mechanism consults with when performing authentication. The authentication mechanism is responsible for creating a *credential*, which is an internal product representation of a successfully authenticated client user. Not all credentials are created equally. The abilities of the credential are determined by the configured authentication mechanism.

Although this product provides several authentication mechanisms, you can configure only a single *active* authentication mechanism at a time. The active authentication mechanism is selected when configuring WebSphere Application Server global security.




## Authentication Process

The figure demonstrates the authentication process. Basically, authentication is required for enterprise bean clients and Web clients when they access protected resources. Enterprise bean clients (a servlet or other enterprise beans or a pure client) send the authentication information to a Web application server using one of the following protocols:

- Common Secure Interoperability Version 2 (CSIv2)
- Secure Authentication Service (SAS)

Web clients use the HTTP or HTTPS protocol to send the authentication information as shown in the previous figure.

The authentication information can be BasicAuth (user ID and password), credential token (in case of Lightweight Third Party Authentication (LTPA) on all platforms), or client certificate. The Web authentication is performed by the Web Authentication module and the enterprise bean authentication is performed by the Enterprise JavaBean (EJB) authentication module, which resides in the CSIv2 and SAS layer. The authentication module is implemented using the Java Authentication and Authorization Service (JAAS) login module. The Web authenticator and the EJB authenticator pass the authentication data to the login module (2), which can use any of the following mechanisms to authenticate the data:

- Lightweight Third Party Authentication (LTPA).
- Simple WebSphere Authentication Mechanism (SWAM)

The authentication module uses the registry that is configured on the system to perform the authentication (4). Three types of registries are supported: LocalOS, Lightweight Directory Access Protocol (LDAP), and custom registry. External registry implementation following the registry interface specified by IBM can replace either the LocalOS or the LDAP user registry.

The login module creates a JAAS subject after authentication and stores the Common Object Request Broker Architecture (CORBA) credential derived from the authentication data in the public credentials list of the subject. The credential is returned to the Web authenticator or EJB authenticator (5).

The Web authenticator and the EJB authenticator store the received credentials in the Object Request Broker (ORB) current for the authorization service to use in performing further access control checks.

WebSphere Application Server provides two authentication mechanisms: SWAM and LTPA. These authentication mechanisms differ primarily in the distributed security features that each supports.

## Configuring authentication mechanisms

Configure authentication mechanisms by clicking **Authentication Mechanisms** under **Security > Global security** in the administrative console.

- If you are using Simple WebSphere Authentication Mechanism (SWAM), no setup is needed. Follow the instructions in "Configuring Lightweight Third Party Authentication" on page 161 to set up Lightweight Third Party Authentication (LTPA).
- For LTPA, follow the steps in "Configuring single signon" on page 172 for most situations. If trust association is required, follow the steps in "Configuring trust association interceptors" on page 169.

## Simple WebSphere authentication mechanism

The Simple WebSphere authentication mechanism (SWAM) is intended for simple, non-distributed, single application server run-time environments. The single application server restriction is due to the fact that SWAM does not support *forwardable* credentials. If a servlet or enterprise bean in application server process 1, invokes a remote method on an enterprise bean living in another application server process 2, the identity of the caller identity in process 1 is not transmitted to server process 2. What is transmitted is an unauthenticated credential, which, depending on the security permissions configured on the EJB methods, can cause authorization failures.

Since SWAM is intended for a single application server process, single signon (SSO) is not supported.

The SWAM authentication mechanism is suitable for simple environments, software development environments, or other environments that do not require a distributed security solution.

## Lightweight Third Party Authentication

Lightweight Third Party Authentication (LTPA) is intended for distributed, multiple application server and machine environments. It supports forwardable credentials and single signon (SSO). LTPA can support security in a distributed environment through cryptography. This support permits LTPA to encrypt, digitally sign, and securely transmit authentication-related data, and later decrypt and verify the signature.

The Lightweight Third Party Authentication (LTPA) protocol enables the WebSphere Application Server to provide security in a distributed environment using cryptography. Application servers distributed in multiple nodes and cells can securely communicate using this protocol. It also provides the single signon (SSO) feature wherein a user is required to authenticate only once in a domain name system (DNS) domain and can access resources in other WebSphere Application Server cells without getting prompted. The realm names on each system in the SSO domain are case sensitive and must match identically.

Windows For local OS on the Windows platform, the realm name is the domain name, if a domain is in use, or the machine name.

UNIX On the UNIX platform, the realm name is the same as the host name.

For the Lightweight Directory Access Protocol (LDAP), the realm name is the host:port of the LDAP server.

The LTPA protocol uses cryptographic keys (LTPA keys) to encrypt and decrypt user data that passes between the servers. These keys need to be shared between the different cells for the resources in one cell to access resources in other cells (assuming that all the cells involved use the same LDAP or custom registry).

When using LTPA, a token is created with the user information and an expiration time and is signed by the keys. The LTPA token is time sensitive. All product servers participating in a protection domain must have their time, date, and time zone synchronized. If not, LTPA tokens appear prematurely expired and cause authentication or validation failures.

This token passes to other servers, in the same cell or in a different cell through cookies (for Web resources when SSO is enabled) or through the authentication layer (Security Authentication Service (SAS) or Common Secure Interoperability Version 2 (CSIv2) for enterprise beans).

If the receiving servers share the same keys as the originating server, the token can be decrypted to obtain the user information, which then is validated to make sure it has not expired and the user information in the token is valid in its registry. On successful validation, the resources in the receiving servers are accessible after the authorization check.

All the WebSphere Application Server processes in a cell (cell, nodes, application servers) share the same set of keys. If key sharing is required between different cells, export them from one cell and import them to the other. For security purposes, the exported keys are encrypted with a user-defined password. This same password is needed when importing the keys into another cell.

In the base version of WebSphere Application Server, LTPA, ICSF, and the Simple WebSphere Authentication Mechanism (SWAM) protocols are supported.

When security is enabled for the first time with LTPA, configuring LTPA is normally the initial step performed.

LTPA requires that the configured user registry be a centrally shared repository such as LDAP or a Windows domain type registry so that users and groups are the same regardless of the machine.

The following table summarizes the authentication mechanism capabilities and user registries with which LTPA can work.

| | **Forwardable Credentials** | **SSO** | **LocalOS User Registry** | **LDAP User Registry** | **Custom User Registry** |
|---|---|---|---|---|---|
| SWAM | No | No | Yes | Yes | Yes |
| LTPA | Yes | Yes | Yes | Yes | Yes |
| ICSF | Yes | Yes | Yes | Yes | Yes |

## Configuring Lightweight Third Party Authentication

The following steps are needed to configure Lightweight Third Party Authentication (LTPA) when setting up security for the first time:

1. Access the administrative console by typing `http://localhost:port_number/ibm/console` in a Web browser. Port 9060 is the default port number for accessing the administrative console. During installation, however, you might have specified a different port number. Use the appropriate port number.
2. Click **Security > Global security**.
3. Under Authentication mechanisms, click **LTPA**.
4. Enter the password and confirm it in the password fields. This password is used to encrypt and decrypt the LTPA keys during export and import of the keys. Remember this password because you enter it again when the keys from this cell are exported to another cell.
5. Enter a positive integer value in the **Timeout** field. This timeout value refers to how long an LTPA token is valid in minutes. The token contains this expiration time so that any server that receives the token can verify that the token is valid before proceeding further.

When the token expires, the user is prompted to log in.

An optimal value for this field depends on your configuration. The default value is 30 minutes.

6. **Optional:** In the **Key file name** field, specify the name of the file that is used when you import or export keys. You can use this field in conjunction with the **Import keys** and **Export keys** buttons at the top of the panel.
7. Click **Apply** or **OK**. The LTPA configuration is now set. Do not generate the LTPA keys in this step because they are automatically generated later. Proceed with the rest of the steps required to enable security, starting with single signon (SSO) (if SSO is required).
8. Complete the information in the Global Security panel and click **OK**. The LTPA keys are generated automatically the first time. Do not generate the keys manually.

The previous steps configure LTPA by setting passwords that generate LTPA keys.

After configuring LTPA, complete the following steps to work with your key files:
1. Generate key files.
2. Export key files.
3. Import key files.
4. If you are enabling security, make sure that you complete the remaining steps starting with enabling SSO.
5. If you generated a new set of keys or imported a new set of keys, verify that the keys are saved by clicking **Save** at the top of the panel. Because LTPA authentication uses time sensitive tokens, verify that the time, date, and time zone are synchronized among all product servers that are participating in the protection domain. If the clock skew is too high between servers, the LTPA token appears prematurely expired and causes authentication or validation failures.

## Configuring Lightweight Third Party Authentication keys

***Generating keys:***

Lightweight Third Party Authentication (LTPA) keys are automatically generated when a password change is detected. The first time that you set the LTPA password, as part of enabling security, the LTPA keys are automatically generated after **OK** or **Apply** is clicked in the LTPA panel. You do not have to click **Generate Keys** in this situation. Complete the following steps in the administrative console to generate a new set of LTPA keys:

1. Access the administrative console by typing `http://localhost:9060/ibm/console` in a Web browser.
2. Verify that all the WebSphere Application Server processes are running (cell, nodes, and all of the application servers). If any of the servers are down at the time of key generation and then brought back up later, these servers might contain old keys. Copy the new set of keys to these servers to bring them back up.
3. Click **Security > Authentication mechanisms > LTPA** in the navigation panel on the left.
4. Click **Generate Keys** if you want to use the existing password. This action generates a new set of keys that are encrypted with the same password as the old set of keys. Regardless of the password change, a new set of keys is generated when you click **Generate Keys**. This new set of keys is not propagated to the run time unless saved; save the files immediately.
5. Enter the new password and confirm it, to use a new password to generate keys. Click **OK** or **Apply**. A new set of keys is generated. A message indicating that a new set of keys is generated displays on the console. Do not click **Generate Keys**. These new keys are propagated to the run time after you save them.
6. Click **Save** to save the keys. After a new set of keys is generated and saved, the key propagation is dynamic. All of the processes running at that time (cells, node agents, application servers) are updated with the new set of keys. The next sections describe the process of exporting and importing the keys.

***Exporting keys:***

To support single signon (SSO) in WebSphere Application Server across multiple WebSphere Application Server domains or cells, share the LTPA keys and the password among the domains. Make sure that the time on the domains is similar to prevent the tokens from appearing as expired between the cells. You can use **Export Keys** to export the LTPA keys to other domains or cells. Complete the following steps in the administrative console to export key files for LTPA:

1. Access the administrative console by typing `http://localhost:9060/ibm/console` in a Web browser.
2. Click **Security** > **Authentication mechanisms** > **LTPA** in the navigation panel on the left.
3. In the **Key File Name** field, enter the full path of a file for key storage. This file needs write permissions.
4. Click **Export Keys**. A file is created with the LTPA keys. Exporting keys fails if a new set of keys is generated or imported and not saved prior to exporting. To avoid failure, make sure that you save the new set of keys (if any) prior to exporting them.
5. Click **Save** to save the configuration.

***Importing keys:***

To support SSO in WebSphere Application Server across multiple WebSphere Application Server domains or cells, share the LTPA keys and the password among the domains. You can use **Import Keys** to import the LTPA keys from other domains. Verify that key files are exported from one of the cells involved, into a file. Complete the following steps in the administrative console to import key files for LTPA.

Importing keys is a dynamic operation. All of the servers that are running at this time are updated with the new set of keys. Any back-level tokens signed with the back-level keys fail validation and the user is prompted to log in again.

1. Access the administrative console by typing `http://localhost:9060/ibm/console` in a Web browser.
2. Click **Security** > **Authentication mechanisms** > **LTPA** in the navigation panel on the left.
3. Change the password in the **password** fields to match the password in the cell from which you are importing the keys.
4. Click **Save** to save the new set of keys in the repository. This step is important to complete before importing the keys. If the password and the keys do not match, the servers fail. If the servers fail, turn off security and redo these steps.
5. In the **Key File Name** field, enter the full path of a file for key storage. This file needs read permissions.
6. Click **Import Keys**. The keys are now imported into the system.
7. Click **Save** to save the new set of keys in the repository. It is important to save the new set of keys to match the new password so that no problems are encountered starting the servers later.

## Lightweight Third Party Authentication settings

Use this page to configure Lightweight Third Party Authentication (LTPA) settings.

To view this administrative console page, complete the following steps:

1. Click **Security > Global security**.
2. Under Authentication, click **Authentication mechanisms > LTPA**.

If you are configuring security for the first time, only the password is required. After the password is entered, click **Apply**. Under Additional Properties, click **Single signon (SSO)** and enter the domain name. Make sure that SSO is enabled. Click **Apply**.

To complete the security setup, make sure that the appropriate registry is set up and click **Apply** from the Global security panel. When security is enabled and any of these properties change, go to the Global security panel under **Security > Global security** and click **Apply** to validate the changes.

***Generate Keys:***

Specifies whether the server generates new Lightweight Third Party Authentication (LTPA) keys.

When security is turned on for the first time with LTPA as the authentication mechanism, the LTPA keys are automatically generated with the password entered in the panel. If you need a new set of keys to generate using the previously set password, click **Generate Keys**. If a new password is used, do not click this option. After the new password is entered and **OK** or **Apply** is clicked, a new set of keys is generated. *A new set of generated keys is not used until you save them.*

***Import Keys:***

Specifies whether the server imports new LTPA keys.

To support single signon (SSO) in the WebSphere product across multiple WebSphere domains (cells), share the LTPA keys and the password among the domains. You can use the **Import Keys** option to import the LTPA keys from other domains. The LTPA keys are exported from one of the cells to a file. To import a new set of LTPA keys, enter the appropriate password, click **OK** and click **Save**. Then, enter the directory location where the LTPA keys are located prior to clicking **Import keys**. Do not click **OK** or **Apply**, but save the settings.

***Export Keys:***

Specifies whether the server exports LTPA keys.

To support single signon (SSO) in the WebSphere product across multiple WebSphere Application Server domains (cells), share the LTPA keys and the password among the domains. Use the **Export Keys** option to export the LTPA keys to other domains.

To export the LTPA keys, make sure that the system is running with security enabled and is using LTPA. Enter the file name in the **Key file name** field and click **Export Keys**. The encrypted keys are stored in the specified file.

***Password:***

Specifies the password to encrypt and decrypt the LTPA keys. Use this password when importing these keys into other WebSphere Application Server administrative domain configurations (if any) and when configuring SSO for a Lotus Domino server.

After the keys are generated or imported, they are used to encrypt and decrypt the LTPA token. Whenever the password is changed, a new set of LTPA keys are automatically generated when you click **OK** or **Apply**. The new set of keys is used after the configuration changes are saved.

| **Data type** | String |
|---|---|

***Confirm password:***

Specifies the confirmed password used to encrypt and decrypt the LTPA keys.

Use this password when importing these keys into other WebSphere Application Server administrative domain configurations (if any) and when configuring SSO for a Lotus Domino server.

| **Data type** | String |
|---|---|

***Timeout:***

Specifies the time period in minutes at which an LTPA token expires. Verify that this time period is longer than the cache timeout configured in the Global security panel.

| | |
|---|---|
| **Data type** | Integer |
| **Units** | Minutes |
| **Default** | 120 |

***Key file name:***

Specifies the name of the file used when importing or exporting keys.

Enter a fully qualified key file name, and click **Import Keys** or **Export Keys**.

| | |
|---|---|
| **Data type** | String |

## Trust associations

*Trust association* enables the integration of IBM WebSphere Application Server security and third-party security servers. More specifically, a reverse proxy server can act as a front-end authentication server while the product applies its own authorization policy onto the resulting credentials passed by the proxy server.

Demand for such an integrated configuration has become more compelling, especially when a single product cannot meet all of the customer needs or when migration is not a viable solution. This article provides a conceptual background behind the approach.

The demand is growing to provide customers with a trust association solution between IBM WebSphere Application Server and other Web authentication servers that act as a reverse proxy security server (IBM Tivoli Access Manager for e-business - WebSEAL, Caching Proxy) as an entry point to all service requests (See the first figure). This implementation design intends to have the proxy server as the only exposed entry point. The proxy server authenticates all requests that come in and provides coarse, granularity junction point authorization.

In this setup, the WebSphere Application Server is used as a back-end server to further exploit its fine-grained access control. The reverse proxy server passes the HTTP request to the WebSphere Application Server that includes the credentials of the authenticated user. WebSphere Application Server then uses these credentials to authorize the request.

**Trust association model**

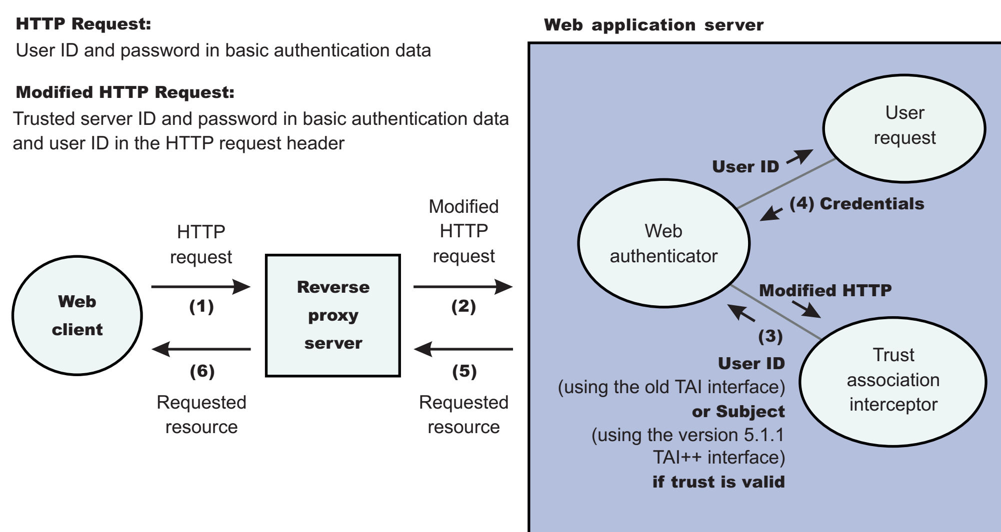
The idea that WebSphere Application Server can support trust association implies that the product application security recognizes and processes HTTP requests received from a reverse proxy server. WebSphere Application Server and the proxy server engage in a contract in which the product gives its full trust to the proxy server and the proxy server applies its authentication policies on every Web request that is dispatched to WebSphere Application Server. This trust is validated by the interceptors that reside in the product environment for every request received. The method of validation is agreed upon by the proxy server and the interceptor.

Running in trust association mode does not prohibit WebSphere Application Server from accepting requests that did not pass through the proxy server. In this case, no interceptor is needed for validating trust. It is possible, however, to configure WebSphere Application Server to strictly require that all HTTP requests go through a reverse proxy server. In this case, all requests that do not come from a proxy server are immediately denied by WebSphere Application Server.

WebSphere Application Server supports the following trust association interceptor (TAI) interfaces:

**com.ibm.ws.security.web.WebSealTrustAssociationInterceptor**
This Tivoli TAI interceptor that implements WebSphere Application Serve TAI interface is provided to support WebSEAL Version 4.1. If you plan to use WebSEAL 5.1, it is recommended that you migrate to use the new com.ibm.ws.security.web.TAMTrustAssociationInterceptorPlus interceptor which implements the new com.ibm.wsspi.security.tai.TrustAssociationInterceptor interface.

**com.ibm.ws.security.web.TAMTrustAssociationInterceptorPlus**
This TAI interceptor implementation that implements the new WebSphere Application Server interface supports WebSphere Application Server Version 5.1.1 and later. The interface supports WebSEAL Version 5.1, but does not support WebSEAL Version 4.1. For an explanation of security attribute propagation, see "Security attribute propagation" on page 275.




## IBM WebSphere Application Server: WebSEAL Integration

The integration of WebSEAL and WebSphere Application Server security is achieved by placing the WebSEAL server at the front-end as a reverse proxy server. See Figure 2. From a WebSEAL management perspective, a junction is created with WebSEAL on one end, and the product Web server on the other end. A junction is a logical connection created to establish a path from the WebSEAL server to another server.

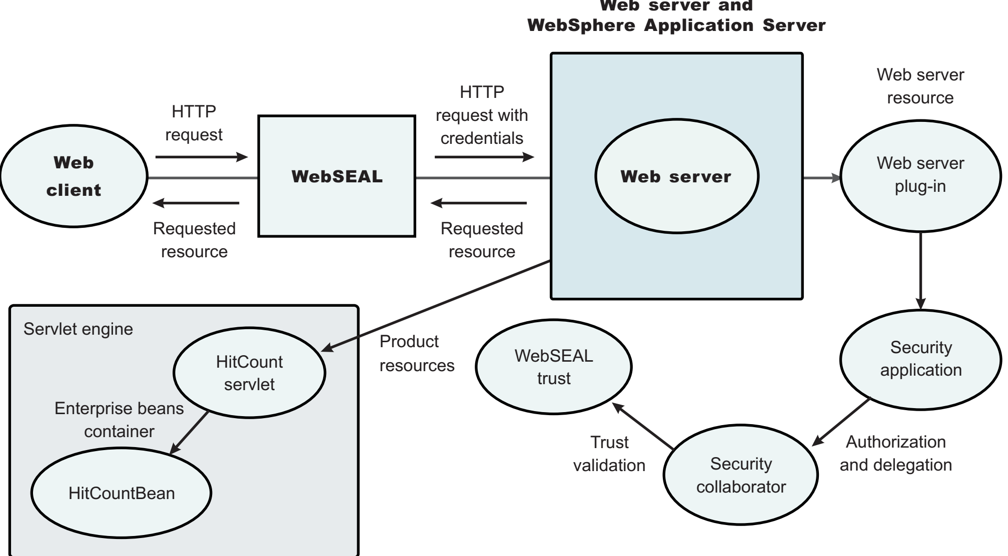
In this setup, a request for Web resources stored in a protected domain of the product is submitted to the WebSEAL server where it is authenticated against the WebSEAL security realm. If the requesting user has access to the junction, the request is transmitted to the WebSphere Application Server HTTP server through the junction, and then to the application server.

Meanwhile, the WebSphere Application Server validates every request that comes through the junction to ensure that the source is a trusted party. This process is referenced as *validating the trust* and it is performed by a WebSEAL product-designated interceptor. If the validation is successful, the WebSphere Application Server authorizes the request by checking whether the client user has the required permissions to access the Web resource. If so, the Web resource is delivered to the WebSEAL server, through the Web server, which then gives it to the client user.

## WebSEAL server

The policy director delegates all of the Web requests to its Web component, the WebSEAL server. One of the major functions of the server is to perform authentication of the requesting user. The WebSEAL server

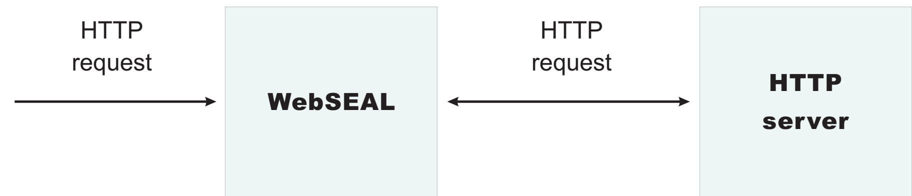
consults a Lightweight Directory Access Protocol (LDAP) directory. It can also map the original user ID to another user ID, such as when global single signon (GSO) is used.

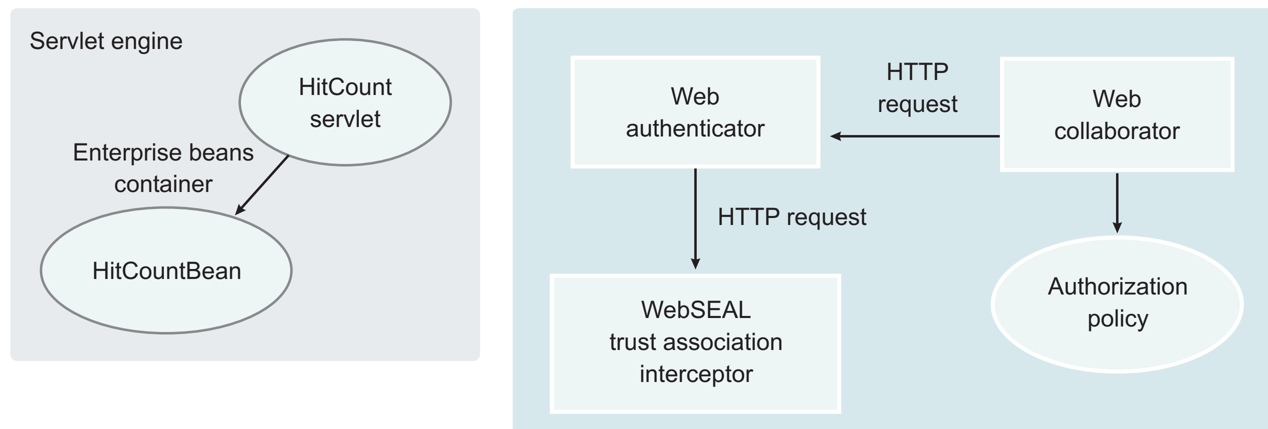



For successful authentication, the server plays the role of a client to WebSphere Application Server when channeling the request. The server needs its own user ID and password to identify itself to WebSphere Application Server. This identity must be valid in the security realm of WebSphere Application Server. The WebSEAL server replaces the basic authentication information in the HTTP request with its own user ID and password. In addition, WebSphere Application Server must determine the credentials of the requesting client so that the application server has an identity to use as a basis for its authorization decisions. This information is transmitted through the HTTP request by creating a header called `iv-creds` with the Tivoli Access Manager user credentials as its value.

### HTTP server

The junction created in the WebSEAL server must get to the HTTP server that serves as the product front end. However, the HTTP server is shielded from knowing that trust association is used. As far as it is concerned, the WebSEAL product is just another HTTP client, and as part of its normal routines, it sends the HTTP request to the product. The only requirement on the HTTP server is a Secure Sockets Layer (SSL) configuration using server authentication only. This requirement protects the requests that flow within the junction.




**Web collaborator**

When trust association is enabled, the Web collaborator manages the interceptors that are configured in the system. It loads and initializes these interceptors when you restart your servers. When a request is passed to WebSphere Application Server by the Web server, the Web collaborator eventually receives the request for a security check. Two actions must take place:

1. The request must be authenticated.
2. The request must be authorized.

The Web authenticator is called to authenticate the request by passing the HTTP request. If successful, a good credential record is returned by the authenticator, which the Web collaborator uses to base its authorization for the requested resource. If the authorization succeeds, the Web collaborator indicates to WebSphere Application Server that the security check has succeeded and that the requested resource can be served.

**Web authenticator**

The Web authenticator is asked by the Web collaborator to authenticate a given HTTP request. Knowing that trust association is enabled, the task of the Web authenticator is to find the appropriate trust association interceptor to direct the request for processing. The Web authenticator queries every available interceptor. If no target interceptor is found, the Web authenticator processes the request as though trust association is not enabled.

For an HTTP request sent by the WebSEAL server, the WebSEAL trust association interceptor replies with a positive response to the Web authenticator. Subsequently, the interceptor is asked to validate its trust association with the WebSEAL server and retrieve the Subject, using the new trust association interface (TAI) interface, or user ID, using the old TAI interface, of the original user client.

**Note:** The new Trust Association Interceptor (TAI) interface, `com.ibm.wsspi.security.tai.TrustAssociationInterceptor`, supports several new features and is

different from the existing `com.ibm.websphere.security.TrustAssociationInterceptor` interface. Although the existing interface is still supported, it is being deprecated in a future release. Refer to X for more information.

WebSphere Application Server Version 4 through WebSphere Application Server Version 5.x support the `com.ibm.websphere.security.TrustAssociationInterceptor.java` interface. WebSphere Application Server Version 6 supports the `com.ibm.wsspi.security.tai.TrustAssociationInterceptor` interface

For more information, see“Trust association interceptor support for Subject creation” on page 112.

**Trust association interceptor interface**

The intent of the trust association interceptor interface is to have reverse proxy security servers (RPSS) exist as the exposed entry points to perform authentication and coarse-grained authorization, while the WebSphere Application Server enforces further fine-grained access control. Trust associations improve security by reducing the scope and risk of exposure.

In a typical e-business infrastructure, the distributed environment of a company consists of Web application servers, Web servers, legacy systems, and one or more RPSS, such as the Tivoli WebSEAL product. Such reverse proxy servers, front-end security servers, or security plug-ins registered within Web servers, guard the HTTP access requests to the Web servers and the Web application servers. While protecting access to the Uniform Resource Identifiers (URIs), these RPSS perform authentication, coarse-grained authorization, and request routing to the target application server.

**Using the trust association interceptor feature**

The following points further describe the benefits of the trust association interceptor (TAI) feature:
- RPSS can authenticate WebSphere Application Server users up front and send credential information about the authenticated user to the product so that the product can trust the RPSS to perform authentication and not prompt the end user for authentication data later. The strength of the trust relationship between RPSS and the product is based on the criteria of trust association that is particular to a RPSS and enforced through the TAI implementation. This level of trust might need relaxing based on the environment. Be aware of the vulnerabilities in cases where the RPSS is not trusted, based on a security technology.
- The end user credentials most likely are sent in a special format as part of the Hypertext Transfer Protocol (HTTP) headers as in the case of RPSS authentication. The credentials can be a special header or a cookie. The data that passes is implementation specific, and the TAI feature considers this fact and accommodates the idea. The TAI implementation works with the credential data and returns a Subject, using the new TAI interface, or a user ID, using the old TAI interface, that represents the end user. WebSphere Application Server uses the information to enforce security policies.

## Configuring trust association interceptors

These steps are required to use either a WebSEAL trust association interceptor or your own trust association interceptor with a reverse proxy security server. WebSphere Application Server enables you to use multiple trust association interceptors. The Application Server uses the first interceptor that can handle the request.

1. Access the administrative console by typing `http://localhost:`*`port_number`*`/ibm/console` in a Web browser. Port 9060 is the default port number for accessing the administrative console. During installation, however, you might have specified a different port number. Use the appropriate port number.
2. Click **Security** > **Global security**.
3. Under Authentication mechanisms, click **LTPA**.
4. Under Additional properties, click **Trust Association**.

5. Select the **Enable trust association** option.
6. Under Additional properties, click **Interceptors**. The default value appears.
7. Verify that the appropriate trust association interceptors are listed. If you need to use a WebSEAL trust association interceptor, see "Configuring single signon using the trust association interceptor" on page 181 or "Configuring single signon using trust association interceptor ++" on page 182. If you are not using WebSEAL and need to use a different interceptor, complete the following steps:
   a. Select both the `com.ibm.ws.security.web.WebSealTrustAssociationInterceptor` and the `com.ibm.ws.security.web.TAMTrustAssociationInterceptorPlus` class name and click **Delete**.
   b. Click **New** and specify a trust association interceptor.

Enables trust association.
1. If you are enabling security, make sure that you complete the remaining steps for enabling security.
2. Save, stop and restart all of the product servers (deployment managers, nodes and Application Servers) for the changes to take effect.

## Trust association settings

Trust association enables the integration of IBM WebSphere Application Server security and third-party security servers. More specifically, a reverse proxy server can act as a front-end authentication server while the product applies its own authorization policy onto the resulting credentials passed by the proxy server. Use this page to configure trust association settings.

To view this administrative console page, complete the following steps:
1. Click **Security > Global security**.
2. Under Authentication, click **Authentication mechanisms > LTPA**.
3. Under Additional properties, click **Trust association**.

When security is enabled and any of these properties change, go to the **Global security** panel and click **Apply** to validate the changes.

***Enable trust association:***

Specifies whether trust association is enabled.

| | |
|---|---|
| **Data type:** | Boolean |
| **Default:** | Disable |
| **Range:** | Enable or Disable |

## Trust association interceptor collection

Use this page to specify trust information for reverse security proxy servers.

To view this administrative console page, complete the following steps:
1. Click **Security > Global security**.
2. Under Authentication, click **Authentication mechanisms > LTPA**.
3. Under Additional Properties, click **Trust association > Interceptors**.

When security is enabled and any of these properties are changed, go to the Global security panel and click **Apply** to validate the changes.

***Interceptor class name:***

Specifies the trust association interceptor class name.

**Data type**
String

**Default**
`com.ibm.ws.security.web.WebSealTrustAssociationInterceptor`

# Single signon

With single signon (SSO) support, Web users can authenticate once when accessing both WebSphere Application Server resources, such as HTML, JavaServer Pages (JSP) files, servlets, enterprise beans, and Lotus Domino resources, such as documents in a Domino database, or accessing resources in multiple WebSphere Application Server domains.

Web users can authenticate once to a WebSphere Application Server or to a Domino server. Without logging in again, Web users can access any other WebSphere Application Servers or Domino servers in the same Domain Name Service (DNS) domain that are enabled for SSO. This authentication is accomplished by configuring the WebSphere Application Servers and the Domino servers to share authentication information.

Enable SSO among WebSphere Application Servers by configuring SSO for WebSphere Application Server. To enable SSO between WebSphere Application Servers and Domino servers, you must configure SSO for both WebSphere Application Server and for Domino.

**Prerequisites and conditions**

To take advantage of support for single signon between WebSphere Application Servers or between WebSphere Application Server and a Domino server, applications must meet the following prerequisites and conditions:

- Verify that all servers are configured as part of the same DNS domain. For example, if the DNS domain is specified as `mycompany.com`, then SSO is effective with any Domino server or WebSphere Application Server on a host that is part of the mycompany.com domain, for example, a.mycompany.com and b.mycompany.com.
- Verify that all servers share the same user registry.

  This registry can be either a supported Lightweight Directory Access Protocol (LDAP) directory server or, if SSO is configured between two WebSphere Application Servers, a custom user registry.Domino servers do not support custom registries, but you can use a Domino-supported registry as a custom registry within WebSphere Application Server. For more information on custom registries, see *Introduction to custom registries*.

  You can use a Domino directory (configured for LDAP access) or other LDAP directory for the user registry. The LDAP directory product must have WebSphere Application Server support. Supported products include both Domino and IBM SecureWay LDAP directory servers. Regardless of the choice to use an LDAP or a custom registry, the SSO configuration is the same. The difference is in the configuration of the registry.
- Define all users in a single LDAP directory. Using LDAP referrals to connect more than one directory together is not supported. Using multiple Domino directory assistance documents to access multiple directories also is not supported.
- Enable HTTP cookies in browsers because the authentication information that is generated by the server is transported to the browser in a cookie. The cookie is then used to propagate the authentication information for the user to other servers, exempting the user from entering the authentication information for every request to a different server.
- For a Domino server:
  - Domino Release 5.0.6a for iSeries 400 or later and Domino Release 5.0.5 or later for other platforms are supported.
  - A Lotus Notes client Release 5.0.5 or later is required for configuring the Domino server for SSO.
  - You can share authentication information across multiple Domino domains.
- For WebSphere Application Server:

- WebSphere Application Server Version 3.5 or later for all platforms is supported.
- You can use any HTTP Web server supported by WebSphere Application Server.
- You can share authentication information across multiple product administrative domains.
- Basic authentication (user ID and password) using the basic and form-login mechanisms is supported.
- By default, WebSphere Application Server does a case-sensitive comparison for authorization. This comparison implies that a user who is authenticated by Domino matches the entry exactly (including the base distinguished name) in the WebSphere Application Server authorization table. If case sensitivity is not considered for the authorization, enable the **Ignore Case** property in the LDAP user registry settings.

## Configuring single signon

With single signon (SSO) support, Web users can authenticate once when accessing Web resources across multiple WebSphere Application Servers. Form login mechanisms for Web applications require that SSO is enabled.

SSO is supported only when Lightweight Third Party Authentication (LTPA) is the authentication mechanism.

When SSO is enabled, a cookie is created containing the LTPA token and inserted into the HTTP response. When the user accesses other Web resources in any other WebSphere Application Server process in the same domain name service (DNS) domain, the cookie is sent in the request. The LTPA token is then extracted from the cookie and validated. If the request is between different cells of WebSphere Application Servers, you must share the LTPA keys and the user registry between the cells for SSO to work. The realm names on each system in the SSO domain are case sensitive and must match identically.

Windows For local OS on the Windows platform, the realm name is the domain name if a domain is in use or the machine name.

UNIX On the Linux or UNIX platforms, the release name is the same as the host name.

For the Lightweight Directory Access Protocol (LDAP) the realm name is the host:port realm name of the LDAP server. The LTPA authentication mechanism requires that you enable SSO if any of the Web applications have form login as the authentication method.

Because single signon is a subset of LTPA, it is recommended that you read “Lightweight Third Party Authentication” on page 160 for more information.

When you enable security attribute propagation, the following cookies are added to the response:

**LtpaToken**
: The LtpaToken is used for interoperating with previous releases of WebSphere Application Server. This token contains the authentication identity attribute only.

**LtpaToken2**
: LtpaToken2 contains stronger encryption and enables you to add multiple attributes to the token. This token contains the authentication identity and additional information such as the attributes used for contacting the original login server and the unique cache key for looking up the Subject when considering more than just the identity in determining uniqueness.

For more information, see "Security attribute propagation" on page 275.

| Token type | Purpose | How to specify |
|---|---|---|
| LtpaToken only | This token type is used for the same SSO behavior existing in WebSphere Application Server Version 5.1 and previous releases. Also, this token type is interoperable with those previous releases. | Disable the **Web inbound security attribute propagation** option located in the SSO configuration panel in the administrative console. To access this panel, complete the following steps:<br>1. Click **Security > Global security**.<br>2. Under Authentication, click **Authentication mechanisms > LTPA**.<br>3. Under Additional properties, click **Single signon (SSO)**. |
| LtpaToken2 only | This token type is used for Web inbound security attribute propagation and uses the AES, CBC, PKCS5 padding encryption strength (128 bit key size). However, this token type is not interoperable with releases prior to WebSphere Application Server Version 5.1.1. The token type allows for multiple attributes specified in the token (mostly containing information to contact the original login server). | Enable the **Web inbound security attribute propagation** option in the SSO configuration panel within the administrative console. Disable the **Interoperability mode** option in the SSO configuration panel within the administrative console. To access this panel, complete the following steps:<br>1. Click **Security > Global security**.<br>2. Under Authentication, click **Authentication mechanisms > LTPA**.<br>3. Under Additional properties, click **Single signon (SSO)**. |
| LtpaToken and LtpaToken2 | These tokens together support both of the previous two options. The token types are interoperable with releases prior to WebSphere Application Server Version 5.1.1 because LtpaToken is present. The security attribute propagation function is enabled because the LtpaToken2 is present. | Enable the **Web inbound security attribute propagation** option in the SSO configuration panel within the administrative console. Enable the **Interoperability mode** option in the SSO configuration panel within the administrative console. To access this panel, complete the following steps:<br>1. Click **Security > Global security**.<br>2. Under Authentication, click **Authentication mechanisms > LTPA**.<br>3. Under Additional properties, click **Single signon (SSO)**. |

The following steps are required to configure SSO for the first time.

1. Access the administrative console by typing `http://localhost:`*`port_number`*`/ibm/console` in a Web browser. Port 9060 is the default port number for accessing the administrative console. During installation, however, you might have specified a different port number. Use the appropriate port number.
2. Click **Security > Global security** .
3. Under Authentication, click **Authentication mechanisms > LTPA**.
4. Under Additional properties, click **Single signon (SSO)**.
5. Click the **Enabled** option if SSO is disabled. After you click **Enabled**, make sure that you complete the remaining steps to enable security.
6. Click the **Requires SSL** option if all of the requests are expected to use HTTPS.

7. Enter the fully-qualified domain names in the **Domain name** field where SSO is effective. The cookie is sent for all of the servers that are contained within the domains that you specify in this field. If you specify domain names, they must be fully qualified. If the domain name is not fully qualified, WebSphere Application Server does not set a domain name value for the LtpaToken cookie and SSO is valid only for the server that created the cookie.

   You can configure the **Domain name** field using any of the following values:

| Domain name value type | Example |
|---|---|
| Blank | |
| Single domain name | austin.ibm.com |
| UseDomainFromURL | UseDomainFromURL |
| Multiple domain names | austin.ibm.com;raleigh.ibm.com |
| Multiple domain names and UseDomainFromURL | • austin.ibm.com;raleigh.ibm.com<br>• UseDomainFromURL |

   If you specify the UseDomainFromURL, WebSphere Application Server sets the SSO domain name value to the domain of the host that makes the request. For example, if an HTTP request comes from server1.raleigh.ibm.com, WebSphere Application Server sets the SSO domain name value to raleigh.ibm.com.

   **Tip:** The value, `UseDomainFromURL`, is case insensitive. You can type `usedomainfromurl` to use this value.

   When you specify multiple domains, you can use the following delimiters: a semicolon (;), a space ( ), a comma (,), or a pipe (|). WebSphere Application Server searches the specified domains in order from left to right. Each domain is compared with the host name of the HTTP request until the first match is located. For example, if you specify ibm.com; austin.ibm.com and a match is found in the ibm.com domain first, WebSphere Application server does not continue to search for a match in the austin.ibm.com domain. However, if a match is not found in either the ibm.com or austin.ibm.com domains, then WebSphere Application Server does not set a domain for the LtpaToken cookie.

   For more information, see “Single signon settings” on page 175.
8. **Optional:** Enable the **Interoperability mode** option if you want to allow SSO connections in WebSphere Application Server version 5.1.1 or later to interoperate with previous versions of the application server. This option sets the old-style LtpaToken into the response so it can be sent to other servers that work only with this token type. However, this option applies only when the **Web inbound security attribute propagation** option is enabled. In this case, both the LtpaToken and LtpaToken2 are added to the response. Otherwise, only the LtpaToken2 is added to the response. If the **Web inbound security attribute propagation** option is disabled, then only the LtpaToken is added to the response.
9. **Optional:** Enable the **Web inbound security attribute propagation** option if you want information added during the login at a specific front-end server to propagate to other front-end servers. The SSO token does not contain any sensitive attributes, but does understand where the original login server exists in cases where it needs to contact that server to retrieve serialized information. It also contains the cache look up value for finding the serialized information in DynaCache, if both front-end servers are configured in the same DRS replication domain. For more information, see “Security attribute propagation” on page 275.

   **Important:** If the following statements are true, it is recommended that you disable the **Web inbound security attribute propagation** option for performance reasons:
   - You do not have any specific information added to the Subject during a login that cannot be obtained at a different front-end server.
   - You did not add custom attributes to the PropagationToken using WSSecurityHelper application programming interfaces (APIs).

If you find you are missing custom information in the Subject, re-enable the **Web inbound security attribute propagation** option to see if the information is propagated successfully to other front-end application servers. If you disable SSO, but use a trust association interceptor instead, you might still need to enable the **Web inbound security attribute propagation** option if you want to retrieve the same Subject generated at different front-end servers.

10. Click **OK**.

For the changes to take effect, save, stop, and restart all the product servers (deployment managers, nodes and Application Servers).

## Single signon settings

Use this page to set the configuration values for single signon (SSO).

To view this administrative console page, complete the following steps:

1. Click **Security > Global Security**.
2. Under Authentication mechanisms, click **LTPA**.
3. Under Additional properties, click **Single signon (SSO)**.

***Enabled:***

Specifies that the single signon function is enabled.

Web applications that use J2EE FormLogin style login pages (such as the WebSphere Application Server administrative console) require single signon (SSO) enablement. Only disable SSO for certain advanced configurations where LTPA SSO-type cookies are not required.

| | |
|---|---|
| **Data type:** | Boolean |
| **Default:** | Enabled |
| **Range:** | Enabled or Disabled |

***Requires SSL:***

Specifies that the single signon function is enabled only when requests are made over HTTPS Secure Sockets Layer (SSL) connections.

| | |
|---|---|
| **Data type:** | Boolean |
| **Default:** | Disable |
| **Range:** | Enable or Disable |

***Domain name:***

Specifies the domain name (.ibm.com, for example) for all single signon hosts.

WebSphere Application Server uses all the information after the first period, from left to right, for the domain names. If this field is not defined, the Web browser defaults the domain name to the host name where the Web application is running. Also, single signon is then restricted to the application server host name and does not work with other application server host names in the domain.

You can specify multiple domains separated by a semicolon (;), a space ( ), a comma (,), or a pipe (|). Each domain is compared with the host name of the HTTP request until the first match is located. For example, if you specify ibm.com;austin.ibm.com and a match is found in the ibm.com domain first,

WebSphere Application server does not match the austin.ibm.com domain. However, if a match is not found in either ibm.com or austin.ibm.com, then WebSphere Application Server does not set a domain for the LtpaToken cookie.

If you specify `UseDomainFromURL`, WebSphere Application Server sets the SSO domain name value to the domain of the host used in the URL. For example, if an HTTP request comes from `server1.raleigh.ibm.com`, WebSphere Application Server sets the SSO domain name value to `raleigh.ibm.com`.

**Tip:** The UseDomainFromURL value is case insensitive. You can type `usedomainfromurl` to use this value.

| **Data type:** | String |
| --- | --- |

***Interoperability mode:***

Specifies that an interoperable cookie is sent to the browser to support back-level servers.

In WebSphere Application Server, Version 6 and later, a new cookie format is needed by the security attribute propagation functionality. When the interoperability mode flag is enabled, the server can send a maximum of two single signon (SSO) cookies back to the browser. In some cases, the server just sends the interoperable SSO cookie.

***Web inbound security attribute propagation:***

When Web inbound security attribution propagation is enabled, security attributes are propagated to front-end application servers. When this option is disabled, the single signon (SSO) token is used to log in and recreate the Subject from the user registry. If you disable this option, the Web inbound login module functions the same as it did in previous releases.

If the application server is a member of a cluster and the cluster is configured with a distributed replication service (DRS) domain, then propagation occurs. If DRS is not configured, then the SSO token contains the originating server information. With this information the receiving server can contact the originating server using an MBean call to get the original serialized security attributes.

## Troubleshooting single signon configurations

This article describes common problems in configuring single signon (SSO) between a WebSphere Application Server and a Domino server and suggests possible solutions.

- Failure to save the Domino Web SSO configuration document

  The client must find Domino server documents for the participating SSO Domino servers. The Web SSO configuration document is encrypted for the servers that you specify. The home server that is indicated by the client location record must point to a server in the Domino domain where the participating servers reside. This pointer ensures that lookups can find the public keys of the servers.

  If you receive a message stating that one or more of the participating Domino servers cannot be found, then those servers cannot decrypt the Web SSO configuration document or perform SSO.

  When the Web SSO configuration document is saved, the status bar indicates how many public keys are used to encrypt the document by finding the listed servers, authors, and administrators in the document.
- Failure of the Domino server console to load the Web SSO configuration document at Domino HTTP server startup

  During configuration of SSO, the server document is configured for **Multi-Server** in the **Session Authentication** field. The Domino HTTP server tries to find and load a Web SSO configuration document during startup. The Domino server console reports the following information if a valid document is found and decrypted: `HTTP: Successfully loaded Web SSO Configuration`.

If a server cannot load the Web SSO configuration document, SSO does not work. In this case, a server reports the following message: `HTTP: Error Loading Web SSO configuration`. Reverting to single-server session authentication.

Verify that only one Web SSO Configuration document is in the Web Configurations view of the Domino directory and in the $WebSSOConfigs hidden view. You cannot create more than one document, but you can insert additional documents during replication.

If you can verify only one Web SSO Configuration document, consider another condition. When the public key of the Server document does not match the public key in the ID file, this same error message can display. In this case, attempts to decrypt the Web SSO configuration document fail and the error message is generated.

This situation can occur when the ID file is created multiple times but the Server document is not updated correctly. Usually, an error message is displayed on the Domino server console stating that the public key does not match the server ID. If this situation occurs, then SSO does not work because the document is encrypted with a public key for which the server does not possess the corresponding private key.

To correct a key-mismatch problem:

1. Copy the public key from the server ID file and paste it into the Server document.
2. Create the Web SSO configuration document again.

- Authentication fails when accessing a protected resource.

  If a Web user is repeatedly prompted for a user ID and password, SSO is not working because either the Domino or the WebSphere Application Server security server cannot authenticate the user with the Lightweight Directory Access Protocol (LDAP) server. Check the following possibilities:

  – Verify that the LDAP server is accessible from the Domino server machine. Use the **TCP/IP ping** utility to check TCP/IP connectivity and to verify that the host machine is running.

  – Verify that the LDAP user is defined in the LDAP directory. Use the **ldapsearch** utility to confirm that the user ID exists and that the password is correct. For example, you can run the following command, entered as a single line:

    You can use the OS/400 Qshell, a UNIX shell, or a Windows DOS prompt

    ```
    % ldapsearch -D "cn=John Doe, ou=Rochester, o=IBM, c=US" -w mypassword
    -h myhost.mycompany.com -p 389
    -b "ou=Rochester, o=IBM, c=US" (objectclass=*)
    ```

    (The percent character (%) indicates the prompt and is not part of the command.) A list of directory entries is expected. Possible error conditions and causes are contained in the following list:

    - No such object: This error indicates that the directory entry referenced by either the user's distinguished name (DN) value, which is specified after the -D option, or the base DN value, which is specified after the -b option, does not exist.
    - Invalid credentials: This error indicates that the password is invalid.
    - Cannot contact the LDAP server: This error indicates that the host name or port specified for the server is invalid or that the LDAP server is not running.
    - An empty list means that the base directory specified by the -b option does not contain any directory entries.

  – If you are using the user's short name (or user ID) instead of the distinguished name, verify that the directory entry is configured with the short name. For a Domino directory, verify the **Short name/UserID** field of the Person document. For other LDAP directories, verify the userid property of the directory entry.

  – If Domino authentication fails when using an LDAP directory other than a Domino directory, verify the configuration settings of the LDAP server in the Directory assistance document in the Directory assistance database. Also verify that the Server document refers to the correct Directory assistance document. The following LDAP values specified in the Directory Assistance document must match the values specified for the user registry in the WebSphere administrative domain:

    - Domain name
    - LDAP host name
    - LDAP port

- Base DN

Additionally, the rules defined in the Directory assistance document must refer to the base distinguished name (DN) of the directory containing the directory entries of the users.

You can trace Domino server requests to the LDAP server by adding the following line to the server `notes.ini` file:

```
webauth_verbose_trace=1
```

After restarting the Domino server, trace messages are displayed in the Domino server console as Web users attempt to authenticate to the Domino server.

- Authorization failure when accessing a protected resource.

  After authenticating successfully, if an authorization error message is displayed, security is not configured correctly. Check the following possibilities:

  – For Domino databases, verify that the user is defined in the access-control settings for the database. Refer to the Domino Administrative documentation for the correct way to specify the user's DN. For example, for the `DN cn=John Doe, ou=Rochester, o=IBM, c=US`, the value on the access-control list must be set as `John Doe/Rochester/IBM/US`.

  – For resources protected by WebSphere Application Server, verify that the security permissions are set correctly.

    - If granting permissions to selected groups, make sure that the user attempting to access the resource is a member of the group. For example, you can verify the members of the groups by using the following Web site to display the directory contents: `Ldap://myhost.mycompany.com:389/ou=Rochester, o=IBM, c=US??sub`
    - If you have changed the LDAP configuration information (host, port, and base DN) in a WebSphere Application Server administrative domain since the permissions were set, the existing permissions are probably invalid and need to be recreated.

- SSO failure when accessing protected resources.

  If a Web user is prompted to authenticate with each resource, SSO is not configured correctly. Check the following possibilities:

  1. Configure both the WebSphere Application Server and the Domino server to use the same LDAP directory. The HTTP cookie used for SSO stores the full DN of the user, for example, `cn=John Doe, ou=Rochester, o=IBM, c=US`, and the domain name service (DNS) domain.
  2. Define Web users by hierarchical names if the Domino Directory is used. For example, update the **User name** field in the Person document to include names of this format as the first value: `John Doe/Rochester/IBM/US`.
  3. Specify the full DNS server name, not just the host name or TCP/IP address for Web sites issued to Domino servers and WebSphere Application Servers configured for SSO. For browsers to send cookies to a group of servers, the DNS domain must be included in the cookie, and the DNS domain in the cookie must match the Web address. (This requirement is why you cannot use cookies across TCP/IP domains.)
  4. Configure both Domino and the WebSphere Application Server to use the same DNS domain. Verify that the DNS domain value is exactly the same, including capitalization. The DNS domain value is found on the Configure Global Security Settings panel of the WebSphere Application Server administrative console and in the Web SSO Configuration document of a Domino server. If you make a change to the Domino Web SSO Configuration document, replicate the modified document to all of the Domino servers participating in SSO.
  5. Verify that the clustered Domino servers have the host name populated with the full DNS server name in the Server document. By using the full DNS server name, Domino Internet Cluster Manager (ICM) can redirect to cluster members using SSO. If this field is not populated, by default, ICM redirects Web addresses to clustered Web servers by using the host name of the server only. It cannot send the SSO cookie because the DNS domain is not included in the Web address. To correct the problem:
     a. Edit the Server document.
     b. Click **Internet Protocols > HTTP** tab.
     c. Enter the full DNS name of the server in the **Host names** field.

6. If a port value for an LDAP server was specified for a WebSphere Application Server administrative domain, edit the Domino Web SSO configuration document and insert a backslash character (\) into the value of the **LDAP Realm** field before the colon character (:). For example, replace `myhost.mycompany.com:389` with `myhost.mycompany.com\:389`.

## Single signon using WebSEAL or the Tivoli Access Manager plug-in for Web servers

Either Tivoli Access Manager WebSEAL or Tivoli Access Manager plug-in for Web servers can be used as reverse proxy servers to provide access management and single signon (SSO) capability to WebSphere Application Server resources. With such an architecture, either WebSEAL or the plug-in authenticates users and forwards the collected credentials to WebSphere Application Server in the form of an IV Header. Two types of single signon are available, the TAI interface and the new TAI interface, so named as both use WebSphere Application Server trust association interceptors (TAIs). With TAI, the end-user name is extracted from the HTTP header and forwarded to embedded Tivoli Access Manager where it is used to construct the client credential information and authorize the user. The difference with the new TAI interface is that all user credential information is available in the HTTP header (not just user name). The new TAI is the more efficient of the two solutions as an Lightweight Directory Access Protocol (LDAP) call is not required as it is with TAI. TAI functionality is retained for backwards compatibility.

The following tasks need to be completed to enable single signon to WebSphere Application Server using either WebSEAL or the plug-in for Web servers. These tasks assume that embedded Tivoli Access Manager is configured for use.

1. "Creating a trusted user account in Tivoli Access Manager"
2. "Configuring WebSEAL for use with WebSphere Application Server" or "Configuring Tivoli Access Manager plug-in for Web servers for use with WebSphere Application Server" on page 180
3. "Configuring single signon using the trust association interceptor" on page 181 or "Configuring single signon using trust association interceptor ++" on page 182

## Creating a trusted user account in Tivoli Access Manager

Tivoli Access Manager Trust Association Interceptors require the creation of a trusted user account in the shared LDAP user registry. This is the ID and password that WebSEAL uses to identify itself to WebSphere Application Server. To prevent potential vulnerabilities, do not use sec_master as the trusted user account and ensure the password you use is unique and generated randomly. The trusted user account should be used for the TAI or TAI++ only.

Use either the Tivoli Access Manager **pdadmin** command line utility or Web Portal Manager to create the trusted user. For example, from the **pdadmin** command line:

```
pdadmin> user create webseal_userid webseal_userid_DN firstname surname password
pdadmin> user modify webseal_userid account-valid yes
```

"Configuring WebSEAL for use with WebSphere Application Server" or "Configuring Tivoli Access Manager plug-in for Web servers for use with WebSphere Application Server" on page 180

## Configuring WebSEAL for use with WebSphere Application Server

A junction must be created between WebSEAL and WebSphere Application Server. This junction will carry the iv-creds (for TAI++) or iv-user (for TAI) and the HTTP basic authentication headers with the request. While WebSEAL can be configured to pass the end user identity in other ways, the iv-creds header is the only one supported by the TAI++ and iv-user the only one supported by TAI.

We recommend that communications over the junction use SSL for increased security. Setting up SSL across this junction requires that you configure the HTTP Server used by WebSphere Application Server, and WebSphere Application Server itself, to accept inbound SSL traffic and route it correctly to WebSphere

Application Server. This requires importing the necessary signing certificates into the WebSEAL certificate keystore, and possibly also the HTTP Server certificate keystore.

Create the junction between WebSEAL and the WebSphere Application Server using the **-c iv-creds** option for TAI++ and **-c iv-user** for TAI. For example (commands are entered as one line):

**TAI++**

```
server task webseald-server create -t ssl -b supply -c iv-creds
 -h host_name -p websphere_app_port_number junction_name
```

**TAI**

```
server task webseald-server create -t ssl -b supply -c iv-user
 -h host_name -p websphere_app_port_number junction_name
```

**Notes:**

1. If warning messages are displayed about the incorrect setup of certificates and key databases, delete the junction, correct problems with the key databases and re-create the junction.
2. The junction can be created as `-t tcp` or `-t ssl` depending on your requirements.

For single signon to WebSphere Application Server the SSO password must be set in WebSEAL. To set the password, complete the following steps:

1. Edit the WebSEAL configuration file, *webseal_install_directory*/etc/webseald-default.conf and set the following parameter, **basicauth-dummy-passwd=*webseal_userid_passwd***. Where **webseal_userid_passwd** is the SSO password for the trusted user account set in “Creating a trusted user account in Tivoli Access Manager” on page 179.
2. Restart WebSEAL.

For more details and options about how to configure junctions between WebSEAL and WebSphere Application Server, including other options for specifying the WebSEAL server identity, refer to the *Tivoli Access Manager WebSEAL Administration Guide* as well as to the documentation for the HTTP Server you are using with your WebSphere Application Server. Tivoli Access Manager documentation is available at http://publib.boulder.ibm.com/tividd/td/tdprodlist.html.

## Configuring Tivoli Access Manager plug-in for Web servers for use with WebSphere Application Server

Tivoli Access Manager plug-in for Web servers can be used as a security gateway for your protected WebSphere Application resources. With such an arrangement the plug-in authorizes all user requests before passing the authorized user’s credentials onto WebSphere Application Server in the form of an iv-creds header. Trust between the plug-in and WebSphere Application Server is established through use of basic authentication headers containing the single signon (SSO) user password.

In the following example Tivoli Access Manager plug-in for Web Servers Version 5.1 configuration shows IV headers configured for post-authorization processing and basic authentication configured as the authentication mechanism and for post-authorization processing. After a request has been authorized the basic authentication header is removed from the request (`strip-hdr = always`) and a new one added (`add-hdr = supply`). Included in this new header is the password set when the SSO user was created in “Creating a trusted user account in Tivoli Access Manager” on page 179. This password needs to be specified in the **supply-password** parameter and is passed in the newly created header. This basic authentication header enables trust between WebSphere Application Server and the plug-in.

An iv-creds header is also added (`generate = iv-creds`) which contains the credential information of the user passed onto WebSphere Application Server. Note also that session cookies are used to maintain session state.

```
[common-modules]
authentication = BA
session = session-cookie
post-authzn = BA
post-authzn = iv-headers

[iv-headers]
accept = all
generate = iv-creds

[BA]
strip-hdr = always
add-hdr = supply
supply-password = sso_user_password
```

“Configuring single signon using the trust association interceptor” or “Configuring single signon using trust association interceptor ++” on page 182

## Configuring single signon using the trust association interceptor

The following steps are required when setting up security for the first time. Ensure that Lightweight Third Party Authentication (LTPA) is the active authentication mechanism:

1. From the WebSphere Application Server console click **Security > Global security**.
2. Ensure that the **Active authentication mechanism** field is set to *Lightweight Third Party Authentication (LTPA)*. If not, set it and save your changes.

This task is performed to enable single signon using the trust association interceptor. The steps involve setting up trust association and creating the interceptor properties.

1. From the WebSphere Application Server console, click **Security > Global security**.
2. Under Authentication mechanisms, click **LTPA**.
3. Under Additional properties, click **Trust association**.
4. Select the **Enable trust association** option.
5. Under Additional properties, click the **Interceptors** link.
6. Click the **com.ibm.ws.security.web.WebSealTrustAssociationInterceptor** link to use the WebSEAL interceptor. This interceptor is the default.
7. Under Additional properties, click **Custom Properties** .
8. Click **New** to enter the property name and value pairs. Ensure the following parameters are set:

| Option | Description |
|---|---|
| **com.ibm.websphere.security.trustassociation.types** | Ensure *webseal* is listed. |
| **com.ibm.websphere.security.webseal.loginId** | The WebSEAL trusted user as created in “Creating a trusted user account in Tivoli Access Manager” on page 179 The format of the username is the short name representation. This is a mandatory property. If it is not set in WebSphere then TAI initialization will fail. |
| **com.ibm.websphere.security.webseal.id** | The *iv-user* header. That is; com.ibm.websphere.security.webseal.id=iv-user |

| Option | Description |
| --- | --- |
| **com.ibm.websphere.security.webseal.hostnames** | Do not set this property if using Tivoli Access Manager Plug-in for Web Servers. The host names (case sensitive) that are trusted and expected in the request header. For example;com.ibm.websphere.security.webseal.hostnames=host1<br><br>This should also include the proxy host names (if any) unless the com.ibm.websphere.security.webseal.ignoreProxy is set to *true*. A list of servers can be obtained using the server list **pdadmin** command. |
| **com.ibm.websphere.security.webseal.ports** | Do not set this property if using Tivoli Access Manager Plug-in for Web Servers. The corresponding port number of the host names that are expected in the request header. This should also include the proxy ports (if any) unless the com.ibm.websphere.security.webseal.ignoreProxy is set to *true*. For example: com.ibm.websphere.security.webseal.ports=80,443 |
| **com.ibm.websphere.security.webseal.ignoreProxy** | An optional property that if set to *true* or *yes* ignores the proxy host names and ports in the IV header. By default this property is set to *false*. |

9. Click **OK**.
10. Save configuration and logout.
11. Restart WebSphere Application Server.

## Configuring single signon using trust association interceptor ++

The following steps are required when setting up security for the first time. Ensure that LTPA is the active authentication mechanism:

1. From the WebSphere Application Server console, click **Security** > **Global Security**.
2. Ensure that the **Active Authentication Mechanism** field is set to Lightweight Third Party Authentication (LTPA). Save your changes.

This task is performed to enable single signon using trust association interceptor ++. The steps involve setting up trust association and creating the interceptor properties.

1. From the WebSphere Application Server console select, click **Security > Global security**.
2. Under Authentication, click **Authentication mechanisms > LTPA**
3. Under Additional properties, click **Trust association**.
4. Select the **Enable Trust Association** option.
5. Click the **Interceptors** link.
6. Click **com.ibm.ws.security.web.TAMTrustAssociationInterceptorPlus** to use the WebSEAL interceptor. This interceptor is the default.
7. Click the **Custom Properties** link.

8. Click **New** to enter the property name and value pairs. Ensure the following parameters are set:

| Option | Description |
|---|---|
| **com.ibm.websphere.security.webseal.checkViaHeader** | The TAI can be configured so that the via header can be ignored when validating trust for a request. Set this property to *false* if none of the hosts in the via header need to be trusted. When set to *false* the trusted **hostnames** and host **ports** properties do not need to be set. Therefore the only mandatory property when check via header is *false* is com.ibm.websphere.security.webseal.loginId<br><br>The default value of the check via header property is *false*. When using Tivoli Access Manager Plug-in for Web Servers this property should be set to *false*.<br>**Note:** The via header is part of the standard HTTP header that records the server names the request has passed through. |
| **com.ibm.websphere.security.webseal.loginId** | The WebSEAL trusted user as created in "Creating a trusted user account in Tivoli Access Manager" on page 179 The format of the username is the short name representation. This is a mandatory property. If it is not set in WebSphere Application Server, then the TAI initialization fails. |
| **com.ibm.websphere.security.webseal.id** | A comma-separated list of headers that should exist in the request. If not all of the configured headers exist in the request then trust can not be established. The default value for the id property is *iv-creds*. Any other values set in WebSphere Application Server are added to the list along with *iv-creds*, separated by commas. |
| **com.ibm.websphere.security.webseal.hostnames** | Do not set this property if using Tivoli Access Manager Plug-in for Web Servers. The property specifies the host names (case sensitive) that are trusted and expected in the request header. Requests arriving from un-listed hosts might not be trusted. If the checkViaHeader property is not set or is set to false then the trusted host names property has no influence. If the checkViaHeader property is set to true and the trusted host names property is not set then TAI initialization will fail. |
| **com.ibm.websphere.security.webseal.ports** | Do not set this property if using Tivoli Access Manager Plug-in for Web Servers. This property is a comma-separated list of trusted host ports. Requests arriving from unlisted ports might not be trusted. If the checkViaHeader property is not set or is set to false then this property has no influence. If the checkViaHeader property is set to true and the trusted host ports property is not set in WebSphere Application Server then the TAI initialization fails. |

| Option | Description |
| --- | --- |
| **com.ibm.websphere.security.webseal.viaDepth** | A positive integer specifying the number of source hosts in the via header to check for trust. By default, every host in the via header is checked and if any are not trusted then trust cannot be established. The via depth property is used when not all hosts in the via header are required to be trusted. The setting indicates the number of hosts that are required to be trusted.<br><br>As an example, consider the following header:<br><br>`Via: HTTP/1.1 webseal1:7002, 1.1 webseal2:7001`<br><br>If the viaDepth property is not set, is set to 2 or is set to 0, and a request with the previous via header is received then both webseal1:7002 and webseal2:7001 need to be trusted. The following configuration applies:<br><br>`com.ibm.websphere.security.webseal.hostnames = webseal1,webseal2`<br>`com.ibm.websphere.security.webseal.ports = 7002,7001`<br><br>If the via depth property is set to 1 and the previous request is received then only the last host in the via header needs to be trusted. The following configuration applies:<br><br>`com.ibm.websphere.security.webseal.hostnames = webseal2`<br>`com.ibm.websphere.security.webseal.ports = 7001`<br><br>The viaDepth property is set to 0 by default which means all hosts in the via header are checked for trust. |
| **com.ibm.websphere.security.webseal.ssoPwdExpiry** | Once trust has been established for a request the single signon user password is cached saving the need to have the TAI re-authenticate the single signon user with Tivoli Access Manager for every request. The cache timeout period can be modified by setting the single signon password expiry property to the required time in seconds. If the password expiry property is set to 0, the cached password will never expire. The default value for the password expiry property is 600. |
| **com.ibm.websphere.security.webseal.ignoreProxy** | This property can be used to tell the TAI to ignore proxies as trusted hosts. If set to true the comments field of the hosts entry in the via header is checked to determine if a host is a proxy. It must be remembered that not all proxies insert comments in the via header indicating that they are proxies. The default value of the ignoreProxy property is false. If the checkViaHeader property is set to false then the ignoreProxy property has no influence in establishing trust. |

| Option | Description |
| --- | --- |
| **com.ibm.websphere.security.webseal.configURL** | For the TAI to be able to establish trust for a request it requires that SvrSslCfg has been run for the WebSphere Java Virtual Machine resulting in a properties file being created. If this properties file is not at the default URL file://java.home/PdPerm.properties then the correct URL of the properties file must be set in the config URL property. If this property is not set and the SvrSslCfg generated properties file is not in the default location, the TAI initialization fails. The default value for the config URL property is file://${WAS_INSTALL_ROOT}/java/jre/PdPerm.properties |

9. Click **OK**.
10. Save configuration and logout.
11. Restart WebSphere Application Server.

## Global signon principal mapping

The Tivoli Access Manager JACC provider can be used to manage authentication to WebSphere Enterprise Information Systems (EIS) such as databases, transaction processing systems and message queue systems, located within the WebSphere Application Server security domain. Such authentication is achieved using the Global single signon (GSO) Principal Mapper JAAS login module for J2EE Connector Architecture (J2C) resources.

With GSO principal mapping, a special-purpose JAAS login module inserts a credential into the subject header. This is used by the resource adapter to authenticate to the Enterprise Information System (EIS). The JAAS login module used is configured on a per-connection factory basis. The default principal mapping module retrieves the user name and password information from XML configuration files. The Tivoli Access Manager JACC provider bypasses the credential stored in the XML configuration files and instead uses the Tivoli Access Manager GSO database to provide the EIS security domain authentication information.

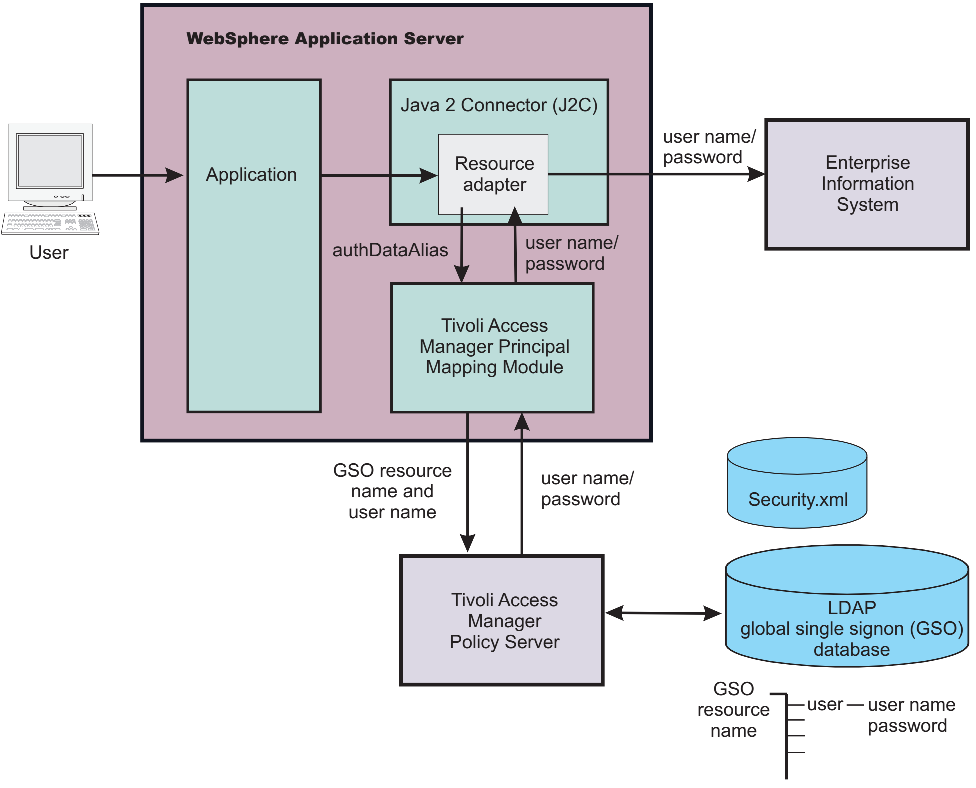
WebSphere Application Server provides a default principal mapping module that associates user credential information with EIS resources. The default mapping module is defined in the WebSphere Application Server administration console on the application login panel. To access the panel, click **Security** > **Global security**. Under JAAS configuration, click **Application logins**. The mapping module name is **DefaultPrincipalMapping**.

The EIS security domain user ID and password are defined under each connection factory by an authDataAlias attribute. The authDataAlias attribute does not contain the user name and password, it contains an alias that refers to a user name and password pair defined elsewhere.

The Tivoli Access Manager Principal Mapping module uses the authDataAlias to determine the GSO resource name and user name required to perform the lookup on the Tivoli Access Manager GSO database. It is the Tivoli Access Manager Policy Server which retrieves the GSO data from the registry.

Tivoli Access Manager stores authentication information on the Tivoli Access Manager GSO database against a resource/user name pair.

**GSO principal mapping architecture**




## Configuring global signon principal mapping

To create a new application login that uses the Tivoli Access Manager GSO database to store the login credentials:

1. Select **Security** > **Global security**.
2. Under Authentication, click **JAAS Configuration > Application logins**
3. Click **New** to create a new JAAS login configuration.
4. Enter the alias name of the new application login. Click **Apply**.
5. Under Additional properties, click **JAAS Login Modules** link to define the JAAS Login Modules.
6. Click **New** and enter the following:

   **Module class name**: com.tivoli.pd.as.gso.AMPrincipalMapper
   **Use Login Module Proxy**: enable
   **Authentication strategy**: REQUIRED

   Click **Apply**
7. In the *Additional Properties* section, click **Custom Properties** to define Login Module-specific values which are passed directly to the underlying Login Modules.
8. Click **New**.

   The Tivoli Access Manager principal mapping module uses the configuration string, authDataAlias, to retrieve the correct user name and password from the security configuration.

   The authDataAlias passed to the module is configured for the J2C ConnectionFactory. Since the authDataAlias is an arbitrary string entered at configuration time, the following scenarios are possible:

   - The authDataAlias contains both the GSO Resource name and the user name. The format of this string is ″Resource/User″

- The authDataAlias contains only the GSO Resource name. The user name is determined using the Subject of the current session.

Which scenario to use is determined by a JAAS configuration option. The details of these options are:

**Name**: com.tivoli.pd.as.gso.AliasContainsUserName
**Value**: True if the alias contains the user name, false if the user name should be retrieved from the security

When entering authDataAliases through the WebSphere Application Server console, the node name is automatically pre-pended to the alias. The JAAS configuration entry is to determine whether this node name should be removed or included as part of the resource name.

**Name**: com.tivoli.pd.as.gso.AliasContainsNodeName
**Value**: True if the alias contains the node name.

Enter each new parameter using the following scenario information as a guide.

**Note:** If the PdPerm.properties configuration file is not located in the default location, JAVA_HOME/PdPerm.properties, then you will also need to add the following property:

Name = com.tivoli.pd.as.gso.AMCfgURL
Value = file:///*path to PdPerm.properties*

**Scenario 1**
**Auth Data Alias** - BackendEIS/eisUser
**Resource** - BackEndEIS
**User** - eisUser
**Principal Mapping Parameters**

| Name | Value |
|---|---|
| delegate | com.tivoli.pdwas.gso.AMPrincipalMapper |
| com.tivoli.pd.as.gso.AliasContainsUserName | true |
| com.tivoli.pd.as.gso.AliasContainsNodeName | false |
| com.tivoli.pd.as.gso.AMLoggingURL | file:///*jlog_props_path* |
| debug | false |

**Scenario 2**
**Auth Data Alias** - BackendEIS
**Resource** - BackEndEIS
**User** - Currently authenticated WAS user
**Principal Mapping Parameters**

| Name | Value |
|---|---|
| delegate | com.tivoli.pdwas.gso.AMPrincipalMapper |
| com.tivoli.pd.as.gso.AliasContainsUserName | false |
| com.tivoli.pd.as.gso.AliasContainsNodeName | false |
| com.tivoli.pd.as.gso.AMLoggingURL | file:///*jlog_props_path* |
| debug | false |

**Scenario 3**
**Auth Data Alias** - nodename/BackendEIS/eisUser
**Resource** - BackEndEIS
**User** - eisUser
**Principal Mapping Parameters**

| Name | Value |
| --- | --- |
| delegate | com.tivoli.pdwas.gso.AMPrincipalMapper |
| com.tivoli.pd.as.gso.AliasContainsUserName | true |
| com.tivoli.pd.as.gso.AliasContainsNodeName | true |
| com.tivoli.pd.as.gso.AMLoggingURL | file:///*jlog_props_path* |
| debug | false |

**Scenario 4**
**Auth Data Alias** - nodename/BackendEIS/eisUser
**Resource** - nodename/BackEndEIS (notice that node name was not removed)
**User** - eisUser
**Principal Mapping Parameters**

| Name | Value |
| --- | --- |
| delegate | com.tivoli.pdwas.gso.AMPrincipalMapper |
| com.tivoli.pd.as.gso.AliasContainsUserName | true |
| com.tivoli.pd.as.gso.AliasContainsNodeName | false |
| com.tivoli.pd.as.gso.AMLoggingURL | file:///*jlog_props_path* |
| debug | false |

**Scenario 5**
**Auth Data Alias** - BackendEIS/eisUser
**Resource** - BackEndEIS
**User** - eisUser
**Principal Mapping Parameters**

| Name | Value |
| --- | --- |
| delegate | com.tivoli.pdwas.gso.AMPrincipalMapper |
| com.tivoli.pd.as.gso.AliasContainsUserName | false |
| com.tivoli.pd.as.gso.AliasContainsNodeName | true |
| com.tivoli.pd.as.gso.AMLoggingURL | file:///*jlog_props_path* |
| debug | false |

**Scenario 6**
**Auth Data Alias** - nodename/BackendEIS/eisUser
**Resource** - nodename/BackendEIS/eisUser (notice that the Resource is the same as Auth Data Alias).
**User** - Currently authenticated WAS user
**Principal Mapping Parameters**

| Name | Value |
| --- | --- |
| delegate | com.tivoli.pdwas.gso.AMPrincipalMapper |
| com.tivoli.pd.as.gso.AliasContainsUserName | false |
| com.tivoli.pd.as.gso.AliasContainsNodeName | false |
| com.tivoli.pd.as.gso.AMLoggingURL | file:///*jlog_props_path* |
| debug | false |

You now need to create the J2C authentication aliases. The user name and password assigned to these alias entries is irrelevant as Tivoli Access Manager is responsible for providing user names and passwords. However, the user name and password assigned to the J2C authentication aliases need to exist so they can be selected for the J2C connection factory in the console.

To create the J2C authentication aliases, from the WebSphere Application Server administrative console, click **Security** >**Global security**. Under **JAAS Configuration** > **J2C Authentication Data** and click **New** for each entry. Refer to the table above for scenario inputs.

The connection factories for each resource adapter that needs to use the GSO database must be configured to use the Tivoli Access Manager Principal Mapping module. To do this:

a. From the WebSphere Application Server console, select **Applications** > **Enterprise Applications** > *application_name*.
b. Under Related items, click the **Connector Modules** link.
c. Click the **.rar** link.
d. Under Additional properties, click the **Resource Adapter** link.

   **Note:** The resource adapter does not need to be packaged with the application. It can be standalone. For such a scenario the resource adapter is configured from **Resources** > **Resource Adapters**.
e. Under Additional properties, click the **J2C Connection Factories** link.
f. Click **New** and enter the connection factory properties.

**Note:** Configuring custom mapping on connection factory is deprecated in WebSphere Application Server Version 6. To configure the GSO credential mapping, it is recommended that you use the Map Resource References to Resources panel on the administrative console. For more information, refer to “J2EE Connector security” on page 256.

## User registries

Information about users and groups reside in a user registry.

With WebSphere Application Server, a user registry authenticates a user and retrieves information about users and groups to perform security-related functions, including authentication and authorization.

WebSphere Application Server provides several implementations to support multiple types of operating system base user registries. You can use the custom Lightweight Directory Access Protocol (LDAP) feature to support any LDAP server by setting up the correct configuration (user and group filters). However, support is not extended to these custom LDAP servers because many configuration possibilities exist.

In addition to Local operating system (OS) and LDAP registries, WebSphere Application Server also provides a plug-in that supports any user registry by using the custom registry feature (also referred to as a *custom user registry*). The custom registry feature supports any user registry that is not implemented by WebSphere Application Server. You can use any registry used in the product environment by implementing the *UserRegistry interface* interface.

The UserRegistry interface is very helpful in situations where the current user and group information exists in some other format (for example, a database) and cannot move to Local OS or LDAP. In such a case, implement the UserRegistry interface so that WebSphere Application Server can use the existing registry for all of the security-related operations. Using a custom registry is a software implementation effort; it is expected that the implementation does not depend on other WebSphere Application Server resources, for example, data sources, for its operation.

Although WebSphere Application Server supports different types of user registries, only one user registry can be active. This active registry is shared by all of the product server processes. If the product

processes in one node or cell need to communicate with other product processes in other nodes or cells using Lightweight Third Party Authentication (LTPA) (or Integrated Cryptographic Service Facility (ICSF) on the z/OS platform), all of the nodes and cells share the same user registry.

## Configuring user registries

Before configuring the user registry, decide which registry to use. Though different types of registries are supported, all of the processes in WebSphere Application Server can use one active registry. Configuring the correct registry is a prerequisite to assigning users and groups to roles for applications. When no registry is configured, the LocalOS registry is used by default. So, if your choice of registry is not Local OS you need to first configure the registry, which is normally done as part of enabling security, restart the servers, and then assign users and groups to roles for all your applications.

After the applications are assigned users and groups, and you need to change the registries (for example from Lightweight Directory Access Protocol (LDAP) to Custom), delete all the users and groups (including any RunAs role) from the applications, and reassign them after changing the registry through the administrative console or by using wsadmin scripting. The following wsadmin command removes all of the users and groups (including the RunAs role) from any application:

```
$AdminApp deleteUserAndGroupEntries yourAppName
```

where *yourAppName* is the name of the application. Backing up the old application is advised before performing this operation. However, if both of the following conditions are true, you might be able to switch the registries without having to delete the users and groups information:

- All of the user and group names (including the password for the RunAs role users) in all of the applications match in both registries.
- The application bindings file does not contain the `accessIDs`, which are unique for each registry even for the same user or group name.

By default, an application does not contain `accessIDs` in the bindings file (these IDs are generated when the applications start). However, if you migrated an existing application from an earlier release, or if you used the wsadmin script to add accessIDs for the applications to improve performance you have to remove the existing user and group information and add the information after configuring the new registry.

For more information on updating accessIDs, see updateAccessIDs in the AdminApp object for scripted administration article.

Complete one of the following steps to configure your user registry:

- Configure the local operating system user registry.
- Configure the LDAP user registry.
- Configure the custom user registry.

This step is required as part of enabling security in WebSphere Application Server.

1. If you are enabling security, make sure that you complete the remaining steps. Verify that the Active User Registry field in the **Global Security** panel is set to the appropriate registry. As the final step, validate the user ID and the password by clicking **OK** or **Apply** in the Global Security panel. Save, stop and start all WebSphere Application Servers.
2. For any changes in user registry panels to be effective, you must validate the changes by clicking **OK** or **Apply** in the Global Security panel. After validation, save the configuration, stop and start all WebSphere Application Servers (cells, nodes and all the application servers). To avoid inconsistencies between the WebSphere Application Server processes, make sure that any changes to the registry are done when all of the processes are running. If any of the processes are down, force synchronization to make sure that the process can start later.

   If the server or servers start without any problems, the setup is correct.

## Local operating system user registries

With the local operating system, or Local OS, user registry implementation, the WebSphere Application Server authentication mechanism can use the user accounts database of the local operating system.

WebSphere Application Server provides implementations for the Windows local accounts registry and domain registry, as well as implementations for the Linux, Solaris, and AIX user accounts registries. Windows Active Directory is supported through the Lightweight Directory Access Protocol (LDAP) user registry implementation discussed later.

**Note:** For an Active Directory (domain controller), the three group scopes are Domain Local Group, Global Group, and Universal Group. For an Active Directory (Domain Controller), the two group types are Security and Distribution.

When a group is created, the default value is Global (and the default type is Security). With Windows NT domain registry support for Windows 2000 and 20003 domain controllers, WebSphere Application Server only supports Global groups that are the Security type. It is recommended that you use the Active Directory registry support (rather than an NT domain registry) if you use Windows 2000 and 2003 domain controllers because the Active Directory supports all group scopes and types. The Active Directory also supports a nested group that is not support by NT domain registry. The Active Directory is a centralized control registry.

WebSphere Application Server does not have to install the member of the domain because it can be installed on any machine on any platform. Note that the NT domain native call only returns the support group (with no error).

A Local OS user registry is not a centralized user registry like LDAP

Do not use a Local OS user registry in a WebSphere Application Server environment, where application servers are dispersed across more than one machine because each machine has its own user registry. Exceptions include a Windows domain registry, which is a centralized registry and Network Information Services (NIS), which is not supported by WebSphere Application Server.

As mentioned previously, the access IDs taken from the user registry are used during authorization checks. Because these IDs are typically unique identifiers, they vary from machine to machine, even if the exact users and passwords exist on each machine.

Web client certificate authentication is not currently supported when using the local operating system user registry. However, Java client certificate authentication does function with a local operating user registry. Java client certificate authentication maps the first attribute of the certificate domain name to the user ID in the user registry.

Even though Java client certificates function correctly, the following error displays in the `SystemOut.log` file:

```
CWSCJ0337E: The mapCertificate method is not supported
```

The error is intended for Web client certificates; however, it also displays for Java client certificates. Ignore this error for Java client certificates.

**Using Windows operating system registries**

When enabling security on Windows operating systems, if the local operating system (LocalOS) is selected as the registry, consider the following points:

**Required privileges**

The user that is running the WebSphere Application Server process requires enough operating system privilege to call the Windows systems application programming interface (API) for authenticating and obtaining user and group information from the Windows operating system. This user logs into the machine, or if running as a service, is the Log On As user. Depending on the machine and whether the machine is a stand-alone machine or a machine that is part of a domain or is the domain controller, the access requirements vary.

- For a stand-alone machine, the user:
  - Is a member of the administrative group.
  - Has the Act as part of the operating system privilege.
  - Has the Log on as a service privilege, if the server is run as a service.
- For a machine that is a member of a domain, only a domain user can start the server process and:
  - Is a member of the domain administrative groups in the domain controller.
  - Has the Act as part of the operating system privilege in the Domain security policy on the domain controller.
  - Has the Act as part of the operating system privilege in the Local security policy on the local machine.
  - Has the Log on as a service privilege on the local machine, if the server is run as a service.

    The user is a domain user and not a local user, which implies that when a machine is part of a domain, only a domain user can start the server.
- For a domain controller machine, the user:
  - Is a member of the domain administrative groups in the domain controller.
  - Has the Act as part of the operating system privilege in the Domain security policy on the domain controller.
  - Has the Log on as a service privilege on the domain controller, if the server is run as a service.

To give a user the Act as part of the operating system or Log on as a service on Windows 2000 systems:

1. Click **Start** > **Settings** > **Control Panel** > **Administrative Tools** > **Local Security Policy** > **Local Policies** > **User Rights Assignments** > **Act as part of the operating system** (or **Log on as a service**) .

   **Note:** If the machine is a stand-alone machine and not a member of a domain, you must add a machineName*userID*, where the *userID* is the owner of the process, such as WebSphere Application Server. If you run WebSphere Application Server as a service, you can log on with `localsystem` as the service.

   If the machine is a member of a domain, add domainName*userID*, where the *userID* is the owner of process (such as WebSphere Application Server). Start WebSphere Application Server as a service with login ID domainName\userID. If WebSphere Application Server is already in service, go to the service and right-click **IBM WebSphere Application Server > properties >Logon to change the logon ID and password** to restart WebSphere Application Server.
2. Add the user name by clicking **Add**.
3. Restart the machine.

   **Windows 2000 domain controller users:** For a Windows 2000 domain controller, replace **Local Security Policy** with **Domain Security Policy** in the previous step.

   **Note:** In all of the previous configurations, the server can be run as a service using `LocalSystem` for the Log On As entry. The `LocalSystem` entry has the required privileges and there is no need to give special privileges to any user. However, because the `LocalSystem` entry has special privileges, make sure that it is appropriate to use in your environment.

If the user running the server does not have the required privilege, you might see one of the following exception messages in the log files:

- `A required privilege is not held by the client.`

- `Access is denied.`

### Domain and local registries

When WebSphere Application Server is started, the security run-time initialization process dynamically attempts to determine if the local machine is a member of a Windows domain. If the machine is part of a domain then by default both the local registry users or groups and the domain registry users or groups can be used for authentication and authorization purposes with the domain registry taking precedence. The list of users and groups that is presented during the security role mapping includes users and groups from both the local user registry and the domain user registry. The users and groups can be distinguished by the associated host names.

*WebSphere Application Server does not support trusted domains.*

If the machine is not a member of a Windows system domain, the user registry local to that machine is used.

### Using both the domain registry and the local registry

When the machine that hosts the WebSphere Application Server process is a member of a domain, both the local and the domain registries are used by default. The following section describes more on this topic and recommends some best practices to avoid unfavorable consequences.

- **Best practices**

  In general, if the local and the domain registries do not contain common users or groups, it is simpler to administer and it eliminates unfavorable side effects. If possible, give users and groups access to unique security roles, including the server ID and administrative roles). In this situation, select the users and groups from either the local registry or the domain registry to map to the roles.

  In cases where the same users or groups exist in both the local registry and the domain registry, it is recommended that at least the server ID and the users and groups that are mapped to the administrative roles be unique in the registries and exist only in the domain.

  If a common set of users exists, set a different password to make sure that the appropriate user is authenticated.

- **How it works**

  When a machine is part of a domain, the domain user registry takes precedence over the local user registry. For example, when a user logs into the system, the domain registry tries to authenticate the user first. If the authentication fails the local registry is used. When a user or a group is mapped to a role, the user and group information is first obtained from the domain registry. In case of failure, the local registry is tried. However, when a fully qualified user or a group name (one with an attached domain or host name) is mapped to a role, then only that registry is used to get the information. Use the administrative console or scripts to get the fully qualified user and group names, which is the recommended way to map users and groups to roles.

  **Note:** A user **Bob** on one machine (the local registry, for example) is not the same as the user **Bob** on another machine (say the domain registry) because the `uniqueID` of **Bob** (the security identifier [SID], in this case) is different in different registries.

- **Examples**

  The machine `MyMachine` is part of the domain `MyDomain`. `MyMachine` contains the following users and groups:

  - MyMachine\user2
  - MyMachine\user3
  - MyMachine\group2

  MyDomain contains the following users and groups:

  - MyDomain\user1
  - MyDomain\user2

– MyDomain\group1
– MyDomain\group2

Here are some scenarios that assume the previous set of users and groups.

1. When user2 logs into the system, the domain registry is used for authentication. If the authentication fails (the password is different) the local registry is used.
2. If the user MyMachine\user2 is mapped to a role, only the user2 in MyMachine has access. So if the user2 password is the same on both the local and the domain registries, user2 cannot access the resource, because user2 is always authenticated using the domain registry. Hence, if both registries have common users, it is recommended that the password be different.
3. If the group2 is mapped to a role, only the users who are members of the MyDomain\group2 can access the resource because group2 information is first obtained from the domain registry.
4. If the group MyMachine\group2 is mapped to a role, only the users who are members of the MyMachine\group2 can access the resource. A specific group is mapped to the role (MyMachine\group2 instead of just group2).
5. Use either user3 or MyMachine\user3 to map to a role, because user3 is unique; it exists in one registry only.

Authorizing with the domain user registry first can cause problems if a user exists in both the domain and local user registries with the same password. Role-based authorization can fail in this situation because the user is first authenticated within the domain user registry. This authentication produces a unique domain security ID that is used in WebSphere Application Server during the authorization check. However, the local user registry is used for role assignment. The domain security ID does not match the unique security ID that is associated with the role. To avoid this problem, map security roles to domain users instead of local users.

**Using either the local or the domain registry**. If you want to access users and groups from either the local registry or the domain registry, instead of both, set the com.ibm.websphere.registry.UseRegistry property. This property can be set to either localor domain. When this property is set to local(case insensitive) only the local registry is used. When this property is set to domain, (case insensitive) only the domain registry is used. Set this property by clicking **Custom Properties** in the **Security** > **User Registries** > **Local OS** panel in the administrative console or by using scripts. When the property is set, the privilege requirement for the user who is running the product process does not change. For example, if this property is set to local, the user that is running the process requires the same privilege, as if the property was not set.

**Using UNIX system registries**

When using UNIX system registries, the process ID that runs the WebSphere Application Server process needs the root authority to call the local operating system APIs for authentication and for obtaining user or group information.

**Note:** In UNIX systems, only the local machine registry is used. Network Information Service (NIS) (Yellow Pages) is not supported.

**Using Linux and Solaris system registries**

For WebSphere Application Server Local OS security registry to work on the Linux and Solaris platforms, a shadow password file must exist. The shadow password file is named shadow and is located in the /etc directory. If the shadow password file does not exist, an error occurs after enabling global security and configuring the user registry as Local OS.

To create the shadow file, run the **pwconv** command (with no parameters). This command creates an /etc/shadow file from the /etc/passwd file. After creating the shadow file, you can enable local operating system security successfully.

**Remote registries**

By default, the registry is local to all of the product processes. The performance is higher, (no need for remote calls) and the registry also increases availability. Any process failing does not effect other processes.

When using LocalOS as the registry, every product process must run with privilege access (`root` in UNIX, `Act as part of operating system` in Windows systems).

If this process is not practical in some situations, you can use a remote registry from the node (or in very rare situations from the cell). Using a remote registry affects performance and creates a single point of failure. **Use remote registries only in rare situations.**

The node and the cell processes are meant for manipulating configuration information and for hosting the registry for all the application servers that create traffic and cause problems.

Using a node agent (instead of the cell) to host the remote registry is preferable because the cell process is not designed to be highly available. Also, using a node to host the remote registry indicates that only the application servers in that node are using it. Because the node agent does not contain any application code, giving it the access required privilege is not a concern.

You can set up a remote registry by setting the WAS_UseRemoteRegistry property in the Global Security panel using the **Custom Properties** link at the bottom of the administrative console panel. Use either the`Cell` or the `Node`(case insensitive) value. If the value is `Cell`, the cell registry is used by all of the product processes including the node agent and all of the application servers. If the cell process is down for any reason, restart all of the processes after the cell is restarted. If the node agent registry is used for the remote registry, set the WAS_UseRemoteRegistry value to `node`. In this case, all the application server processes use the node agent registry. In this case, if the node agent fails and does not start automatically, you might need to restart all the application servers after the node agent is started.

## Configuring local operating system user registries

For security purposes, the WebSphere Application Server provides and supports the implementation for Windows operating system registries, AIX, Solaris and multiple versions of Linux operating systems. The respective operating system APIs are called by the product processes (servers) for authenticating a user and other security-related tasks (for example, getting user or group information). Access to these APIs are restricted to users who have special privileges. These privileges depend on the operating system and are described below.

Before configuring the LocalOS registry you need to know the user name (ID) and password to use here. This user can be any valid user in the registry. This user is referred to as either a product security server ID, a server ID or a server user ID in the documentation. Having a server ID means that a user has special privileges when calling protected internal methods. Normally, this ID and password are used to log into the administrative console after security is turned on. You can use other users to log in if those users are part of the administrative roles. When security is enabled in the product, this server ID and password are authenticated with the registry during product startup. If authentication fails, the server does not come up. So it is important to choose an ID and password that do not expire or change often. If the product server user ID or password need to change in the registry, ensure that the changes are performed when all the product servers are up and running. After the changes are completed in the registry, use the following steps to change the ID and the password information. Save, stop, and restart all the servers so that the product can use the new ID or password. If any problem arises after starting the product because of authentication problems (that cannot be fixed), disable security before the server can start up. To avoid this step, make sure that the changes are validated in the Global Security panel. After the server is up, change the ID and password information and enable security.

When using the Windows operating system, consider the following issues:

- The server ID needs to be different from the Windows machine name where the product is installed. For example, if the Windows machine name is *vicky* and the security server ID is vicky, the Windows system fails when getting the information (group information, for example) for user *vicky*.
- WebSphere Application Server dynamically determines whether the machine is a member of a Windows system domain.
- WebSphere Application Server does not support Windows trusted domains.
- If a machine is a member of a Windows domain, both the domain user registry and the local user registry of the machine participate in authentication and security role mapping.
- The domain user registry takes precedence over the local user registry of the machine and can have undesirable implications if users with the same password exist in both user registries.
- The user that the product processes run under requires the `Administrative` and `Act as part of the operating system` privileges to call the Windows operating system APIs that authenticate or collect user and group information. The process needs special authority, which is given by these privileges. The user in this example might not be the same as the security server ID (the requirement for which is a valid user in the registry). This user logs into the machine (if using the command line to start the product process) or the Log On User setting in the services panel if the product processes have started using the services. If the machine is also part of a domain, this user is a part of the Domain Admin group in the domain to call the operating system APIs in the domain in addition to having the Act as part of operating system privilege in the local machine.

When using the UNIX operating systems (AIX and Solaris) and Linux, consider the following points:
- The user that the product processes run under requires the `root` privilege. This privilege is needed to call the UNIX operating system APIs to authenticate or to collect user and group information. The process needs special authority, which is given by the `root` privilege. This user might not be the same as the security server ID (the requirement is that it should be a valid user in the registry). This user logs into the machine and is running the product processes.
- On a UNIX operating system, the user that enables global security must have the `root` privilege if you use the local OS user registry. Otherwise, a failed validation error is displayed.
- When using the Linux operating system, you might need to have the password shadow file in your system.

The following steps are needed to perform this task initially when setting up security for the first time.
1. Click **Security > Global security**.
2. Under User registries, click **Local OS**,
3. Enter a valid user name in the Server user ID field.
4. Enter the user password in the Server user password field.
5. Click **OK**. Validation of the user and password does not happen in this panel. Validation is only done when you click **OK** or **Apply** in the Global Security panel. If you are enabling security for the first time, complete the other steps and go to the Global Security panel. Make sure that LocalOS is the Active User Registry. If security was already enabled and you had changed either the user or the password information in this panel, make sure to go to the Global Security panel and click **OK** or **Apply** to validate your changes. If your changes are not validated, the server might not come up.

The Local OS user registry has been configured.
1. If you are enabling security, complete the remaining steps. As the final step, ensure that you validate the user and password by clicking **OK** or **Apply** in the Global Security panel. Save, stop, and start all the product servers.
2. For any changes in this panel to be effective, you need to save, stop and start all the product servers (deployment managers, nodes and Application Servers).
3. If the server comes up without any problems the setup is correct.

## Local operating system user registry settings

Use this page to configure local operating system user registry settings.

To view this administrative console page, click **Security** > **Global Security**. Under User registries, click **Local OS**.

***Server user ID:***

Specifies a valid user ID in the local OS registry.

This ID is the security server ID, which is only used for WebSphere Application Server security and is not associated with the system process that runs the server. The server calls the Local OS registry to authenticate and obtain privilege information about users by calling the native APIs in that particular registry. Access to native APIs is normally restricted to users having special privileges (for example, **root** in UNIX systems and **Act as part of operating system** in Windows systems). To use security in the application server, the process ID (not the security server ID) on which WebSphere Application Server runs requires enough privileges to call the system APIs. The special privilege means that the process running the WebSphere Application Server needs to be part of the **Administrators** group and have the **Act as part of operating system** privilege on Windows systems, and be **root**, or have root authority on UNIX systems.

**Note:** Windows If you are configuring Local OS security on Windows and you encounter the `A required privilege is not helped by the client` error message, you must follow the procedure documented to give the user those privileges. To set the privilege, click **Start > Settings > Control Panel > Administrative Tools > Local Security Policy > Local Policies > User Rights Assignments > Act as part of the operating system**.

When using a Windows system registry, this ID cannot match the name of the Windows machine. Windows systems treat the machine name bob as having an account similar to user bob.

| | |
|---|---|
| **Data type:** | String |
| **Units:** | Alphanumeric characters |

***Server user password:***

Specifies a valid user password that corresponds to a valid user ID in the local OS registry.

| | |
|---|---|
| **Data type** | String |

## Lightweight Directory Access Protocol

Lightweight Directory Access Protocol (LDAP) is a user registry in which authentication is performed using an LDAP binding.

WebSphere Application Server security provides and supports implementation of most major LDAP directory servers, which can act as the repository for user and group information. These LDAP servers are called by the product processes (servers) for authenticating a user and other security-related tasks (for example, getting user or group information). This support is provided by using different user and group filters to obtain the user and group information. These filters have default values that you can modify to fit your needs. The custom LDAP feature enables you to use any other LDAP server (which is not in the product supported list of LDAP servers) for its user registry by using the appropriate filters.

To use LDAP as the user registry, you need to know a valid user name (ID), the user password, the server host and port, the base distinguished name (DN) and if necessary the bind DN and the bind password. You can choose any valid user in the registry that is searchable. In some LDAP servers, the administrative users are not searchable and cannot be used (for example, `cn=root` in SecureWay). This user is referred to as WebSphere Application Server security server ID, server ID, or server user ID in the documentation. Being a server ID means a user has special privileges when calling some protected internal methods.

Normally, this ID and password are used to log into the administrative console after security is turned on. You can use other users to log in if those users are part of the administrative roles.

When security is enabled in the product, this server ID and password are authenticated with the registry during the product startup. If authentication fails, the server does not start. Choosing an ID and password that do not expire or change often is important. If the product server user ID or password need to change in the registry, make sure that the changes are performed when all the product servers are up and running.

When the changes are done in the registry, use the steps described in Configuring LDAP user registries. Change the ID, password, and other configuration information, save, stop, and restart all the servers so that the new ID or password is used by the product. If any problems occur starting the product when security is enabled, disable security before the server can start up (to avoid these problems, make sure that any changes in this panel are validated in the Global Security panel). When the server is up, you can change the ID, password and other configuration information and then enable security.

## Configuring Lightweight Directory Access Protocol user registries

Review the article on Lightweight Directory Access Protocol (LDAP) before beginning this task.

1. In the administrative console, click **Security > Global security**.
2. Under User registries, click **LDAP**.
3. Enter a valid user name in the Server user ID field. You can either enter the complete distinguished name (DN) of the user or the short name of the user as defined by the User Filter in the Advanced LDAP settings panel. For example, enter the user ID for Netscape.
4. Enter the password of the user in the Server user password field.
5. Select the type of LDAP server that is used from the Type list. The type of LDAP server determines the default filters that are used by the WebSphere Application Server. These default filters change the **Type** field to **Custom**, which indicates that custom filters are used. This action occurs after you click **OK** or **Apply** in the Advanced LDAP settings panel. Choose the **Custom** type from the list and modify the user and group filters to use other LDAP servers, if required. If either the IBM Directory Server or the iPlanet Directory Server is selected, also select the Ignore Case field.
6. Enter the fully qualified host name of the LDAP server in the Host field.
7. Enter the LDAP server port number in the Port field. The host name and the port number represent the realm for this LDAP server in the WebSphere Application Server cell. So, if servers in different cells are communicating with each other using Lightweight Third Party Authentication (LTPA) tokens, these realms must match exactly in all the cells.
8. Enter the Base distinguished name (DN) in the Base distinguished name field. The Base DN indicates the starting point for searches in this LDAP directory server. For example, for a user with a DN of `cn=John Doe, ou=Rochester, o=IBM, c=US`, specify the Base DN as any of the following options (assuming a suffix of `c=us`): `ou=Rochester, o=IBM, c=us or o=IBM c=us` or `c=us`. This field can be case sensitive. Match the case in your directory server. This field is required for all LDAP directories except the Domino Directory. The Base DN field is optional for the Domino server.
9. Enter the Bind DN name in the Bind distinguished name field, if necessary. The Bind DN is required if anonymous binds are not possible on the LDAP server to obtain user and group information. If the LDAP server is set up to use anonymous binds, leave this field blank.
10. Enter the password corresponding to the Bind DN in the Bind password field, if necessary.
11. Modify the Search time out value if required. This timeout value is the maximum amount of time that the LDAP server waits to send a response to the product client before aborting the request. The default is 120 seconds.
12. Deselect the **Reuse connection** option only if you use routers to send requests to multiple LDAP servers, and if the routers do not support affinity. Leave this field enabled for all other situations.

13. Select the **Ignore case for authorization** option, if required. When this flag is enabled, the authorization check is case insensitive. Normally, an authorization check involves checking the complete DN of a user, which is unique in the LDAP server and is case sensitive. However, when using either the IBM Directory Server or the iPlanet Directory Server LDAP servers, this flag needs enabling because the group information obtained from the LDAP servers is not consistent in case. This inconsistency only effects the authorization check.
14. Enable Secure Sockets Layer (SSL) if the communication to the LDAP server is through SSL. For more information on setting up LDAP for SSL, refer to Configuring SSL for LDAP clients.
15. Select the **SSL enabled** option if you want to use secure sockets layer communications with the LDAP server. If you select the **SSL enabled** option, select the appropriate SSL alias configuration from the list in the SSL configuration field.
16. Click **OK**. The validation of the user, password, and the setup do not take place in this panel. Validation is only done when you click **OK** or **Apply** in the **Global Security** panel. If you are enabling security for the first time, complete the remaining steps and go to the **Global Security** panel. Select **LDAP** as the active user registry. If security is already enabled, but information on this panel changes, go to the **Global Security** panel and click **OK** or **Apply** to validate your changes. If your changes are not validated, the server might not come up.

Sets the LDAP registry configuration. This step is required to set up the LDAP registry. This step is required as part of enabling security in the WebSphere Application Server.
1. If you are enabling security, complete the remaining steps. As the final step, validate this setup by clicking **OK** or **Apply** in the Global Security panel.
2. Save, stop, and restart all the product servers (deployment managers, nodes and Application Servers) for changes in this panel to take effect.
3. If the server comes up without any problems the setup is correct.

## Lightweight Directory Access Protocol settings

Use this page to configure Lightweight Directory Access Protocol (LDAP) settings when users and groups reside in an external LDAP directory.

To view this administrative console page, click **Security > Global security**. Under User registries, click **LDAP**.

When security is enabled and any of these properties change, go to the Global security panel and click **Apply** to validate the changes.

***Server user ID:***

Specifies the user ID that is used to run the WebSphere Application Server for security purposes.

Although this ID is not the LDAP administrator user ID, specify a valid entry in the LDAP directory located under the Base Distinguished Name.

***Server user password:***

Specifies the password corresponding to the security server ID.

***Type:***

Specifies the type of LDAP server to which you connect.

IBM SecureWay Directory Server is not supported.

For a list of supported LDAP servers, see ″Supported directory services.″ in the documentation.

***Host:***

Specifies the host ID (IP address or domain name service (DNS) name) of the LDAP server.

***Port:***

Specifies the host port of the LDAP server.

If multiple WebSphere Application Servers are installed and configured to run in the same single signon domain, or if the WebSphere Application Server interoperates with a previous version of the WebSphere Application Server, then it is important that the port number match all configurations. For example, if the LDAP port is explicitly specified as `389` in a Version 4.0.x configuration, and a WebSphere Application Server at Version 5 is going to interoperate with the Version 4.0.x server, then verify that port `389` is specified explicitly for the Version 5 server.

| | |
|---|---|
| **Default:** | 389 |

***Base distinguished name (DN):***

Specifies the base distinguished name of the directory service, indicating the starting point for LDAP searches of the directory service.

For example, for a user with a distinguished name (DN) of `cn=John Doe, ou=Rochester, o=IBM, c=US`, you can specify the base DN as (assuming a suffix of `c=us`): `ou=Rochester, o=IBM, c=us`. For authorization purposes, this field is case sensitive. This specification implies that if a token is received (for example, from another cell or Domino) the base DN in the server must match the base DN from the other cell or Domino server exactly. If case sensitivity is not a consideration for authorization, enable the **Ignore case** field. This field is required for all Lightweight Directory Access Protocol (LDAP) directories except for the Domino Directory, where this field is optional.

If you need to interoperate between WebSphere Application Server Version 5 and a Version 5.0.1 or later server, you must enter a normalized base distinguished name. A normalized base distinguished name does not contain spaces before or after commas and equal symbols. An example of a non-normalized base distinguished name is `o = ibm, c = us` or `o=ibm, c=us`. An example of a normalized base distinguished name is `o=ibm,c=us`. In WebSphere Application Server, Version 5.0.1 or later, the normalization occurs automatically during run time

***Bind distinguished name (DN):***

Specifies the distinguished name for the application server to use when binding to the directory service.

If no name is specified, the application server binds anonymously. See the Base Distinguished Name field description for examples of distinguished names.

***Bind password:***

Specifies the password for the application server to use when binding to the directory service.

***Search timeout:***

Specifies the timeout value in seconds for an Lightweight Directory Access Protocol (LDAP) server to respond before aborting a request.

| | |
|---|---|
| **Default:** | 120 |

***Reuse connection:***

Specifies whether the server reuses the Lightweight Directory Access Protocol (LDAP) connection. Clear this option only in rare situations where a router is used to spray requests to multiple LDAP servers and when the router does not support affinity.

| | |
|---|---|
| **Default:** | Enabled |
| **Range:** | Enabled or Disabled |

***Ignore case for authorization:***

Specifies that a case insensitive authorization check is performed when using the default authorization.

This field is required when IBM Directory Server is selected as the LDAP directory server.

This field is required when Sun ONE Directory Server is selected as the LDAP directory server. For more information, see ″Using specific directory servers as the LDAP server″ in the documentation.

Otherwise, this field is optional and can be enabled when a case-sensitive authorization check is required. For example, use this field when the certificates and the certificate contents do not match the case used for the entry in the LDAP server. You can enable the **Ignore case** field when using single signon (SSO) between WebSphere Application Server and Lotus Domino.

| | |
|---|---|
| **Default:** | Enabled |
| **Range:** | Enabled or Disabled |

***SSL enabled:***

Specifies whether secure socket communication is enabled to the Lightweight Directory Access Protocol (LDAP) server. When enabled, the LDAP Secure Sockets Layer (SSL) settings are used, if specified.

***SSL configuration:***

Specifies the Secure Sockets Layer configuration to use for the Lightweight Directory Access Protocol (LDAP) connection. This configuration is used only when SSL is enabled for LDAP.

| | |
|---|---|
| **Default:** | DefaultSSLSettings |

## Advanced Lightweight Directory Access Protocol user registry settings

Use this page to configure the advanced Lightweight Directory Access Protocol (LDAP) user registry settings when users and groups reside in an external LDAP directory.

To view this administrative page, complete the following steps:

1. Click **Security** > **Global security**.
2. Under User registries, click **LDAP**.
3. Under Additional properties, click **Advanced Lightweight Directory Access Protocol (LDAP) user registry settings**.

Default values for all the user and group related filters are already completed in the appropriate fields. You can change these values depending on your requirements. These default values are based on the type of LDAP server selected in the **LDAP settings** panel. If this type changes (for example from Netscape to Secureway) the default filters automatically change. When the default filter values change, the LDAP server type changes to `Custom` to indicate that custom filters are used. When security is enabled and any of these properties change, go to the **Global security** panel and click **Apply** or **OK** to validate the changes.

*User filter:*

Specifies the LDAP user filter that searches the user registry for users.

This option is typically used for security role to user assignments. It specifies the property by which to look up users in the directory service. For example, to look up users based on their user IDs, specify (%(uid=%v)(objectclass=inetOrgPerson)). For more information about this syntax, see the LDAP directory service documentation.

**Data type:** String

*Group filter:*

Specifies the LDAP group filter that searches the user registry for groups

This option is typically used for security role to group assignments. It specifies the property by which to look up groups in the directory service. For more information about this syntax, see the LDAP directory service documentation.

**Data type:** String

*User ID map:*

Specifies the LDAP filter that maps the short name of a user to an LDAP entry.

Specifies the piece of information that represents users when users appear. For example, to display entries of the type `object class = inetOrgPerson` by their IDs, specify `inetOrgPerson:uid`. This field takes multiple objectclass:property pairs delimited by a semicolon (;).

**Data type:** String

*Group ID Map:*

Specifies the LDAP filter that maps the short name of a group to an LDAP entry.

Specifies the piece of information that represents groups when groups appear. For example, to display groups by their names, specify `*:cn`. The asterisk (*) is a wildcard character that searches on any object class in this case. This field takes multiple `objectclass:property` pairs delimited by a semicolon (;).

**Data type:** String

*Group member ID map:*

Specifies the LDAP filter that identifies user to group relationships.

For directory types SecureWay, Netscape, and Domino, this field takes multiple objectclass:property pairs, delimited by a semicolon (;). In an objectclass:property pair, the objectclass value is the same objectclass that is defined in the Group Filter, and the property is the member attribute. If the objectclass value does not match the objectclass in Group Filter, authorization might fail if groups are mapped to security roles. For more information about this syntax, see your LDAP directory service documentation.

For IBM Directory Server, iPlanet Directory Server and Active Directory, this field takes multiple (`group attribute:member attribute`) pairs delimited by a semicolon (;). They are used to find the group memberships of a user by enumerating all the group attributes possessed by a given user. For example,

attribute pair (memberof:member) is used by Active Directory, and (ibm-allGroup:member) is used by IBM Directory Server . This field also specifies which property of an objectclass stores the list of members belonging to the group represented by the objectclass. For supported LDAP directory servers, see ″Supported directory services″.

**Data type:** String

***Perform a nested group search:***

Specifies a recursive nested group search.

Select this option if the Lightweight Directory Access Protocol (LDAP) server does not support recursive server-side group member searches (and if recursive group member search is required). It is not recommended that you select this option to locate recursive group memberships for LDAP servers. WebSphere security leverages the LDAP server's recursive search functionality to search a user's group memberships, including recursive group memberships. For example:

- IBM Directory Server is pre-configured by WebSphere Application Server security to recursively calculate a user's group memberships using the `ibm-allGroup` attribute
- SunONE directory server is pre-configured to calculate nested group memberships using the `nsRole` attribute

**Data type:** String

***Certificate map mode:***

Specifies whether to map X.509 certificates into an LDAP directory by EXACT_DN or CERTIFICATE_FILTER. Specify CERTIFICATE_FILTER to use the specified certificate filter for the mapping.

**Data type:** String

***Certificate filter:***

Specifies the filter certificate mapping property for the LDAP filter. The filter is used to map attributes in the client certificate to entries in the LDAP registry.

If more than one LDAP entry matches the filter specification at run time, then authentication fails because it results in an ambiguous match. The syntax or structure of this filter is: `LDAP attribute=${Client certificate attribute}` (for example, `uid=${SubjectCN}`). The left side of the filter specification is an LDAP attribute that depends on the schema that your LDAP server is configured to use. The right side of the filter specification is one of the public attributes in your client certificate. The right side must begin with a dollar sign ($) and open bracket ({) and end with a close bracket (}). You can use the following certificate attribute values on the right side of the filter specification. The case of the strings is important:

- `${UniqueKey}`
- `${PublicKey}`
- `${PublicKey}`
- `${Issuer}`
- `${NotAfter}`
- `${NotBefore}`
- `${SerialNumber}`
- `${SigAlgName}`
- `${SigAlgOID}`
- `${SigAlgParams}`
- `${SubjectCN}`

- ${Version}

**Data type:** String

## Configuring Lightweight Directory Access Protocol search filters

WebSphere Application Server uses Lightweight Directory Access Protocol (LDAP) filters to search and obtain information about users and groups from an LDAP directory server. A default set of filters is provided for each LDAP server that the product supports. You can modify these filters to fit your LDAP configuration. After the filters are modified (and you click **OK** or **Apply**) the directory type in the LDAP Registry panel changes to `custom`, which indicates that custom filters are used. Also, you can develop filters to support any additional type of LDAP server. The effort to support additional LDAP directories is optional and other LDAP directory types are not supported.

1. In the administrative console, click **Security > Global security.**.
2. Under User registries, click **LDAP**.
3. Under Additional properties, click **Advanced Lightweight Directory Access Protocol (LDAP) user registry settings**.
4. Modify the User filter, if necessary. The user filter is used for searching the registry for users and is typically used for the security role to user assignment. Also, the filter is used to authenticate a user using the attribute that is specified in the filter. The filter specifies the property that is used to look up users in the directory service.

   In the following example, the property that is assigned to `%v`, which is the short name of the user, must be a unique key. Two LDAP entries with the same object class cannot have the same short name. To look up users based on their user IDs (uid) and to use the inetOrgPerson object class, specify the following syntax:

   ```
   (&(uid=%v)(objectclass=inetOrgPerson)
   ```

   For more information about this syntax, see the LDAP directory service documentation.
5. Modify the Group filter, if necessary. The group filter is used in searching the registry for groups and is typically used for the security role to group assignment. Also, the filter is used to specify the property by which to look up groups in the directory service.

   In the following example, the property that is assigned to `%v`, which is the short name of the group, must be a unique key. Two LDAP entries with the same object class cannot have the same short name. To look up groups based on their common names (CN) and to use either the groupOfNames or the groupOfUniqueNames object class, specify the following syntax:

   ```
   (&(cn=%v)(|(objectclass=groupOfNames)(objectclass=groupOfUniqueNames)))
   ```

   For more information about this syntax, see the LDAP directory service documentation.
6. Modify the User ID map, if necessary. This filter maps the short name of a user to an LDAP entry. It specifies the piece of information that represents users when these users are displayed with their short names. For example, to display entries of the type object class = inetOrgPerson by their IDs, specify `inetOrgPerson:uid`. This field takes multiple objectclass:property pairs delimited by a semicolon (;). To provide a consistent value for methods like the getCallerPrincipal( ) method and the getUserPrincipal() method, the short name that is obtained by using this filter is used. For example, the user `CN=Bob Smith`, `ou=austin.ibm.com`, `o=IBM`, `c=US` can log in using any attributes that are defined (for example, e-mail address, social security number, and so on) but when these methods are called, the user ID `bob` is returned no matter how the user logs in.
7. Modify the Group ID map filter, if necessary. This filter maps the short name of a group to an LDAP entry. It specifies the piece of information that represents groups when groups display. For example, to display groups by their names, specify `*:cn`. The asterisk (*) is a wildcard character that searches on any object class in this case. This field takes multiple objectclass:property pairs delimited by a semicolon (;).

8. Modify the Group Member ID Map filter, if necessary. This filter identifies user to group memberships. For SecureWay, Netscape, and Domino directory types, this field is used to query all the groups that match the specified object classes to see if the user is contained in the specified attribute. For example, to get all the users belonging to groups with the groupOfNames object class and the users that are contained in the member attributes, specify `groupOfNames:member`. This syntax, which is a property of an objectclass, stores the list of members that belong to the group that is represented by the objectclass. This field takes multiple objectclass:property pairs that are delimited by a semicolon (;). For more information about this syntax, see the LDAP directory service documentation.

   For the IBM Directory Server, iPlanet Directory Server, and Active Directory, this field is used to query all users in a group by using the information that is stored in the user object (instead of querying all the groups individually to find if the user exists in that group). For example, the memberof:member filter (for Active Directory) is used to get the memberof attribute of the user object to obtain all the groups to which the user belongs. The member attribute is used to get all the users in a group that use the group object. Using the user object to obtain the group information improves performance.
9. Select the **Perform a nested group search** option if your LDAP server does not support recursive server-side searches.
10. Modify the Certificate map mode, if necessary. You can use the X.590 certificates for user authentication when LDAP is selected as the user registry. This field is used to indicate whether to map the X.509 certificates into an LDAP directory user by **EXACT_DN** or **CERTIFICATE_FILTER**. If **EXACT_DN** is selected, the DN in the certificate must exactly match the user entry in the LDAP server (including case and spaces).

    Select the Ignore case for authorization field on the LDAP settings to make the authorization case insensitive. To access the LDAP setting panel, complete the following steps:
    a. Click **Security > Global security.**.
    b. Under User registries, click **LDAP**.
11. If you select **CERTIFICATE_FILTER**, specify the LDAP filter for mapping attributes in the client certificate to entries in LDAP. If more than one LDAP entry matches the filter specification at run time, authentication fails because an ambiguous match results. The syntax or structure of this filter is: `LDAP attribute=${Client certificate attribute}` (for example, `uid=${SubjectCN}`).

    The left side of the filter specification is an LDAP attribute that depends on the schema that your LDAP server is configured to use. The right side of the filter specification is one of the public attributes in your client certificate. Note that the right side must begin with a dollar sign ($), open bracket ({), and end with a close bracket (}). Use the following certificate attribute values on the right side of the filter specification. The case of the strings is important.
    - ${UniqueKey}
    - ${PublicKey}
    - ${Issuer}
    - ${NotAfter}
    - ${NotBefore}
    - ${SerialNumber}
    - ${SigAlgName}
    - ${SigAlgOID}
    - ${SigAlgParams}
    - ${SubjectDN}
    - ${Version}

    To enable this field, select **CERTIFICATE_FILTER** for the certificate mapping.
12. Click **Apply**.

    When any LDAP user or group filter is modified in the Advanced LDAP Settings panel click **Apply**. Clicking **OK** navigates you to the LDAP User Registry panel, which contains the previous LDAP directory type, rather than the custom LDAP directory type. Clicking **OK** or **Apply** in the LDAP User Registry panel saves the back-level LDAP directory type and the default filters of that directory. This action overwrites any changes to the filters that you made. To avoid overwriting changes, you can take either of the following actions:

- Click **Apply** in the Advanced Lightweight Directory Access Protocol (LDAP) user registry settings panel. To proceed to another panel, use the left navigation. Using the navigation to access the LDAP User Registry panel changes the directory type to `Custom`.
- Choose **Custom** type from the LDAP User Registry panel. Click **Apply** and then change the filters by clicking the Advanced Lightweight Directory Access Protocol (LDAP) user registry settings panel. After you complete your changes, click **Apply** or **OK**.

The validation of the changes (if any) does not take place in this panel. Validation is done when you click **OK** or **Apply** in the Global Security panel. If you are in the process of enabling security for the first time, complete the remaining steps and go to the Global Security panel. Select **LDAP** as the Active User Registry. If security is already enabled and any information on this panel changes, go to the Global Security panel and click **OK** or **Apply** to validate your changes. If your changes are not validated, the server might not start.

Setting the LDAP search filters. This step is required to modify existing user and group filters for a particular LDAP directory type. It is also used to set up certificate filters to map certificates to entries in the LDAP server.
1. If you are enabling security, complete the remaining steps. As the final step make sure that you validate this setup by clicking **OK** or **Apply** in the Global Security panel.
2. Save, stop, and start all the product servers (cell, nodes and all the application servers) for any changes in this panel to become effective.
3. After the server starts, go through all the security-related tasks (getting users, getting groups, and so on) to verify that the changes to the filters function.

## Using specific directory servers as the LDAP server

For *Using MS Active Directory server as the LDAP server* below, note that to use Microsoft Active Directory as the LDAP server for authentication with WebSphere Application Server you must take specific steps. By default, Microsoft Active Directory does not permit anonymous LDAP queries. To create LDAP queries or to browse the directory, an LDAP client must bind to the LDAP server using the distinguished name (DN) of an account that belongs to the administrator group of the Windows system. A group membership search in the Active Directory is done by enumerating the memberof attribute possessed by a given user entry, rather than browsing through the member list in each group. If you change this default behavior to browse each group, you can change the **Group Member ID Map** field from `memberof:member` to `group:member`.

### Using IBM Tivoli Directory Server as the LDAP server

To use IBM Tivoli Directory Server (formerly IBM Directory Server), choose **IBM Tivoli Directory Server** as the directory type.

For supported directory servers, refer to the article, Supported directory services. The difference between these two types is group membership lookup. It is recommended that you choose the IBM Tivoli Directory Server for optimum performance during run time. In the IBM Tivoli Directory Server, the group membership is an operational attribute. With this attribute, a group membership lookup is done by enumerating the ibm-allGroups attribute for the entry, All group memberships, including the static groups, dynamic groups, and nested groups, can be returned with the `ibm-allGroups` attribute. WebSphere Application Server supports dynamic groups, nested groups, and static groups in IBM Tivoli Directory Server using the `ibm-allGroups` attribute. To utilize this attribute in a security authorization application, use a case-insensitive match so that attribute values returned by the ibm-allGroups attribute are all in uppercase.

**Important:** It is recommended that you do not install IBM Tivoli Directory Server Version 5.2 on the same machine that you install WebSphere Application Server, Version 6.x. IBM Tivoli Directory Server, Version 5.2 includes WebSphere Application Server Express, Version 5.0.2, which the directory server uses for its administrative console. Install the Web Administration tool Version 5.2 and WebSphere Application Server Express, Version 5.0.2, which are both bundled with

IBM Tivoli Directory Server, Version 5.2, on a different machine from WebSphere Application Server, Version 6.x. You cannot use WebSphere Application Server, Version 6.x as the administrative console for IBM Tivoli Directory Server. If IBM Tivoli Directory Server, Version 5.2 and WebSphere Application Server, Version 6.x are installed on the same machine, you might encounter port conflicts.

If you must install IBM Tivoli Directory Server Version 5.2 and WebSphere Application Server Version 6.x on the same machine, consider the following information:

- During the IBM Tivoli Directory Server installation process, you must select both the **Web Administration tool** and **WebSphere Application Server Express, Version 5.0.2**.
- Install WebSphere Application Server, Version 6.x.
- When you install WebSphere Application Server, Version 6.x, change the port number for the application server.
- You might need to adjust the WebSphere Application Server environment variables on the version 6.x application server for *WAS_HOME* and *WAS_INSTALL_ROOT*. To change the variables using the administrative console, click **Environment > WebSphere Variables**.

**Using a Lotus Domino Server as the LDAP server**

If you choose the Lotus Domino LDAP server Version 6 and the attribute short name is not defined in the schema, you can take either of the following actions:

- Change the schema to add the short name attribute.
- Change the user ID map filter to replace the short name with any other defined attribute (preferably to UID). For example, change `person:shortname` to `person:uid`.

The userID map filter has been changed to use the **uid** attribute instead of the **shortname** attribute as the current version of Lotus Domino does not create the **shortname** attribute by default. If you want to use the **shortname** attribute, define the attribute in the schema and change the userID map filter to the following:

```
User ID Map :    person:shortname
```

**Using Sun ONE Directory Server as the LDAP server**

You can choose **Sun ONE Directory Server** for your Sun ONE Directory Server system. For supported directory servers, refer to the article, Supported directory services. In Sun ONE Directory Server, the default object class is `groupOfUniqueName` when you create a group. For better performance, WebSphere Application Server uses the user object to locate the user group membership from the *nsRole* attribute. Thus, create the group from the role. If you want to use `groupOfUniqueName` to search groups, specify your own filter setting. Roles unify entries. Roles are designed to be more efficient and easier to use for applications. For example, an application can locate the role of an entry by enumerating all the roles possessed by a given entry, rather than selecting a group and browsing through the members list. When using roles, you can create a group could be created using a:

- Managed role
- Filtered role
- Nested role

All of these roles are computable by `nsRole` attribute.

**Using Microsoft Active Directory server as the LDAP server**

To set up Microsoft Active Directory as your LDAP server, complete the following steps.

1. Determine the full distinguished name (DN) and password of an account in the **administrators** group.

For example, if the Active Directory administrator creates an account in the Users folder of the Active Directory Users and Computers Windows control panel and the DNS domain is `ibm.com`, the resulting DN has the following structure:

```
cn=<adminUsername>, cn=users, dc=ibm,
dc=com
```

2. Determine the short name and password of any account in the Microsoft Active Directory. This password does not have to be the same account that is used in the previous step.
3. Use the WebSphere Application Server administrative console to set up the information needed to use Microsoft Active Directory
   a. Click **Security > Global security**.
   b. Under Authentication, click **Authentication mechanisms > LDAP**.
   c. Set up LDAP with Active Directory at the directory type. Based on the information determined in the previous steps, you can specify the following values on the LDAP settings panel:

      **Server user ID**
         Specify the short name of the account that was chosen in the second step.

      **Server user password**
         Specify the password of the account that was chosen in the second step.

      **Type** Specify Active Directory

      **Host** Specify the domain name service (DNS) name of the machine that is running Microsoft Active Directory.

      **Base distinguished name (DN)**
         Specify the domain components of the DN of the account that was chosen in the first step. For example: `dc=ibm, dc=com Bind`

      **Bind distinguished name (DN)**
         Specify the full distinguished name of the account that was chosen in the first step. For example: `cn=<adminUsername>, cn=users, dc=ibm, dc=com`

      **Bind password**
         Specify the password of the account that was chosen in the first step.
   d. Click **OK** to save the changes.
   e. Stop and restart the administrative server so that the changes take effect.
4. **Optional:** Set ObjectCategory as the filter in the Group member ID map field to improve LDAP performance.
   a. Under Additional properties, click **Advanced Lightweight Directory Access Protocol (LDAP) user registry settings** .
   b. Add `;objectCategory:group` to the end of the Group member ID map field.
   c. Click **OK** to save the changes
   d. Stop and restart the administrative server so that the changes take effect.

## Supported directory services

WebSphere Application Server security supports several different LDAP servers. For a list of supported LDAP servers, refer to the **Supported hardware, software and APIs** prerequisite Web site in the "Security: Resources for learning" on page 21 article.

It is expected that other LDAP servers follow the LDAP specification function. Support is limited to these specific directory servers only. You can use any other directory server by using the custom directory type in the list and by filling in the filters required for that directory.

To improve performance for LDAP searches, the default filters for IBM Directory Server, iPlanet Directory Server, and Active Directory are defined such that when you search for a user, the result contains all the relevant information about the user (user ID, groups, and so on). As a result, the product does not call the LDAP server multiple times. This definition is possible only in these directory types, which support searches where the complete user information is obtained.

If you use the IBM Directory Server, enable the Ignore case flag. This flag is required because when the group information is obtained from the user object attributes, the case is not the same as when you get the group information directly. For the authorization to work in this case, perform a case insensitive check and verify the requirement for the Ignore case flag.

## Locating a user's group memberships in Lightweight Directory Access Protocol

WebSphere Application Server security can be configured to search group memberships directly or indirectly. It can also be configured to search only a static group, or it can be configured to search static groups, recursive (or nested) groups, and dynamic groups for some Lightweight Directory Access Protocol (LDAP) servers.

**Evaluate group memberships from user object directly**

Several popular LDAP servers enable user objects to contain information about the groups to which they belong (such as Microsoft Active Directory Server, or eDirectory). Some user group memberships can be computable attributes from the user object (such as IBM Directory Server or Sun ONE directory server). In some LDAP servers, this attribute can be used to include a user's dynamic group memberships, nesting group memberships, and static group memberships to locate all group memberships from a single attribute.

For example, in IBM Directory Server all group memberships, including the static groups, dynamic groups, and nested groups, can be returned using the ibm-allGroups attribute. In Sun ONE, all roles, including managed roles, filtered roles, and nested roles, are calculated using the nsRole attribute. If an LDAP server has such an attribute in a user object to include dynamic groups, nested groups, and static groups, WebSphere Application Server security can be configured to use this attribute to support dynamic groups, nested groups, and static groups.

**Evaluate group memberships from a group object indirectly**

Some LDAP servers enable only group objects such as the Lotus Domino LDAP server to contain information about users. The LDAP server does not enable the user object to contain information about groups. For this type of LDAP server, group membership searches are performed by locating the user on the member list of groups. The member list evaluation is currently used in the static group membership search for all of the releases before WebSphere Application Server Version 5.

Use the direct method for searching group memberships if your LDAP server has such an attribute in user object to include group information. To use the direct method or the indirect method, enter the appropriate value in the Group Member ID Map field on the Advanced LDAP Settings panel using:

- objectclass:attribute pairs for the indirect method
- attribute:attribute pairs for the direct method

Sample entries of attribute:attribute pairs in Group Member ID Map fields include:

- ibm-allGroups:member for IBM Directory server
- nsRole:nsRole for Sun ONE directory if groups are created with Role inside Sun ONE
- memberOf:member in Microsoft Active Directory Server

Sample entries of objectClass:attribute pairs in the Group Member ID Map field include:

- dominoGroup:member for Domino
- groupOfNames:member for eDirectory

While using the direct method, dynamic groups, recursive groups, and static groups can be returned as multiple values of a single attribute. For example, in IBM Directory Server all group memberships, including the static groups, dynamic groups, and nested groups, can be returned using the ibm-allGroups attribute. In Sun ONE, all roles, including managed roles, filtered roles, and nested roles, are calculated using the nsRole attribute. If an LDAP server can use the nsRole attribute, dynamic groups, nested groups, and static groups are all supported by WebSphere Application Server.

Some LDAP servers do not have recursive computing functionality. For example, although Microsoft Active Directory server has direct group search capability using the memberOf attribute, memberOf lists the groups beneath which the group is directly nested only and does not contain the recursive list of nested predecessors. Another example is that the Lotus Domino LDAP server, which only supports the indirect method to locate the group memberships for a user (you cannot obtain recursive group memberships from a Domino server directly). For LDAP servers without recursive searching capability, WebSphere Application Server security provides a recursive function that is enabled by clicking **Perform a Nested Group Search** in the Advanced LDAP user registry settings. Select this option only if your LDAP server does not provide recursive searches and you want a recursive search.

## Dynamic groups and nested group support

Dynamic groups contain a group name and membership criteria:

- The group membership information is as current as the information on the user object.
- There is no need to manually maintain members on the group object.
- Dynamic groups are designed such so an application does not need to pull a large amount of information from the directory to find out if someone is a member of a group.

*Nested groups* enable the creation of hierarchical relationships that are used to define inherited group membership. A nested group is defined as a child group entry whose distinguished name (DN) is referenced by a parent group entry attribute.

Dynamic and nested groups simplify WebSphere Application Server security management and increase its effectiveness and flexibility. You only need to assign a larger parent group if all nested groups share the same privilege. Assigning a role to a single parent group simplifies the runtime authorization table.

## Dynamic and nested group support for the SunONE or iPlanet Directory Server

The SunONE or iPlanet Directory Server uses two grouping mechanisms:

**Groups**
Groups are entries that name other entries as a list of members or as a filter for members.

**Roles**
Roles are also entries that name other entries as a list of members or as a filter for members. Additional functionality is provided by generating the nsrole attribute on each role member.

Three types of roles are available:

**Filtered roles**
Entries are members if they match a specified Lightweight Directory Access Protocol (LDAP) filter. In this way, the role depends upon the attributes that are contained in each entry. This role is equivalent to a dynamic group.

**Nested roles**
Creates roles that contain other roles. This role is equivalent to a nested group.

**Managed roles**
Explicitly assigns a role to member entries. This role is equivalent to a static group.

Refer to “Configuring dynamic and nested group support for the SunONE or iPlanet Directory Server” for more information.

## Configuring dynamic and nested group support for the SunONE or iPlanet Directory Server

To use dynamic and nested groups with WebSphere Application Server security, you must be running WebSphere Application Server Version 5.1.1 or later. Refer to “Dynamic and nested group support for the SunONE or iPlanet Directory Server” on page 210 for more information on this topic.

1. On the LDAP registry panel, select `SunONE` for the LDAP server.
2. Select the **Ignore case** option
3. On the LDAP settings panel, change the Group Filter setting to `&(cn=%v)(objectclass=ldapsubentry))`
4. On the LDAP settings panel, change the Group Member ID Map setting to `nsRole:nsRole`.

## Dynamic groups and nested group support for the IBM Directory Server

WebSphere Application Server Version 5.x or later supports all Lightweight Directory Access Protocol (LDAP) dynamic and nested groups when using IBM Directory Server 4.x and later. This function is enabled by default by taking advantage of a new feature in IBM Directory Server. IBM Directory Server V4.1 uses the ibm-allGroups forward reference group attribute that automatically calculates all the group memberships (including dynamic and recursive memberships) for a user. Security directly locates a user group membership from a user object rather than indirectly search all the groups to match group members.

Refer to “Configuring dynamic and nested group support for the IBM Directory Server” for more information.

## Configuring dynamic and nested group support for the IBM Directory Server

When creating groups, ensure that nested and dynamic group memberships work correctly.

1. In the WebSphere Application Server security LDAP user registry configuration panel, select `IBM_Directory_Server` for the LDAP server.
2. On the LDAP settings panel change the Group Filter setting. Change the setting to the following value:

   ```
   (&(cn=%v)(|(objectclass=groupOfNames)(objectclass=groupOfUniqueNames)
   (objectclass=groupOfURLs))).
   ```
3. On the LDAP settings panel change the Group Member ID Map setting. Change the setting to the following value:

   ```
   ibm-allGroups:member;ibm-allGroups:uniqueMember
   ```
4. On the Add an LDAP entry panel the Auxiliary object class value is `ibm-nestedGroup` when creating a nested group. On the Add an LDAP entry panel, the Auxiliary object class value is `ibm-dynamicGroup` when creating a dynamic group.

## Custom user registries

A *custom user registry* is a customer-implemented user registry, that implements the UserRegistry Java interface, as provided by the product. A custom-implemented user registry can support virtually any type of an account repository from a relational database, flat file, and so on. The custom user registry provides

considerable flexibility in adapting product security to various environments where some form of a user registry, other than Lightweight Directory Access Protocol (LDAP) or Local Operating System (LocalOS), already exists in the operational environment.

WebSphere Application Server security provides an implementation that uses various local operating system-based registries (Windows, AIX, Solaris, Linux) and various Lightweight Directory Access Protocol (LDAP)-based registries. However, situations can exist where your user and group data resides in other repositories or custom registries (a database, for example) and moving this information to either a LocalOS or an LDAP registry implementation might not be feasible. For these situations WebSphere Application Server security provides a service provider interface (SPI) that you can implement to interact with your current registry. The SPI is the UserRegistry interface. This interface has a set of methods to implement for the product security to interact with your registries for all security-related tasks. The LocalOS and LDAP registry implementations that are provided also implement this interface. Custom user registries are sometimes called the *pluggable user registries* or *custom registries* for short. Your custom user registry implementation is expected to be thread-safe.

The *UserRegistry interface* is a collection of methods that are required to authenticate individual users using either password or certificates and to collect information about the user (privilege attributes) for authorization purposes. This interface also includes methods that obtain user and group information so that they can be given access to resources. When implementing the methods in the interface, you must decide how to map the information that is manipulated by the UserRegistry interface to the information in your registry.

Make sure that your implementation of the custom registry does not depend on any WebSphere Application Server components such as data sources, enterprise beans, and so on. Do not have this dependency because security is initialized and enabled prior to most of the other WebSphere Application Server components during startup. If your previous implementation used these components, make a change that eliminates the dependency.

The methods in the UserRegistry interface operate on the following information for users:

**User Security Name**
: The user name, which is similar to the user name in the Windows, Linux and UNIX systems Local OS registries. This name is used to log in when prompted by a secured application. By default, the Enterprise JavaBeans (EJB) getCallerPrincipal method and the getRemoteUser and getUserPrincipal servlet methods return this name. The user security name is also referred to as *userSecurityName*, *userName*, or *user name*.

**Unique ID**
: This ID represents a unique identifier for the user. The UserRegistry interface requires this identifier to be unique. The unique ID similar to the system ID (SID) in Windows systems, the Unique ID (UID) in Linux and UNIX systems, and the distinguished name (DN) in Lightweight Directory Authentication Protocol (LDAP). This ID is also referred to as *uniqueUserId*. The unique ID is used to make the authorization decisions for protected resources.

**Display name**
: This name is an optional string that describes a user, and it is similar to the FullName attribute in Windows operating systems. The implementation can use display names for informational purposes only; these names are not required to exist or to be unique. The user interface can use the display name to present more information about the user.

**Group Security name**
: This name, which represents the security group, is also referred to as *groupSecurityName*, *groupName* and *group name*.

**Unique ID**
: The unique ID is the identifier for a group. This name is also referred to as *uniqueGroupId*.

**Display name**
: The display name is an optional string that describes a group.

The article on UserRegistry interface describes each of the methods in the UserRegistry interface that need implementing. An explanation of each of the methods and their usage in the sample and any changes from the Version 4 interface are provided. The Related references section provides links to all other custom user registries documentation, including a file-based registry sample. The Sample provided is very simple and is intended to familiarize you with this feature. Do not use this sample in an actual production environment.

## Configuring custom user registries

Before you begin this task, implement and build the UserRegistry interface. For more information on developing custom user registries refer to the article, “Developing custom user registries” on page 104. The following steps are required to configure custom user registries through the administrative console.

1. Click **Security** > **Global security**
2. Under User registries, click **Custom**.
3. Enter a valid user name in the Server user ID field.
4. Enter the password of the user in the Server user password field.
5. Enter the full name of the location of the implementation class file in the Custom registry class name field as a dot-separated file name. For the sample, this file name is `com.ibm.websphere.security.FileRegistrySample`. The file exists in the WebSphere Application Server class path (preferably in the *install_root*/lib/ext directory). This file exists in all the product processes. So, if you are operating in a Network Deployment environment, this file exists in the cell class path and in all of the node class paths.
6. Select the **Ignore case for authorization** option for the authorization to perform a case insensitive check. Enabling this option is necessary only when your registry is case insensitive and does not provide a consistent case when queried for users and groups.
7. Click **Apply** if you have any other additional properties to enter for the registry initialization. Otherwise click **OK** and complete the steps required to turn on security.
8. If you need to enter additional properties to initialize your implementation, click **Custom properties**. Click **New**. Enter the property name and value. Click **OK**. Repeat this step to add other additional properties. For the sample, enter the following two properties. It is assumed that the `users.props` and the `groups.props` file are in the *customer_sample* directory under the product installation directory. You can place these properties in any directory that you chose and reference their location through Custom properties. However, make sure that the directory has the appropriate access permissions.

| Property name | Property value |
| --- | --- |
| usersFile | $*USER_INSTALL_ROOT*/*customer_sample*/users.props |
| groupsFile | $*USER_INSTALL_ROOT*/*customer_sample*/groups.props |

Samples of these two properties are available in the “users.props file” on page 239 and the “groups.props file” on page 239 article.

The Description, Required, and Validation Expression fields are not used and you can leave them blank.

**Note:** In a Network Deployment environment where multiple WebSphere Application Server processes exist (cell and multiple nodes in different machines), these properties are available for each process. Use the relative name *USER_INSTALL_ROOT* to locate any files, as this name expands to the product installation directory. If this name is not used, ensure that the files exist in the same location in all the nodes. To change the value for the *USER_INSTALL_ROOT* variable

This step is required to set up the custom user registry and to enable security in WebSphere Application Server.

1. Complete the remaining steps, if you are enabling security.

2. After security is turned on, save, stop, and start all the product servers (cell, nodes and all the application servers) for any changes in this panel to take effect.
3. If the server comes up without any problems, the setup is correct.
4. Validate the user and password by clicking **OK** or **Apply** on the Global security panel. Save, synchronize (in the cell environment), stop and restart all the product servers.

## UserRegistry.java files

```
// 5639-D57, 5630-A36, 5630-A37, 5724-D18
// (C) COPYRIGHT International Business Machines Corp. 1997, 2004
// All Rights Reserved * Licensed Materials - Property of IBM
//
// DESCRIPTION:
//
//    This file is the UserRegistry interface that custom registries in WebSphere
//    Application Server implement to enable WebSphere security to use the custom
//    registry.
//
package com.ibm.websphere.security;

import java.util.*;
import java.rmi.*;
import java.security.cert.X509Certificate;
import com.ibm.websphere.security.cred.WSCredential;/**
 * Implementing this interface enables WebSphere Application Server Security
 * to use custom registries. This interface extends java.rmi.Remote because the
 * registry can be in a remote process.
 *
 * Implementation of this interface must provide implementations for:
*
* initialize(java.util.Properties)
* checkPassword(String,String)
* mapCertificate(X509Certificate[])
* getRealm
* getUsers(String,int)
* getUserDisplayName(String)
* getUniqueUserId(String)
* getUserSecurityName(String)
* isValidUser(String)
* getGroups(String,int)
* getGroupDisplayName(String)
* getUniqueGroupId(String)
* getUniqueGroupIds(String)
* getGroupSecurityName(String)
* isValidGroup(String)
* getGroupsForUser(String)
* getUsersForGroup(String,int)
* createCredential(String)
**/

public interface UserRegistry extends java.rmi.Remote
{

  /**
   * Initializes the registry. This method is called when creating the
   * registry.
   *
   * @param props the registry-specific properties with which to
   *              initialize the  custom registry
   * @exception CustomRegistryException
   *                   if there is any registry specific problem
   * @exception RemoteException
   *    as this extends java.rmi.Remote
```

```
**/
public void initialize(java.util.Properties props)
   throws CustomRegistryException,
          RemoteException;  /**
* Checks the password of the user. This method is called to authenticate a
* user when the user's name and password are given.
*
* @param userSecurityName the name of user
* @param password the password of the user
* @return a valid userSecurityName. Normally this is
*   the name of same user whose password was checked but if the
*  implementation wants to return any other valid
*  userSecurityName in the registry it can do so
* @exception CheckPasswordFailedException if userSecurityName/
*  password combination does not exist in the registry
* @exception CustomRegistryException if there is any registry specific
*           problem
* @exception RemoteException as this extends java.rmi.Remote
**/
public String checkPassword(String userSecurityName, String password)
   throws PasswordCheckFailedException,
          CustomRegistryException,
          RemoteException;  /**
* Maps a certificate (of X509 format) to a valid user in the registry.
* This is used to map the name in the certificate supplied by a browser
* to a valid userSecurityName in the registry
*
* @param cert the X509 certificate chain
* @return the mapped name of the user userSecurityName
* @exception CertificateMapNotSupportedException if the particular
*           certificate is not supported.
* @exception CertificateMapFailedException if the mapping of the
*           certificate fails.
* @exception CustomRegistryException if there is any registry specific
*           problem
* @exception RemoteException as this extends java.rmi.Remote
**/
public String mapCertificate(X509Certificate[] cert)
   throws CertificateMapNotSupportedException,
          CertificateMapFailedException,
          CustomRegistryException,
          RemoteException;  /**
* Returns the realm of the registry.
*
* @return the realm. The realm is a registry-specific string indicating
*           the realm or domain for which this registry
*           applies.  For example, for OS400 or AIX this would be the
*           host name of the system whose user registry this object
*           represents.
*           If null is returned by this method realm defaults to the
*           value of "customRealm". It is recommended that you use
*           your own value for realm.
* @exception CustomRegistryException if there is any registry specific
*           problem
* @exception RemoteException as this extends java.rmi.Remote
**/
public String getRealm()
   throws CustomRegistryException,
          RemoteException;  /**
* Gets a list of users that match a pattern in the registry.
* The maximum number of users returned is defined by the limit
* argument.
* This method is called by administrative console and by scripting (command
```

```
* line) to make available the users in the registry for adding them (users)
* to roles.
*
* @parameter pattern the pattern to match. (For example., a* will match all
*   userSecurityNames starting with a)
* @parameter limit the maximum number of users that should be returned.
*  This is very useful in situations where there are thousands of
*           users in the registry and getting all of them at once is not
*           practical. A value of 0 implies get all the users and hence
*           must be used with care.
* @return a Result object that contains the list of users
*   requested and a flag to indicate if more users exist.
* @exception CustomRegistryException if there is any registry specific
*           problem
* @exception RemoteException as this extends java.rmi.Remote
**/
public Result getUsers(String pattern, int limit)
   throws CustomRegistryException,
          RemoteException;  /**
* Returns the display name for the user specified by userSecurityName.
*
* This method is called only when the user information displays
* (information purposes only, for example, in the administrative console) and not used
* in the actual authentication or authorization purposes. If there are no
* display names in the registry return null or empty string.
*
* In WebSphere Application Server Version 4.0 custom registry, if you had a display
* name for the user and  if it was different from the security name, the display name
* was returned for the EJB methods getCallerPrincipal() and the servlet methods
* getUserPrincipal() and  getRemoteUser().
* In WebSphere Application Server Version 5.0 for the same methods the security
* name is returned by default. This is the recommended way as the display name
* is not unique and might create security holes.
* However, for backward compatibility if one needs the display name to
* be returned set the property WAS_UseDisplayName to true.
*
* See the documentation for more information.
*
* @parameter userSecurityName the name of the user.
* @return the display name for the user. The display name
*   is a registry-specific string that represents a descriptive, not
*  necessarily unique, name for a user. If a display name does
*           not exist return null or empty string.
* @exception EntryNotFoundException if userSecurityName does not exist.
* @exception CustomRegistryException if there is any registry specific
*           problem
* @exception RemoteException as this extends java.rmi.Remote
**/
public String getUserDisplayName(String userSecurityName)
   throws EntryNotFoundException,
          CustomRegistryException,
          RemoteException;  /**
* Returns the unique ID for a userSecurityName. This method is called when
* creating a credential for a user.
*
* @parameter userSecurityName the name of the user.
* @return the unique ID of the user. The unique ID for an user is
*   the stringified form of some unique, registry-specific, data
*  that serves to represent the user.  For example, for the UNIX
*  user registry, the unique ID for a user can be the UID.
* @exception EntryNotFoundException if userSecurityName does not exist.
* @exception CustomRegistryException if there is any registry specific
*           problem
```

```
 * @exception RemoteException as this extends java.rmi.Remote
 **/
 public String getUniqueUserId(String userSecurityName)
    throws EntryNotFoundException,
           CustomRegistryException,
           RemoteException;  /**
 * Returns the name for a user given its unique ID.
 *
 * @parameter uniqueUserId the unique ID of the user.
 * @return the userSecurityName of the user.
 * @exception EntryNotFoundException if the uniqueUserID does not exist.
 * @exception CustomRegistryException if there is any registry specific
 *           problem
 * @exception RemoteException as this extends java.rmi.Remote
 **/
 public String getUserSecurityName(String uniqueUserId)
    throws EntryNotFoundException,
           CustomRegistryException,
           RemoteException;

/**
 * Determines if the userSecurityName exists in the registry
 *
 * @parameter userSecurityName the name of the user
 * @return true if the user is valid. false otherwise
 * @exception CustomRegistryException if there is any registry specific
 *           problem
 * @exception RemoteException as this extends java.rmi.Remote
 **/
 public boolean isValidUser(String userSecurityName)
    throws CustomRegistryException,
           RemoteException;

/**
 * Gets a list of groups that match a pattern in the registy.
 * The maximum number of groups returned is defined by the limit
 * argument.
 * This method is called by the administrative console and scripting
 * (command line) to make available the groups in the registry for adding
 * them (groups) to roles.
 *
 * @parameter pattern the pattern to match. (For e.g., a* will match all
 *   groupSecurityNames starting with a)
 * @parameter limit the maximum number of groups to return.
 *  This is very useful in situations where there are thousands of
 *           groups in the registry and getting all of them at once is not
 *           practical. A value of 0 implies get all the groups and hence
 *           must be used with care.
 * @return a Result object that contains the list of groups
 *   requested and a flag to indicate if more groups exist.
 * @exception CustomRegistryException if there is any registry-specific
 *           problem
 * @exception RemoteException as this extends java.rmi.Remote
 **/
 public Result getGroups(String pattern, int limit)
    throws CustomRegistryException,
           RemoteException;

/**
 * Returns the display name for the group specified by groupSecurityName.
 *
 * This method may be called only when the group information displayed
 * (for example, the administrative console) and not used in the actual
```

```
 * authentication or authorization purposes. If there are no display names
 * in the registry return null or empty string.
 *
 * @parameter groupSecurityName the name of the group.
 * @return the display name for the group. The display name
 *   is a registry-specific string that represents a descriptive, not
 *  necessarily unique, name for a group. If a display name does
 *           not exist return null or empty string.
 * @exception EntryNotFoundException if groupSecurityName does not exist.
 * @exception CustomRegistryException if there is any registry specific
 *           problem
 * @exception RemoteException as this extends java.rmi.Remote
 **/
 public String getGroupDisplayName(String groupSecurityName)
    throws EntryNotFoundException,
           CustomRegistryException,
           RemoteException;

/**
 * Returns the unique ID for a group.

 * @parameter groupSecurityName the name of the group.
 * @return the unique ID of the group. The unique ID for
 *   a group is the stringified form of some unique,
 *   registry-specific, data that serves to represent the group.
 *           For example, for the UNIX user registry, the unique IDd could
 *  be the GID.
 * @exception EntryNotFoundException if groupSecurityName does not exist.
 * @exception CustomRegistryException if there is any registry specific
 *           problem
 * @exception RemoteException as this extends java.rmi.Remote
 **/
 public String getUniqueGroupId(String groupSecurityName)
    throws EntryNotFoundException,
           CustomRegistryException,
           RemoteException;


/**
 * Returns the unique IDs for all the groups that contain the unique ID of
 * a user.
 * Called during creation of a user's credential.
 *
 * @parameter uniqueUserId the unique ID of the user.
 * @return a list of all the group unique IDs that the unique user ID
 *   belongs to. The unique ID for an entry is the stringified
 *  form of some unique, registry-specific, data that serves
 *  to represent the entry.  For example, for the
 *   UNIX user registry, the unique ID for a group could be the GID
 *  and the unique ID for the user could be the UID.
 * @exception EntryNotFoundException if unique user ID does not exist.
 * @exception CustomRegistryException if there is any registry specific
 *           problem
 * @exception RemoteException as this extends java.rmi.Remote
 **/
 public List getUniqueGroupIds(String uniqueUserId)
    throws EntryNotFoundException,
           CustomRegistryException,
           RemoteException;

/**
 * Returns the name for a group given its unique ID.
 *
```

```
 * @parameter uniqueGroupId the unique ID of the group.
 * @return the name of the group.
 * @exception EntryNotFoundException if the uniqueGroupId does not exist.
 * @exception CustomRegistryException if there is any registry-specific
 *            problem
 * @exception RemoteException as this extends java.rmi.Remote
 **/
 public String getGroupSecurityName(String uniqueGroupId)
    throws EntryNotFoundException,
           CustomRegistryException,
           RemoteException;

/**
 * Determines if the groupSecurityName exists in the registry
 *
 * @parameter groupSecurityName the name of the group
 * @return true if the groups exists, false otherwise
 * @exception CustomRegistryException if there is any registry specific
 *            problem
 * @exception RemoteException as this extends java.rmi.Remote
 **/
 public boolean isValidGroup(String groupSecurityName)
    throws CustomRegistryException,
           RemoteException;

/**
 * Returns the securityNames of all the groups that contain the user
 *
 * This method is called by administrative console and scripting
 * (command line) to verify the user entered for RunAsRole mapping belongs
 * to that role  in the roles to user mapping. Initially, the check is done
 * to see if the role contains the user. If the role does not contain the user
 * explicitly, this method is called to get the groups that this user
 * belongs to so that checks are made on the groups that the role contains.
 *
 * @parameter userSecurityName the name of the user
 * @return a List of all the group securityNames that the user
 *   belongs to.
 * @exception EntryNotFoundException if user does not exist.
 * @exception CustomRegistryException if there is any registry specific
 *            problem
 * @exception RemoteException as this extends java.rmi.Remote
 **/
 public List getGroupsForUser(String userSecurityName)
    throws EntryNotFoundException,
           CustomRegistryException,
           RemoteException;

/**
 * Gets a list of users in a group.
 *
 * The maximum number of users returned is defined by the limit
 * argument.
 *
 * This method is used by the WebSphere Business Integration
 * Server Foundation process choreographer when staff assignments
 * are modeled using groups.
 *

 * In rare situations if you are working with a registry where getting all of
 * the users from any of your groups is not practical (for example if
 * a large number of users exist) you can throw the NotImplementedException
 * for that particular groups. Make sure that if the WebSphere Business
```

```
   * Integration Server Foundation Process Choreographer is installed (or
   * if installed later) that are not modeled using these particular groups.
   * If no concern exists about the staff assignments returning the users from
   * groups in the registry it is recommended that this method be implemented
   * without throwing the NotImplemented exception.
   *
   * @parameter groupSecurityName that represents the name of the group
   * @parameter limit the maximum number of users to return.
   *           This option is very useful in situations where lots of
   *           users are in the registry and getting all of them at
   *           once is not practical. A value of 0 means  get all of
   *           the users and must be used with care.
   * @return    a Result object that contains the list of users
   *           requested and a flag to indicate if more users exist.
   * @deprecated This method will be deprecated in the future.
   * @exception NotImplementedException throw this exception in rare situations
   *           if it is not practical to get this information for any of the
   *           groups from the registry.
   * @exception EntryNotFoundException if the group does not exist in
   *           the registry
   * @exception CustomRegistryException if any registry-specific
   *           problem occurs
   * @exception RemoteException as this extends java.rmi.Remote interface
   **/
   public Result getUsersForGroup(String groupSecurityName, int limit)
      throws NotImplementedException,
             EntryNotFoundException,
             CustomRegistryException,
             RemoteException;

  /**
   * This method is implemented internally by the WebSphere Application Server
   * code in this release. This method is not called for the custom registry
   * implementations for this release. Return null in the implementation.
   *
   * Note that because this method is not called you can also return the
   * NotImplementedException as the previous documentation says.
   *
   **/
   public com.ibm.websphere.security.cred.WSCredential
                             createCredential(String userSecurityName)
      throws NotImplementedException,
      EntryNotFoundException,
             CustomRegistryException,
             RemoteException;
}
```

## FileRegistrySample.java file

The user and group information required by this sample is contained in the users.props and groups.props files.

The contents of the `FileRegistrySample.java` file:

```
//
// 5639-D57, 5630-A36, 5630-A37, 5724-D18
// (C) COPYRIGHT International Business Machines Corp. 1997, 2003
// All Rights Reserved * Licensed Materials - Property of IBM
////----------------------------------------------------------------------
// This program may be used, executed, copied, modified and distributed
// without royalty for the purpose of developing, using, marketing, or
// distributing.
//----------------------------------------------------------------------
//
```

```
// This sample is for the custom user registry feature in WebSphere
// Application Server.

import java.util.*;
import java.io.*;
import java.security.cert.X509Certificate;
import com.ibm.websphere.security.*;
/**
 *  The main purpose of this sample is to demonstrate the use of the
 *  custom user registry feature available in WebSphere Application Server. This
 *  sample is a file-based registry sample where the users and the groups
 *  information is listed in files (users.props and groups.props). As such
 *  simplicity and not the performance was a major factor behind this. This
 *  sample should be used only to get familiarized with this feature. An
 *  actual implementation of a realistic registry should consider various
 *  factors like performance, scalability, thread safety, and so on.
 **/
public class FileRegistrySample implements UserRegistry {

   private static String USERFILENAME = null;
   private static String GROUPFILENAME = null;

   /** Default Constructor **/
   public FileRegistrySample() throws java.rmi.RemoteException {
   }

  /**
   * Initializes the registry. This method is called when creating the
   * registry.
   *
   * @param     props - The registry-specific properties with which to
   *                    initialize the custom registry
   * @exception CustomRegistryException
   *                    if there is any registry-specific problem
   **/
  public void initialize(java.util.Properties props)
         throws CustomRegistryException {
      try {
         /* try getting the USERFILENAME and the GROUPFILENAME from
          * properties that are passed in (For example, from the
          * administrative console). Set these values in the administrative
          * console. Go to the special custom settings in the custom
          * user registry section of the Authentication panel.
          * For example:
          * usersFile   c:/temp/users.props
          * groupsFile  c:/temp/groups.props
          */
          if (props != null) {
             USERFILENAME = props.getProperty("usersFile");
             GROUPFILENAME = props.getProperty("groupsFile");
          }

      } catch(Exception ex) {
         throw new CustomRegistryException(ex.getMessage(),ex);
      }

      if (USERFILENAME == null || GROUPFILENAME == null) {
         throw new CustomRegistryException("users/groups information missing");
      }

   }  /**
   * Checks the password of the user. This method is called to authenticate
```

```
* a user when the user's name and password are given.
*
* @param  userSecurityName the name of user
* @param  password the password of the user
* @return a valid userSecurityName. Normally this is
*         the name of same user whose password was checked
*         but if the implementation wants to return any other
*         valid userSecurityName in the registry it can do so
* @exception CheckPasswordFailedException if userSecurityName/
*            password combination does not exist
*            in the registry
* @exception CustomRegistryException if there is any registry-
*            specific problem
**/
public String checkPassword(String userSecurityName, String passwd)
   throws PasswordCheckFailedException,
          CustomRegistryException {
   String s,userName = null;
   BufferedReader in = null;

   try {
      in = fileOpen(USERFILENAME);
      while ((s=in.readLine())!=null)
      {
         if (!(s.startsWith("#") || s.trim().length() <=0 )) {
            int index = s.indexOf(":");
            int index1 = s.indexOf(":",index+1);
            // check if the userSecurityName:passwd combination exists
            if ((s.substring(0,index)).equals(userSecurityName) &&
                    s.substring(index+1,index1).equals(passwd)) {
               // Authentication successful, return the userId.
               userName = userSecurityName;
               break;
            }
         }
      }
   } catch(Exception ex) {
      throw new CustomRegistryException(ex.getMessage(),ex);
   } finally {
      fileClose(in);
   }


   if (userName == null) {
      throw new PasswordCheckFailedException("Password check failed for user: "
      + userSecurityName);
   }

   return userName;
}  /**
* Maps a X.509 format certificate to a valid user in the registry.
* This is used to map the name in the certificate supplied by a browser
* to a valid userSecurityName in the registry
*
* @param     cert the X509 certificate chain
* @return    The mapped name of the user userSecurityName
* @exception CertificateMapNotSupportedException if the
*            particular certificate is not supported.
* @exception CertificateMapFailedException if the mapping of
*            the certificate fails.
* @exception CustomRegistryException if there is any registry
*            -specific problem
**/
```

```
public String mapCertificate(X509Certificate[] cert)
   throws CertificateMapNotSupportedException,
          CertificateMapFailedException,
          CustomRegistryException {
   String name=null;
   X509Certificate cert1 = cert[0];
   try {
      // map the SubjectDN in the certificate to a userID.
      name = cert1.getSubjectDN().getName();
   } catch(Exception ex) {
      throw new CertificateMapNotSupportedException(ex.getMessage(),ex);
   }

   if(!isValidUser(name)) {
      throw new CertificateMapFailedException("user: " + name
      + " is not valid");
   }
   return name;
}  /**
* Returns the realm of the registry.
*
* @return the realm. The realm is a registry-specific string
* indicating the realm or domain for which this registry
* applies. For example, for OS/400 or AIX this would be
* the host name of the system whose user registry this
* object represents. If null is returned by this method,
* realm defaults to the value of "customRealm". It is
* recommended that you use your own value for realm.
*
* @exception CustomRegistryException if there is any registry-
* specific problem
**/
public String getRealm()
   throws CustomRegistryException {
   String name = "customRealm";
   return name;
}  /**
* Gets a list of users that match a pattern in the registry.
* The maximum number of users returned is defined by the limit
* argument.
* This method is called by the administrative console and scripting
* (command line) to make the users in the registry available for
* adding them (users) to roles.
*
* @param      pattern the pattern to match. (For example, a* will
*             match  all userSecurityNames starting with a)
* @param      limit the maximum number of users that should be
*             returned. This is very useful in situations where
*             there are thousands of users in the registry and
*             getting all of them at once is not practical. The
*             default is 100. A value of 0 implies get all the
*             users and hence must be used with care.
* @return     a Result object that contains the list of users
*             requested and a flag to indicate if  more users
*             exist.
* @exception  CustomRegistryException if there is any registry-
*             specificproblem
**/
public Result getUsers(String pattern, int limit)
   throws CustomRegistryException {
   String s;
   BufferedReader in = null;
   List allUsers = new ArrayList();
```

```
   Result result = new Result();
   int count = 0;
   int newLimit = limit+1;
   try {
      in = fileOpen(USERFILENAME);
      while ((s=in.readLine())!=null)
      {
         if (!(s.startsWith("#") || s.trim().length() <=0 )) {
            int index = s.indexOf(":");
            String user = s.substring(0,index);
            if (match(user,pattern)) {
               allUsers.add(user);
               if (limit !=0 && ++count == newLimit) {
                  allUsers.remove(user);
                  result.setHasMore();
                  break;
               }
            }
         }
      }
   } catch (Exception ex) {
      throw new CustomRegistryException(ex.getMessage(),ex);
   } finally {
      fileClose(in);
   }

   result.setList(allUsers);
   return result;
}  /**
* Returns the display name for the user specified by
*  userSecurityName.
*
* This method may be called only when the user information
* is displayed (information purposes only, for example, in
* the administrative console) and hence not used in the actual
* authentication or authorization purposes. If there are no
* display names in the registry return null or empty string.
*
* In WebSphere Application Server 4 custom registry, if you
* had a display name for the user and if it was different from the
* security name, the display name was returned for the EJB
* methods getCallerPrincipal() and the servlet methods
* getUserPrincipal() and  getRemoteUser().
* In WebSphere Application Server Version 5, for the same
* methods, the security name will be returned by default. This
* is the recommended way as the display name is not unique
* and might create security holes. However, for backward
* compatibility if one needs the display name to be returned
* set the property WAS_UseDisplayName to true.
*
*See the InfoCenter documentation for more information.
*
* @param     userSecurityName the name of the user.
* @return    the display name for the user. The display
*            name is a registry-specific string that
*            represents a descriptive, not necessarily
*            unique, name for a user. If a display name
*            does not exist return null or empty string.
* @exception EntryNotFoundException if userSecurityName
*            does not exist.
* @exception CustomRegistryException if there is any registry-
 *           specific problem
**/
```

```
 public String getUserDisplayName(String userSecurityName)
    throws CustomRegistryException,
           EntryNotFoundException {

    String s,displayName = null;
    BufferedReader in = null;

    if(!isValidUser(userSecurityName)) {
       EntryNotFoundException nsee = new EntryNotFoundException("user: "
       + userSecurityName + " is not valid");
       throw nsee;
    }

    try {
       in = fileOpen(USERFILENAME);
       while ((s=in.readLine())!=null)
       {
          if (!(s.startsWith("#") || s.trim().length() <=0 )) {
             int index = s.indexOf(":");
             int index1 = s.lastIndexOf(":");
             if ((s.substring(0,index)).equals(userSecurityName)) {
                displayName = s.substring(index1+1);
                break;
             }
          }
       }
    } catch(Exception ex) {
       throw new CustomRegistryException(ex.getMessage(), ex);
    } finally {
       fileClose(in);
    }

    return displayName;
 }

/**
 * Returns the unique ID for a userSecurityName. This method is called
 * when creating a credential for a user.
 *
 * @param    userSecurityName - The name of the user.
 * @return   The unique ID of the user. The unique ID for an user
 *           is the stringified form of some unique, registry-specific,
 *           data that serves to represent the user. For example, for
 *           the UNIX user registry, the unique ID for a user can be
 *           the UID.
 * @exception EntryNotFoundException if userSecurityName does not
 *            exist.
 * @exception CustomRegistryException if there is any registry-
 *            specific problem
 **/
 public String getUniqueUserId(String userSecurityName)
    throws CustomRegistryException,
           EntryNotFoundException {

    String s,uniqueUsrId = null;
    BufferedReader in = null;
    try {
       in = fileOpen(USERFILENAME);
       while ((s=in.readLine())!=null)
       {
          if (!(s.startsWith("#") || s.trim().length() <=0 )) {
             int index = s.indexOf(":");
             int index1 = s.indexOf(":", index+1);
```

```
            if ((s.substring(0,index)).equals(userSecurityName)) {
               int index2 = s.indexOf(":", index1+1);
               uniqueUsrId = s.substring(index1+1,index2);
               break;
            }
         }
      }
   } catch(Exception ex) {
      throw new CustomRegistryException(ex.getMessage(),ex);
   } finally {
      fileClose(in);
   }

   if (uniqueUsrId == null) {
      EntryNotFoundException nsee =
      new EntryNotFoundException("Cannot obtain uniqueId for user: "
      + userSecurityName);
      throw nsee;
   }

   return uniqueUsrId;
}  /**
* Returns the name for a user given its uniqueId.
*
* @param      uniqueUserId  - The unique ID of the user.
* @return     The userSecurityName of the user.
* @exception  EntryNotFoundException if the unique user ID does not exist.
* @exception  CustomRegistryException if there is any registry-specific
*             problem
**/
public String getUserSecurityName(String uniqueUserId)
   throws CustomRegistryException,
          EntryNotFoundException {
   String s,usrSecName = null;
   BufferedReader in = null;
   try {
      in = fileOpen(USERFILENAME);
      while ((s=in.readLine())!=null)
      {
         if (!(s.startsWith("#") || s.trim().length() <=0 )) {
            int index = s.indexOf(":");
            int index1 = s.indexOf(":", index+1);
            int index2 = s.indexOf(":", index1+1);
            if ((s.substring(index1+1,index2)).equals(uniqueUserId)) {
               usrSecName = s.substring(0,index);
               break;
            }
         }
      }
   } catch (Exception ex) {
      throw new CustomRegistryException(ex.getMessage(), ex);
   } finally {
      fileClose(in);
   }

   if (usrSecName == null) {
      EntryNotFoundException ex =
         new EntryNotFoundException("Cannot obtain the
         user securityName for " + uniqueUserId);
      throw ex;
   }

   return usrSecName;
```

```
}  /**
* Determines if the userSecurityName exists in the registry
*
* @param     userSecurityName - The name of the user
* @return    True if the user is valid; otherwise false
* @exception CustomRegistryException if there is any registry-
*            specific problem
* @exception RemoteException as this extends java.rmi.Remote
*            interface
**/
public boolean isValidUser(String userSecurityName)
   throws CustomRegistryException {
   String s;
   boolean isValid = false;
   BufferedReader in = null;
   try {
      in = fileOpen(USERFILENAME);
      while ((s=in.readLine())!=null)
      {
         if (!(s.startsWith("#") || s.trim().length() <=0 )) {
            int index = s.indexOf(":");
            if ((s.substring(0,index)).equals(userSecurityName)) {
               isValid=true;
               break;
            }
         }
      }
   } catch (Exception ex) {
      throw new CustomRegistryException(ex.getMessage(), ex);
   } finally {
      fileClose(in);
   }

   return isValid;
}
/**
 * Gets a list of groups that match a pattern in the registry
 * The maximum number of groups returned is defined by the
 * limit argument. This method is called by administrative console
 * and scripting (command line) to make available the groups in
 * the registry for adding them (groups) to roles.
 *
 * @param       pattern the pattern to match. (For example, a* matches
 *              all groupSecurityNames starting with a)
 * @param       Limits the maximum number of groups to return
 *              This is very useful in situations where there
 *              are thousands of groups in the registry and getting all
 *              of them at once is not practical. The default is 100.
 *              A value of 0 implies get all the groups and hence must
 *              be used with care.
 * @return      A Result object that contains the list of groups
 *              requested and a flag to indicate if more groups exist.
 * @exception CustomRegistryException if there is any registry-specific
 *              problem
 **/
 public Result getGroups(String pattern, int limit)
    throws CustomRegistryException {
    String s;
    BufferedReader in = null;
    List allGroups = new ArrayList();      Result result = new Result();
    int count = 0;
    int newLimit = limit+1;
```

```
      try {
         in = fileOpen(GROUPFILENAME);
         while ((s=in.readLine())!=null)
         {
            if (!(s.startsWith("#") || s.trim().length() <=0 )) {
               int index = s.indexOf(":");
               String group = s.substring(0,index);
               if (match(group,pattern)) {
                  allGroups.add(group);
                  if (limit !=0 && ++count == newLimit) {
                     allGroups.remove(group);
                     result.setHasMore();
                     break;
                  }
               }
            }
         }
      } catch (Exception ex) {
         throw new CustomRegistryException(ex.getMessage(),ex);
      } finally {
         fileClose(in);
      }

      result.setList(allGroups);
      return result;
   }

  /**
   * Returns the display name for the group specified by groupSecurityName.
   * For this version of WebSphere Application Server, the only usage of
   * this method is by the clients (administrative console and scripting)
   * to present a descriptive name of the user if it exists.
   *
   * @param groupSecurityName the name of the group.
   * @return  the display name for the group. The display name
   *          is a registry-specific string that represents a
   *          descriptive, not necessarily unique, name for a group.
   *          If a display name does not exist return null or empty
   *          string.
   * @exception EntryNotFoundException if groupSecurityName does
   *          not exist.
   * @exception CustomRegistryException if there is any registry-
   *          specific problem
   **/
   public String getGroupDisplayName(String groupSecurityName)
      throws CustomRegistryException,
             EntryNotFoundException {
      String s,displayName = null;
      BufferedReader in = null;

      if(!isValidGroup(groupSecurityName)) {
         EntryNotFoundException nsee = new EntryNotFoundException("group: "
         + groupSecurityName + " is not valid");
         throw nsee;
      }

      try {
         in = fileOpen(GROUPFILENAME);
         while ((s=in.readLine())!=null)
         {
            if (!(s.startsWith("#") || s.trim().length() <=0 )) {
               int index = s.indexOf(":");
               int index1 = s.lastIndexOf(":");
```

```
            if ((s.substring(0,index)).equals(groupSecurityName)) {
               displayName = s.substring(index1+1);
               break;
            }
         }
      }
   } catch(Exception ex) {
      throw new CustomRegistryException(ex.getMessage(),ex);
   } finally {
      fileClose(in);
   }

   return displayName;
}

/**
 * Returns the Unique ID for a group.

 * @param     groupSecurityName the name of the group.
 * @return    The unique ID of the group. The unique ID for
 *            a group is the stringified form of some unique,
 *            registry-specific, data that serves to represent
 *            the group. For example, for the UNIX user registry,
 *            the unique ID might be the GID.
 * @exception EntryNotFoundException if groupSecurityName does
 *            not exist.
 * @exception CustomRegistryException if there is any registry-
 *            specific problem
 * @exception RemoteException as this extends java.rmi.Remote
 **/
public String getUniqueGroupId(String groupSecurityName)
   throws CustomRegistryException,
          EntryNotFoundException {
   String s,uniqueGrpId = null;
   BufferedReader in = null;
   try {
      in = fileOpen(GROUPFILENAME);
      while ((s=in.readLine())!=null)
      {
         if (!(s.startsWith("#") || s.trim().length() <=0 )) {
            int index = s.indexOf(":");
            int index1 = s.indexOf(":", index+1);
            if ((s.substring(0,index)).equals(groupSecurityName)) {
               uniqueGrpId = s.substring(index+1,index1);
               break;
            }
         }
      }
   } catch(Exception ex) {
      throw new CustomRegistryException(ex.getMessage(),ex);
   } finally {
      fileClose(in);
   }

   if (uniqueGrpId == null) {
      EntryNotFoundException nsee =
      new EntryNotFoundException("Cannot obtain the uniqueId for group: "
      + groupSecurityName);
      throw nsee;
   }

   return uniqueGrpId;
}
```

```
/**
 * Returns the Unique IDs for all the groups that contain the UniqueId
 * of a user. Called during creation of a user's credential.
 *
 * @param     uniqueUserId the unique ID of the user.
 * @return    A list of all the group unique IDs that the unique user
 *            ID belongs to. The unique ID for an entry is the
 *            stringified form of some unique, registry-specific, data
 *            that serves to represent the entry.  For example, for the
 *            UNIX user registry, the unique ID for a group might be
 *            the GID and the Unique ID for the user might be the UID.
 * @exception EntryNotFoundException if uniqueUserId does not exist.
 * @exception CustomRegistryException if there is any registry-specific
 *            problem
 **/
 public List getUniqueGroupIds(String uniqueUserId)
    throws CustomRegistryException,
           EntryNotFoundException {
    String s,uniqueGrpId = null;
    BufferedReader in = null;
    List uniqueGrpIds=new ArrayList();
    try {
       in = fileOpen(USERFILENAME);
       while ((s=in.readLine())!=null)
       {
          if (!(s.startsWith("#") || s.trim().length() <=0 )) {
             int index = s.indexOf(":");
             int index1 = s.indexOf(":", index+1);
             int index2 = s.indexOf(":", index1+1);
             if ((s.substring(index1+1,index2)).equals(uniqueUserId)) {
                int lastIndex = s.lastIndexOf(":");
                String subs = s.substring(index2+1,lastIndex);
                StringTokenizer st1 = new StringTokenizer(subs, ",");
                while (st1.hasMoreTokens())
                   uniqueGrpIds.add(st1.nextToken());
                break;
             }
          }
       }
    } catch(Exception ex) {
       throw new CustomRegistryException(ex.getMessage(),ex);
    } finally {
       fileClose(in);
    }

    return uniqueGrpIds;
 }

/**
 * Returns the name for a group given its uniqueId.
 *
 * @param     uniqueGroupId the unique ID of the group.
 * @return    The name of the group.
 * @exception EntryNotFoundException if the uniqueGroupId does
 *            not exist.
 * @exception CustomRegistryException if there is any registry-
 *            specific problem
 **/
 public String getGroupSecurityName(String uniqueGroupId)
    throws CustomRegistryException,
           EntryNotFoundException {
    String s,grpSecName = null;
```

```
    BufferedReader in = null;
    try {
       in = fileOpen(GROUPFILENAME);
       while ((s=in.readLine())!=null)
       {
          if (!(s.startsWith("#") || s.trim().length() <=0 )) {
             int index = s.indexOf(":");
             int index1 = s.indexOf(":", index+1);
             if ((s.substring(index+1,index1)).equals(uniqueGroupId)) {
                grpSecName = s.substring(0,index);
                break;
             }
          }
       }
    } catch (Exception ex) {
       throw new CustomRegistryException(ex.getMessage(),ex);
    } finally {
       fileClose(in);
    }

    if (grpSecName == null) {
       EntryNotFoundException ex =
          new EntryNotFoundException("Cannot obtain the group
         security name for: " + uniqueGroupId);
       throw ex;
    }

    return grpSecName;

 }

/**
 * Determines if the groupSecurityName exists in the registry
 *
 * @param     groupSecurityName the name of the group
 * @return    True if the groups exists; otherwise false
 * @exception CustomRegistryException if there is any registry-
 *            specific problem
 **/
 public boolean isValidGroup(String groupSecurityName)
    throws CustomRegistryException {
    String s;
    boolean isValid = false;
    BufferedReader in = null;
    try {
       in = fileOpen(GROUPFILENAME);
       while ((s=in.readLine())!=null)
       {
          if (!(s.startsWith("#") || s.trim().length() <=0 )) {
             int index = s.indexOf(":");
             if ((s.substring(0,index)).equals(groupSecurityName)) {
                isValid=true;
                break;
             }
          }
       }
    } catch (Exception ex) {
       throw new CustomRegistryException(ex.getMessage(),ex);
    } finally {
       fileClose(in);
    }

    return isValid;
```

```
  }

 /**
  * Returns the securityNames of all the groups that contain the user
  *
  * This method is called by the administrative console and scripting
  * (command line) to verify the user entered for RunAsRole mapping
  * belongs to that role in the roles to user mapping. Initially, the
  * check is done to see  if the role contains the user. If the role does
  * not contain the user explicitly, this method is called to get the groups
  * that this user belongs to so that check can be made on the groups that
  * the role contains.
  *
  * @param     userSecurityName the name of the user
  * @return    A list of all the group securityNames that the user
  *            belongs to.
  * @exception EntryNotFoundException if user does not exist.
  * @exception CustomRegistryException if there is any registry-
  *            specific problem
  * @exception RemoteException as this extends the java.rmi.Remote
  *            interface
  **/
  public List getGroupsForUser(String userName)
     throws CustomRegistryException,
            EntryNotFoundException {
     String s;
     List grpsForUser = new ArrayList();
     BufferedReader in = null;
     try {
        in = fileOpen(GROUPFILENAME);
        while ((s=in.readLine())!=null)
        {
           if (!(s.startsWith("#") || s.trim().length() <=0 )) {
              StringTokenizer st = new StringTokenizer(s, ":");
              for (int i=0; i<2; i++)
                 st.nextToken();
              String subs = st.nextToken();
              StringTokenizer st1 = new StringTokenizer(subs, ",");
              while (st1.hasMoreTokens()) {
                 if((st1.nextToken()).equals(userName)) {
                    int index = s.indexOf(":");
                    grpsForUser.add(s.substring(0,index));
                 }
              }
           }
        }
     } catch (Exception ex) {
        if (!isValidUser(userName)) {
           throw new EntryNotFoundException("user: " + userName
           + " is not valid");
        }
        throw new CustomRegistryException(ex.getMessage(),ex);
     } finally {
        fileClose(in);
     }

     return grpsForUser;
  }

 /**
  * Gets a list of users in a group.
  *
  * The maximum number of users returned is defined by the
```

```
* limit argument.
*
* This method is being used by the WebSphere Application
* Server Enterprise Process Choreographer (Enterprise) when
* staff assignments are modeled using groups.

*
* In rare situations, if you are working with a registry where
* getting all the users from any of your groups is not practical
* (for example if there are a large number of users) you can throw
* the NotImplementedException for that particular group. Make sure
* that if the process choreographer is installed (or if installed later)
* the staff assignments are not modeled using these particular groups.
* If there is no concern about returning the users from groups
* in the registry it is recommended that this method be implemented
* without throwing the NotImplemented exception.
* @param         groupSecurityName the name of the group
* @param         Limits the maximum number of users that should be
*                returned. This is very useful in situations where there
*                are lot of users in the registry and getting all of
*                them at once is not practical. A value of 0 implies
*                get all the users and hence must be used with care.
* @return        A result object that contains the list of users
*                requested and a flag to indicate if more users exist.
* @deprecated    This method will be deprecated in future.
* @exception     NotImplementedException throw this exception in rare
*                situations if it is not practical to get this information
*                for any of the group or groups from the registry.
* @exception     EntryNotFoundException if the group does not exist in
*                the registry
* @exception     CustomRegistryException if there is any registry-specific
*                problem
**/
public Result getUsersForGroup(String groupSecurityName, int limit)
   throws NotImplementedException,
          EntryNotFoundException,
          CustomRegistryException {
   String s, user;
   BufferedReader in = null;
   List usrsForGroup = new ArrayList();
   int count = 0;
   int newLimit = limit+1;
   Result result = new Result();

   try {
      in = fileOpen(GROUPFILENAME);
      while ((s=in.readLine())!=null)
      {
         if (!(s.startsWith("#") || s.trim().length() <=0 )) {
            int index = s.indexOf(":");
            if ((s.substring(0,index)).equals(groupSecurityName))
            {
               StringTokenizer st = new StringTokenizer(s, ":");
               for (int i=0; i<2; i++)
                  st.nextToken();
               String subs = st.nextToken();
               StringTokenizer st1 = new StringTokenizer(subs, ",");
               while (st1.hasMoreTokens()) {
                  user = st1.nextToken();
                  usrsForGroup.add(user);
                  if (limit !=0 && ++count == newLimit) {
                     usrsForGroup.remove(user);
                     result.setHasMore();
```

```
                  break;
               }
            }
         }
      }
   }
} catch (Exception ex) {
   if (!isValidGroup(groupSecurityName)) {
      throw new EntryNotFoundException("group: "
      + groupSecurityName
      + " is not valid");
   }
   throw new CustomRegistryException(ex.getMessage(),ex);
} finally {
   fileClose(in);
}

result.setList(usrsForGroup);
return result;
}

/**
 * This method is implemented internally by the WebSphere Application
 * Server code in this release. This method is not called for the custom
 * registry implementations for this release. Return null in the
 * implementation.
 *
 **/
public com.ibm.websphere.security.cred.WSCredential
      createCredential(String userSecuritytName)
      throws CustomRegistryException,
             NotImplementedException,
             EntryNotFoundException {

   // This method is not called.
   return null;
}

// private methods
private BufferedReader fileOpen(String fileName)
   throws FileNotFoundException {
   try {
      return new BufferedReader(new FileReader(fileName));
   } catch(FileNotFoundException e) {
      throw e;
   }
}

private void fileClose(BufferedReader in) {
   try {
      if (in != null) in.close();
   } catch(Exception e) {
      System.out.println("Error closing file" + e);
   }
}

private boolean match(String name, String pattern) {
   RegExpSample regexp = new RegExpSample(pattern);
   boolean matches = false;
   if(regexp.match(name))
       matches = true;
   return matches;
}
```

```
}


//----------------------------------------------------------------------
// The program provides the Regular Expression implementation
// used in the sample for the custom user registry (FileRegistrySample).
// The pattern matching in the sample uses this program to search for the
// pattern (for users and groups).
//----------------------------------------------------------------------

class RegExpSample
{

    private boolean match(String s, int i, int j, int k)
    {
        for(; k < expr.length; k++)
label0:
            {
                Object obj = expr[k];
                if(obj == STAR)
                {
                    if(++k >= expr.length)
                        return true;
                    if(expr[k] instanceof String)
                    {
                        String s1 = (String)expr[k++];
                        int l = s1.length();
                        for(; (i = s.indexOf(s1, i)) >= 0; i++)
                            if(match(s, i + l, j, k))
                                return true;

                        return false;
                    }
                    for(; i < j; i++)
                        if(match(s, i, j, k))
                            return true;

                    return false;
                }
                if(obj == ANY)
                {
                    if(++i > j)
                        return false;
                    break label0;
                }
                if(obj instanceof char[][])
                {
                    if(i >= j)
                        return false;
                    char c = s.charAt(i++);
                    char ac[][] = (char[][])obj;
                    if(ac[0] == NOT)
                    {
                        for(int j1 = 1; j1 < ac.length; j1++)
                            if(ac[j1][0] <= c && c <= ac[j1][1])
                                return false;

                        break label0;
                    }
                    for(int k1 = 0; k1 < ac.length; k1++)
                        if(ac[k1][0] <= c && c <= ac[k1][1])
                            break label0;
```

```
                return false;
            }
            if(obj instanceof String)
            {
                String s2 = (String)obj;
                int i1 = s2.length();
                if(!s.regionMatches(i, s2, 0, i1))
                    return false;
                i += i1;
            }
        }

    return i == j;
}

public boolean match(String s)
{
    return match(s, 0, s.length(), 0);
}

public boolean match(String s, int i, int j)
{
    return match(s, i, j, 0);
}

public RegExpSample(String s)
{
    Vector vector = new Vector();
    int i = s.length();
    StringBuffer stringbuffer = null;
    Object obj = null;
    for(int j = 0; j < i; j++)
    {
        char c = s.charAt(j);
        switch(c)
        {
        case 63: /* '?' */
            obj = ANY;
            break;

        case 42: /* '*' */
            obj = STAR;
            break;

        case 91: /* '[' */
            int k = ++j;
            Vector vector1 = new Vector();
            for(; j < i; j++)
            {
                c = s.charAt(j);
                if(j == k && c == '^')
                {
                    vector1.addElement(NOT);
                    continue;
                }
                if(c == '\\')
                {
                    if(j + 1 < i)
                        c = s.charAt(++j);
                }
                else
                if(c == ']')
                    break;
```

```
                char c1 = c;
                if(j + 2 < i && s.charAt(j + 1) == '-')
                    c1 = s.charAt(j += 2);
                char ac1[] = {
                    c, c1
                };
                vector1.addElement(ac1);
            }

            char ac[][] = new char[vector1.size()][];
            vector1.copyInto(ac);
            obj = ac;
            break;

        case 92: /* '\\' */
            if(j + 1 < i)
                c = s.charAt(++j);
            break;

        }
        if(obj != null)
        {
            if(stringbuffer != null)
            {
                vector.addElement(stringbuffer.toString());
                stringbuffer = null;
            }
            vector.addElement(obj);
            obj = null;
        }
        else
        {
            if(stringbuffer == null)
                stringbuffer = new StringBuffer();
            stringbuffer.append(c);
        }
    }

    if(stringbuffer != null)
        vector.addElement(stringbuffer.toString());
    expr = new Object[vector.size()];
    vector.copyInto(expr);
}

static final char NOT[] = new char[2];
static final Integer ANY = new Integer(0);
static final Integer STAR = new Integer(1);
Object expr[];

}
```

## Result.java file

This module is used by user registries in WebSphere Application Server when calling the getUsers and getGroups methods. The user registries use this method to set the list of users and groups and to indicate if there are more users and groups in the registry than requested.

```
//  @(#) 1.20 src/en/ae/rsec_result.xml, WEBSJAVA.INFO.DOCSRC,
//  ASVINFO1 10/17/02 16:43:01 [10/18/02 07:31:30]
//  5639-D57, 5630-A36, 5630-A37, 5724-D18
// (C) COPYRIGHT International Business Machines Corp. 1997, 2003
//  All Rights Reserved * Licensed Materials - Property of IBM
//
package com.ibm.websphere.security;
```

```
import java.util.List;

public class Result implements java.io.Serializable {
    /**
      Default constructor
    */
    public Result() {
    }

    /**
       Returns the list of users and groups
       @return the list of users and groups
    */
    public List getList() {
      return list;
    }

    /**
       indicates if there are more users and groups in the registry
    */
    public boolean hasMore() {
      return more;
    }
    /**
       Set the flag to indicate that there are more users and groups
       in the registry to true
    */
    public void setHasMore() {
      more = true;
    }

    /*
      Set the list of users and groups
      @param list   list of users/groups
    */
    public void setList(List list) {
      this.list = list;
    }

    private boolean more = false;
    private List list;
}
```

## Custom user registry settings

Use this page to configure the custom user registry.

To view this administrative console page, click **Security > Global security**. Under User registries, click **Custom**.

After the properties are set in this panel, click **Apply**. Under Additional Properties, click **Custom properties** to include additional properties that the custom registry requires. The following property is predefined by the product; set this property when required only:

**WAS_UseDisplayName**
: When this property is set to `true`, the getCallerPrincipal(), getUserPrincipal(), and getRemoteUser() methods return the display name. By default, the securityName of the user is returned. This property is introduced to support backward compatibility with the version 4 custom registry.

When security is enabled and any of these custom user registry settings change, go to the Global security panel and click **Apply** to validate the changes.

***Server user ID:***

Specifies the user ID under which the server runs, for security purposes.

This server ID represents a valid user in the custom registry.

| | |
|---|---|
| **Data type:** | String |

***Server user password:***

Specifies the password corresponding to the security server ID.

| | |
|---|---|
| **Data type:** | String |

***Custom registry class name:***

Specifies a dot-separated class name that implements the com.ibm.websphere.security.UserRegistry interface.

Put the custom registry class name in the class path. A suggested location is the `%install_root%/lib/ext` directory. Although the custom registry implements the com.ibm.websphere.security.UserRegistry interface, for backward compatibility, a user registry can alternately implement the com.ibm.websphere.security.CustomRegistry interface.

| | |
|---|---|
| **Data type:** | String |
| **Default:** | com.ibm.websphere.security.FileRegistrySample |

***Ignore case for authorization:***

Specifies that a case insensitive authorization check is performed when you use the default authorization.

| | |
|---|---|
| **Default:** | Disabled |
| **Range:** | Enabled or Disabled |

## users.props file

Following is the format for the `users.props` file:

```
# 5639-D57, 5630-A36, 5630-A37, 5724-D18
# (C) COPYRIGHT International Business Machines Corp. 1997, 2004
# All Rights Reserved * Licensed Materials - Property of IBM
#
# Format:
# name:passwd:uid:gids:display name
# where name   = userId/userName of the user
#       passwd = password of the user
#       uid    = uniqueId of the user
#       gid    = groupIds of the groups that the user belongs to
#       display name = a (optional) display name for the user.
bob:bob1:123:567:bob
dave:dave1:234:678:
jay:jay1:345:678,789:Jay-Jay
ted:ted1:456:678:Teddy G
jeff:jeff1:222:789:Jeff
vikas:vikas1:333:789:vikas
bobby:bobby1:444:789:
```

## groups.props file

The following example illustrates the format for the `groups.props` file:

```
# 5639-D57, 5630-A36, 5630-A37, 5724-D18
# (C) COPYRIGHT International Business Machines Corp. 1997, 2003
# All Rights Reserved * Licensed Materials - Property of IBM
#
# Format:
# name:gid:users:display name
# where name   = groupId of the group
#       gid    = uniqueId of the group
#       users  = list of all the userIds that the group contains
#       display name = a (optional) display name for the group.
admins:567:bob:Administrative group
operators:678:jay,ted,dave:Operators group
users:789:jay,jeff,vikas,bobby:
```

## Java Authentication and Authorization Service

The standard Java 2 security API helps enforce access control, based on the location of the code and the user. The current principal of the running thread is not considered in the Java 2 security authorization. Instances where authorization is based on the principal (as opposed to the code base) and the user exist. The Java Authentication and Authorization Service is a standard Java API that supports the Java 2 security authorization to extend the code base on the principal as well as the code base and users.

The Java Authentication and Authorization Service (JAAS) Version 1.0 extends the Java 2 security architecture of the Java 2 platform with additional support to authenticate and enforce access control with principals and users. It implements a Java version of the standard Pluggable Authentication Module (PAM) framework, and extends the access control architecture of the Java 2 platform in a compatible fashion to support user-based authorization or principal-based authorization. WebSphere Application Server fully supports the JAAS architecture and extends the access control architecture to support role-based authorization for Java 2 Platform, Enterprise Edition (J2EE) resources including servlets, JavaServer Pages (JSP) files, and Enterprise JavaBeans (EJB) components. Refer to Java 2 security for more information.

The following sections cover the JAAS implementation and programming model:
- Java Authentication and Authorization Service login configuration
- Programmatic Login
- Java Authentication and Authorization Service authorization

The JAAS documentation can be found at http://www.ibm.com/developerworks/java/jdk/security. Scroll down to find the JAAS documentation for your platform.

## Java Authentication and Authorization Service authorization

Java 2 security architecture uses a security policy to specify which access rights are granted to running code. This architecture is *code-centric*. That is, the permissions are granted based on code characteristics including where the code is coming from, whether it is digitally signed, and by whom. Authorization of the Java Authentication and Authorization Service (JAAS) augments the existing code-centric access controls with new user-centric access controls. Permissions are granted based on what code is running and who is running it.

When using JAAS authentication to authenticate a user, a *subject* is created to represent the authenticated user. A subject is comprised of a set of principals, where each principal represents an identity for that user. You can grant permissions in the policy to specific principals. After the user is authenticated, the application can associate the subject with the current access control context. For each subsequent security-checked operation, the Java run time automatically determines whether the policy grants the required permission to a specific principal only. If so, the operation is supported if the subject associated with the access control context contains the designated principal only.

Associate a subject with the current access control context by calling the static doAs method from the subject class, passing it an authenticated subject and java.security.PrivilegedAction or java.security.PrivilegedExceptionAction. The doAs method associates the provided subject with the current access control context and then invokes the run method from the action. The run method implementation contains all the code that ran as the specified subject. The action runs as the specified subject.

In the Java 2 Platform, Enterprise Edition (J2EE) programming model, when invoking the EJB method from an enterprise bean or servlet, the method runs under the user identity that is determined by the run-as setting. The J2EE Version 1.4 Specification does not indicate which user identity to use when invoking an enterprise bean from a Subject.doAs action block within either the EJB code or the servlet code. A logical extension is to use the proper identity specified in the subject when invoking the EJB method within the Subject doAs action block.

This simple rule of letting Subject.doAs overwrite the run-as identity setting is an ideal way to integrate the JAAS programming model with the J2EE run-time environment. However, a general JAAS design oversight was introduced into IBM Software Development Kit (SDK), Java Technology Edition Version 1.3 or later when integrating the JAAS Version 1.0 or later implementation with the Java 2 security architecture. A subject, which is associated with the access control context is cut off by a doPrivileged call when a doPrivileged call occurs within the Subject.doAs action block. Until this problem is corrected, no reliable and run-time efficient way is available to guarantee the correct behavior of Subject.doAs in a J2EE run-time environment.

The problem can be explained better with the following example:

```
Subject.doAs(subject, new java.security.PrivilegedAction() {
   Public Object run() {
       // Subject is associated with the current thread context
       java.security.AccessController.doPrivileged( new
           java.security.PrivilegedAction() {
                         public Object run() {
                         // Subject was cut off from the current
                         // thread context
   return null;
              }
        });
        // Subject is associated with the current thread context
        return null;
   }
});
```

At line three, the subject object is associated with the context of the current thread. As indicated on line 7 within the run method of a doPrivileged action block, the subject object is removed from the thread context. After leaving the doPrivileged block, the subject object is restored to the current thread context. Because doPrivileged blocks can be placed anywhere along the running path and instrumented quite often in a server environment, the run-time behavior of a doAs action block becomes difficult to manage.

To resolve this difficulty, WebSphere Application Server provides a WSSubject helper class to extend the JAAS authorization to a J2EE EJB method invocation as described previously. The WSSubject class provides static doAs and doAsPrivileged methods that have identical signatures to the subject class. The WSSubject.doAs method associates the Subject to the currently running thread. The WSSubject.doAs and WSSubject.doAsPrivileged methods then invoke the corresponding Subject.doAs and Subject.doAsPrivileged methods. The original credential is restored and associated with the running thread upon leaving the WSSubject.doAs and WSSubject.doAsPrivileged methods.

Note that the WSSubject class is not a replacement of the subject object, but rather a helper class to ensure consistent run-time behavior as long as an EJB method invocation is a concern.

The following example illustrates the run-time behavior of the WSSubject.doAs method:

```
WSSubject.doAs(subject, new java.security.PrivilegedAction() {
   Public Object run() {
       // Subject is associated with the current thread context
       java.security.AccessController.doPrivileged( new
           java.security.PrivilegedAction() {
                      public Object run() {
                         // Subject was cut off from the current thread
                         // context.
   return null;
              }
        });
        // Subject is associated with the current thread context
   return null;
   }
});
```

The Subject.doAs and Subject.doAsPrivileged methods are not integrated with the J2EE run-time environment. EJB methods that are invoked within the Subject.doAs and Subject.doAsPrivileged action blocks run under the identity specified by the run-as setting and not by the subject identity.

- The subject object generated by the WSLoginModuleImpl instance and the WSClientLoginModuleImpl instance contains a principal that implements the WSPrincipal interface. Using the getCredential() method for a WSPrincipal object returns an object that implements the WSCredential interface. You can also find the WSCredential object instance in the PublicCredentials list of the subject instance. Retrieve the WSCredential object from the PublicCredentials list instead of using the getCredential() method.
- The getCallerPrincipal() method for the WSSubject class returns a string representing the caller security identity. The return type differs from the getCallerPrincipal method of the EJBContext interface, which is java.security.Principal.
- The Subject object generated by the J2C DefaultPrincipalMapping module contains a resource principal and a PasswordCredentials list. The resource principal represents the RunAs identity.

Refer to “J2EE Connector security” on page 256 for more information

## Configuring application logins for Java Authentication and Authorization Service

Java Authentication and Authorization Service (JAAS) is a new feature in WebSphere Application Server. It is a collection of WebSphere Application Server strategic authentication APIs and replaces the Common Object Request Broker Architecture (CORBA) programmatic login APIs.

WebSphere Application Server provides some extensions to JAAS:

- **com.ibm.websphere.security.auth.WSSubject**. The com.ibm.websphere.security.auth.WSSubject API extends the JAAS authorization model to Java 2 Platform, Enterprise Edition (J2EE) resources.
- You can configure JAAS login in the administrative console and store this configuration in the WebSphere configuration application programming interface (API). However, WebSphere Application Server still supports the default JAAS login configuration format (plain text file) provided by the JAAS default implementation. If duplicate login configurations are defined in both the WebSphere configuration API and the plain text file format, the one in the WebSphere configuration API takes precedence. Advantages to defining the login configuration in the WebSphere configuration API include:
  - User interface support in defining JAAS login configuration
  - Central management of the JAAS login configuration
  - Distribution of the JAAS login configuration in a Network Deployment product installation

Due to a design oversight in the JAAS V1.0, javax.security.auth.Subject.getSubject() method does not return the subject associated with the running thread inside a java.security.AccessController.doPrivileged() code block. This problem presents an inconsistent behavior that is problematic and causes undesirable effort. The com.ibm.websphere.security.auth.WSSubject API provides a workaround to associate the subject to a running thread.

- **Proxy LoginModule**. The Proxy LoginModule loads the actual LoginModule. The default JAAS implementation does not use the thread context class loader to load classes. The LoginModule module cannot load if the LoginModule class file is not in the application class loader or the Java extension class loader class path. Due to this class loader visibility problem, WebSphere Application Server provides a proxy LoginModule module to load the JAAS LoginModule using the thread context class loader. You do not need to place the LoginModule implementation on the application class loader or the Java extension class loader class path with this proxy LoginModule module.

  If you do not want to use the Proxy LoginModule, you can place the LoginModule in the `jre/lib/ext` directory. However, this is not recommended due to the security risks.

Two JAAS login configurations are defined in the WebSphere Configuration API security document for applications to use. In the left navigation pane, click **Security > Global Security > JAAS Configuration > Application Login >WSLogin** and **ClientContainer**. The following three JAAS login configurations are available:

**WSLogin**
: Defines a login configuration and a LoginModule implementation that applications can use in general.

**ClientContainer**
: Defines a login configuration and a LoginModule implementation that is similar to that of the WSLogin configuration, but enforces the requirements of the WebSphere Application Server client container. Refer to Configuration entry settings for Java Authentication and Authorization Service for more information.

**DefaultPrincipalMapping,**
: Defines a special LoginModule module that is typically used by J2EE Connector to map an authenticated WebSphere user identity to a set of user authentication data (user ID and password) for the specified back-end enterprise information system (EIS). For more information about J2EE Connector and the DefaultMappingModule module, refer to the J2EE security section.

A new JAAS login configuration can be added and modified using the administrative console. The changes are saved in the cell-level security document and are available to all managed application servers. An application server restart is required for the changes to take effect at run time.

**Attention:** Do not remove or delete the predefined JAAS login configurations (ClientContainer, WSLogin and DefaultPrincipalMapping). Deleting or removing them can cause other enterprise applications to fail.

1. Delete a JAAS login configuration.
   a. Click **Security** in the navigation tree and expand the **Global Security** panel.
   b. Click **JAAS Configuration** > **Application Logins**. The Application Login Configuration panel appears.
   c. Select the check box for the login configurations to delete and click **Delete**.
2. Create a new JAAS login configuration.
   a. Click **Security** in the navigation tree and expand the **Global Security** panel.
   b. Click **JAAS Configuration** > **Application Logins**.
   c. Click **New**. The Application Login Configuration panel appears.
   d. Specify the alias name of the new JAAS login configuration and click **Apply**. This value is the name of the login configuration that you pass in the javax.security.auth.login.LoginContext implementation for creating a new LoginContext.

Click **Apply** to save changes and to add the extra node name that precedes the original alias name. Clicking **OK** does not save the new changes in the `security.xml` file.

e. Click **JAAS Login Modules**.

f. Click **New**.

g. Specify the Module Classname. Specify WebSphere Proxy LoginModule because of the limitation of the class loader visibility problem.

h. Specify the LoginModule implementation as the delegate property of the Proxy LoginModule. The WebSphere Proxy LoginModule class name is com.ibm.ws.security.common.auth.module.proxy.WSLoginModuleProxy.

i. Select **Authentication Strategy** from the list and click **Apply**.

j. Click **Custom Properties**. The **Custom Properties** panel is displayed for the selected LoginModule.

k. Create a new property with the name `delegate` and the value of the real LoginModule implementation. You can specify other properties like `debug` with the value `true`. These properties are passed to the LoginModule class as options to the initialize() method of the LoginModule instance.

l. Click **Save**.

There are several locations within the WebSphere Application Server directory structure where you can place a JAAS login module. The following list provides locations for the JAAS login module in order of recommendation:

- Within an Enterprise Archive (EAR) file for a specific Java 2 Enterprise Edition (J2EE) application.

  If you place the login module within the EAR file, it is accessible to the specific application only.
- In the WebSphere Application Server shared library.

  If you place the login module in the shared library, you must specify which applications can access the module. For more information on shared libraries, see Managing shared libraries.
- In the Java extensions directory (`*WAS_HOME*\jre\lib\ext`)

  If you place the JAAS login module in the Java extensions directory, the login module is available to all applications.

Although the Java extensions directory provides the greatest availability for the login module, it is recommended that you place the login module in an application EAR file. If other applications need to access the same login module, consider using shared libraries.

3. Change the plain text file. WebSphere Application Server supports the default JAAS login configuration format (plain text file) provided by the JAAS default implementation. However, a tool is not provided that edits plain text files in this format. You can define the JAAS login configuration in the plain text file (*install_root*`/properties/wsjaas.conf`). Any syntax errors can cause the incorrect parsing of the plain JAAS login configuration text file. This problem can cause other applications to fail.

   Java client programs that use the Java Authentication and Authorization Service (JAAS) for authentication must invoke with the JAAS configuration file specified. This configuration file is set in the `/`*install_root*`/bin/launchClient.bat` file as `set JAAS_LOGIN_CONFIG=-Djava.security.auth.login.config=%install_root%\properties\wsjaas_client.conf`. If the `launchClient.bat` file is not used to invoke the Java client program, verify that the appropriate JAAS configuration file is passed to the Java virtual machine with the `-Djava.security.auth.login.config` flag.

A new JAAS login configuration is created or an old JAAS login configuration is removed. An enterprise application can use a newly created JAAS login configuration without restarting the application server process.

However, new JAAS login configurations defined in the *install_root*`/properties/wsjaas.conf` file, do not refresh automatically. Restart the application servers to validate changes. These JAAS login configurations are specific to a particular node and are not available for other application servers running on other nodes.

Create new JAAS login configurations used by enterprise applications to perform custom authentication. Use these newly defined JAAS login configurations to perform programmatic login.

## Login configuration for Java Authentication and Authorization Service

Java Authentication and Authorization Service (JAAS) is a new feature in WebSphere Application Server. JAAS is WebSphere strategic APIs for authentication and it will replace the CORBA programmatic login APIs. WebSphere Application Server provides some extensions to JAAS:

- **com.ibm.websphere.security.auth.WSSubject**: The com.ibm.websphere.security.auth.WSSubject API extends the JAAS authorization model to J2EE resources. You can configure JAAS login in the administrative console (or by using the scripting functions) and store this configuration in the WebSphere configuration application programming interface (API). However, WebSphere Application Server still supports the default JAAS login configuration format (plain text file) provided by the JAAS default implementation. If duplicate login configurations are defined in both the WebSphere configuration API and the plain text file format, the one in the WebSphere configuration API takes precedence. Advantages to defining the login configuration in the WebSphere configuration API include:
  - User interface support in defining JAAS login configuration
  - Central management of the JAAS login configuration
  - Distribution of the JAAS login configuration in a Network Deployment product installation

  Due to a design oversight in the JAAS 1.0, javax.security.auth.Subject.getSubject() does not return the Subject associated with the thread of execution inside a java.security.AccessController.doPrivileged() code block. This can present a inconsistent behavior that is problematic and causes undesirable effort. com.ibm.websphere.security.auth.WSSubject provides a work around to associate Subject to thread of execution. com.ibm.websphere.security.auth.WSSubject extends the JAAS authorization model to J2EE resources.

  **Note: Why WebSphere Application Server has its own subject class**: You can retrieve the subjects in a Subject.doAs() block with the Subject.getSubject() call. However, this procedure does not work if there is an AccessController.doPrivileged() call within the Subject.doAs() block. In the following example, s1 is equal to s, but s2 is null:

  * AccessController.doPrivileged() not only truncates the Subject propagation,
  * but also reduces the permissions. It does not include the JAAS security
  * policy defined for the principals in the Subject.

```
Subject.doAs(s, new PrivilegedAction() {
  public Object run() {
    System.out.println("Within Subject.doAsPrivileged()");
    Subject s1 = Subject.getSubject(AccessController.getContext());
    AccessController.doPrivileged(new PrivilegedAction() {
      public Object run() {
      Subject s2 = Subject.getSubject(AccessController.getContext());
      return null;
    }
  });
  return null;
}
});
```

- JAAS Login Configuration can be configured in administrative console (or by using the scripting functions) and stored in WebSphere configuration application programming interface (API). An application can define new JAAS login configuration in the Admin Console and the the data is persisted in the configuration respository (stored in the WebSphere configuration API). However, WebSphere still support the default JAAS login configuration format (plan text file) provided by the JAAS default implementation. But if there are duplication login configurations defined in both the WebSphere configuration API and the plan text file format, the one in the WebSphere configuration API takes precedence. There are advantages to define the login configuration in the WebSphere configuration API:

- – UI support in defining JAAS login configuration.
  – The JAAS configuration login configuration can be managed centrally.
  – The JAAS configuration login configuration is distributed in a Network Deployment installation.
- **Proxy LoginModule**: The Proxy.LoginModule API stores login module .jar files. The default JAAS implementation does not use the thread context class loader to load classes, the LoginModule could not be loaded if the LoginModule class file is not in the application class loader or the Java extension class loader classpath. Due to this class loader visibility problem, WebSphere provides a proxy LoginModule to load JAAS LoginModule using the thread context class loader. The LoginModule implementation does not have to be placed on the application class loader or the Java extension class loader classpath with this proxy LoginModule.

**Note:** Do not remove or delete the pre-defined JAAS Login Configurations (ClientContainer, WSLogin and DefaultPrincipalMapping). Deleting or removing them could cause other enterprise applications to fail.

A system administrator determines the authentication technologies, or LoginModules, to be used for each application and configures them in a login configuration. The source of the configuration information (for example, a file or a database) is up to the current javax.security.auth.login.Configuration implementation. The WebSphere Application Server implementation permits the login configuration to be defined in both the WebSphere configuration API security document and in a JAAS configuration file where the former takes precedence.

Two JAAS login configurations are defined in the WebSphere configuration API security document for applications to use. To access the configurations, click **Security** > **JAAS Configuration** > **Application Login Config**: **WSLogin** and **ClientContainer**. The **WSLogin** defines a login configuration and LoginModule implementation that may be used by applications in general. The **ClientContainer** defines a login configuration and LoginModule implementation that is similar to that of WSLogin but enforces the requirements of the WebSphere Application Server Client Container. The third entry, **DefaultPrincipalMapping**, defines a special LoginModule that is typically used by Java 2 Connector to map an authenticated WebSphere user identity to a set of user authentication data (user ID and password) for the specified back end enterprise information system (EIS). For more information about Java 2 Connector and the DefaultMappingModule please refer to the Java 2 Security section.

New JAAS login configuration may be added and modified using Security Center. The changes are saved in the cell level security document and are available to all managed application servers. An application server restart is required for the changes to take effect at run time and for the client container login configuration to be made available.

WebSphere Application Server also reads JAAS Configuration information from the wsjaas.conf file under the properties sub directory of the root directory under which WebSphere Application Server is installed. Changes made to the wsjaas.conf file is used only by the local application server and will take effect after restarting the application server. Note that JAAS configuration in the WebSphere configuration API security document takes precedence over that defined in the `wsjaas.conf` file. In other words, a configuration entry in `wsjaas.conf` will be overridden by an entry of the same alias name in the WebSphere configuration API security document.

**Note:** The Java Authentication and Authorization Service (JAAS) login configuration entries in the Security Center are propagated to the server run time when they are created, not when the configuration is saved. However, the deleted JAAS login configuration entries are not removed from the server run time. To remove the entries, save the new configuration, then stop and restart the server.

The samples gallery provides a JAAS login sample that demonstrates how to use JAAS with WebSphere Application Server. The sample uses a server-side login with JAAS to authenticate a user with the security run time for WebSphere Application Server. The sample demonstrates the following technology:

- Java 2 Platform, Enterprise Edition (J2EE) Java Authentication and Authorization Service (JAAS)

- JAAS for WebSphere Application Server
- WebSphere Application Server security

The form login sample is component of the technology samples. For more information on how to access the form login sample, see Accessing the Samples (Samples Gallery).

## Configuration entry settings for Java Authentication and Authorization Service

Use this page to specify a list of Java Authentication and Authorization Service (JAAS) login configurations for the application code to use, including J2EE artifacts such as enterprise beans, JavaServer Pages (JSP) files, servlets, resource adapters, and message data blocks (MDBs).

To view this administrative console page, complete the following steps:

1. Click **Security > Global security**.
2. Under Authentication, click **JAAS configuration > Application logins**.

Read the JAAS documentation before you begin defining additional login modules for authenticating to the WebSphere Application Server security run time. You can define additional login configurations for your applications. However, if the WebSphere Application Server LoginModule (`com.ibm.ws.security.common.auth.module.WSLoginModuleImpl`) is not used or the LoginModule does not produce a credential that is recognized by WebSphere Application Server, then the WebSphere Application Server security run time cannot use the authenticated subject from these login configurations for an authorization check for resource access.

**Note:** You must invoke Java client programs that use Java Authentication and Authorization Service (JAAS) for authentication with a JAAS configuration file specified. The WebSphere product supplies the default JAAS configuration file, wsjaas_client.conf under the `*install_root*/properties` directory. This configuration file is set in the `/*install_root*/bin/launchClient.bat` file as: `set JAAS_LOGIN_CONFIG=-Djava.security.auth.login.config=%WAS_HOME%\properties\wsjaas_client.conf`

If `launchClient.bat` file is not used to invoke Java client programs, make sure that the appropriate JAAS configuration file is passed to the Java virtual machine with the *-Djava.security.auth.login.config* flag.

### ClientContainer

Specifies the login configuration used by the client container application, which uses the CallbackHandler API defined in the client container deployment descriptor.

The ClientContainer configuration is the default login configuration for the WebSphere Application Server. Do not remove this default, as other applications that use it fail.

| **Default:** | ClientContainer |
|---|---|

### DefaultPrincipalMapping

Specifies the login configuration used by Java 2 Connectors to map users to principals that are defined in the J2C Authentication Data Entries.

ClientContainer is the default login configuration for the WebSphere Application Server. Do not remove this default, as other applications that use it fail.

| **Default:** | ClientContainer |
|---|---|

### WSLogin

Specifies whether all applications can use the WSLogin configuration to perform authentication for the WebSphere Application Server security run time.

This login configuration does not honor the CallbackHandler defined in the client container deployment descriptor. To use this functionality, use the ClientContainer login configuration.

The WSLogin configuration is the default login configuration for the WebSphere Application Server. Do not remove this default because other administrative applications that use it will fail. This login configuration authenticates users for the WebSphere Application Server security run time. Use credentials from the authenticated subject returned from this login configurations as an authorization check for access to WebSphere Application Server resources.

| | |
|---|---|
| **Default:** | ClientContainer |

## System login configuration entry settings for Java Authentication and Authorization Service

Use this page to specify a list of Java Authentication and Authorization Service (JAAS) system login configurations.

To view this administrative console page, click **Security > Global security**. Under Authentication, click **JAAS configuration > System logins**.

Read the Java Authentication and Authorization Service documentation before you begin defining additional login modules for authenticating to the WebSphere Application Server security run time. Do not remove the following system login modules:

- RMI_INBOUND
- WEB_INBOUND
- DEFAULT
- RMI_OUTBOUND
- SWAM
- wssecurity.IDAssertion
- wssecurity.signature
- wssecurity.PKCS7
- wssecurity.PkiPath
- wssecurity.UsernameToken
- wssecurity.X509BST
- LTPA
- LTPA_WEB

### RMI_INBOUND, WEB_INBOUND, DEFAULT

Processes inbound login requests for Remote Method Invocation (RMI), Web applications, and most of the other login protocols. These login configurations are used by WebSphere Application Server Version 5.1.1 and later.

**RMI_INBOUND**
The RMI_INBOUND login configuration handles logins for inbound RMI requests. Typically, these logins are requests for authenticated access to Enterprise JavaBeans (EJB) files. Also, these logins might be Java Management Extensions (JMX) requests when using the RMI connector.

**WEB_INBOUND**
The WEB_INBOUND login configuration handles logins for Web application requests, which

includes servlets and JavaServer Pages (JSP) files. This login configuration can interact with the output that is generated from a trust association interceptor (TAI), if configured. The Subject passed into the WEB_INBOUND login configuration might contain objects generated by the TAI.

**DEFAULT**
The DEFAULT login configuration handles the logins for inbound requests made by most of the other protocols and internal authentications.

These three login configurations can pass in the following callback information, which is handled by the login modules within these configurations. These callbacks are not passed in at the same time. However, the combination of these callbacks determines how WebSphere Application Server authenticates the user.

**Callback**

```
callbacks[0] = new javax.security.auth.callback.
NameCallback("Username: ");
```

**Responsibility**
Collects the user name that is provided during a login. This information can be the user name for the following types of logins:

- User name and password login, which is known as basic authentication.
- User name only for identity assertion.

**Callback**

```
callbacks[1] = new javax.security.auth.callback.
PasswordCallback("Password: ", false);
```

**Responsibility**
Collects the password that is provided during a login.

**Callback**

```
callbacks[2] = new com.ibm.websphere.security.auth.callback.
WSCredTokenCallbackImpl("Credential Token: ");
```

**Responsibility**
Collects the Lightweight Third Party Authentication (LTPA) token (or other token type) during a login. Typically, this information is present when a user name and a password are not present.

**Callback**

```
callbacks[3] = new com.ibm.wsspi.security.auth.callback.
WSTokenHolderCallback("Authz Token List: ");
```

**Responsibility**
Collects the ArrayList of the TokenHolder objects that are returned from the call to the WSOpaqueTokenHelper. createTokenHolderListFromOpaqueToken () method using the Common Secure Interoperability version 2 (CSIv2) authorization token as input.

**Restriction:** This callback is present only when the **Security Attribute Propagation** option is enabled and this login is a propagation login. In a propagation login, sufficient security attributes are propagated with the request to prevent having to access the user registry for additional attributes.

In system login configurations, WebSphere Application Server authenticates the user based upon the information collected by the callbacks. However, a custom login module does not need to act upon any of these callbacks. The following list explains the typical combinations of these callbacks:

- The `callbacks[0] = new javax.security.auth.callback.NameCallback("Username: ");` callback only

  This callback occurs for CSIv2 Identity Assertion; Web and CSIv2 X509 certificate logins; old-style trust association interceptor logins, and so on. In Web and CSIv2 X509 certificate logins, WebSphere Application Server maps the certificate to a user name. This callback is used by any login type that establishes trust using the user name only.

- Both the `callbacks[0] = new javax.security.auth.callback.NameCallback("Username: ");` callback and the `callbacks[1] = new javax.security.auth.callback.PasswordCallback("Password: ", false);` callbacks.

  This combination of callbacks is typical for basic authentication logins. Most user authentications occur using these two callbacks.

- The `callbacks[2] = new com.ibm.websphere.security.auth.callback.WSCredTokenCallbackImpl("Credential Token: ");` only

  This callback is used to validate a Lightweight Third Party Authentication (LTPA) token. This validation typically occurs during an single signon (SSO) or downstream login. Any time a request originates from a WebSphere Application Server, instead of a pure client, the LTPA token typically flows to the target server. For single signon (SSO), the LTPA token is received in the cookie and the token is used for login. If a custom login module needs the user name from an LTPA token, the module can use the following method to retrieve the unique ID from the token:

  ```
  com.ibm.wsspi.security.token.WSSecurityPropagationHelper.
  validateLTPAToken(byte[])
  ```

  After retrieving the unique ID, use the following method to get the user name:

  ```
  com.ibm.wsspi.security.token.WSSecurityPropagationHelper.
  getUserFromUniqueID(uniqueID)
  ```

  **Important:** Any time a custom login module is plugged in ahead of the WebSphere Application Server login modules and it changes the identity using the credential mapping services, it is important that this login module validates the LTPA token, if present. Calling the following method is sufficient to validate the trust in the LTPA token:

  ```
  com.ibm.wsspi.security.token.WSSecurityPropagationHelper.
  validateLTPAToken(byte[])
  ```

  The receiving server must have the same LTPA keys as the sending server in order for this to be successful. There is a possible security exposure if you do not validate this LTPA token, when present.

- A combination of any of the previously mentioned callbacks plus the `callbacks[3] = new com.ibm.wsspi.security.auth.callback.WSTokenHolderCallback("Authz Token List: ");` callback.

  This callback indicates that some propagated attributes arrived at the server. The propagated attributes still require one of the following authentication methods:

  – `callbacks[0] = new javax.security.auth.callback.NameCallback("Username: ");`
  – `callbacks[1] = new javax.security.auth.callback.PasswordCallback("Password: ", false);`
  – `callbacks[2] = new com.ibm.websphere.security.auth.callback.WSCredTokenCallbackImpl("Credential Token: ");`

  If the attributes are added to the Subject from a pure client, then the NameCallback and PasswordCallback callbacks authenticate the information and the objects that are serialized in the token holder are added to the authenticated Subject.

  If both CSIv2 identity assertion and propagation are enabled, WebSphere Application Server uses the NameCallback and the token holder, which contains all of the propagated attributes, to deserialize most of the objects. WebSphere Application Server uses the NameCallback because trust is established with the servers that you indicate in the CSIv2 trusted server list. To specify trusted servers, complete the following steps:

  1. Click **Security > Global security**.
  2. Under Authentication, click **Authentication protocol > CSIv2 Inbound authentication**.

Custom serialization needs to be handled by a custom login module. For more information, see ″Security attribute propagation″.

In addition to the callbacks defined previously, the WEB_INBOUND login configuration only can contain the following additional callbacks

**Callback**

```
callbacks[4] = new com.ibm.websphere.security.auth.callback.
WSServletRequestCallback("HttpServletRequest: ");
```

**Responsibility**

Collects the HTTP servlet request object, if presented. This callback enables login modules to retrieve information from the HTTP request to use during a login.

**Callback**

```
callbacks[5] = new com.ibm.websphere.security.auth.callback.
WSServletResponseCallback("HttpServletResponse: ");
```

**Responsibility**

Collects the HTTP servlet response object, if presented. This callback enables login modules to add information into the HTTP response as a result of the login. For example, login modules might add the SingleSignonCookie to the response.

**Callback**

```
callbacks[6] = new com.ibm.websphere.security.auth.callback.
WSAppContextCallback("ApplicationContextCallback: ");
```

**Responsibility**

Collects the Web application context used during the login. This callback consists of a Hashtable, which contains the application name and the redirect Web address, if present.

The following login modules are predefined for the RMI_INBOUND, WEB_INBOUND, and DEFAULT system login configurations. You can add custom login modules before, between, or after any of these login modules, but you cannot remove these predefined login modules.

- com.ibm.ws.security.server.lm.ltpaLoginModule

  This login module performs the primary login when attribute propagation is either enabled or disabled. A primary login uses normal authentication information such as a user ID and password; an LTPA token; or a trust association interceptor (TAI) and a certificate distinguished name (DN). If any of the following scenarios are true, this login module is not used and the com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule module performs the primary login:

  – The java.util.Hashtable object with the required user attributes is contained in the Subject.

  – The java.util.Hashtable object with the required user attributes is present in the sharedState HashMap of the LoginContext.

  – The WSTokenHolderCallback callback is present without a specified password. If a user name and a password are present with a WSTokenHolderCallback, callback, which indicates propagated information, the request likely originates from either a pure client or a server from a different realm that mapped the existing identity to a user ID and password.

- com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule

  This login module performs the primary login using the normal authentication information if any of the following conditions are true:

  – A java.util.Hashtable object with required user attributes is contained in the Subject

  – A java.util.Hashtable object with required user attributes is present in the sharedState HashMap of the LoginContext

  – The WSTokenHolderCallback callback is present without a PasswordCallback callback.

  When the java.util.Hashtable object is present, the login module maps the object attributes into a valid Subject. When the WSTokenHolderCallback is present, the login module deserializes the byte token

objects and regenerates the serialized Subject contents. The java.util.Hashtable takes precedence over all of the other forms of login. Be careful to avoid duplicating or overriding what WebSphere Application Server might have propagated previously. By specifying a java.util.Hashtable to take precedence over other authentication information, the custom login module must have already verified the LTPA token, if present, to establish sufficient trust. The custom login module can use the com.ibm.wsspi.security.token.WSSecurityPropagationHelper.validationLTPAToken(byte[]) method to validate the LTPA token present in the WSCredTokenCallback. Failure to validate the LTPA token presents a security risk.

For more information on adding a Hashtable containing well-known and well-formed attributes used by WebSphere Application Server as sufficient login information, see ″Configuring inbound identity mapping″.

## RMI_OUTBOUND

Processes Remote Method Invocation (RMI) requests that are sent outbound to another server when either the com.ibm.CSI.rmiOutboundLoginEnabled or the com.ibm.CSIOutboundPropagationEnabled properties are true.

These properties are set in the CSIv2 authentication panel. To access the panel, click **Security > Global security**. Under Authentication protocol, click **CSIv2 Outbound authentication**. To set the com.ibm.CSI.rmiOutboundLoginEnabled property, select **Custom outbound mapping**. To set the com.ibm.CSIOutboundPropagationEnabled property, select the **Security attribute propagation** option.

This login configuration determines the security capabilities of the target server and its security domain. For example, if WebSphere Application Server Version 5.1.1 or later communicates with a version 5.x Application Server, then the Version 5.1.1 Application Server sends the authentication information only, using an LTPA token, to the Version 5.x Application Server. However, if WebSphere Application Server Version 5.1.1 or later communicates with a version 5.1.x Application Server, the authentication and authorization information is sent to the receiving application server if propagation is enabled at both the sending and receiving servers. When the application server sends both the authentication and authorization information downstream, it removes the need to re-access the user registry and look up the security attributes of the user for authorization purposes. Additionally, any custom objects added at the sending server should be present in the Subject at the downstream server.

The following callback is available to in the RMI_OUTBOUND login configuration. You can use the com.ibm.wsspi.security.csiv2.CSIv2PerformPolicy object that is returned by this callback to query the security policy for this particular outbound request. This query can help determine if the target realm is different than the current realm and if WebSphere Application Server must map the realm. For more information, see ″Configuring outbound mapping to a different target realm″.

**Callback**

```
callbacks[0] = new WSProtocolPolicyCallback(″Protocol Policy Callback: ″);
```

**Responsibility**

Provides protocol-specific policy information for the login modules on this outbound invocation. This information is used to determine the level of security, including the target realm, target security requirements, and coalesced security requirements.

The following method obtains the CSIv2PerformPolicy from this specific login module:

```
csiv2PerformPolicy = (CSIv2PerformPolicy)
((WSProtocolPolicyCallback)callbacks[0]).getProtocolPolicy();
```

A different protocol other than RMI might have a different type of policy object.

The following login module is predefined in the RMI_OUTBOUND login configuration. You can add custom login modules before, between, or after any of these login modules, but you cannot remove these predefined login modules.

**com.ibm.ws.security.lm.wsMapCSIv2OutboundLoginModule**

Retrieves the following tokens and objects before creating an opaque byte that is sent to another server by using the Common Secure Interoperability version 2 (CSIv2) authorization token layer:

- Forwardable com.ibm.wsspi.security.token.Token implementations from the Subject
- Serializable custom objects from the Subject
- Propagation tokens from the thread

You can use a custom login module prior to this login module to perform credential mapping. However, it is recommended that the login module change the contents of the Subject that is passed in during the login phase. If this recommendation is followed, the login modules processed after this login module act on the new Subject contents.

For more information, see ″ Configuring outbound mapping to a different target realm″.

## SWAM

Processes login requests in a single server environment when Simple WebSphere Authentication Mechanism (SWAM) is used as the authentication method.

SWAM does not support forwardable credentials. When SWAM is the authentication method, WebSphere Application Server cannot send requests from server to server. In this case, you must use LTPA.

## wssecurity.IDAssertion

Processes login configuration requests for Web services security using identity assertion. This login configuration is for version 5.x systems.

## wssecurity.PKCS7

This login configuration is for version 6.x systems.

## wssecurity.PkiPath

This login configuration is for version 6.x systems.

## wssecurity.signature

Processes login configuration requests for Web services security using digital signature validation. This login configuration is for version 5.x systems.

## wssecurity.UsernameToken

This login configuration is for version 6.x systems.

## wssecurity.X509BST

This login configuration is for version 6.x systems.

## LTPA_WEB

Processes login requests used by the Web container such as servlets and JavaServer pages (JSP) files.

This login configuration is used by WebSphere Application Server Version 5.1. This login configuration was introduced in version 5.1 and is no longer used in version 5.1.1.

The com.ibm.ws.security.web.AuthenLoginModule login module is predefined in the LTPA login configuration. You can add custom login modules before or after this module in the LTPA_WEB login configuration.

The LTPA_WEB login configuration can process the HttpServletRequest object, the HttpServletResponse object, and the Web application name that are passed in using a callback handler. For more information, see ″Customizing a server-side Java Authentication and Authorization Service authentication and login configuration″ in the documentation.

### LTPA

Processes login requests that are not handled by the LTPA_WEB login configuration.

This login configuration is used by WebSphere Application Server Version 5.1 and previous versions.

The com.ibm.ws.security.server.lm.ltpaLoginModule login module is predefined in the LTPA login configuration. You can add custom login modules before or after this module in the LTPA login configuration. For more information, see ″Customizing a server-side Java Authentication and Authorization Service authentication and login configuration″ in the documentation.

## Login module settings for Java Authentication and Authorization Service

Use this page to define the login module for a Java Authentication and Authorization Service (JAAS) login configuration.

You can define the JAAS login modules for application and system logins. To define these login modules in the administrative console, use one of the following paths:

- To view this administrative console page, click **Security > Global security**. Under Authentication, click **JAAS configuration > Application logins** or **System logins** > *alias_name*. Under Additional properties, click **JAAS login modules**.

### Module class name

Specifies the class name of the given login module.

| | |
|---|---|
| **Data type:** | String |

### Proxy class name

Specifies the name of the proxy login module class.

The default login modules defined by the WebSphere product use the proxy LoginModule class, com.ibm.ws.security.common.auth.module.WSLoginModuleProxy. This proxy class loads the WebSphere Application Server login module with the thread context class loader and delegates all the operations to the *real* login module implementation. The real login module implementation is specified as the delegate option in the option configuration. The proxy class is needed because the Developer Kit application class loaders do not have visibility of the WebSphere Application Server product class loaders.

| | |
|---|---|
| **Data type:** | String |

### Authentication Strategy

Specifies the authentication behavior as authentication proceeds down the list of login modules.

A Java Authentication and Authorization Service (JAAS) authentication provider supplies the authentication strategy. In JAAS, an authentication strategy is implemented through the LoginModule interface.

| | |
|---|---|
| **Data type:** | String |
| **Default:** | Required |
| **Range:** | Required, Requisite, Sufficient and Optional |

**Required**
: The LoginModule is required to succeed. If it succeeds or fails, authentication still continues to proceed down the LoginModule list for each realm.

**Requisite**
: The LoginModule is required to succeed. If it succeeds, authentication continues down the

LoginModule list in the realm entry. If it fails, control immediately returns to the application--that is, authentication does not proceed down the LoginModule list.

**Sufficient**
The LoginModule is not required to succeed. If it does succeed, control immediately returns to the application--again, authentication does not proceed down the LoginModule list. If it fails, authentication continues down the list.

**Optional**
The LoginModule is not required to succeed. If it succeeds or fails, authentication still continues to proceed down the LoginModule list.

Specify additional options by clicking **Custom Properties** under Additional Properties. These name and value pairs are passed to the login modules during initialization. This process is one of the mechanisms that is used to passed information to login modules.

### Module order

Specifies the order in which the Java Authentication and Authorization Service (JAAS) login modules are processed.

Click **Set Order** to change the processing order of the login modules.

## Login module order settings for Java Authentication and Authorization Service

Use this page to specify the order in which WebSphere Application Server processes the login configuration modules.

You can specify the order of the login modules for application and system logins. To define these login modules in the administrative console, complete the following steps:

1. Click **Security > Global security**.
2. Under Authentication, click **JAAS Configuration > Application logins** or **System logins** > *login_configuration*. You can create a new configuration by clicking **New**.
3. Under Additional properties, click **JAAS login modules**.
4. Click **Set order**.

When you select one of the JAAS login module class names, you can move that class name up and down the list. After you press **OK** and save the changes, the new order is reflected on either the Application login configuration or System login configuration panel.

## Login configuration settings for Java Authentication and Authorization Service

Use this page to configure application login configurations.

To view this administrative console page, click **Security > Global security**. Under Authentication, click **JAAS configuration > Application logins** or **System logins** > *alias_name*.

Click **Apply** to save changes and to add the extra node name that precedes the original alias name. Clicking **OK** does not save the new changes in the `security.xml` file.

### Alias

Specifies the alias name of the application login.

Do not use the forward slash character (/) in the alias name when defining JAAS login configuration entries. The JAAS login configuration parser cannot process the forward slash character.

**Data type:** String

## J2EE Connector security

The J2EE connector architecture defines a standard architecture for connecting the Java 2 Platform, Enterprise Edition (J2EE) to heterogeneous enterprise information systems (EIS). Examples of EIS include Enterprise Resource Planning (ERP), mainframe transaction processing (TP) and database systems.

The connector architecture enables an EIS vendor to provide a standard *resource adapter* for its EIS. A *resource adapter* is a system-level software driver that is used by a Java application to connect to an EIS. The resource adapter plugs into an application server and provides connectivity between the EIS, the application server, and the enterprise application. Accessing information in EIS typically requires access control to prevent unauthorized accesses. J2EE applications must authenticate to the EIS to open a connection to it.

The J2EE Connector security architecture is designed to extend the end-to-end security model for J2EE-based applications to include integration with EISs. An application server and an EIS collaborate to ensure the proper authentication of a resource principal, which establishes a connection to an underlying EIS. The connector architecture identifies the following mechanisms as the commonly-supported authentication mechanisms although other mechanisms can be defined:

- BasicPassword: Basic user-password-based authentication mechanism that is specific to an EIS
- Kerbv5: Kerberos Version 5-based authentication mechanism

Applications define whether to use application-managed sign-on or container-managed sign-on in the resource-ref elements in the deployment descriptor. Each resource-ref element describes a single connection factory reference binding. The `res-auth` element in a resource-ref element, whose value is either Application or Container, indicates whether the enterprise bean code should perform sign-on or whether it should enable the WebSphere Application Server to sign-on to the resource manager using the principal mapping configuration. The resource-ref element is defined at application assembly time. Use the WebSphere Development Toolkit to configure the resource `-ref`.

**Application managed sign-on**

To access an EIS system, applications locate a connection factory from the JNDI namespace and invoke the getConnection method on that connection factory object. The getConnection method might require a user ID and password argument. A J2EE application can pass in a user ID and password to getConnection, which subsequently passes the information to the resource adapter. Specifying a user ID and password in the application code has some security implications, however.

The user ID and password, if coded into the Java source code, are available to developers and testers in the organization. Also, the user ID and password are visible to users if they de-compile the Java class.

The user ID and password cannot be changed without first requiring a synchronized code change. Alternatively, application code might retrieve sets of user IDs and passwords from persistent storage or from an external service. This approach requires that IT administrators configure and manage a user ID and password using the application-specific mechanism.

WebSphere Application Server allows a component-managed authentication alias to be specified on a resource. This authentication data is common to all references to the resource. On the **Resource Adapter**>**Connection Factory** configuration panel, select **component-managed authentication alias**.

With res-auth=Application, the authentication data is taken from, in order:

1. user id and password passed to getConnection(...)
2. component-managed auth alias on the Connection Factory or DataSource
3. Custom Properties UserName and Password on the DataSource

The username and password properties can be initially defined in the RAR file, and can also be defined in the admininstrative console or wsadmin scripting under custom properties. Do not use the custom properties, which enable users to connect to the resources.

**Container-managed sign-on**

The user ID and password for the target EIS can be supplied by the application server. WebSphere Application Server provides container-managed sign-on functionality. It locates the proper authentication data for the target EIS to enable the client to establish a connection. Application code does not have to provide a user ID and password in the getConnection call when it is configured to use container-managed sign-on, nor does authentication data have to be common to all references to a resource. WebSphere Application Server uses a Java Authentication and Authorization Service (JAAS) pluggable authentication mechanism to use a pre-configured JAAS login configuration, and LoginModule(s) to map a client security identity and credentials on the thread of execution to a pre-configured user ID and password.

WebSphere Application Server ships a default many-to-one credential mapping LoginModule that maps any client identity on the thread of execution to a pre-configured user ID and password for a specified target EIS. The default mapping module is a special purpose JAAS LoginModule that returns a PasswordCredential specified by the configured J2C authentication data entry. The default mapping LoginModule performs a table lookup, but does not perform actual authentication. The user ID and password are stored together with an alias in the J2C Authentication data list. The J2C Authentication data list is located on the Global Security panel under **Authentication** > **JAAS Configuration**. The default principal and credential mapping function is defined by the DefaultPrincipalMapping application JAAS login configuration.

The DefaultPrincipalMapping login configuration should not be modified since WebSphere Application Server added performance enhancements to this frequently used default mapping configuration. WebSphere Application Server does not support modifying the DefaultPrincipalMapping configuration, changing the default LoginModule, or stacking a custom LoginModule in the configuration.

For most systems, the default configuration with a many-to-one mapping is sufficient. However, WebSphere Application Server does support custom principal and credential mapping configurations. Custom mapping modules can be added to the application logins JAAS configuration by creating a new JAAS login configuration with a unique name. For example, a custom mapping module can provide one-to-one mapping or Kerberos functionality.

You also can use the WebSphere Application Server administrative console to bind the resource manager connection factory references to one of the configured resource factories. If the value of the res-auth element is Container, you must configure the mapping configuration using the **Map resource references to resources** link on an enterprise application panel.

**Map resource references to references**

To map resource references to resources, do the following:

1. Click **Applications** > **Enterprise Applications**.
2. Select an application.
3. Under Additional Properties, select **Map resource references to resources**.
4. Select a connection factory reference binding from the table that has a login configuration of Resource authorization: Container. You must specify an authentication method for the selected connection factory reference binding. Choose either **Use default method** or **Use custom login configuration**. If you choose the **Use default method** option, the WebSphere Application Server DefaultPrincipalMapping login configuration is selected. You must select an authentication data alias from the drop-down list.
5. After you make a selection, click **Apply** for the configuration to take effect.

6. If you choose **Use custom login configuration**, you must select a mapping JAAS login configuration from the drop-down list.
7. Click **Apply**. The selected login configuration name and an **Update** button appear in the login configuration field of the particular connection factory reference binding.
8. Click **Update** to define mapping properties that you might need to pass to the mapping LoginModule(s).

### J2C mapping modules and mapping properties

Mapping modules are special JAAS login modules that provide principal and credential mapping functionality. You can define and configure custom mapping modules using the administrative console.

You also can define and pass context data to mapping modules by using login options in each JAAS login configuration. In WebSphere Application Server Version 6, you also can define context data using mapping properties on each connection factory reference binding.

Login options that are defined under each JAAS login configuration are shared among all resources that use the same JAAS login configuration and mapping modules. Mapping properties that are defined for each connection factory reference binding are used exclusively by that resource reference.

Consider a usage scenario where an external mapping service is used, (such the as Tivoli Access Manager Global Sign-On (GSO) service). You have two EIS servers: DB2 and MQ.

Use the Tivoli Access Manager GSO to locate authentication data for both backend servers. The authentication data for DB2 is different from that for MQ, however. Use the login option in a mapping JAAS login configuration to specify the parameters that are required to establish a connection to the TAM GSO service. Use the mapping properties in a connection factory reference binding to specify which EIS server the user ID and password are required for.

For more detailed information about developing a mapping module, see the Developing your own Java 2 security mapping module article.

**Note:**

- WebSphere Application Server Version 6 configures container-managed sign-on under each enterprise application. This is different than WebSphere Application Server Version 5, which configures container-managed sign-on for each connection factory.
- The deprecated way of configuration at the **Resource Adapter** > **Connection factory** panel still works in WebSphere Application Server Version 6. The advantage to configuring at the connection factory reference level is that the configuration has application scope and is not visible to other applications. However, the mapping configuration defined at the connection factory is visible to other applications.
- The mapping configuration at the connection factory has moved to the resource manager connection factory reference. The mapping LoginModules that were developed using WebSphere Application Server Version 5 JAAS Callback types can be used by the resource manager connection factory reference, but the mapping LoginModules cannot take advantage of the custom mapping properties feature.
- Connection factory reference binding supports mapping properties, and passes those properties to mapping LoginModules by way of a new WSMappingPropertiesCallback Callback type. In addition, WSMappingPropertiesCallback and the new WSManagedConnectionFactoryCallback are defined in the com.ibm.wsspi package. New mapping LoginModules should use the new Callback types.

## Managing J2EE Connector Architecture authentication data entries

Java 2 Platform, Enterprise Edition (J2EE) Connector authentication data entries are used by resource adapters and Java DataBase Connectivity (JDBC) data sources. A J2EE Connector authentication data entry contains authentication data, which includes the following information:

**Alias**
An identifier that identifies the authenticated data entry. When configuring resource adapters or data sources, the administrator can specify which authentication data to choose using the corresponding alias.

**User ID**
A user identity of the intended security domain. For example, if a particular authentication data entry is used to open a new connection to DB2, this entry contains a DB2 user identity.

**Password**
The password of the user identity is encoded in the configuration respository.

**Description**
A short text description.

This task creates and deletes Java 2 Connector (J2C) authentication data entries.

1. Delete a J2C authentication data entry.
   a. Click **Security** in the navigation tree and select **Global Security**, and then click **JAAS Configuration** > **J2C Authentication Data**. The **J2C Authentication Data Entries** panel is displayed.
   b. Select the check boxes for the entries to delete and click **Delete**. Before deleting or removing an authentication data entry, make sure that it is not used or referenced by any resource adapter or data source. If the deleted authentication data entry is used or referenced by a resource, the application that uses the resource adapter or the data source fails to connect to the resources.
2. Create a new J2C authentication data entry.
   a. Click **Security** in the navigation tree and select **Global Security**, and then click **JAAS Configuration** > **J2C Authentication Data**. The **J2C Authentication Data Entries** panel is displayed.
   b. Click **New**.
   c. Enter a unique alias, a valid user ID, a valid password, and a short description (optional).
   d. Click **OK** or **Apply**. No validation for the user ID and password is required.
   e. Click **Save**. For a Network Deployment installation, make sure that a file synchronized operation is performed to propagate the changes to other nodes.

A new J2C authentication data entry is created or an old entry is removed. The newly created entry is visible without restarting the application server process to use in the data source definition. But the entry is only in effect after the server is restarted. Specifically, the authentication data is loaded by an application server when starting an application and is shared among applications in the same application server.

If you create or update a data source that points to a newly created J2C authentication data alias, the test connection fails to connect until you restart the deployment manager. After you restart the deployment manager, the J2C authentication data is reflected in the run-time configuration. Any changes to the J2C authentication data fields require a deployment manager restart for the changes to take effect.

This step defines authentication data that you can share among resource adapters and data sources. Use the authentication data entry that is defined in the resource adapters or the data sources.

### Java 2 Connector authentication data entry settings

Use this page as a central place for administrators to define authentication data, which includes user identities and passwords. These values can reference authentication data entries by resource adapters, data sources, and other configurations that require authentication data using an alias.

You can display this page directly from the JAAS configuration page or from other pages for resources that use J2EE Connector (J2C) authentication data entries. For example, to view this administrative page, you can click either **Security > Global security**. Under Authentication, click **JAAS configuration > J2C authentication data**.

**Deleting authentication data entries:** Be careful when deleting authentication data entries. If the deleted authentication data is used by other configurations, the initializing resources process fails.

Define a new authentication data entry by clicking **New**.

***Alias:***

Specifies the name of the authentication data entry.

| | |
|---|---|
| **Data type:** | String |
| **Units:** | String |
| **Default:** | None |

***User ID:***

Specifies the user identity.

| | |
|---|---|
| **Data type:** | String |

***Description:***

Specifies an optional description of the authentication data entry. For example, this authentication data entry is used to connect to DB2.

| | |
|---|---|
| **Data type:** | String |

## Identity mapping

*Identity mapping* is a one-to-one mapping of a user identity between two servers so that the proper authorization decisions are made by downstream servers. Identity mapping is necessary when the integration of servers is needed, but the user registries are different and not shared between the systems.

In most cases, requests flow downstream between two servers that are part of the same security domain. In WebSphere Application Server, two servers that are members of the same cell are also members of the same security domain. In the same cell, the two servers have the same user registry and the same Lightweight Third Party Authentication (LTPA) keys for token encryption. These two commonalities ensure that the LTPA token (among other user attributes), which flows between the two servers, not only can be decrypted and validated, but also the user identity in the token can be mapped to attributes that are recognized by the authorization engine.

The most reliable and recommended configuration involves two servers within the same cell. However, sometimes you need to integrate multiple systems that cannot use the same user registry. When the user registries are different between two servers, the security domain or realm of the target server does not match the security domain of the sending server.

WebSphere Application Server enables mapping to occur either before sending the request outbound or before enabling the existing security credentials to flow to the target server as-is. The credentials are mapped inbound with the specification that the target realm is trusted.

An alternative to mapping is to send the user identity without the token or the password to a target server without actually mapping the identity. The use of the user identity is based on trust between the two servers. Use Common Secure Interoperability version 2 (CSIv2) identity assertion. When enabled, it sends just the X.509 certificate, principal name, or distinguished name (DN) based upon what was used by the original client to perform the initial authentication. During CSIv2 identity assertion, trust is established between the WebSphere Application Servers.

The user identity must exist in the target user registry for identity assertion to work. This process can also enable interoperability between other Java 2 Platform, Enterprise Edition (J2EE) Version 1.3 and higher compliant application servers. When using identity assertion, if both the sending server and target servers have identity assertion configured, WebSphere Application Server always uses this method of authentication, even when both servers are in the same security domain. For more information on CSIv2 identity assertion, see "Identity assertion" on page 382.

When the user identity is not present in the user registry of the target server, identity mapping must occur either before the request is sent outbound or when the request comes inbound. This decision depends upon your environment and requirements. However, it is typically easier to map the user identity before the request is sent outbound for the following reasons:

- You know the user identity of the existing credential as it comes from the user registry of the sending server.
- You do not have to worry about sharing Lightweight Third Party Authentication (LTPA) keys with the other target realm because you are not mapping the identity to LTPA credentials. Typically, you are mapping the identity to a user ID and password that are present in the user registry of the target realm.

When you do perform outbound mapping, in most cases, it is recommended that you use Secure Sockets Layer (SSL) to protect the integrity and confidentiality of the security information sent across the network. If LTPA keys are not shared between servers, an LTPA token cannot be validated at the inbound server. In this case, outbound mapping is necessary because the user identity can not be determined at the inbound server to do inbound mapping. For more information, see "Configuring outbound mapping to a different target realm" on page 270.

When you need inbound mapping, potentially due to the mapping capabilities of the inbound server, you must ensure that both servers have the same LTPA keys so that you can get access to the user identity. Typically, in secure communications between servers, an LTPA token is passed into the WSCredTokenCallback of the inbound JAAS login configuration for the purposes of client authentication. A method is available that enables you to open the LTPA token, if valid, and get access to the user unique ID so that mapping can be performed. For more information, see "Configuring inbound identity mapping." In other cases, such as identity assertion, you might receive a user name in the NameCallback of the inbound login configuration that enables you to map the identity.

## Configuring inbound identity mapping

For inbound identity mapping, it is recommend that you write a custom login module and configure WebSphere Application Server to run the login module first within the system login configurations. Consider the following steps when you write your custom login module:

1. Get the inbound user identity from the callbacks and map the identity, if necessary This step occurs in the login() method of the login module. A valid authentication has either or both of the following callbacks present: NameCallback and the WSCredTokenCallback. The following code sample shows you how to determine the user identity:

```
javax.security.auth.callback.Callback callbacks[] =
   new javax.security.auth.callback.Callback[3];
 callbacks[0] = new javax.security.auth.callback.NameCallback("");
 callbacks[1] = new javax.security.auth.callback.PasswordCallback
     ("Password: ", false);
```

```
callbacks[2] = new com.ibm.websphere.security.auth.callback.
    WSCredTokenCallbackImpl("");
callbacks[3] = new com.ibm.wsspi.security.auth.callback.
    WSTokenHolderCallback("");

try
{
 callbackHandler.handle(callbacks);
}
catch (Exception e)
{
 // Handles exceptions
 throw new WSLoginFailedException (e.getMessage(), e);
}

// Shows which callbacks contain information
boolean identitySwitched = false;
String uid = ((NameCallback) callbacks[0]).getName();
char password[] = ((PasswordCallback) callbacks[1]).getPassword();
byte[] credToken = ((WSCredTokenCallbackImpl) callbacks[2]).getCredToken();
java.util.List authzTokenList = ((WSTokenHolderCallback)
    callbacks[3]).getTokenHolderList();

if (credToken != null)
{
 try
 {
  String uniqueID = WSSecurityPropagationHelper.validateLTPAToken(credToken);
  String realm = WSSecurityPropagationHelper.getRealmFromUniqueID (uniqueID);
      // Now set the string to the UID so that you can use the result for either
      // mapping or logging in.
  uid = WSSecurityPropagationHelper.getUserFromUniqueID (uniqueID);
 }
 catch (Exception e)
 {
  // Handles the exception
 }
}
else if (uid == null)
{
    // Throws an except if invalid authentication data exists.
    // You must have either UID or CredToken
 throw new WSLoginFailedException("invalid authentication data.");
}
else if (uid != null && password != null)
{
    // This is a typical authentication. You can choose to map this ID to
    // another ID or you can skip it and allow WebSphere Application Server
    // to login for you. When passwords are presented, be very careful to not
    // validate the password because this is the initial authentication.

 return true;
}

   // If desired, map this uid to something else and set the identitySwitched
   // boolean. If the identity was changed, clear the propagated attributes
```

```
    // below so they are not used incorrectly.
 uid = myCustomMappingRoutine (uid);

    // Clear the propagated attributes because they no longer applicable to the
    // new identity
 if (identitySwitched)
 {
  ((WSTokenHolderCallback) callbacks[3]).setTokenHolderList(null);
 }
```

2. Check to see if attribute propagation occurred and if the attributes for the user are already present when the identity remains the same. Check to see if the user attributes are already present from the sending server to avoid duplicate calls to the user registry lookup. To check for the user attributes, use a method on the WSTokenHolderCallback that analyzes the information present in the callback to determine if the information is sufficient for WebSphere Application Server to create a Subject. The following code sample checks for the user attributes:

   ```
   boolean requiresLogin =
   ((com.ibm.wsspi.security.auth.callback.WSTokenHolderCallback)
   callbacks[2]).requiresLogin();
   ```

   If sufficient attributes are not present to form the WSCredential and WSPrincipal objects needed to perform authorization, the previous code sample returns a true result. When the result is false, you can choose to discontinue processing as the necessary information exists to create the Subject without performing additional remote user registry calls.
3. **Optional:** Look up the required attributes from the user registry, put the attributes in hashtable, and add the hashtable to the shared state. If the identity is switched in this login module, you must complete the following steps:
   a. Create the hashtable of attributes as shown in the following example.
   b. Add the hashtable to shared state.

   If the identity is not switched, but the value of the `requiresLogin` code sample shown previously is true, you can create the hashtable of attributes. However, you are not required to create a hashtable in this situation as WebSphere Application Server handles the login for you. However, you might consider creating a hashtable to gather attributes in special cases where you are using your own special user registry. Creating a UserRegistry implementation, using a hashtable, and letting WebSphere Application Server gather the user attributes for you might be the easiest solution. The following table shows how to create a hashtable of user attributes:

   ```
   if (requiresLogin || identitySwitched)
    {
     // Retrives the default InitialContext for this server.
     javax.naming.InitialContext ctx = new javax.naming.InitialContext();

     // Retrieves the local UserRegistry implementation.
     com.ibm.websphere.security.UserRegistry reg = (com.ibm.websphere.
           security.UserRegistry)
     ctx.lookup("UserRegistry");

        // Retrieves the user registry uniqueID based on the uid specified
        // in the NameCallback.
     String uniqueid = reg.getUniqueUserId(uid);
      uid = WSSecurityPropagationHelper.getUserFromUniqueID (uniqueID);

        // Retrieves the display name from the user registry based on the uniqueID.
     String securityName = reg.getUserSecurityName(uid);
   ```

```
   // Retrieves the groups associated with the uniqueID.
java.util.List groupList = reg.getUniqueGroupIds(uid);

   // Creates the java.util.Hashtable with the information that you gathered
   // from the UserRegistry implementation.
java.util.Hashtable hashtable = new java.util.Hashtable();
hashtable.put(com.ibm.wsspi.security.token.AttributeNameConstants.
     WSCREDENTIAL_UNIQUEID, uniqueid);
   hashtable.put(com.ibm.wsspi.security.token.AttributeNameConstants.
     WSCREDENTIAL_SECURITYNAME, securityName);
hashtable.put(com.ibm.wsspi.security.token.AttributeNameConstants.
     WSCREDENTIAL_GROUPS, groupList);

   // Adds a cache key that is used as part of the look up mechanism for
   // the created Subject. The cache key can be an object, but should have
   // an implemented toString() method. Make sure that the cacheKey contains
   // enough information to scope it to the user and any additional attributes
   // that you are using. If you do not specify this property the Subject is
   // scoped to the returned WSCREDENTIAL_UNIQUEID, by default.
hashtable.put(com.ibm.wsspi.security.token.AttributeNameConstants.
      WSCREDENTIAL_CACHE_KEY, "myCustomAttribute" + uniqueid);
// Adds the hashtable to the sharedState of the Subject.
_sharedState.put(com.ibm.wsspi.security.token.AttributeNameConstants.
      WSCREDENTIAL_PROPERTIES_KEY, hashtable);
}
```

The following rules define in more detail how a hashtable login is performed. You must use a java.util.Hashtable object in either the Subject (public or private credential set) or shared state HashMap. The com.ibm.wsspi.security.token.AttributeNameConstants class defines the keys that contain the user information. If the hashtable object is put into the shared state of the login context using a custom login module that is listed prior to the Lightweight Third Party Authentication (LTPA) login module, the value of the java.util.Hashtable object is searched using the following key within the shared state hashMap:

**Property**
: com.ibm.wsspi.security.cred.propertiesObject

**Reference to the property**
: AttributeNameConstants.WSCREDENTIAL_PROPERTIES_KEY

**Explanation**
: This key searches for the hashtable object that contains the required properties in sharedState of the login context.

**Expected result**
: A java.util.Hashtable object.

If a java.util.Hashtable object is found either inside the Subject or within the sharedState area, verify that the following properties are present in the hashtable:

**Property**
: com.ibm.wsspi.security.cred.uniqueId

**Reference to the property**
: AttributeNameConstants.WSCREDENTIAL_UNIQUEID

**Returns**
: java.util.String

**Explanation**
The value of the property must be a unique representation of the user. For the WebSphere Application Server default implementation, this property represents the information that is stored in the application authorization table. The information is located in the application deployment descriptor after it is deployed and user-to-role mapping is performed. See the expected format examples if the user to role mapping is performed using a lookup to a WebSphere Application Server user registry implementation. If a third-party authorization provider overrides the user to role mapping, then the third-party authorization provider defines the format. To ensure compatibility with the WebSphere Application Server default implementation for the unique ID value, call the WebSphere Application Server `public String getUniqueUserId(String userSecurityName)` UserRegistry method.

**Expected format examples**

| Realm | Format (uniqueUserId) |
|---|---|
| Lightweight Directory Access Protocol (LDAP) | `ldaphost.austin.ibm.com:389/cn=user,o=ibm,c=us` |
| Windows | `MYWINHOST/S-1-5-21-963918322-163748893-4247568029-500` |
| UNIX | `MYUNIXHOST/32` |

The com.ibm.wsspi.security.cred.uniqueId property is required.

**Property**
com.ibm.wsspi.security.cred.securityName

**Reference to the property**
AttributeNameConstants. WSCREDENTIAL_ SECURITYNAME

**Returns**
java.util.String

**Explanation**
This property searches for the securityName of the authentication user. This name is commonly called the display name or short name. WebSphere Application Server uses the securityName attribute for the getRemoteUser(), getUserPrincipal() and getCallerPrincipal() application programming interfaces (APIs). To ensure compatibility with the WebSphere Application Server default implementation for the securityName value, call the WebSphere Application Server `public String getUserSecurityName(String uniqueUserId)` UserRegistry method.

**Expected format examples**

| Realm | Format (uniqueUserId) |
|---|---|
| LDAP | user (*LDAP UID*) |
| Windows | user (*Windows username*) |
| UNIX | user (*UNIX username*) |

The com.ibm.wsspi.security.cred.securityName property is required.

**Property**
com.ibm.wsspi.security.cred.groups

**Reference to the property**
AttributeNameConstants. WSCREDENTIAL_GROUPS

**Returns**
java.util.ArrayList

**Explanation**

This key searches for the ArrayList of groups to which the user belongs. The groups are specified in the `realm_name/user_name format`. The format of these groups is important as the groups are used by the WebSphere Application Server authorization engine for group-to-role mappings in the deployment descriptor. The format provided must match the format expected by the WebSphere Application Server default implementation. When you use a third-party authorization provider, you must use the format expected by the third-party provider. To ensure compatibility with the WebSphere Application Server default implementation for the unique group IDs value, call the WebSphere Application Server `public List getUniqueGroupIds(String uniqueUserId)` UserRegistry method.

**Expected format examples for each group in the ArrayList**

| Realm | Format |
|---|---|
| LDAP | `ldap1.austin.ibm.com:389/cn=group1,o=ibm,c=us` |
| Windows | `MYWINREALM/S-1-5-32-544` |
| UNIX | `MY/S-1-5-32-544` |

The com.ibm.wsspi.security.cred.groups property is not required. A user is not required to have associated groups.

**Property**

com.ibm.wsspi.security.cred.cacheKey

**Reference to the property**

AttributeNameConstants. WSCREDENTIAL_CACHE_KEY

**Returns**

java.lang.Object

**Explanation**

This key property can specify an Object that represents the unique properties of the login including the user-specific information and the user dynamic attributes that might affect uniqueness. For example, when the user logs in from location A, which might affect their access control, the cacheKey needs to include location A so that the Subject received is the correct Subject for the current location.

This com.ibm.wsspi.security.cred.cacheKey property is not required. When this property is not specified, the cache lookup is the value specified for WSCREDENTIAL_UNIQUEID. When this information is found in the java.util.Hashtable object, WebSphere Application Server creates a Subject similar to the Subject that goes through the normal login process (at least for LTPA). The new Subject contains a WSCredential object and a WSPrincipal object that is fully populated with the information found in the Hashtable object.

4. Add your custom login module into the RMI_INBOUND, WEB_INBOUND, and DEFAULT Java Authentication and Authorization Service (JAAS) system login configurations. Configure the RMI_INBOUND login configuration so that WebSphere Application Server loads your new custom login module first.
   a. Click **Security > Global security**.
   b. Under Authentication, click **JAAS configuration > System logins > RMI_INBOUND**
   c. Under Additional Properties, click **JAAS login modules > New** to add your login module to the RMI_INBOUND configuration.
   d. Return to the JAAS login modules panel for RMI_INBOUND and click **Set order** to change the order that the login modules are loaded so that WebSphere Application Server loads your custom login module first.
   e. Repeat the previous three steps for the WEB_INBOUND and DEFAULT login configurations.

This process configures identity mapping for an inbound request.

The “Example: Custom login module for inbound mapping” article shows a custom login module that creates a java.util.Hashtable based on the specified NameCallback. The java.util.Hashtable is added to the sharedState java.util.Map so that the WebSphere Application Server login modules can locate the information in the Hashtable.

## Example: Custom login module for inbound mapping

This sample shows a custom login module that creates a java.util.Hashtable hastable that is based on the specified NameCallback callback. The java.util.Hashtable hash table is added to the sharedState java.util.Map so that the WebSphere Application Server login modules can locate the information in the Hashtable.

```
public customLoginModule()
{

public void initialize(Subject subject, CallbackHandler callbackHandler,
   Map sharedState, Map options)
{
 // (For more information on initialization, see
 //  "Custom login module development for a system login configuration" on page 78.)
 _sharedState = sharedState;
}

public boolean login() throws LoginException
{
 // (For more information on what to do during login, see
 //  "Custom login module development for a system login configuration" on page 78.)

   // Handles the WSTokenHolderCallback to see if this is an initial or
   // propagation login.
 javax.security.auth.callback.Callback callbacks[] =
     new javax.security.auth.callback.Callback[3];
 callbacks[0] = new javax.security.auth.callback.NameCallback("");
 callbacks[1] = new javax.security.auth.callback.PasswordCallback(
     "Password: ", false);
 callbacks[2] = new com.ibm.websphere.security.auth.callback.
     WSCredTokenCallbackImpl("");
 callbacks[3] = new com.ibm.wsspi.security.auth.callback.
     WSTokenHolderCallback("");

 try
 {
  callbackHandler.handle(callbacks);
 }
 catch (Exception e)
 {
  // Handles the exception
 }

 // Determines which callbacks contain information
 boolean identitySwitched = false;
 String uid = ((NameCallback) callbacks[0]).getName();
 char password[] = ((PasswordCallback) callbacks[1]).getPassword();
 byte[] credToken = ((WSCredTokenCallbackImpl) callbacks[2]).getCredToken();
 java.util.List authzTokenList = ((WSTokenHolderCallback) callbacks[3]).
     getTokenHolderList();
```

```
if (credToken != null)
{
 try
 {
  String uniqueID = WSSecurityPropagationHelper.validateLTPAToken(credToken);
  String realm = WSSecurityPropagationHelper.getRealmFromUniqueID (uniqueID);
      // Set the string to the UID so you can use the information to either
      // map or login.
  uid = WSSecurityPropagationHelper.getUserFromUniqueID (uniqueID);
 }
 catch (Exception e)
 {
  // handle exception
 }
}
else if (uid == null)
{
 // Invalid authentication data. You must have either UID or CredToken
 throw new WSLoginFailedException("invalid authentication data.");
}
else if (uid != null && password != null)
{
    // This is a typical authentication. You can choose to map this ID to
    // another ID or you can skip it and allow WebSphere Application Server
    // to login for you. When passwords are presented, be very careful not to
    // validate the password because this is the initial authentication.

 return true;
}

 // If desired, map this uid to something else and set the identitySwitched
 // boolean. If the identity is changed, clear the propagated attributes below
 // so they are not used incorrectly.
uid = myCustomMappingRoutine (uid);

// Clear the propagated attributes because they no longer apply to the new identity
if (identitySwitched)
{
 ((WSTokenHolderCallback) callbacks[3]).setTokenHolderList(null);
}
boolean requiresLogin = ((com.ibm.wsspi.security.auth.callback.
    WSTokenHolderCallback) callbacks[2]).requiresLogin();

if (requiresLogin || identitySwitched)
{
 // Retrieves the default InitialContext for this server.
 javax.naming.InitialContext ctx = new javax.naming.InitialContext();

 // Retrieves the local UserRegistry object.
 com.ibm.websphere.security.UserRegistry reg =
        (com.ibm.websphere.security.UserRegistry) ctx.lookup("UserRegistry");

 // Retrieves the registry uniqueID based on the uid that is specified
    // in the NameCallback.
 String uniqueid = reg.getUniqueUserId(uid);
```

```
   uid = WSSecurityPropagationHelper.getUserFromUniqueID (uniqueID);

  // Retrieves the display name from the user registry based on the uniqueID.
  String securityName = reg.getUserSecurityName(uid);

  // Retrieves the groups associated with this uniqueID.
  java.util.List groupList = reg.getUniqueGroupIds(uid);

  // Creates the java.util.Hashtable with the information that you gathered
     // from the UserRegistry.
  java.util.Hashtable hashtable = new java.util.Hashtable();
  hashtable.put(com.ibm.wsspi.security.token.AttributeNameConstants.
       WSCREDENTIAL_UNIQUEID, uniqueid);
     hashtable.put(com.ibm.wsspi.security.token.AttributeNameConstants.
       WSCREDENTIAL_SECURITYNAME, securityName);
  hashtable.put(com.ibm.wsspi.security.token.AttributeNameConstants.
       WSCREDENTIAL_GROUPS, groupList);

  // Adds a cache key that is used as part of the look up mechanism for
  // the created Subject. The cache key can be an object, but should have
  // an implemented toString() method. Make sure the cacheKey contains enough
  // information to scope it to the user and any additional attributes you are
     // using. If you do not specify this property, the Subject is scoped to the
  // WSCREDENTIAL_UNIQUEID returned, by default.
  hashtable.put(com.ibm.wsspi.security.token.AttributeNameConstants.
       WSCREDENTIAL_CACHE_KEY, "myCustomAttribute" + uniqueid);
  // Adds the hashtable to the sharedState of the Subject.
  _sharedState.put(com.ibm.wsspi.security.token.AttributeNameConstants.
       WSCREDENTIAL_PROPERTIES_KEY, hashtable);
 }
 else if (requiresLogin == false)
 {
  // For more information on this section, see
     // "Security attribute propagation" on page 275.
  // If you added a custom Token implementation, you can search through the
     // token holder list for it to deserialize.
  // Note: Any Java objects are automatically deserialized by
     // wsMapDefaultInboundLoginModule

  for (int i=0; i<authzTokenList.size(); i++)
  {
   if (authzTokenList[i].getName().equals("com.acme.MyCustomTokenImpl")
   {
    byte[] myTokenBytes = authzTokenList[i].getBytes();

          // Passes these bytes into the constructor of your implementation
          // class for deserialization.
    com.acme.MyCustomTokenImpl myTokenImpl =
          new com.acme.MyCustomTokenImpl(myTokenBytes);
   }
  }
 }
}

public boolean commit() throws LoginException
{
```

```
 // (For more information on what to do during a commit, see
  //   "Custom login module development for a system login configuration" on page 78.)

 // Not doing anything here for this specific example
}

// Defines your login module variables
com.ibm.wsspi.security.token.AuthorizationToken customAuthzToken = null;
com.ibm.wsspi.security.token.AuthenticationToken defaultAuthToken = null;
java.util.Map _sharedState = null;
}
```

## Configuring outbound mapping to a different target realm

By default, when WebSphere Application Server makes an outbound request from one server to another server in a different security realm, the request is rejected. This request is rejected to protect against a rogue server reading potentially sensitive information if successfully impersonating the home of the object. The following alternatives are available to enable one server to send outbound requests to a target server in a different realm:

- Do not perform mapping, instead, allow the existing security information to flow to a trusted target server even if the target server resides in a different realm. To allow information to flow to a server in a different realm, complete the following steps in the administrative console:
  1. Click **Security > Global security**.
  2. 
  3. Specify the target realms in the **Trusted target realms** field. You can specify each trusted target realm that is separated by a pipe (|) character. For example, specify *server_name.domain:port_number* for a Lightweight Directory Access Protocol (LDAP) server or the machine name for Local OS. If you want to propagate security attributes to a different target realm, you must specify that target realm in the **Trusted target realms** field.
- Use the Java Authentication and Authorization Service (JAAS) WSLogin application login configuration to create a basic authentication Subject that contains the credentials of the new target realm. This configuration enables you to log in with a realm, user ID, and password that are specific to the user registry of the target realm. You can provide the login information from within the Java 2 Platform, Enterprise Edition (J2EE) application that is making the outbound request or from within the RMI_OUTBOUND system login configuration. These two login options are described in the following information:
  1. Use the WSLogin application login configuration from within the J2EE application to log in and get a Subject that contains the user ID and the password of the target realm. The application then can wrap the remote call with a WSSubject.doAs call. For an example, see "Example: Using the WSLogin configuration to create a basic authentication subject" on page 271.
  2. Use the code sample in "Example: Using the WSLogin configuration to create a basic authentication subject" on page 271 from this plug point within the RMI_OUTBOUND login configuration. Every outbound Remote Method Invocation (RMI) request passes through this login configuration when it is enabled. Complete the following steps to enable and plug in this login configuration:
     a. Click **Security > Global security**.
     b. Under Authentication, click **Authentication protocol > CSIv2 outbound authentication**.
     c. Select the **Custom outbound mapping** option. If the **Security Attribute Propagation** option is selected, then WebSphere Application Server is already using this login configuration and you do not need to enable custom outbound mapping.
     d. Write a custom login module. For more information, see "Custom login module development for a system login configuration" on page 78.

The "Example: Sample login configuration for RMI_OUTBOUND" on page 273 article shows a custom login module that determines whether the realm names match. In this example, the realm names do not match so the WSLogin is used to create a basic authentication Subject based on custom mapping rules. The custom mapping rules are specific to the customer environment and must be implemented using a realm to user ID and password mapping utility.

e. Configure the RMI_OUTBOUND login configuration so that your new custom login module is first in the list.
   1) Click **Security** > **Global security**.
   2) Under Authentication, click **JAAS configuration** > **System logins** > **RMI_OUTBOUND**.
   3) Under Additional Properties, click **JAAS login modules** > **New** to add your login module to the RMI_OUTBOUND configuration.
   4) Return to the JAAS login modules panel for RMI_OUTBOUND and click **Set order** to change the order that the login modules are loaded so that your custom login is loaded first.

- Add the use_realm_callback and use_appcontext_callback options to the outbound mapping module for WSLogin. To add these options, complete the following steps:
  1. Click **Security > Global security**.
  2. Under Authentication, click **JAAS Configuration** > **Application logins** > **WSLogin**.
  3. Under Additional Properties, click **JAAS Login Modules > com.ibm.ws.security.common.auth.module.WSLoginModuleImpl**.
  4. Under Additional Properties, click **Custom Properties** > **New**.
  5. On the Custom Properties panel, enter `use_realm_callback` in the **Name** field and `true` in the **Value** field.
  6. Click **OK**.
  7. Click **New** to enter the second custom property.
  8. On the Custom Properties panel, enter `use_appcontext_callback` in the **Name** field and `true` in the **Value** field.
  9. Click **OK**.

  The following changes are made to the `security.xml` file:

```
<entries xmi:id="JAASConfigurationEntry_2" alias="WSLogin">
 <loginModules xmi:id="JAASLoginModule_2"
  moduleClassName="com.ibm.ws.security.common.auth.module.proxy.WSLoginModuleProxy"
  authenticationStrategy="REQUIRED">
  <options xmi:id="Property_2" name="delegate"
   value="com.ibm.ws.security.common.auth.module.WSLoginModuleImpl"/>
  <options xmi:id="Property_3" name="use_realm_callback" value="true"/>
  <options xmi:id="Property_4" name="use_appcontext_callback" value="true"/>
 </loginModules>
</entries>
```

## Example: Using the WSLogin configuration to create a basic authentication subject

This example shows how to use the WSLogin application login configuration from within a Java 2 Platform, Enterprise Edition (J2EE) application to login and get a Subject that contains the user ID and the password of the target realm

```
 javax.security.auth.Subject subject = null;

try
{
  // Create a login context using the WSLogin login configuration and specify a
```

```
  // user ID, target realm, and password. Note: If the target_realm_name is the
  // same as the current realm, an authenticated Subject is created. However, if
  // the target_realm_name is different from the current realm, a basic
  // authentication Subject is created that is not validated. This unvalidated
  // Subject is created so that you can send a request to the different target
  // realm with valid security credentials for that realm.
  javax.security.auth.login.LoginContext ctx = new LoginContext("WSLogin",
   new WSCallbackHandlerImpl("userid", "target_realm_name", "password"));

  // Note: The following is an alternative that  validates the user ID and
  // password specified against the target realm. It will perform a remote call
  // to the target server and will return  true if the user ID and password are
  // valid and false if the user ID and password are invalid. If false is
  // returned, a WSLoginFailedException is thrown. You can catch that exception and
  // perform a retry or stop the request from flowing by allowing that exception to
  // surface out of this login.

  // ALTERNATIVE LOGIN CONTEXT THAT VALIDATES THE USER ID AND PASSWORD TO THE
  // TARGET REALM

  /****  currently remarked out ****
  java.util.Map appContext = new java.util.HashMap();
            appContext.put(javax.naming.Context.INITIAL_CONTEXT_FACTORY,
                         "com.ibm.websphere.naming.WsnInitialContextFactory");
            appContext.put(javax.naming.Context.PROVIDER_URL,
                         "corbaloc:iiop:target_host:2809");

  javax.security.auth.login.LoginContext ctx = new LoginContext("WSLogin",
   new WSCallbackHandlerImpl("userid", "target_realm_name", "password", appContext));
  **** currently remarked out  ****/

  // Starts the login
  ctx.login();

  // Gets the Subject from the context
  subject = ctx.getSubject();
 }
 catch (javax.security.auth.login.LoginException e)
 {
  throw new com.ibm.websphere.security.auth.WSLoginFailedException (e.getMessage(), e);
 }

 if (subject != null)
 {
  // Defines a privileged action that encapsulates your remote request.
java.security.PrivilegedAction myAction = java.security.PrivilegedAction()
  {
   public Object run()
   {
    // Assumes a proxy is already defined. This example method returns a String
    return proxy.remoteRequest();
   }
  });

  // Executes this action using the basic authentication Subject needed for
     // the target realm security requirements.
```

```
 String myResult = (String) com.ibm.websphere.security.auth.WSSubject.doAs
       (subject, myAction);
}
```

## Example: Sample login configuration for RMI_OUTBOUND

This example shows a sample login configuration for RMI_OUTBOUND that determines whether the realm names match between two servers.

```
public customLoginModule()
{
 public void initialize(Subject subject, CallbackHandler callbackHandler,
     Map sharedState, Map options)
 {
     // (For more information on what to do during initialization, see
     // "Custom login module development for a system login configuration" on page 78.)
 }

 public boolean login() throws LoginException
 {
     // (For more information on what to do during login, see
     // "Custom login module development for a system login configuration" on page 78.)

  // Gets the WSProtocolPolicyCallback object
  Callback callbacks[] = new Callback[1];
   callbacks[0] = new com.ibm.wsspi.security.auth.callback.
          WSProtocolPolicyCallback("Protocol Policy Callback: ");

  try
  {
   callbackHandler.handle(callbacks);
  }
  catch (Exception e)
  {
   // Handles the exception
  }

     // Receives the RMI (CSIv2) policy object for checking the target realm
     // based upon information from the IOR.
     // Note: This object can be used to perform additional security checks.
     // See the Javadoc for more information.
  csiv2PerformPolicy = (CSIv2PerformPolicy) ((WSProtocolPolicyCallback)callbacks[0]).
       getProtocolPolicy();

  // Checks if the realms do not match. If they do not match, then login to
     // perform a mapping
  if (!csiv2PerformPolicy.getTargetSecurityName().equalsIgnoreCase(csiv2PerformPolicy.
        getCurrentSecurityName()))
  {
   try
   {
    // Do some custom realm -> user ID and password mapping
    MyBasicAuthDataObject myBasicAuthData = MyMappingLogin.lookup
          (csiv2PerformPolicy.getTargetSecurityName());

          // Creates the login context with basic authentication data gathered from
```

```
          // custom mapping
     javax.security.auth.login.LoginContext ctx = new LoginContext("WSLogin",
      new WSCallbackHandlerImpl(myBasicAuthData.userid,
        csiv2PerformPolicy.getTargetSecurityName(),
                    myBasicAuthData.password));

     // Starts the login
     ctx.login();

             // Gets the Subject from the context. This subject is used to replace
             // the passed-in Subject during the commit phase.
     basic_auth_subject = ctx.getSubject();
    }
    catch (javax.security.auth.login.LoginException e)
    {
     throw new com.ibm.websphere.security.auth.
               WSLoginFailedException (e.getMessage(), e);
    }
  }
 }

 public boolean commit() throws LoginException
 {
     // (For more information on what to do during commit, see
     // "Custom login module development for a system login configuration" on page 78.)

  if (basic_auth_subject != null)
  {
       // Removes everything from the current Subject and adds everything from the
       // basic_auth_subject
   try
   {
    public final Subject basic_auth_subject_priv = basic_auth_subject;
          // Do this in a doPrivileged code block so that application code
          // does not need to add additional permissions
    java.security.AccessController.doPrivileged(new java.security.
             PrivilegedExceptionAction()
    {
     public Object run() throws WSLoginFailedException
     {
               // Removes everything user-specific from the current outbound
               // Subject. This a temporary Subject for this specific invocation
               // so you are not affecting the Subject set on the thread. You may
               // keep any custom objects that you want to propagate in the Subject.
               // This example removes everything and adds just the new information
               // back in.
      try
      {
       subject.getPublicCredentials().clear();
       subject.getPrivateCredentials().clear();
       subject.getPrincipals().clear();
      }
      catch (Exception e)
      {
       throw new WSLoginFailedException (e.getMessage(), e);
      }
```

```
              // Adds everything from basic_auth_subject into the login subject.
              // This completes the mapping to the new user.
      try
      {
       subject.getPublicCredentials().addAll(basic_auth_subject.
                   getPublicCredentials());
       subject.getPrivateCredentials().addAll(basic_auth_subject.
                   getPrivateCredentials());
       subject.getPrincipals().addAll(basic_auth_subject.
                   getPrincipals());
      }
      catch (Exception e)
      {
       throw new WSLoginFailedException (e.getMessage(), e);
      }

      return null;
     }
    });
   }
   catch (PrivilegedActionException e)
   {
    throw new WSLoginFailedException (e.getException().getMessage(),
             e.getException());
   }
  }
 }

 // Defines your login module variables
 com.ibm.wsspi.security.csiv2.CSIv2PerformPolicy csiv2PerformPolicy = null;
 javax.security.auth.Subject basic_auth_subject = null;
}
```

## Security attribute propagation

*Security attribute propagation* enables WebSphere Application Server to transport security attributes (authenticated Subject contents and security context information) from one server to another in your configuration. WebSphere Application Server might obtain these security attributes from either an enterprise user registry, which queries static attributes, or a custom login module, which can query static or dynamic attributes. Dynamic security attributes, which are custom in nature, might include the authentication strength used for the connection, the identity of the original caller, the location of the original caller, the IP address of the original caller, and so on.

Security attribute propagation provides propagation services using Java serialization for any objects that are contained in the Subject. However, Java code must be able to serialize and de-serialize these objects. The Java programming language specifies the rules for how Java code can serialize an object. Because problems can occur when dealing with different platforms and versions of software, WebSphere Application Server also offers a token framework that enables custom serialization functionality. The token framework has other benefits that include the ability to identify the uniqueness of the token. This uniqueness determines how the Subject gets cached and the purpose of the token. The token framework defines four marker token interfaces that enable the WebSphere Application Server run time to determine how to propagate the token.

**Important:** Any custom tokens that are used in this framework are not used by WebSphere Application Server for authorization or authentication. The framework serves as a way to notify

WebSphere Application Server that you want these tokens propagated in a particular way. WebSphere Application Server handles the propagation details, but does not handle serialization or deserialization of custom tokens. Serialization and deserialization of these custom tokens are carried out by the implementation and handled by a custom login module.

With WebSphere Application Server 6.0 and later, a custom Java Authorization Contract for Container (JACC) provider can be configured to enforce access control for Java 2 Platform, Enterprise Edition (J2EE) applications. A custom JACC provider can explore the custom security attributes in the caller JAAS subject in making access control decisions.

When a request is being authenticated, a determination is made by the login modules whether this is an *initial login* or a *propagation login*. An initial login is the process of authenticating the user information, typically a user ID and password, and then calling the application programming interfaces (APIs) for the remote user registry to look up secure attributes that represent the user access rights. A propagation login is the process of validating the user information, typically an Lightweight Third Party Authentication (LTPA) token, and then deserializing a series of tokens that constitute both custom objects and token framework objects known to the WebSphere Application Server.

The following marker tokens are introduced in the framework:

**Authorization token**

The authorization token contains most of the authorization-related security attributes that are propagated. The default authorization token is used by the WebSphere Application Server authorization engine to make Java 2 Platform, Enterprise Edition (J2EE) authorization decisions. Service providers can use custom authorization token implementations to isolate their data in a different token; perform custom serialization and de-serialization; and make custom authorization decisions using the information in their token at the appropriate time. For information on how to use and implement this token type, see “Default PropagationToken” on page 282 and “Implementing a custom PropagationToken” on page 287.

**Single signon (SSO) token**

A custom SingleSignonToken token that is added to the Subject is automatically added to the response as an HTTP cookie and contains the attributes sent back to Web browsers. The token interface getName() method together with the getVersion method defines the cookie name. WebSphere Application Server defines a default SingleSignonToken with the LtpaToken name and version 2. The cookie name added is LtpaToken2. Do not add sensitive information, confidential information, or unencrypted data to the response cookie.

It is also recommended that any time that you use cookies, use the Secure Sockets Layer (SSL) protocol to protect the request. Using an SSO token, Web users can authenticate once when accessing Web resources across multiple WebSphere Application Servers. A custom SSO token extends this functionality by adding custom processing to the single signon scenario. For more information on SSO tokens, see “Configuring single signon” on page 172. For information on how to use and implement this token type, see “Default SingleSignonToken” on page 309 and “Implementing a custom SingleSignonToken” on page 310.

**Propagation token**

The propagation token is not associated with the authenticated user so it is not stored in the Subject. Instead, the propagation token is stored on the thread and follows the invocation wherever it goes. When a request is sent outbound to another server, the propagation tokens on that thread are sent with the request and the tokens are executed by the target server. The attributes stored on the thread are propagated regardless of the Java 2 Platform, Enterprise Edition (J2EE) RunAs user switches.

The default propagation token monitors and logs all user switches and host switches. You can add additional information to the default propagation token using the WSSecurityHelper application programming interfaces (APIs). To retrieve and set custom implementations of a propagation

token, you can use the WSSecurityPropagationHelper class. For information on how to use and implement this token type, see "Default PropagationToken" on page 282 and "Implementing a custom PropagationToken" on page 287.

**Authentication token**

The authentication token flows to downstream servers and contains the identity of the user. This token type serves the same function as the Lightweight Third Party Authentication (LTPA) token in previous versions. Although this token type is typically reserved for internal WebSphere Application Server purposes, you can add this token to the Subject and the token is propagated using the getBytes method of the token interface.

A custom authentication token is used solely for the purpose of the service provider that adds it to the Subject. WebSphere Application Server do not use it for authentication purposes, because a default authentication token exists that is used for WebSphere Application Server authentication. This token type is available for the service provider to identify the purpose of the custom data to use the token to perform custom authentication decisions. For information on how to use and implement this token type, see "Default AuthenticationToken" on page 321 and "Implementing a custom AuthenticationToken" on page 322.

### Horizontal propagation versus downstream propagation

In WebSphere Application Server, both horizontal propagation, which is uses single signon for Web requests, and downstream propagation, which uses Remote Method Invocation over the Internet Inter-ORB Protocol (RMI/IIOP) to access enterprise beans, are available.

### Horizontal propagation

In horizontal propagation, security attributes are propagated amongst front-end servers. The serialized security attributes, which are the Subject contents and the propagation tokens, can contain both static and dynamic attributes. The single signon (SSO) token stores additional system-specific information that is needed for horizontal propagation. The information contained in the SSO token tells the receiving server where the originating server is located and how to communicate with that server. Additionally, the SSO token also contains the key to look up the serialized attributes. In order to enable horizontal propagation, you must configure the single signon token and the Web inbound security attribute propagation features. You can configure both of these features using the administrative console by completing the following steps:

1. Click **Security > Global security**.
2. Under Authentication, click **Authentication Mechanisms > LTPA**.
3. Under Additional properties, click **Single signon (SSO)**

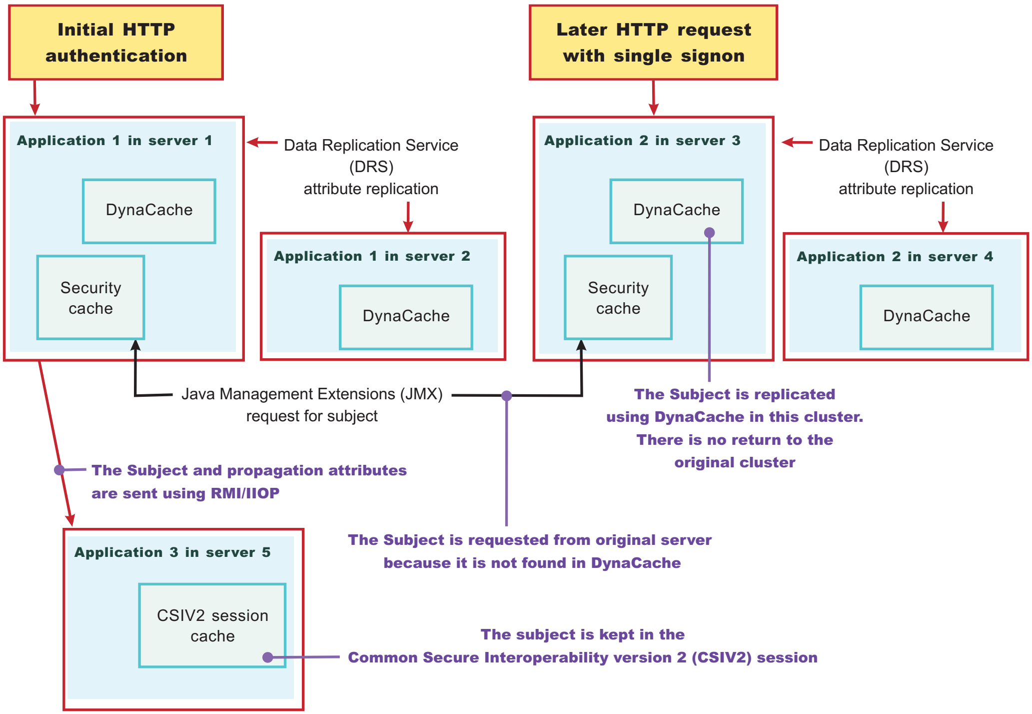
For more information, see "Enabling security attribute propagation" on page 279.

When front-end servers are configured and in the same distributed replication service (DRS) replication domain, the application server automatically propagates the serialized information to all of the servers within the same domain. In figure 1, application 1 is deployed on server 1 and server 2, and both servers are members of the same DRS replication domain. If a request originates from application 1 on server 1 and then gets redirected to application 1 on server 2, the original login attributes are found on server 2 without additional remote requests. However, if the request originates from application 1 on either server 1 or server 2, but the request is redirected to application 2 on either server 1 or server 2, the serialized information is not found in the DRS cache because the servers are not configured in the same replication domain. As a result, a remote Java Management Extensions (JMX) request is sent back to the originating server that hosts application 1 to obtain the serialized information so that original login information is available to the application. By getting the serialized information using a single JMX remote call back to the originating server, the following benefits are realized:

- You gain the function of retrieving login information from the original server.

- You do not need to perform any remote user registry calls because the application server can regenerate the Subject from the serialized information. Without this ability, the application server might make 5 to 6 separate remote calls.

**Figure 1**




## Performance implications for horizontal propagation

The performance implications of either the DRS or JMX remote call depends upon your environment. THE DRS or JMX remote call is used for obtaining the original login attributes. Horizontal propagation reduces many of the remote user registry calls in cases where these calls cause the most performance problems for an application. However, the de-serialization of these objects also might cause performance degradation, but this degradation might be less than the remote user registry calls. It is recommended that you test your environment with horizontal propagation enabled and disabled. In cases where you must use horizontal propagation for preserving original login attributes, test whether DRS or JMX provides better performance in your environment. Typically, it is recommended that you configure DRS both for failover and performance reasons. However, because DRS propagates the information to all of the servers in the same replication domain (whether the servers are accessed or not), there might be a performance degradation if too many servers are in the same replication domain. In this case, either reduce the number of servers in the replication domain or do not configure the servers in a DRS replication domain. The later suggestion causes a JMX remote call to retrieve the attributes, when needed, which might be quicker overall.

## Downstream propagation

In *downstream propagation*, a Subject is generated at the Web front-end server, either by a propagation login or a user registry login. WebSphere Application Server propagates the security information

downstream for enterprise bean invocations when both Remote Method Invocation (RMI) outbound and inbound propagation are enabled.

**Benefits of propagating security attributes**

The security attribute propagation feature of WebSphere Application Server has the following benefits:

- Enables WebSphere Application Server to use the security attribute information for authentication and authorization purposes. The propagation of security attributes can eliminate the need for user registry calls at each remote hop along an invocation. Previous versions of WebSphere Application Server propagated only the user name of the authenticated user, but ignored other security attribute information that needed to be regenerated downstream using remote user registry calls. To accentuate the benefits of this new functionality, consider the following example:

  In previous releases, you might use a reverse proxy server (RPSS), such as WebSEAL, to authenticate the user, gather group information, and gather other security attributes. As stated previously, WebSphere Application Server accepted the identity of the authenticated user, but disregarded the additional security attribute information. To create a Java Authentication and Authorization Service (JAAS) Subject containing the needed WSCredential and WSPrincipal objects, WebSphere Application Server made 5 to 6 calls to the user registry. The WSCredential object contains various security information that is required to authorize a J2EE resource. The WSPrincipal object contains the realm name and the user that represents the principal for the Subject.

  In the current release of the Application Server, information that is obtained from the reverse proxy server can be used by WebSphere Application Server and propagated downstream to other server resources without additional calls to the user registry. The retaining of the security attribute information enables you to protect server resources properly by making appropriate authorization and trust-based decisions User switches that occur because of J2EE RunAs configurations do not cause the application server to lose the original caller information. This information is stored in the PropagationToken located on the running thread.
- Enables third-party providers to plug in custom tokens. The token interface contains a getBytes method that enables the token implementation to define custom serialization, encryption methods, or both.
- Provides the ability to have multiple tokens of the same type within a Subject created by different providers. WebSphere Application Server can handle multiple tokens for the same purpose. For example, you might have multiple authorization tokens in the Subject and each token might have distinct authorization attributes that are generated by different providers.
- Provides the ability to have a unique ID for each token type that is used to formulate a more unique subject identifier than just the user name in cases where dynamic attributes might change the context of a user login. The token type has a getUniqueId() method that is used for returning a unique string for caching purposes. For example, you might need to propagate a location ID, which indicates the location from which the user logs into the system. This location ID can be generated during the original login using either an reverse proxy server or the WEB_INBOUND login configuration and added to the Subject prior to serialization. Other attributes might be added to the Subject as well and use a unique ID. All of the unique IDs must be considered for the uniqueness of the entire Subject. WebSphere Application Server has the ability to specify what is unique about the information in the Subject, which might affect how the user accesses the Subject later.

## Enabling security attribute propagation

The security attribute propagation feature of WebSphere Application Server enables you to send security attribute information regarding the original login to other servers using a token. To fully enable security attribute propagation, you must configure the single signon (SSO), CSIv2 inbound, and CSIv2 outbound panels in the WebSphere Application Server Administrative Console. You can enable just the portions of security attribute propagation relevant to your configuration. For example, you can enable Web propagation, which is propagation amongst front-end application servers, using either the push technique (DynaCache) or the pull technique (remote method to originating server). You also can choose whether to enable Remote Method Invocation (RMI) outbound and inbound propagation, which is commonly called

downstream propagation. Typically both types of propagation are enabled for any given cell. In some cases, you might want to choose a different option for a specific application server using the server security panel within the specific application server settings. To access the server security panel in the administrative console, click **Servers > Application Servers >** ***server_name***. Under Security, click **Server security**. Under Additional properties, click **Server-level security**.

Complete the following steps to configure WebSphere Application Server for security attribute propagation:

1. Access the WebSphere Application Server administrative console by typing http://*server_name*:9060/ibm/console The administrative console address might differ if you have previously changed the port number.
2. Click **Security > Global security**. Under Authentication, click **Authentication mechanisms > LTPA**. Under Additional Properties, click **Single Signon (SSO)**.
3. **Optional:** Select the **Interoperability Mode** option if you need to interoperate with servers that do not support security attribute propagation. Servers that do not support security attribute propagation receive the Lightweight Third Party Authentication (LTPA) token and the PropagationToken, but ignore the security attribute information that it does not understand.
4. Select the **Web inbound security attribute propagation** option. The **Web inbound security attribute propagation** option enables horizontal propagation, which allows the receiving SSO token to retrieve the login information from the original login server. If you do not enable this option, downstream propagation can occur if you enable the **Security Attribute Propagation** option on both the CSIv2 Inbound authentication and CSIv2 outbound authentication panels.

   Typically, you enable the **Web inbound security attribute propagation** option if you need to gather dynamic security attributes set at the original login server that cannot be regenerated at the new front-end server. This attributes include any custom attributes that might be set in the PropagationToken using the com.ibm.websphere.security.WSSecurityHelper application programming interfaces (APIs). You must determine whether enabling this option improves or degrades the performance of your system. While the option prevents some remote user registry calls, the deserialization and decryption of some tokens might impact performance. In some cases, propagation is faster especially if your user registry is the bottleneck of your topology. It is recommended that you measurement the performance of your environment using and not using this option. When you test the performance, it is recommended that you test in the operating environment of the typical production environment with the typical number of unique users accessing the system simultaneously.
5. Click **Security > Global security**. Under Authentication, click **Authentication protocol > CSIv2 inbound authentication**. The Login configuration field specifies `RMI_INBOUND` as the system login configuration used for inbound requests. To add custom Java Authentication and Authorization Service (JAAS) login modules, complete the following steps:
   a. Click **Security > Global security**. Under Authentication, click **JAAS Configuration > System logins**. A list of the system login configurations is displayed. WebSphere Application Server provides the following pre-configured system login configurations: DEFAULT, LTPA, LTPA_WEB, RMI_INBOUND, RMI_OUTBOUND, SWAM, WEB_INBOUND, wssecurity.IDAssertion, and wssecurity.Signature. Do not delete these predefined configurations.
   b. Click the name of the login configuration that you want to modify.
   c. Under Additional Properties, click **JAAS Login Modules**. The JAAS Login Modules panel is displayed, which lists all of the login modules processed in the login configuration. Do not delete the required JAAS login modules. Instead, you can add custom login modules before or after the required login modules. If you add custom login modules, do not begin their names with com.ibm.ws.security.server because this prefix is reserved for WebSphere Application Server internal use.

      You can specify the order in which the login modules are processed by clicking **Set Order**.
6. Select the **Security attribute propagation** option on the CSIv2 Inbound authentication panel. When you select **Security Attribute Propagation**, the server advertises to other application servers that it can receive propagated security attributes from another server in the same realm over the Common Secure Interoperability version 2 (CSIv2) protocol.

7. Click **Security**. Under Authentication, click **Authentication protocol > CSIv2 Outbound authentication**. The CSIv2 outbound authentication panel is displayed. The **Login configuration** field specifies `RMI_OUTBOUND` as JAAS login configuration that is used for outbound configuration. You cannot change this login configuration. Instead, you can customize this login configuration by completing the substeps listed previously for CSIv2 Inbound authentication.
8. **Optional:** Verify that the **Security Attribute Propagation** option is selected if you want to enable outbound Subject and security context token propagation for the Remote Method Invocation (RMI) protocol. When you select this option, WebSphere Application Server serializes the Subject contents and the PropagationToken contents. After the contents are serialized, the server uses the Common Secure Interoperability version 2 (CSIv2) protocol to send the Subject and PropagationToken to the target servers that support security attribute propagation. If the receiving server does not support security attribute tokens, WebSphere Application Server sends the Lightweight Third Party Authentication (LTPA) token only.

   **Important:** WebSphere Application Server propagates only the objects within the Subject that it can serialize. The server propagates custom objects on a best-effort basis.

   When **Security Attribute Propagation** is enabled, WebSphere Application Server adds marker tokens to the Subject to enable the target server to add additional attributes during the inbound login. During the commit phase of the login, the marker tokens and the Subject are marked as read-only and cannot be modified thereafter.
9. **Optional:** Select the **Custom Outbound Mapping** option if you deselect the **Security Attribute Propagation** option and you want to use the RMI_OUTBOUND login configuration. If the **Custom Outbound Mapping** option nor the **Security Attribute Propagation** option is selected, WebSphere Application Server does not call the RMI_OUTBOUND login configuration. If you need to plug in a credential mapping login module, you must select the **Custom Outbound Mapping** option.
10. **Optional:** Specify trusted target realm names in the **Trusted Target Realms** field. By specifying these realm names, information can be sent to servers that reside outside the realm of the sending server to allow for inbound mapping to occur at these downstream servers. To perform outbound mapping to a realm different from the current realm, you must specify the realm in this field so that you can get to this point without the request being rejected due to a realm mismatch. If you need WebSphere Application Server to propagate security attributes to another realm when a request is sent, you must specify the realm name in the **Trusted Target Realms** field. Otherwise, the security attributes are not propagated to the unspecified realm. You can add multiple target realms by adding a pipe (|) delimiter between each entry.
11. **Optional:** Enable propagation for a pure client. For a pure client to propagate attributes added to the invocation Subject, you must add the following property to the `sas.client.props` file:

    ```
    com.ibm.CSI.rmiOutboundPropagationEnabled=true
    ```

After completing these steps, you have configured WebSphere Application Server to propagate security attributes to other servers. After you have configured WebSphere Application Server for security attribute propagation and need to disable this functionality, you can disable propagation for either the server level or the cell level. To disable security attribute propagation on the server level, click **Server > Application Servers >***server_name*. Under Security, click **Server security**. You can disable security attribute propagation for inbound requests by clicking **CSI inbound authentication** under Additional Properties and deselecting **Security attribute propagation**. You can disable security attribute propagation for outbound requests by clicking **CSI outbound authentication** under Additional Properties and deselecting **Security attribute propagation**. To disable security attribute propagation on the cell level, undo each of the steps that you completed to enable security attribute propagation in this task.

## Default PropagationToken

A default PropagationToken is located on the thread of execution for applications and the security infrastructure to use. WebSphere Application Server propagates this default PropagationToken downstream and the token stays on the thread where the invocation lands at each hop. The data should be available from within the container of any resource where the PropagationToken lands. Remember that you must enable the propagation feature at each server where a request is sent in order for propagation to work. Make sure that you have enabled security attribute propagation for all of the cells in your environment where you want propagation

There is a WSSecurityHelper class that has application programming interfaces (APIs) for accessing the PropagationToken attributes. This article documents the usage scenarios and includes examples. A close relationship exists between PropagationToken and the WorkArea feature. The main difference between these features is that after you add attributes to the PropagationToken, you cannot change the attributes. You cannot change these attributes so that the security run time can add auditable information and have that information remain there for the life of the invocation. Any time that you add an attribute to a specific key, an ArrayList is stored to hold that attribute. Any new attribute added with the same key is added to the ArrayList. When you call getAttributes, the ArrayList is converted to a String[] and the order is preserved. The first element in the String[] is the first attribute added for that specific key.

In the default PropagationToken, a change flag is kept that logs any data changes to the token. These changes are tracked to enable WebSphere Application Server to know when to re-send the authentication information downstream so that the downstream server has those changes. Normally, Common Secure Interoperability Version 2 (CSIv2) maintains a session between servers for an authenticated client. If the PropagationToken changes, a new session is generated and subsequently a new authentication occurs. Frequent changes to the PropagationToken during a method causes frequent downstream calls. If you change the token prior to making many downstream calls, but you change the token between each downstream call, you might impact security performance.

**Getting the server list from the default PropagationToken**

Every time the PropagationToken is propagated and used to create the authenticated Subject, either horizontally or downstream, the name of the receiving application server is logged into the PropagationToken. The format of the host is ″Cell:Node:Server″, which provides you access to the cell name, node name, and server name of each application server that receives the invocation. The following code provides you with this list of names and can be called from a Java 2 Platform, Enterprise Edition (J2EE) application:

```
String[] server_list = null;

// If security is disabled on this application server, do not bother checking
if (com.ibm.websphere.security.WSSecurityHelper.isServerSecurityEnabled())
{
 try
 {
  // Gets the server_list string array
  server_list = com.ibm.websphere.security.WSSecurityHelper.getServerList();
 }
 catch (Exception e)
 {
  // Performs normal exception handling for your application
 }

 if (server_list != null)
 {
```

```
   // print out each server in the list, server_list[0] is the first server
   for (int i=0; i<server_list.length; i++)
   {
    System.out.println("Server[" + i + "] = " + server_list[i]);
   }
  }
 }
```

The format of each server in the list is: *cell*:*node*:*server*. The output, for example, is: `myManager:node1:server1`

**Getting the caller list from the default PropagationToken**

A default PropagationToken is generated any time an authenticated user is set on the thread of execution or any one tries to add attributes to the PropagationToken. Whenever an authenticated user is set on the thread, the user is logged in the default PropagationToken. There may be some pushing and popping of Subjects by the authorization code. At times, the same user might be logged in multiple times if the RunAs user is different from the caller. The following list provides the rules that are used to determine if a user added to the thread gets logged into the PropagationToken:

- The current Subject must be authenticated. For example, an unauthenticated Subject is not logged.
- The current authenticated Subject is logged if a Subject has not been previously logged.
- The current authenticated Subject is logged if the last authenticated Subject logged does not contain the same user.
- The current authenticated Subject is logged on each unique application server involved in the propagation process.

The following code sample shows how to use the getCallerList() API:

```
 String[] caller_list = null;

 // If security is disabled on this application server, do not check the caller list
 if (com.ibm.websphere.security.WSSecurityHelper.isServerSecurityEnabled())
 {
  try
  {
   // Gets the caller_list string array
   caller_list = com.ibm.websphere.security.WSSecurityHelper.getCallerList();
  }
  catch (Exception e)
  {
   // Performs normal exception handling for your application
  }

  if (caller_list != null)
  {
   // Prints out each caller in the list, caller_list[0] is the first caller
   for (int i=0; i<caller_list.length;i++)
   {
    System.out.println("Caller[" + i + "] = " + caller_list[i]);
   }
  }
 }
```

The format of each caller in the list is: *cell*:*node*:*server*:*realm*/*securityName*. The output, for example, is: `myManager:node1:server1:ldap.austin.ibm.com:389/jsmith`.

### Getting the first caller from the default PropagationToken

Whenever you want to know which authenticated caller started the request, you can call the getFirstCaller method and the caller list is parsed. However, this method returns the securityName of the caller only. If you need to know more than the securityName, call the getCallerList() method and retrieve the first entry in the String[]. This entry provides the entire caller information. The following code sample retrieves the securityName of the first authenticated caller using the getFirstCaller() API:

```
String first_caller = null;

// If security is disabled on this application server, do not bother checking
if (com.ibm.websphere.security.WSSecurityHelper.isServerSecurityEnabled())
{
 try
 {
  // Gets the first caller
  first_caller = com.ibm.websphere.security.WSSecurityHelper.getFirstCaller();

  // Prints out the caller name
  System.out.println("First caller: " + first_caller);
 }
 catch (Exception e)
 {
  // Performs normal exception handling for your application
 }
}
```

The output, for example, is: `jsmith`.

### Getting the first host from the default PropagationToken

Whenever you want to know what the first application server is for this request, you can call the getFirstServer() method directly. The following code sample retrieves the name of the first application server using the getFirstServer() API:

```
String first_server = null;

// If security is disabled on this application server, do not bother checking
if (com.ibm.websphere.security.WSSecurityHelper.isServerSecurityEnabled())
{
 try
 {
  // Gets the first server
  first_server = com.ibm.websphere.security.WSSecurityHelper.getFirstServer();

  // Prints out the server name
  System.out.println("First server: " + first_server);
 }
 catch (Exception e)
 {
  // Performs normal exception handling for your application
 }
}
```

The output, for example, is: `myManager:node1:server1`.

### Adding custom attributes to the default PropagationToken

You can add custom attributes to the default PropagationToken for application usage. This token follows the request downstream so that the attributes are available when they are needed. When you use the default PropagationToken to add attributes, you must understand the following issues:

- When you add information to the PropagationToken, it affects CSIv2 session caching. Add information sparingly between remote requests.
- After you add information with a specific key, the information cannot be removed.
- You can add as many values to a specific key as your need. However, all of the values must be available from a returned String[] in the order they were added.
- The PropagationToken is available only on servers where propagation and security are enabled.
- The Java 2 Security javax.security.auth.AuthPermission wssecurity.addPropagationAttribute is needed to add attributes to the default PropagationToken.
- An application cannot use keys that begin with either com.ibm.websphere.security or com.ibm.wsspi.security. These prefixes are reserved for system usage.

The following code sample shows how to use the addPropagationAttribute API:

```
// If security is disabled on this application server,
    // do not check the status of server security
if (com.ibm.websphere.security.WSSecurityHelper.isServerSecurityEnabled())
{
 try
 {
  // Specifies the key and values
  String key = "mykey";
  String value1 = "value1";
  String value2 = "value2";

  // Sets key, value1
      com.ibm.websphere.security.WSSecurityHelper.
      addPropagationAttribute (key, value1);

  // Sets key, value2
  String[] previous_values = com.ibm.websphere.security.WSSecurityHelper.
         addPropagationAttribute (key, value2);

  // Note: previous_values should contain value1
 }
 catch (Exception e)
 {
  // Performs normal exception handling for your application
 }
}
```

See "Getting custom attributes from the default PropagationToken" to retrieve attributes using the getPropagationAttributes application programming interface (API).

### Getting custom attributes from the default PropagationToken

Custom attributes are added to the default PropagationToken using the addPropagationAttribute API. These attributes can be retrieved using the getPropagationAttributes API. This token follows the request downstream so the attributes are available when they are needed. When you use the default PropagationToken to retrieve attributes, you must understand the following issues.

- The PropagationToken is available only on servers where propagation and security are enabled.
- The Java 2 Security javax.security.auth.AuthPermission wssecurity.getPropagationAttributes is needed to retrieve attributes from the default PropagationToken.

The following code sample shows how to use the getPropagationAttributes API:

```
// If security is disabled on this application server, do not bother checking
if (com.ibm.websphere.security.WSSecurityHelper.isServerSecurityEnabled())
{
 try
 {
  String key = "mykey";
  String[] values = null;

  // Sets key, value1
      values = com.ibm.websphere.security.WSSecurityHelper.
      getPropagationAttributes (key);

  // Prints the values
  for (int i=0; i<values.length; i++)
  {
   System.out.println("Value[" + i + "] = " + values[i]);
  }
 }
 catch (Exception e)
 {
  // Performs normal exception handling for your application
 }
}
```

The output, for example, is:

```
Value[0] = value1
Value[1] = value2
```

See Adding custom attributes to the default PropagationToken to add attributes using the addPropagationAttributes API.

**Changing the TokenFactory associated with the default PropagationToken**

When WebSphere Application Server generates a default PropagationToken, the application server utilizes the TokenFactory class that is specified using the com.ibm.wsspi.security.token.propagationTokenFactory property. To modify this property using the administrative console, complete the following steps:

1. Click **Security > Global Security**.
2. Under Additional properties, click **Custom properties**.

The default TokenFactory specified for this property is called com.ibm.ws.security.ltpa.AuthzPropTokenFactory. This token factory encodes the data in the PropagationToken and does not encrypt the data. Because the PropagationToken typically flows over Common Secure Interoperability version 2 (CSIv2) using Secure Sockets Layer (SSL), there is no need to encrypt the token itself. However, if you need additional security for the PropagationToken, you can associate a different TokenFactory implementation with this property to get encryption. For example, if you choose to associate com.ibm.ws.security.ltpa.LTPAToken2Factory with this property, the token is AES encrypted. However, you need to weigh the performance impacts against your security needs. Adding

sensitive information to the PropagationToken is a good reason to change the TokenFactory implementation to something that encrypts rather than just encodes.

If you want to perform your own signing and encryption of the default PropagationToken, you must implement the following classes:
- com.ibm.wsspi.security.ltpa.Token
- com.ibm.wsspi.security.ltpa.TokenFactory

Your TokenFactory implementation instantiates and validates your token implementation. You can choose to use the Lightweight Third Party Authentication (LTPA) keys passed into the initialize method of the TokenFactory or you can use your own keys. If you use your own keys, they must be the same everywhere in order to validate the tokens that are generated using those keys. See the Javadoc, available through a link on the front page of the information center, for more information on implementing your own custom TokenFactory. To associate your TokenFactory with the default PropagationToken, using the administrative console, complete the following steps:
1. Click **Security > Global Security**.
2. Under Additional properties, click **Custom properties**.
3. Locate the com.ibm.wsspi.security.token.propagationTokenFactory property and verify that the value of this property matches your custom TokenFactory implementation.
4. Verify that your implementation classes are put into the `*install directory*/classes` directory so that the WebSphere class loader can load the classes.

## Implementing a custom PropagationToken

This task explains how you might create your own PropagationToken implementation, which is set on the thread of execution and propagated downstream. The default PropagationToken usually is sufficient for propagating attributes that are not user-specific. Consider writing your own implementation if you want to accomplish one of the following tasks:
- Isolate your attributes within your own implementation.
- Serialize the information using custom serialization. You must deserialize the bytes at the target and add that information back on the thread by plugging in a custom login module into the inbound system login configurations. This task also might include encryption and decryption.

To implement a custom Propagation token, you must complete the following steps:
1. Write a custom implementation of the PropagationToken interface. There are many different methods for implementing the PropagationToken interface. However, make sure that the methods required by the PropagationToken interface and the token interface are fully implemented. After you implement this interface, you can place it in the *install_dir*/classes directory. Alternatively, you can place the class in any private directory. However, make sure that the WebSphere Application Server class loader can locate the class and that it is granted the appropriate permissions. You can add the Java archive (JAR) file or directory that contains this class into the `server.policy` file so that it has the necessary permissions that are needed by the server code.

   **Tip:** All of the token types defined by the propagation framework have similar interfaces. Basically, the token types are marker interfaces that implement the com.ibm.wsspi.security.token.Token interface. This interface defines most of the methods. If you plan to implement more than one token type, consider creating an abstract class that implements the com.ibm.wsspi.security.token.Token interface. All of your token implementations, including the PropagationToken, might extend the abstract class and then most of the work is completed.

   To see an implementation of PropagationToken, see "Example: com.ibm.wsspi.security.token.PropagationToken implementation" on page 288
2. Add and receive the custom PropagationToken during WebSphere Application Server logins This task is typically accomplished by adding a custom login module to the various application and system login

configurations. You also can add the implementation from an application. However, in order to deserialize the information, you will need to plug in a custom login module, which is discussed in "Propagating a custom Java serializable object" on page 331. The WSSecurityPropagationHelper class has APIs that are used to set a PropagationToken on the thread and to retrieve it from the thread to make updates.

The code sample in "Example: custom PropagationToken login module" on page 294 shows how to determine if the login is an initial login or a propagation login. The difference between these login types is whether the WSTokenHolderCallback contains propagation data. If the callback does not contain propagation data, initialize a new custom PropagationToken implementation and set it on the thread. If the callback contains propagation data, look for your specific custom PropagationToken TokenHolder instance, convert the byte[] back into your customer PropagationToken object, and set it back on the thread. The code sample shows both instances.

You can add attributes any time your custom PropagationToken is added to the thread. If you add attributes between requests and the getUniqueId method changes, then the CSIv2 client session is invalidated so that it can send the new information downstream. Keep in mind that adding attributes between requests can affect performance. In many cases, this is the desired behavior so that downstream requests receive the new PropagationToken information.

To add the custom PropagationToken to the thread, call WSSecurityPropagationHelper.addPropagationToken. This call requires the following Java 2 Security permission: WebSphereRuntimePerMission ″setPropagationToken″

3. Add your custom login module to WebSphere Application Server system login configurations that already contain the com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule for receiving serialized versions of your custom propagation token Also, you can add this login module to any of the application logins where you might want to generate your custom PropagationToken on the thread during the login. Alternatively, you can generate the custom PropagationToken implementation from within your application. However, to deserialize it, you need to add the implementation to the system login modules.

   For information on how to add your custom login module to the existing login configurations, see "Custom login module development for a system login configuration" on page 78

After completing these steps, you have implemented a custom PropagationToken.

## Example: com.ibm.wsspi.security.token.PropagationToken implementation

Use this file to see an example of a PropagationToken implementation. The following sample code does not extend an abstract class, but rather implements the com.ibm.wsspi.security.token.PropagationToken interface directly. You can implement the interface directly, but it might cause you to write duplicate code. However, you might choose to implement the interface directly if there are considerable differences between how you handle the various token implementations.

For information on how to implement a custom PropagationToken, see "Implementing a custom PropagationToken" on page 287.

```
package com.ibm.websphere.security.token;

import com.ibm.websphere.security.WSSecurityException;
import com.ibm.websphere.security.auth.WSLoginFailedException;
import com.ibm.wsspi.security.token.*;
import com.ibm.websphere.security.WebSphereRuntimePermission;
import java.io.ByteArrayOutputStream;
import java.io.ByteArrayInputStream;
import java.io.DataOutputStream;
import java.io.DataInputStream;
import java.io.ObjectOutputStream;
```

```
import java.io.ObjectInputStream;
import java.io.OutputStream;
import java.io.InputStream;
import java.util.ArrayList;

public class CustomPropagationTokenImpl implements com.ibm.wsspi.security.
   token.PropagationToken
{
 private java.util.Hashtable hashtable = new java.util.Hashtable();
 private byte[] tokenBytes = null;
   // 2 hours in millis, by default
  private static long expire_period_in_millis = 2*60*60*1000;
 private long counter = 0;

/**
 * The constructor that is used to create initial PropagationToken instance
 */

 public CustomAbstractTokenImpl ()
 {
  // set the token version
  addAttribute("version", "1");
  // set the token expiration
  addAttribute("expiration", new Long(System.currentTimeMillis() +
        expire_period_in_millis).toString());
 }

/**
 * The constructor that is used to deserialize the token bytes received
 * during a propagation login.
 */
 public CustomAbstractTokenImpl (byte[] token_bytes)
 {
  try
  {
       hashtable = (java.util.Hashtable) com.ibm.wsspi.security.token.
         WSOpaqueTokenHelper.deserialize(token_bytes);
  }
  catch (Exception e)
  {
   e.printStackTrace();
  }
 }

/**
 * Validates the token including expiration, signature, and so on.
 * @return boolean
 */

 public boolean isValid ()
 {
  long expiration = getExpiration();

  // if you set the expiration to 0, it does not expire
  if (expiration != 0)
  {
```

```
   // return if this token is still valid
   long current_time = System.currentTimeMillis();

   boolean valid = ((current_time < expiration) ? true : false);
   System.out.println("isValid: returning " + valid);
   return valid;
  }
  else
  {
   System.out.println("isValid: returning true by default");
   return true;
  }
 }

/**
 * Gets the expiration as a long type.
 * @return long
 */
 public long getExpiration()
 {
  // get the expiration value from the hashtable
  String[] expiration = getAttributes("expiration");

  if (expiration != null && expiration[0] != null)
  {
   // expiration is the first element (should only be one)
   System.out.println("getExpiration: returning " + expiration[0]);
   return new Long(expiration[0]).longValue();
  }

  System.out.println("getExpiration: returning 0");
  return 0;
 }

/**
 * Returns if this token should be forwarded/propagated downstream.
 * @return boolean
 */
 public boolean isForwardable()
 {
     // You can choose whether your token gets propagated. In some cases
     // you might want the token to be local only.
  return true;
 }

/**
 * Gets the principal that this token belongs to. If this token is an
 * authorization token, this principal string must match the authentication
 * token principal string or the message is rejected.
 * @return String
 */
 public String getPrincipal()
 {
  // It is not necessary for the PropagtionToken to return a principal,
     // because it is not user-centric.
  return "";
```

```
 }

/**
 * Returns the unique identifier of the token based upon information that
 * the provider considers makes it a unique token. This identifier is used
 * for caching purposes and might be used in combination with other token
 * unique IDs that are part of the same Subject.
 *
 * This method should return null if you want the accessID of the user to
 * represent its uniqueness. This is the typical scenario.
 *
 * @return String
 */
 public String getUniqueID()
 {
     // If you want to propagate the changes to this token, change the
     // value that this unique ID returns whenever the token is changed.
     // Otherwise, CSIv2 uses an existing session when everything else is
     // the same. This getUniqueID ischecked by CSIv2 to determine the
     // session lookup.
  return counter;
 }

/**
 * Gets the bytes to be sent across the wire. The information in the byte[]
 * needs to be enough to recreate the Token object at the target server.
 * @return byte[]
 */
 public byte[] getBytes ()
 {
  if (hashtable != null)
  {
   try
   {
     // Do this if the object is set to read-only during login commit
     // because this guarantees that no new data is set.
    if (isReadOnly() && tokenBytes == null)
     tokenBytes = com.ibm.wsspi.security.token.WSOpaqueTokenHelper.
               serialize(hashtable);

     // You can deserialize this in the downstream login module using
         //  WSOpaqueTokenHelper.deserialize()
    return tokenBytes;
   }
   catch (Exception e)
   {
    e.printStackTrace();
    return null;
   }
  }

  System.out.println("getBytes: returning null");
  return null;
 }

/**
```

```
 * Gets the name of the token, which is used to identify the byte[] in the
 * protocol message.
 * @return String
 */
 public String getName()
 {
  return this.getClass().getName();
 }

/**
 * Gets the version of the token as an short type. This code also is used
 * to identify the byte[] in the protocol message.
 * @return short
 */
 public short getVersion()
 {
  String[] version = getAttributes("version");

  if (version != null && version[0] != null)
   return new Short(version[0]).shortValue();

  System.out.println("getVersion: returning default of 1");
  return 1;
    }

/**
 * When called, the token becomes irreversibly read-only. The implementation
 * needs to ensure that any setter methods check that this read-only flag has
 * been set.
 */
 public void setReadOnly()
 {
  addAttribute("readonly", "true");
 }

/**
 * Called internally to see if the token is readonly
 */
 private boolean isReadOnly()
 {
  String[] readonly = getAttributes("readonly");

  if (readonly != null && readonly[0] != null)
   return new Boolean(readonly[0]).booleanValue();

  System.out.println("isReadOnly: returning default of false");
  return false;
 }

/**
 * Gets the attribute value based on the named value.
 * @param String key
 * @return String[]
 */
 public String[] getAttributes(String key)
 {
```

```
  ArrayList array = (ArrayList) hashtable.get(key);

  if (array != null && array.size() > 0)
  {
   return (String[]) array.toArray(new String[0]);
  }

  return null;
 }

/**
 * Sets the attribute name and value pair. Returns the previous values set
 * for the key, or returns null if the value is not previously set.
 * @param String key
 * @param String value
 * @returns String[];
 */
 public String[] addAttribute(String key, String value)
 {
  // Gets the current value for the key
  ArrayList array = (ArrayList) hashtable.get(key);

  if (!isReadOnly())
  {
   // Increments the counter to change the uniqueID
   counter++;

   // Copies the ArrayList to a String[] as it currently exists
   String[] old_array = null;
   if (array != null && array.size() > 0)
    old_array = (String[]) array.toArray(new String[0]);

   // Allocates a new ArrayList if one was not found
   if (array == null)
    array = new ArrayList();

   // Adds the String to the current array list
   array.add(value);

   // Adds the current ArrayList to the Hashtable
   hashtable.put(key, array);

   // Returns the old array
   return old_array;
  }

  return (String[]) array.toArray(new String[0]);
 }


/**
 * Gets the list of all of the attribute names present in the token.
 * @return java.util.Enumeration
 */
 public java.util.Enumeration getAttributeNames()
 {
```

```
  return hashtable.keys();
 }

/**
 * Returns a deep clone of this token. This is typically used by the session
 * logic of the CSIv2 server to create a copy of the token as it exists in the
 * session.
 * @return Object
 */
 public Object clone()
 {
  com.ibm.websphere.security.token.CustomPropagationTokenImpl deep_clone =
   new com.ibm.websphere.security.token.CustomPropagationTokenImpl();

  java.util.Enumeration keys = getAttributeNames();

  while (keys.hasMoreElements())
  {
   String key = (String) keys.nextElement();

   String[] list = (String[]) getAttributes(key);

   for (int i=0; i<list.length; i++)
    deep_clone.addAttribute(key, list[i]);
  }

      return deep_clone;
 }
}
```

## Example: custom PropagationToken login module

This file shows how to determine if the login is an initial login or a propagation login

```
public customLoginModule()
{
 public void initialize(Subject subject, CallbackHandler callbackHandler,
        Map sharedState, Map options)
 {
  // (For more information on what to do during initialization, see
     // "Custom login module development for a system login configuration" on page 78.)
 }

 public boolean login() throws LoginException
 {
  // (For more information on what to do during login, see
     // "Custom login module development for a system login configuration" on page 78.)

  // Handles the WSTokenHolderCallback to see if this is an initial
     // or propagation login.
  Callback callbacks[] = new Callback[1];
  callbacks[0] = new WSTokenHolderCallback("Authz Token List: ");

  try
  {
   callbackHandler.handle(callbacks);
```

```
 }
 catch (Exception e)
 {
  // handle exception
 }

 // Receives the ArrayList of TokenHolder objects (the serialized tokens)
 List authzTokenList = ((WSTokenHolderCallback) callbacks[0]).getTokenHolderList();

 if (authzTokenList != null)
 {
  // Iterates through the list looking for your custom token
  for (int i=0; i<authzTokenList.size(); i++)
  {
   TokenHolder tokenHolder = (TokenHolder)authzTokenList.get(i);

   // Looks for the name and version of your custom PropagationToken implementation
   if (tokenHolder.getName().equals("
             com.ibm.websphere.security.token.CustomPropagationTokenImpl") &&
       tokenHolder.getVersion() == 1)
   {
    // Passes the bytes into your custom PropagationToken constructor
          //  to deserialize
    customPropToken = new
     com.ibm.websphere.security.token.CustomPropagationTokenImpl(tokenHolder.
                 getBytes());

   }
  }
 }
 else // This is not a propagation login. Create a new instance of
         // your PropagationToken implementation
 {
  // Adds a new custom propagation token. This is an initial login
  customPropToken = new com.ibm.websphere.security.token.CustomPropagationTokenImpl();

  // Adds any initial attributes
  if (customPropToken != null)
  {
   customPropToken.addAttribute("key1", "value1");
   customPropToken.addAttribute("key1", "value2");
   customPropToken.addAttribute("key2", "value1");
   customPropToken.addAttribute("key3", "something different");
  }
 }

 // Note: You can add the token to the thread during commit in case
    // something happens during the login.
}

public boolean commit() throws LoginException
{
 // For more information on what to do during commit, see
    // "Custom login module development for a system login configuration" on page 78
 if (customPropToken != null)
 {
```

```
  // Sets the propagation token on the thread
  try
  {

   System.out.println(tc, "*** ADDED MY CUSTOM PROPAGATION TOKEN TO THE THREAD ***");
   // Prints out the values in the deserialized propagation token
   java.util.Enumeration keys = customPropToken.getAttributeNames();
   while (keys.hasMoreElements())
   {
    String key = (String) keys.nextElement();
    String[] list = (String[]) customPropToken.getAttributes(key);
    for (int k=0; k<list.length; k++)
    System.out.println("Key/Value: " + key + "/" + list[k]);
   }

   // This sets it on the thread using getName() + getVersion() as the key
   com.ibm.wsspi.security.token.WSSecurityPropagationHelper.addPropagationToken(
             customPropToken);
  }
  catch (Exception e)
  {
   // Handles exception
  }


  // Now you can verify that you have set it properly by trying to get
      // it back from the thread and print the values.
  try
  {
   // This gets the PropagationToken from the thread using getName()
         // and getVersion() parameters.
   com.ibm.wsspi.security.token.PropagationToken tempPropagationToken =
    com.ibm.wsspi.security.token.WSSecurityPropagationHelper.getPropagationToken
     ("com.ibm.websphere.security.token.CustomPropagationTokenImpl", 1);

   if  (tempPropagationToken != null)
   {
    System.out.println(tc, "*** RECEIVED MY CUSTOM PROPAGATION
              TOKEN FROM THE THREAD ***");
    // Prints out the values in the deserialized propagation token
    java.util.Enumeration keys = tempPropagationToken.getAttributeNames();
    while (keys.hasMoreElements())
    {
     String key = (String) keys.nextElement();
     String[] list = (String[]) tempPropagationToken.getAttributes(key);
     for (int k=0; k<list.length; k++)
     System.out.println("Key/Value: " + key + "/" + list[k]);
    }
   }
  }
  catch (Exception e)
  {
   // Handles exception
  }
 }
}
```

```
 // Defines your login module variables
 com.ibm.wsspi.security.token.PropagationToken customPropToken = null;

}
```

## Default AuthorizationToken

This article explains how WebSphere Application Server uses the default AuthorizationToken. Consider using the default AuthorizationToken when you are looking for a place to add string attributes that will get propagated downstream. However, make sure that the attributes that you add to the AuthorizationToken are specific to the user associated with the authenticated Subject. If they are not specific to a user, the attributes probably belong in the PropagationToken, which is also propagated with the request. For more information on the PropagationToken, see “Default PropagationToken” on page 282. To add attributes into the AuthorizationToken, you must plug in a custom login module into the various system login modules that are configured. Any login module configuration that has the com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule implementation configured can receive propagated information and can generate propagation information that can be sent outbound to another server.

If propagated attributes are not presented to the login configuration during an initial login, a default AuthorizationToken is created in the wsMapDefaultInboundLoginModule after the login occurs in the ltpaLoginModule. A reference to the default AuthorizationToken can be obtained from the login() method using the sharedState hashmap. You must plug in the custom login module after the wsMapDefaultInboundLoginModule implementation for WebSphere Application Server to see the default AuthorizationToken..

For more information on the Java Authentication and Authorization Service (JAAS) programming model, see “Security: Resources for learning” on page 21.

**Important:** Whenever you plug in a custom login module into the WebSphere Application Server login infrastructure, you must ensure that the code is trusted. When you add the login module into the *install_dir*/classes directory, it has Java 2 Security AllPermissions. It is recommended that you add your login module and other infrastructure classes into a private directory. However, if you use a private directory, modify the $(WAS_INSTALL_ROOT)/properties/server.policy file so that the private directory, Java archive (JAR) file, or both have the permissions needed to execute the application programming interfaces (API) called from the login module. Because the login module might run after the application code on the call stack, you might consider adding a doPrivileged code block so that you do not need to add additional permissions to your applications.

The following sample code shows you how to obtain a reference to the default AuthorizationToken from the login() method, how to add attributes to the token, and how to read from the existing attributes that are used for authorization.

```
public customLoginModule()
{
 public void initialize(Subject subject, CallbackHandler callbackHandler,
          Map sharedState, Map options)
 {
     // (For more information on initialization, see
     // "Custom login module development for a system login configuration" on page 78.)

  // Get a reference to the sharedState map that is passed in during initialization.
  _sharedState = sharedState;
```

```
}

public boolean login() throws LoginException
{
    // (For more information on what to during login, see
    // "Custom login module development for a system login configuration" on page 78.)

 // Look for the default AuthorizationToken in the shared state
 defaultAuthzToken  = (com.ibm.wsspi.security.token.AuthorizationToken)
      sharedState.get
    (com.ibm.wsspi.security.auth.callback.Constants.WSAUTHZTOKEN_KEY);

 // Might not always have one of these generated. It depends on the login
    // configuration setup.
 if (defaultAuthzToken != null)
 {
  try
  {
   // Add a custom attribute
   defaultAuthzToken.addAttribute("key1", "value1");

   // Determine all of the attributes and values that exist in the token.
   java.util.Enumeration listOfAttributes = defaultAuthorizationToken.
             getAttributeNames();

   while (listOfAttributes.hasMoreElements())
   {
    String key = (String) listOfAttributes.nextElement();

    String[] values = (String[]) defaultAuthorizationToken.getAttributes (key);

    for (int i=0;  i<values.length; i++)
    {
     System.out.println ("Key: " + key + ", Value[" + i + "]: "
                 + values[i]);
    }
   }

   // Read the existing uniqueID attribute.
   String[]  uniqueID = defaultAuthzToken.getAttributes
     (com.ibm.wsspi.security.token.AttributeNameConstants.
              WSCREDENTIAL_UNIQUEID);

    // Getthe uniqueID from the String[]
    String unique_id = (uniqueID != null &&
               uniqueID[0] != null) ? uniqueID[0] : "";

   // Read the existing expiration attribute.
   String[]  expiration = defaultAuthzToken.getAttributes
     (com.ibm.wsspi.security.token.AttributeNameConstants.
              WSCREDENTIAL_EXPIRATION);

    // An example of getting a long expiration value from the string array.
    long expire_time = 0;
    if (expiration != null && expiration[0] != null)
     expire_time = Long.parseLong(expiration[0]);
```

```
    // Read the existing display name attribute.
    String[]  securityName = defaultAuthzToken.getAttributes
      (com.ibm.wsspi.security.token.AttributeNameConstants.
               WSCREDENTIAL_SECURITYNAME);

     // Get the display name from the String[]
     String display_name = (securityName != null &&
                securityName[0] != null) ? securityName[0] : "";


    // Read the existing long securityName attribute.
    String[]  longSecurityName = defaultAuthzToken.getAttributes
     (com.ibm.wsspi.security.token.AttributeNameConstants.
             WSCREDENTIAL_LONGSECURITYNAME);

    // Get the long security name from the String[]
    String long_security_name = (longSecurityName != null &&
              longSecurityName[0] != null) ? longSecurityName[0] : "";


    // Read the existing group attribute.
    String[]  groupList = defaultAuthzToken.getAttributes
      (com.ibm.wsspi.security.token.AttributeNameConstants.
               WSCREDENTIAL_GROUPS);

    // Get the groups from the String[]
    ArrayList groups = new ArrayList();
    if (groupList != null)
    {
     for (int i=0; i<groupList.length; i++)
     {
      System.out.println ("group[" + i + "] = " + groupList[i]);
      groups.add(groupList[i]);
     }
    }
   }
   catch (Exception e)
   {
    throw new WSLoginFailedException (e.getMessage(), e);
   }
  }

 }

 public boolean commit() throws LoginException
 {
  // (For more information on what to do during commit, see
     // "Custom login module development for a system login configuration" on page 78.)

 }

 private java.util.Map _sharedState = null;
 private com.ibm.wsspi.security.token.AuthorizationToken defaultAuthzToken = null;
}
```

### Changing the TokenFactory associated with the default AuthorizationToken

When WebSphere Application Server generates a default AuthorizationToken, the application server utilizes the TokenFactory class that is specified using the com.ibm.wsspi.security.token.authorizationTokenFactory property. To modify this property using the administrative console, complete the following steps:

1. Click **Security > Global Security**.
2. Under Additional properties, click **Custom properties**.

The default TokenFactory that used is called com.ibm.ws.security.ltpa.AuthzPropTokenFactory. This token factory encodes the data, but does not encrypt the data in the AuthorizationToken. Because the AuthorizationToken typically flows over Common Secure Interoperability version 2 (CSIv2) using Secure Sockets Layer (SSL), there is no need to encrypt the token itself. However, if you need addition security for the AuthorizationToken, you can associate a different TokenFactory implementation with this property to get encryption. For example, if you associate com.ibm.ws.security.ltpa.LTPAToken2Factory with this property, the token uses AES encryption. However, you need to weigh the performance impacts against your security needs. Adding sensitive information to the AuthorizationToken is one reason to change the TokenFactory implementation to something that encrypts rather than just encodes.

If you want to perform your own signing and encryption of the default AuthorizationToken you must implement the following classes:

- com.ibm.wsspi.security.ltpa.Token
- com.ibm.wsspi.security.ltpa.TokenFactory

Your TokenFactory implementation instantiates and validates your token implementation. You can use the Lightweight Third Party Authentication (LTPA) keys that are passed into the initialize method of the TokenFactory or you can use your own keys. If you use your own keys, they must be the same everywhere in order to validate the tokens that are generated using those keys. See the Javadoc, available through a link on the front page of the information center, for more information on implementing your own custom TokenFactory. To associate your TokenFactory with the default AuthorizationToken, using the administrative console, complete the following steps:

1. Click **Security > Global Security**.
2. Under Additional properties, click **Custom properties**.
3. Locate the com.ibm.wsspi.security.token.authorizationTokenFactory property and verify that the value of this property matches your custom TokenFactory implementation.
4. Verify that your implementation classes are put into the *install directory*/classes directory so that the WebSphere class loader can load the classes.

## Implementing a custom AuthorizationToken

This task explains how you might create your own AuthorizationToken implementation, which is set in the login Subject and propagated downstream. The default AuthorizationToken usually is sufficient for propagating attributes that are user-specific. Consider writing your own implementation if you want to accomplish one of the following tasks:

- Isolate your attributes within your own implementation.
- Serialize the information using custom serialization. You must deserialize the bytes at the target and add that information back on the thread. This task also might include encryption and decryption.
- Affect the overall uniqueness of the Subject using the getUniqueID() API.

To implement a custom authorization token, you must complete the following steps:

1. Write a custom implementation of the AuthorizationToken interface. There are many different methods for implementing the AuthorizationToken interface. However, make sure that the methods required by

the AuthorizationToken interface and the token interface are fully implemented. After you implement this interface, you can place it in the `install_dir/classes` directory. Alternatively, you can place the class in any private directory. However, make sure that the WebSphere Application Server class loader can locate the class and that it is granted the appropriate permissions. You can add the Java archive (JAR) file or directory that contains this class into the `server.policy` file so that it has the necessary permissions that are needed by the server code.

**Tip:** All of the token types defined by the propagation framework have similar interfaces. Basically, the token types are marker interfaces that implement the com.ibm.wsspi.security.token.Token interface. This interface defines most of the methods. If you plan to implement more than one token type, consider creating an abstract class that implements the com.ibm.wsspi.security.token.Token interface. All of your token implementations, including the AuthorizationToken, might extend the abstract class and then most of the work is completed.

To see an implementation of AuthorizationToken, see "Example: com.ibm.wsspi.security.token.AuthorizationToken implementation"

2. Add and receive the custom AuthorizationToken during WebSphere Application Server logins This task is typically accomplished by adding a custom login module to the various application and system login configurations. However, in order to deserialize the information, you must plug in a custom login module, which is discussed in "Propagating a custom Java serializable object" on page 331. After the object is instantiated in the login module, you can the object to the Subject during the commit() method.

   If you only want to add information to the Subject to get propagated, see "Propagating a custom Java serializable object" on page 331. If you want to ensure that the information is propagated, want to do you own custom serialization, or want to specify the uniqueness for Subject caching purposes, then consider writing your own AuthorizationToken implementation.

   The code sample in "Example: custom AuthorizationToken login module" on page 306 shows how to determine if the login is an initial login or a propagation login. The difference between these login types is whether the WSTokenHolderCallback contains propagation data. If the callback does not contain propagation data, initialize a new custom AuthorizationToken implementation and set it into the Subject. If the callback contains propagation data, look for your specific custom AuthorizationToken TokenHolder instance, convert the byte[] back into your custom AuthorizationToken object, and set it back into the Subject. The code sample shows both instances.

   You can make your AuthorizationToken read-only in the commit phase of the login module. If you do not make the token read-only, then attributes can be added within your applications.

3. Add your custom login module to WebSphere Application Server system login configurations that already contain the com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule for receiving serialized versions of your custom authorization token

   Because this login module relies on information in the sharedState added by the com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule, add this login module after com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule. For information on how to add your custom login module to the existing login configurations, see "Custom login module development for a system login configuration" on page 78

After completing these steps, you have implemented a custom AuthorizationToken.

## Example: com.ibm.wsspi.security.token.AuthorizationToken implementation

Use this file to see an example of a AuthorizationToken implementation. The following sample code does not extend an abstract class, but rather implements the com.ibm.wsspi.security.token.AuthorizationToken interface directly. You can implement the interface directly, but it might cause you to write duplicate code. However, you might choose to implement the interface directly if there are considerable differences between how you handle the various token implementations.

For information on how to implement a custom AuthorizationToken, see “Implementing a custom AuthorizationToken” on page 300.

```
package com.ibm.websphere.security.token;

import com.ibm.websphere.security.WSSecurityException;
import com.ibm.websphere.security.auth.WSLoginFailedException;
import com.ibm.wsspi.security.token.*;
import com.ibm.websphere.security.WebSphereRuntimePermission;
import java.io.ByteArrayOutputStream;
import java.io.ByteArrayInputStream;
import java.io.DataOutputStream;
import java.io.DataInputStream;
import java.io.ObjectOutputStream;
import java.io.ObjectInputStream;
import java.io.OutputStream;
import java.io.InputStream;
import java.util.ArrayList;

public class CustomAuthorizationTokenImpl implements com.ibm.wsspi.security.
   token.AuthorizationToken
{
 private java.util.Hashtable hashtable = new java.util.Hashtable();
 private byte[] tokenBytes = null;
 private static long expire_period_in_millis = 2*60*60*1000;
  // 2 hours in millis, by default

/**
 * Constructor used to create initial AuthorizationToken instance
 */

 public CustomAuthorizationTokenImpl (String principal)
 {
  // Sets the principal in the token
  addAttribute("principal", principal);
  // Sets the token version
  addAttribute("version", "1");
  // Sets the token expiration
  addAttribute("expiration", new Long(System.currentTimeMillis() +
        expire_period_in_millis).toString());
 }

/**
 * Constructor used to deserialize the token bytes received during a
 * propagation login.
 */
 public CustomAuthorizationTokenImpl (byte[] token_bytes)
 {
  try
  {
   hashtable = (java.util.Hashtable) com.ibm.wsspi.security.token.
           WSOpaqueTokenHelper.deserialize(token_bytes);
  }
  catch (Exception e)
  {
   e.printStackTrace();
  }
 }

/**
 * Validates the token including expiration, signature, and so on.
 * @return boolean
 */

 public boolean isValid ()
 {
  long expiration = getExpiration();
```

```
  // if you set the expiration to 0, it does not expire
  if (expiration != 0)
  {
   // return if this token is still valid
   long current_time = System.currentTimeMillis();

   boolean valid = ((current_time < expiration) ? true : false);
   System.out.println("isValid: returning " + valid);
   return valid;
  }
  else
  {
   System.out.println("isValid: returning true by default");
   return true;
  }
 }

/**
 * Gets the expiration as a long.
 * @return long
 */
 public long getExpiration()
 {
  // Gets the expiration value from the hashtable
  String[] expiration = getAttributes("expiration");

  if (expiration != null && expiration[0] != null)
  {
   // The expiration is the first element. There should be only one expiration.
   System.out.println("getExpiration: returning " + expiration[0]);
   return new Long(expiration[0]).longValue();
  }

  System.out.println("getExpiration: returning 0");
  return 0;
 }

/**
 * Returns if this token should be forwarded/propagated downstream.
 * @return boolean
 */
 public boolean isForwardable()
 {
  // You can choose whether your token gets propagated. In some cases,
     // you might want it to be local only.
  return true;
 }

/**
 * Gets the principal that this Token belongs to. If this is an authorization token,
 * this principal string must match the authentication token principal string or the
 * message will be rejected.
 * @return String
 */
 public String getPrincipal()
 {
  // this might be any combination of attributes
  String[] principal = getAttributes("principal");

  if (principal != null && principal[0] != null)
  {
   return principal[0];
  }

  System.out.println("getExpiration: returning null");
  return null;
```

```
 }

/**
 * Returns a unique identifier of the token based upon the information that provider
 * considers makes this a unique token. This will be used for caching purposes
 * and might be used in combination with other token unique IDs that are part of
 * the same Subject.
 *
 * This method should return null if you want the accessID of the user to represent
 * uniqueness.  This is the typical scenario.
 *
 * @return String
 */
 public String getUniqueID()
 {
  // if you don't want to affect the cache lookup, just return NULL here.
  // return null;

  String cacheKeyForThisToken = "dynamic attributes";

  // if you do want to affect the cache lookup, return a string of
     // attributes that you want factored into the lookup.
  return cacheKeyForThisToken;
 }

/**
 * Gets the bytes to be sent across the wire. The information in the byte[]
 * needs to be enough to recreate the Token object at the target server.
 * @return byte[]
 */
 public byte[] getBytes ()
 {
  if (hashtable != null)
  {
   try
   {
    // Do this if the object is set to read-only during login commit,
    // because this makes sure that no new data gets set.
    if (isReadOnly() && tokenBytes == null)
     tokenBytes = com.ibm.wsspi.security.token.WSOpaqueTokenHelper.
                serialize(hashtable);

    // You can deserialize this in the downstream login module using
          // WSOpaqueTokenHelper.deserialize()
    return tokenBytes;
   }
   catch (Exception e)
   {
    e.printStackTrace();
    return null;
   }
  }

  System.out.println("getBytes: returning null");
  return null;
 }

/**
 * Gets the name of the token used to identify the byte[] in the protocol message.
 * @return String
 */
 public String getName()
 {
  return this.getClass().getName();
 }

/**
```

```
 * Gets the version of the token as an short. This also is used to identify the
 * byte[] in the protocol message.
 * @return short
 */
 public short getVersion()
 {
  String[] version = getAttributes("version");

  if (version != null && version[0] != null)
   return new Short(version[0]).shortValue();

  System.out.println("getVersion: returning default of 1");
  return 1;
    }

/**
 * When called, the token becomes irreversibly read-only. The implementation
 * needs to ensure that any setter methods check that this flag has been set.
 */
 public void setReadOnly()
 {
  addAttribute("readonly", "true");
 }

/**
 * Called internally to see if the token is read-only
 */
 private boolean isReadOnly()
 {
  String[] readonly = getAttributes("readonly");

  if (readonly != null && readonly[0] != null)
   return new Boolean(readonly[0]).booleanValue();

  System.out.println("isReadOnly: returning default of false");
  return false;
 }

/**
 * Gets the attribute value based on the named value.
 * @param String key
 * @return String[]
 */
 public String[] getAttributes(String key)
 {
  ArrayList array = (ArrayList) hashtable.get(key);

  if (array != null && array.size() > 0)
  {
   return (String[]) array.toArray(new String[0]);
  }

  return null;
 }

/**
 * Sets the attribute name and value pair. Returns the previous values set for key,
 * or null if not previously set.
 * @param String key
 * @param String value
 * @returns String[];
 */
 public String[] addAttribute(String key, String value)
 {
  // Gets the current value for the key
  ArrayList array = (ArrayList) hashtable.get(key);
```

```
  if (!isReadOnly())
  {
   // Copies the ArrayList to a String[] as it currently exists
   String[] old_array = null;
   if (array != null && array.size() > 0)
    old_array = (String[]) array.toArray(new String[0]);

   // Allocates a new ArrayList if one was not found
   if (array == null)
    array = new ArrayList();

   // Adds the String to the current array list
   array.add(value);

   // Adds the current ArrayList to the Hashtable
   hashtable.put(key, array);

   // Returns the old array
   return old_array;
  }

  return (String[]) array.toArray(new String[0]);
 }


/**
 * Gets the list of all attribute names present in the token.
 * @return java.util.Enumeration
 */
 public java.util.Enumeration getAttributeNames()
 {
  return hashtable.keys();
 }

/**
 * Returns a deep copying of this token, if necessary.
 * @return Object
 */
 public Object clone()
 {
  com.ibm.websphere.security.token.CustomAuthorizationTokenImpl deep_clone =
   new com.ibm.websphere.security.token.CustomAuthorizationTokenImpl();

  java.util.Enumeration keys = getAttributeNames();

  while (keys.hasMoreElements())
  {
   String key = (String) keys.nextElement();

   String[] list = (String[]) getAttributes(key);

   for (int i=0; i<list.length; i++)
    deep_clone.addAttribute(key, list[i]);
  }

      return deep_clone;
 }
}
```

## Example: custom AuthorizationToken login module

This file shows how to determine if the login is an initial login or a propagation login

```
public customLoginModule()
{
 public void initialize(Subject subject, CallbackHandler callbackHandler,
```

```
    Map sharedState, Map options)
{
    // (For more information on  what to do during initialization, see
    // "Custom login module development for a system login configuration" on page 78.)
 _sharedState = sharedState;
}

public boolean login() throws LoginException
{
    // (For more information on what do during login, see
    // "Custom login module development for a system login configuration" on page 78.)

 // Handles the WSTokenHolderCallback to see if this is an initial or
    // propagation login.
 Callback callbacks[] = new Callback[1];
 callbacks[0] = new WSTokenHolderCallback("Authz Token List: ");

 try
 {
  callbackHandler.handle(callbacks);
 }
 catch (Exception e)
 {
  // Handles exception
 }

 // Receives the ArrayList of TokenHolder objects (the serialized tokens)
 List authzTokenList = ((WSTokenHolderCallback) callbacks[0]).getTokenHolderList();

 if (authzTokenList != null)
 {
  // Iterates through the list looking for your custom token
  for (int i=0; i
  for (int i=0; i<authzTokenList.size(); i++)
  {
   TokenHolder tokenHolder = (TokenHolder)authzTokenList.get(i);

         // Looks for the name and version of your custom AuthorizationToken
         // implementation
   if (tokenHolder.getName().equals("com.ibm.websphere.security.token.
            CustomAuthorizationTokenImpl") &&
       tokenHolder.getVersion() == 1)
   {
           // Passes the bytes into your custom AuthorizationToken constructor
           // to deserialize
    customAuthzToken = new
     com.ibm.websphere.security.token.CustomAuthorizationTokenImpl(
                 tokenHolder.getBytes());

   }
  }
 }
 else
    // This is not a propagation login. Create a new instance of your
    // AuthorizationToken implementation
 {
```

```
      // Gets the prinicpal from the default AuthenticationToken. This must match
      // all tokens.
  defaultAuthToken = (com.ibm.wsspi.security.token.AuthenticationToken)
   sharedState.get(com.ibm.wsspi.security.auth.callback.Constants.WSAUTHTOKEN_KEY);
  String principal = defaultAuthToken.getPrincipal();

      // Adds a new custom authorization token. This is an initial login. Pass the
      // principal into the constructor
  customAuthzToken = new com.ibm.websphere.security.token.
          CustomAuthorizationTokenImpl(principal);

  // Adds any initial attributes
  if (customAuthzToken != null)
  {
   customAuthzToken.addAttribute("key1", "value1");
   customAuthzToken.addAttribute("key1", "value2");
   customAuthzToken.addAttribute("key2", "value1");
   customAuthzToken.addAttribute("key3", "something different");
  }
 }

    // Note: You can add the token to the Subject during commit in case something
    // happens during the login.
}

public boolean commit() throws LoginException
{
   // (For more information on what to do during a commit, see
   // "Custom login module development for a system login configuration" on page 78.)

 if (customAut  // (hzToken != null)
 {
  // sSets the customAuthzToken token into the Subject
  try
  {
   public final AuthorizationToken customAuthzTokenPriv = customAuthzToken;
         // Do this in a doPrivileged code block so that application code does not
         // need to add additional permissions
   java.security.AccessController.doPrivileged(new java.security.PrivilegedAction()
   {
    public Object run()
    {
     try
     {
              // Adds the custom authorization token if it is not null
              //  and not already in the Subject
                             if ((customAuthzTokenPriv != null) &&
        (!subject.getPrivateCredentials().contains(customAuthzTokenPriv)))
      {
       subject.getPrivateCredentials().add(customAuthzTokenPriv);
      }
     }
     catch (Exception e)
     {
      throw new WSLoginFailedException (e.getMessage(), e);
     }
```

```
      return null;
     }
    });
   }
   catch (Exception e)
   {
    throw new WSLoginFailedException (e.getMessage(), e);
   }
  }
 }

 // Defines your login module variables
 com.ibm.wsspi.security.token.AuthorizationToken customAuthzToken = null;
 com.ibm.wsspi.security.token.AuthenticationToken defaultAuthToken = null;
 java.util.Map _sharedState = null;
}
```

## Default SingleSignonToken

Do not use the default SingleSignonToken in service provider code. This default token is used by the WebSphere Application Server run-time code only. There are size limitations for this token when it is added as an HTTP cookie. If you need to create an HTTP cookie using this token framework, you can implement a custom SingleSignonToken. To implement a custom SingleSignonToken, see "Implementing a custom SingleSignonToken" on page 310 for more information.

**Changing the TokenFactory associated with the default SingleSignonToken**

When default SingleSignonToken is generated, the application server utilizes the TokenFactory class that is specified using the com.ibm.wsspi.security.token.singleSignonTokenFactory property. To modify this property using the administrative console, complete the following steps:

1. Click **Security > Global Security**.
2. Under Additional properties, click **Custom properties**.

The default TokenFactory specified for this property is called com.ibm.ws.security.ltpa.LTPAToken2Factory. This token factory creates an SSO token called LtpaToken2, which WebSphere Application Server uses for propagation. This TokenFactory uses the AES/CBC/PKCS5Padding cipher. If you change this TokenFactory, you lose the interoperability with any servers running a version of WebSphere Application Server prior to version 5.1.1 that use the default TokenFactory. Only servers running WebSphere Application Server Version 5.1.1 or later with propagation enabled are aware of the LtpaToken2 cookie. However, this is not a problem if all of your application servers use WebSphere Application Server Version 5.1.1 or later and all of your servers use your new TokenFactory.

If you need to perform your own signing and encryption of the default SingleSignonToken, you must implement the following classes:

- com.ibm.wsspi.security.ltpa.Token
- com.ibm.wsspi.security.ltpa.TokenFactory

Your TokenFactory implementation instantiates (createToken) and validates (validateTokenBytes) your token implementation. You can use the LTPA keys passed into the initialize method of the TokenFactory or you can use your own keys. If you use your own keys, they must be the same everywhere in order to validate the tokens that are generated using those keys. See the Javadoc, available through a link on the

front page of the information center, for more information on implementing your own custom TokenFactory. To associate your TokenFactory with the default SingleSignonToken using the administrative console, complete the following steps:

1. Click **Security > Global Security**.
2. Under Additional properties, click **Custom properties**.
3. Locate the com.ibm.wsspi.security.token.singleSignonTokenFactory property and verify that the value of this property matches your custom TokenFactory implementation.
4. Verify that your implementation classes are put into the `install directory/classes` directory so that the WebSphere class loader can load the classes.

## Implementing a custom SingleSignonToken

This task explains how to create your own SingleSignonToken implementation, which is set in the login Subject and added to the HTTP response as an HTTP cookie. The cookie name is the concatenation of the SingleSignonToken.getName() application programming interface (API) and the SingleSignonToken.getVersion() API. There is no delimiter. When you add a SingleSignonToken to the Subject, it also gets propagated horizontally and downstream in case the Subject is used for other Web requests. You must deserialize your custom SingleSignonToken when you receive it from a propagation login. Consider writing your own implementation if you want to accomplish one of the following:

- Isolate your attributes within your own implementation.
- Serialize the information using custom serialization. It is recommended that you encrypt the information because it is out to the HTTP response and is available on the Internet. You must deserialize or decrypt the bytes at the target and add that information back into the Subject.
- Affect the overall uniqueness of the Subject using the getUniqueID() API

To implement a custom SingleSignonToken, you must complete the following steps:

1. Write a custom implementation of the SingleSignonToken interface.

   There are many different methods for implementing the SingleSignonToken interface. However, make sure that the methods required by the SingleSignonToken interface and the token interface are fully implemented. After you implement this interface, you can place it in the `install_dir/classes` directory. Alternatively, you can place the class in any private directory. However, make sure that the WebSphere Application Server class loader can locate the class and that it is granted the appropriate permissions. You can add the Java archive (JAR) file or directory that contains this class into the `server.policy` file so that it has the necessary permissions that are needed by the server code.

   **Tip:** All of the token types defined by the propagation framework have similar interfaces. Basically, the token types are marker interfaces that implement the com.ibm.wsspi.security.token.Token interface. This interface defines most of the methods. If you plan to implement more than one token type, consider creating an abstract class that implements the com.ibm.wsspi.security.token.Token interface. All of your token implementations, including the SingleSignonToken, might extend the abstract class and then most of the work is completed.

   To see an implementation of the SingleSignonToken, see "Example: com.ibm.wsspi.security.token.SingleSignonToken implementation" on page 311
2. Add and receive the custom SingleSignonToken during WebSphere Application Server logins. This task is typically accomplished by adding a custom login module to the various application and system login configurations. However, in order to deserialize the information, you will need to plug in a custom login module, which is discussed in a subsequent step. After the object is instantiated in the login module, you can add it to the Subject during the commit() method.

   The code sample in "Example: custom SingleSignonToken login module" on page 316 shows how to determine if the login is an initial login or a propagation login. The difference is whether the WSTokenHolderCallback contains propagation data. If the token does not contain propagation data, initialize a new custom SingleSignonToken implementation and set it into the Subject. Also, look for the

HTTP cookie from the HTTP request if the HTTP request object is available in the callback. You can get your custom SingleSignonToken both from a horizontal propagation login and from the HTTP request. However, it is recommended that you make the token available in both places because then the information arrives at any front-end application server even if that server that does not support propagation.

You can make your SingleSignonToken read-only in the commit phase of the login module. If you make the token read-only, additional attributes cannot be added within your applications.

**Restriction:**

- HTTP cookies have a size limitation so do not add too much data to this token.
- The WebSphere Application Server run time does not handle cookies that it does not generate, so this cookie is not used by the run time.
- The SingleSignonToken object, when in the Subject, does affect the cache lookup of the Subject if you return something in the getUniqueID() method.

3. Get the HTTP cookie from the HTTP request object during login or from an application. The sample code, found in "Example: HTTP cookie retrieval" on page 318 shows how you can retrieve the HTTP cookie from the HTTP request, decode the cookie so that it is back to your original bytes, and create your custom SingleSignonToken object from the bytes.
4. Add your custom login module to WebSphere Application Server system login configurations that already contain the com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule for receiving serialized versions of your custom propagation token Because this login module relies on information in the sharedState added by the com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule, add this login module after com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule.

   For information on adding your custom login module into the existing login configurations, see "Custom login module development for a system login configuration" on page 78

After completing these steps, you have implemented a custom AuthorizationToken.

## Example: com.ibm.wsspi.security.token.SingleSignonToken implementation

Use this file to see an example of a SingleSignon implementation. The following sample code does not extend an abstract class, but rather implements the com.ibm.wsspi.security.token.SingleSignonToken interface directly. You can implement the interface directly, but it might cause you to write duplicate code. However, you might choose to implement the interface directly if there are considerable differences between how you handle the various token implementations.

For information on how to implement a custom SingleSignonToken, see "Implementing a custom SingleSignonToken" on page 310.

```
package com.ibm.websphere.security.token;

import com.ibm.websphere.security.WSSecurityException;
import com.ibm.websphere.security.auth.WSLoginFailedException;
import com.ibm.wsspi.security.token.*;
import com.ibm.websphere.security.WebSphereRuntimePermission;
import java.io.ByteArrayOutputStream;
import java.io.ByteArrayInputStream;
import java.io.DataOutputStream;
import java.io.DataInputStream;
import java.io.ObjectOutputStream;
import java.io.ObjectInputStream;
import java.io.OutputStream;
import java.io.InputStream;
import java.util.ArrayList;

public class CustomSingleSignonTokenImpl implements com.ibm.wsspi.security.
   token.SingleSignonToken
```

```
{
 private java.util.Hashtable hashtable = new java.util.Hashtable();
 private byte[] tokenBytes = null;
   // 2 hours in millis, by default
 private static long expire_period_in_millis = 2*60*60*1000;

/**
 * Constructor used to create initial SingleSignonToken instance
 */

 public CustomSingleSignonTokenImpl (String principal)
 {
  // set the principal in the token
  addAttribute("principal", principal);
  // set the token version
  addAttribute("version", "1");
  // set the token expiration
  addAttribute("expiration", new Long(System.currentTimeMillis() +
        expire_period_in_millis).toString());
 }

/**
 * Constructor used to deserialize the token bytes received during a propagation login.
 */
 public CustomSingleSignonTokenImpl (byte[] token_bytes)
 {
  try
  {
   // you should implement a decryption algorithm to decrypt the cookie bytes
   hashtable = (java.util.Hashtable) some_decryption_algorithm (token_bytes);
  }
  catch (Exception e)
  {
   e.printStackTrace();
  }
 }

/**
 * Validates the token including expiration, signature, and so on.
 * @return boolean
 */

 public boolean isValid ()
 {
  long expiration = getExpiration();

  // if you set the expiration to 0, it's does not expire
  if (expiration != 0)
  {
   // return if this token is still valid
   long current_time = System.currentTimeMillis();

   boolean valid = ((current_time < expiration) ? true : false);
   System.out.println("isValid: returning " + valid);
   return valid;
  }
  else
  {
   System.out.println("isValid: returning true by default");
   return true;
  }
 }

/**
 * Gets the expiration as a long.
 * @return long
 */
```

```
 public long getExpiration()
 {
  // get the expiration value from the hashtable
  String[] expiration = getAttributes("expiration");

  if (expiration != null && expiration[0] != null)
  {
   // expiration will always be the first element (should only be one)
   System.out.println("getExpiration: returning " + expiration[0]);
   return new Long(expiration[0]).longValue();
  }

  System.out.println("getExpiration: returning 0");
  return 0;
 }

/**
 * Returns if this token should be forwarded/propagated downstream.
 * @return boolean
 */
 public boolean isForwardable()
 {
  // You can choose whether your token gets propagated or not, in some cases
     // you might want it to be local only.
  return true;
 }

/**
 * Gets the principal that this Token belongs to. If this is an authorization token,
 * this principal string must match the authentication token principal string or the
 * message will be rejected.
 * @return String
 */
 public String getPrincipal()
 {
  // this could be any combination of attributes
  String[] principal = getAttributes("principal");

  if (principal != null && principal[0] != null)
  {
   return principal[0];
  }

  System.out.println("getExpiration: returning null");
  return null;
 }

/**
 * Returns a unique identifier of the token based upon information the provider
 * considers makes this a unique token.  This will be used for caching purposes
 * and may be used in combination with outher token unique IDs that are part of
 * the same Subject.
 *
 * This method should return null if you want the accessID of the user to represent
 * uniqueness.  This is the typical scenario.
 *
 * @return String
 */
 public String getUniqueID()
 {
  // this could be any combination of attributes
  return getPrincipal();
 }

/**
 * Gets the bytes to be sent across the wire.  The information in the byte[]
 * needs to be enough to recreate the Token object at the target server.
```

```
 * @return byte[]
 */
 public byte[] getBytes ()
 {
  if (hashtable != null)
  {
   try
   {
    // do this if the object is set read-only during login commit,
    // since this guarantees no new data gets set.
    if (isReadOnly() && tokenBytes == null)
     tokenBytes = some_encryption_algorithm (hashtable);

    // you can deserialize the tokenBytes using a similiar decryption algorithm.
    return tokenBytes;
   }
   catch (Exception e)
   {
    e.printStackTrace();
    return null;
   }
  }

  System.out.println("getBytes: returning null");
  return null;
 }

/**
 * Gets the name of the token, used to identify the byte[] in the protocol message.
 * @return String
 */
 public String getName()
 {
  return "myCookieName";
 }

/**
 * Gets the version of the token as an short.  This is also used to identify the
 * byte[] in the protocol message.
 * @return short
 */
 public short getVersion()
 {
  String[] version = getAttributes("version");

  if (version != null && version[0] != null)
   return new Short(version[0]).shortValue();

  System.out.println("getVersion: returning default of 1");
  return 1;
    }

/**
 * When called, the token becomes irreversibly read-only.  The implementation
 * needs to ensure any setter methods check that this has been set.
 */
 public void setReadOnly()
 {
  addAttribute("readonly", "true");
 }

/**
 * Called internally to see if the token is readonly
 */
 private boolean isReadOnly()
 {
  String[] readonly = getAttributes("readonly");
```

```
  if (readonly != null && readonly[0] != null)
   return new Boolean(readonly[0]).booleanValue();

  System.out.println("isReadOnly: returning default of false");
  return false;
 }

/**
 * Gets the attribute value based on the named value.
 * @param String key
 * @return String[]
 */
 public String[] getAttributes(String key)
 {
  ArrayList array = (ArrayList) hashtable.get(key);

  if (array != null && array.size() > 0)
  {
   return (String[]) array.toArray(new String[0]);
  }

  return null;
 }

/**
 * Sets the attribute name/value pair.  Returns the previous values set for key,
 * or null if not previously set.
 * @param String key
 * @param String value
 * @returns String[];
 */
 public String[] addAttribute(String key, String value)
 {
  // get the current value for the key
  ArrayList array = (ArrayList) hashtable.get(key);

  if (!isReadOnly())
  {
   // copy the ArrayList to a String[] as it currently exists
   String[] old_array = null;
   if (array != null && array.size() > 0)
    old_array = (String[]) array.toArray(new String[0]);

   // allocate a new ArrayList if one was not found
   if (array == null)
    array = new ArrayList();

   // add the String to the current array list
   array.add(value);

   // add the current ArrayList to the Hashtable
   hashtable.put(key, array);

   // return the old array
   return old_array;
  }

  return (String[]) array.toArray(new String[0]);
 }


/**
 * Gets the List of all attribute names present in the token.
 * @return java.util.Enumeration
 */
 public java.util.Enumeration getAttributeNames()
```

```
 {
  return hashtable.keys();
 }

/**
 * Returns a deep copying of this token, if necessary.
 * @return Object
 */
 public Object clone()
 {
  com.ibm.websphere.security.token.CustomSingleSignonImpl deep_clone =
   new com.ibm.websphere.security.token.CustomSingleSignonTokenImpl();

  java.util.Enumeration keys = getAttributeNames();

  while (keys.hasMoreElements())
  {
   String key = (String) keys.nextElement();

   String[] list = (String[]) getAttributes(key);

   for (int i=0; i<list.length; i++)
    deep_clone.addAttribute(key, list[i]);
  }

      return deep_clone;
 }
}
```

## Example: custom SingleSignonToken login module

This file shows how to determine if the login is an initial login or a propagation login

```
public customLoginModule()
{
 public void initialize(Subject subject, CallbackHandler callbackHandler,
     Map sharedState, Map options)
 {
     // (For more information on initialization, see
     // "Custom login module development for a system login configuration" on page 78.)
  _sharedState = sharedState;
 }

 public boolean login() throws LoginException
 {
     // (For more information on what to do during login, see
     // "Custom login module development for a system login configuration" on page 78.)

     // Handles the WSTokenHolderCallback to see if this is an initial or
     // propagation login.
  Callback callbacks[] = new Callback[1];
  callbacks[0] = new WSTokenHolderCallback("Authz Token List: ");

  try
  {
   callbackHandler.handle(callbacks);
  }
  catch (Exception e)
  {
   // handle exception
  }
```

```
 // Receives the ArrayList of TokenHolder objects (the serialized tokens)
 List authzTokenList = ((WSTokenHolderCallback) callbacks[0]).getTokenHolderList();

 if (authzTokenList != null)
 {
  // iterate through the list looking for your custom token
  for (int i=0; i
  for (int i=0; i<authzTokenList.size(); i++)
  {
   TokenHolder tokenHolder = (TokenHolder)authzTokenList.get(i);

   // Looks for the name and version of your custom SingleSignonToken
         // implementation
   if (tokenHolder.getName().equals("myCookieName")
             && tokenHolder.getVersion() == 1)
   {
    // Passes the bytes into your custom SingleSignonToken constructor
           // to deserialize
    customSSOToken = new
     com.ibm.websphere.security.token.CustomSingleSignonTokenImpl
                  (tokenHolder.getBytes());

   }
  }
 }
 else
         // This is not a propagation login. Create a new instance of your
         // SingleSignonToken implementation
 {
      // Gets the principal from the default SingleSignonToken. This principal
      //  must match all tokens.
  defaultAuthToken = (com.ibm.wsspi.security.token.AuthenticationToken)
   sharedState.get(com.ibm.wsspi.security.auth.callback.Constants.WSAUTHTOKEN_KEY);
  String principal = defaultAuthToken.getPrincipal();

  // Adds a new custom single signon (SSO) token. This is an initial login.
      //  Pass the principal into the constructor
  customSSOToken = new com.ibm.websphere.security.token.
          CustomSingleSignonTokenImpl(principal);

  // add any initial attributes
  if (customSSOToken != null)
  {
   customSSOToken.addAttribute("key1", "value1");
   customSSOToken.addAttribute("key1", "value2");
   customSSOToken.addAttribute("key2", "value1");
   customSSOToken.addAttribute("key3", "something different");
  }
 }

    // Note: You can add the token to the Subject during commit in case something
    // happens during the login.
}

public boolean commit() throws LoginException
```

```
{
    // (For more information on what to do during commit, see
    // "Custom login module development for a system login configuration" on page 78.)

 if (customSSOToken != null)
 {
  // Sets the customSSOToken token into the Subject
  try
  {
   public final SingleSignonToken customSSOTokenPriv = customSSOToken;
         // Do this in a doPrivileged code block so that application code does not
         // need to add additional permissions
   java.security.AccessController.doPrivileged(new java.security.PrivilegedAction()
   {
    public Object run()
    {
     try
     {
      // Adds the custom SSO token if it is not null and
                //  not already in the Subject
                               if ((customSSOTokenPriv != null) &&
        (!subject.getPrivateCredentials().
                         contains(customSSOTokenPriv)))
      {
       subject.getPrivateCredentials().
                     add(customSSOTokenPriv);
      }
     }
     catch (Exception e)
     {
      throw new WSLoginFailedException (e.getMessage(), e);
     }

     return null;
    }
   });
  }
  catch (Exception e)
  {
   throw new WSLoginFailedException (e.getMessage(), e);
  }
 }
}

// Defines your login module variables
com.ibm.wsspi.security.token.SingleSignonToken customSSOToken = null;
com.ibm.wsspi.security.token.AuthenticationToken defaultAuthToken = null;
java.util.Map _sharedState = null;
}
```

## Example: HTTP cookie retrieval

Use this file to see an example of how to retrieve a cookie from an HTTP request, decode the cookie so that it is back to your original bytes, and create your custom SingleSignonToken object from the bytes. This example shows how to complete these steps from a login module. However, you also can complete these steps using a servlet.

For information on how to implement a custom SingleSignonToken, see “Implementing a custom SingleSignonToken” on page 310.

```
public customLoginModule()
{
 public void initialize(Subject subject, CallbackHandler callbackHandler,
     Map sharedState, Map options)
 {
     // (For more information on what to do during initialization, see
     // "Custom login module development for a system login configuration" on page 78.)
  _sharedState = sharedState;
 }

 public boolean login() throws LoginException
 {
     // (For more information on what to do during login, see
     // "Custom login module development for a system login configuration" on page 78.)

     // Handles the WSTokenHolderCallback to see if this is an
     // initial or propagation login.
  Callback callbacks[] = new Callback[2];
  callbacks[0] = new WSTokenHolderCallback("Authz Token List: ");
  callbacks[1] = new WSServletRequestCallback("HttpServletRequest: ");

  try
  {
   callbackHandler.handle(callbacks);
  }
  catch (Exception e)
  {
   // Handles the exception
  }

  // receive the ArrayList of TokenHolder objects (the serialized tokens)
  List authzTokenList = ((WSTokenHolderCallback) callbacks[0]).getTokenHolderList();
  javax.servlet.http.HttpServletRequest request =
        ((WSServletRequestCallback) callbacks[1]).getHttpServletRequest();

  if (request != null)
  {

   // Checks if the cookie is present
   javax.servlet.http.Cookie[] cookies = request.getCookies();
   String[] cookieStrings = getCookieValues (cookies, "myCookeName1");

   if (cookieStrings != null)
   {
    String cookieVal = null;
    for (int n=0;n<cookieStrings.length;n++)
    {
     cookieVal = cookieStrings[n];
     if (cookieVal.length()>0)
     {
              // Removes the cookie encoding from the cookie to get
              // your custom bytes
      byte[] cookieBytes =
```

```
      com.ibm.websphere.security.WSSecurityHelper.
                    convertCookieStringToBytes(cookieVal);
     customSSOToken =
      new com.ibm.websphere.security.token.
                    CustomSingleSignonTokenImpl(cookieBytes);

              // Now that you have your cookie from the request,
              // you can do something with it here, or add it
              // to the Subject in the commit() method for use later.
     if (debug || tc.isDebugEnabled())
     {
      Systen.out.println("*** GOT MY CUSTOM SSO TOKEN FROM
                    THE REQUEST ***");
     }
    }
   }
  }
 }

}

public boolean commit() throws LoginException
{
    // (For more information on what to during a commit, see
    // "Custom login module development for a system login configuration" on page 78.)

 if (customSSOToken != null)
 {
  // Sets the customSSOToken token into the Subject
  try
  {
   public final SingleSignonToken customSSOTokenPriv = customSSOToken;
         // Do this in a doPrivileged code block so that application code does not
         // need to add additional permissions
   java.security.AccessController.doPrivileged(new java.security.PrivilegedAction()
   {
    public Object run()
    {
     try
     {
                // Add the custom SSO token if it is not null and not
                // already in the Subject
                               if ((customSSOTokenPriv != null) &&
        (!subject.getPrivateCredentials().
                       contains(customSSOTokenPriv)))
      {
       subject.getPrivateCredentials().add(customSSOTokenPriv);
      }
     }
     catch (Exception e)
     {
      throw new WSLoginFailedException (e.getMessage(), e);
     }

     return null;
    }
```

```
    });
   }
   catch (Exception e)
   {
    throw new WSLoginFailedException (e.getMessage(), e);
   }
  }
 }

 // Private method to get the specific cookie from the request
 private String[] getCookieValues (Cookie[] cookies, String hdrName)
 {
  Vector retValues = new Vector();
  int numMatches=0;
  if (cookies != null)
  {
   for (int i = 0; i < cookies.length; ++i)
   {
    if (hdrName.equals(cookies[i].getName()))
    {
     retValues.add(cookies[i].getValue());
     numMatches++;
     System.out.println(cookies[i].getValue());
    }
   }
  }

  if (retValues.size()>0)
   return (String[]) retValues.toArray(new String[numMatches]);
  else
   return null;
 }

 // Defines your login module variables
 com.ibm.wsspi.security.token.SingleSignonToken customSSOToken = null;
 com.ibm.wsspi.security.token.AuthenticationToken defaultAuthToken = null;
 java.util.Map _sharedState = null;
}
```

## Default AuthenticationToken

Do not use the default AuthenticationToken in service provider code. This default token is used by the WebSphere Application Server run-time code only and is authentication mechanism specific. Any modifications to this token by service provider code can potentially cause interoperability problems. If you need to create an authentication token for custom usage, see “Implementing a custom AuthenticationToken” on page 322 for more information.

### Changing the TokenFactory associated with the default AuthenticationToken

When WebSphere Application Server generates a default AuthenticationToken, the application server utilizes the TokenFactory class that is specified using the com.ibm.wsspi.security.token.authenticationTokenFactory property. To modify this property using the administrative console, complete the following steps:

1. Click **Security > Global Security**.
2. Under Additional properties, click **Custom properties**.

The default TokenFactory specified for this property is called com.ibm.ws.security.ltpa.LTPATokenFactory. The LTPATokenFactory uses the DESede/ECB/PKCS5Padding cipher. This token factory creates an interoperable Lightweight Third Party Authentication (LTPA) token. If you change this TokenFactory, you lose the interoperability with any servers running a version of WebSphere Application Server prior to version 5.1.1 and any other servers that do not support the new TokenFactory implementation. However, this is not a problem if all of your application servers use WebSphere Application Server Version 5.1.1 or later and all of your servers use your new TokenFactory.

If you associate com.ibm.ws.security.ltpa.LTPAToken2Factory with the com.ibm.wsspi.security.token.authenticationTokenFactory property, the token is AES encrypted. However, you need to weigh the performance against your security needs. By doing this, you might add additional attributes to the AuthenticationToken in the Subject during a login that are available downstream.

If you need to perform your own signing and encryption of the default AuthenticationToken, you must implement the following classes:
- com.ibm.wsspi.security.ltpa.Token
- com.ibm.wsspi.security.ltpa.TokenFactory

Your TokenFactory implementation instantiates (createToken) and validates (validateTokenBytes) your token implementation. You can use the LTPA keys passed into the initialize method of the TokenFactory or you can use your own keys. If you use your own keys, they must be the same everywhere in order to validate the tokens that are generated using those keys. See the Javadoc, available through a link on the front page of the information center, for more information on implementing your own custom TokenFactory. To associate your TokenFactory with the default AuthenticationToken using the administrative console, complete the following steps:
1. Click **Security > Global Security**.
2. Under Additional properties, click **Custom properties**.
3. Locate the com.ibm.wsspi.security.token.authenticationTokenFactory property and verify that the value of this property matches your custom TokenFactory implementation.
4. Verify that your implementation classes are put into the *install directory*/`classes` directory so that the WebSphere class loader can load the classes.

## Implementing a custom AuthenticationToken

This task explains how you might create your own AuthenticationToken implementation, which is set in the login Subject and propagated downstream. This implementation enables you to specify an authentication token that can be used by a custom login module or application. Consider writing your own implementation if you want to accomplish one of the following tasks:
- Isolate your attributes within your own implementation.
- Serialize the information using custom serialization. You must deserialize the bytes at the target and add that information back on the thread. This task also might include encryption and decryption.
- Affect the overall uniqueness of the Subject using the getUniqueID() API.

**Important:** Custom AuthenticationToken implementations are not used by the security run time in WebSphere Application Server to enforce authentication. WebSphere Application Security run time uses this token in the following situations only:
- Call the getBytes() method for serialization
- Call the getForwardable() method to determine whether to serialize the AuthenticationToken.
- Call the getUniqueId() method for uniqueness
- Call the getName() and the getVersion() methods for adding serialized bytes to the TokenHolder that is sent downstream

All of the other uses are custom implementations.

To implement a custom authentication token, you must complete the following steps:

1. Write a custom implementation of the AuthenticationToken interface. There are many different methods for implementing the AuthenticationToken interface. However, make sure that the methods required by the AuthenticationToken interface and the token interface are fully implemented. After you implement this interface, you can place it in the *install_dir*/classes directory. Alternatively, you can place the class in any private directory. However, make sure that the WebSphere Application Server class loader can locate the class and that it is granted the appropriate permissions. You can add the Java archive (JAR) file or directory that contains this class into the server.policy file so that it has the necessary permissions that are needed by the server code.

   **Tip:** All of the token types defined by the propagation framework have similar interfaces. Basically, the token types are marker interfaces that implement the com.ibm.wsspi.security.token.Token interface. This interface defines most of the methods. If you plan to implement more than one token type, consider creating an abstract class that implements the com.ibm.wsspi.security.token.Token interface. All of your token implementations, including the AuthenticationToken, might extend the abstract class and then most of the work is completed.

   To see an implementation of AuthenticationToken, see "Example: com.ibm.wsspi.security.token.AuthorizationToken implementation" on page 301

2. Add and receive the custom AuthenticationToken during WebSphere Application Server logins This task is typically accomplished by adding a custom login module to the various application and system login configurations. However, in order to deserialize the information, you must plug in a custom login module. After the object is instantiated in the login module, you can the object to the Subject during the commit() method.

   If you only want to add information to the Subject to get propagated, see "Propagating a custom Java serializable object" on page 331. If you want to ensure that the information is propagated, if you want to do your own custom serialization, or if you want to specify the uniqueness for Subject caching purposes, then consider writing your own AuthenticationToken implementation.

   The code sample in "Example: custom AuthenticationToken login module" on page 329 shows how to determine if the login is an initial login or a propagation login. The difference between these login types is whether the WSTokenHolderCallback contains propagation data. If the callback does not contain propagation data, initialize a new custom AuthenticationToken implementation and set it into the Subject. If the callback contains propagation data, look for your specific custom AuthenticationToken TokenHolder instance, convert the byte[] back into your custom AuthenticationToken object, and set it back into the Subject. The code sample shows both instances.

   You can make your AuthenticationToken read-only in the commit phase of the login module. If you do not make the token read-only, then attributes can be added within your applications.

3. Add your custom login module to WebSphere Application Server system login configurations that already contain the com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule for receiving serialized versions of your custom authorization token

   Because this login module relies on information in the sharedState added by the com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule, add this login module after com.ibm.ws.security.server.lm.wsMapDefaultInboundLoginModule. For information on how to add your custom login module to the existing login configurations, see "Custom login module development for a system login configuration" on page 78

After completing these steps, you have implemented a custom AuthenticationToken.

## Example: com.ibm.wsspi.security.token.AuthenticationToken implementation

Use this file to see an example of a AuthenticationToken implementation. The following sample code does not extend an abstract class, but rather implements the com.ibm.wsspi.security.token.AuthenticationToken interface directly. You can implement the interface directly, but it might cause you to write duplicate code.

However, you might choose to implement the interface directly if there are considerable differences between how you handle the various token implementations.

For information on how to implement a custom AuthenticationToken, see “Implementing a custom AuthenticationToken” on page 322.

```
package com.ibm.websphere.security.token;

import com.ibm.websphere.security.WSSecurityException;
import com.ibm.websphere.security.auth.WSLoginFailedException;
import com.ibm.wsspi.security.token.*;
import com.ibm.websphere.security.WebSphereRuntimePermission;
import java.io.ByteArrayOutputStream;
import java.io.ByteArrayInputStream;
import java.io.DataOutputStream;
import java.io.DataInputStream;
import java.io.ObjectOutputStream;
import java.io.ObjectInputStream;
import java.io.OutputStream;
import java.io.InputStream;
import java.util.ArrayList;

public class CustomAuthenticationTokenImpl implements com.ibm.wsspi.security.
   token.AuthenticationToken
{
 private java.util.Hashtable hashtable = new java.util.Hashtable();
 private byte[] tokenBytes = null;
  // 2 hours in millis, by default
   private static long expire_period_in_millis = 2*60*60*1000;
 private String oidName = "your_oid_name";
  // This string can really be anything if you do not want to use an OID.

/**
 * Constructor used to create initial AuthenticationToken instance
 */
 public CustomAuthenticationTokenImpl (String principal)
 {
  // Sets the principal in the token
  addAttribute("principal", principal);
  // Sets the token version
  addAttribute("version", "1");
  // Sets the token expiration
  addAttribute("expiration", new Long(System.currentTimeMillis()
        + expire_period_in_millis).toString());
 }

/**
 * Constructor used to deserialize the token bytes received during a
 * propagation login.
 */
 public CustomAuthenticationTokenImpl (byte[] token_bytes)
 {
  try
  {
       // The data in token_bytes should be signed and encrypted if the
       // hashtable is acting as an authentication token.
   hashtable = (java.util.Hashtable) custom_decryption_algorithm (token_bytes);
  }
  catch (Exception e)
  {
   e.printStackTrace();
  }
 }

/**
 * Validates the token including expiration, signature, and so on.
 * @return boolean
```

```
 */

 public boolean isValid ()
 {
  long expiration = getExpiration();

  // If you set the expiration to 0, the token does not expire
  if (expiration != 0)
  {
   // Returns a response that identifies whether this token is still valid
   long current_time = System.currentTimeMillis();

   boolean valid = ((current_time < expiration) ? true : false);
   System.out.println("isValid: returning " + valid);
   return valid;
  }
  else
  {
   System.out.println("isValid: returning true by default");
   return true;
  }
 }

/**
 * Gets the expiration as a long type.
 * @return long
 */
 public long getExpiration()
 {
  // Gets the expiration value from the hashtable
  String[] expiration = getAttributes("expiration");

  if (expiration != null && expiration[0] != null)
  {
   // The expiration is the first element and there should only be one expiration
   System.out.println("getExpiration: returning " + expiration[0]);
   return new Long(expiration[0]).longValue();
  }

  System.out.println("getExpiration: returning 0");
  return 0;
 }

/**
 * Returns if this token should be forwarded/propagated downstream.
 * @return boolean
 */
 public boolean isForwardable()
 {
     // You can choose whether your token gets propagated. In some cases
     // you might want it to be local only.
  return true;
 }

/**
 * Gets the principal to which this token belongs. If this is an
 * authorization token, this principal string must match the
 * authentication token principal string or the message is rejected.
 * @return String
 */
 public String getPrincipal()
 {
  // This value might be any combination of attributes
  String[] principal = getAttributes("principal");

  if (principal != null && principal[0] != null)
  {
```

```
   return principal[0];
  }

  System.out.println("getExpiration: returning null");
  return null;
 }

/**
 * Returns a unique identifier of the token based upon information the provider
 * considers makes this a unique token. This identifier is used for caching purposes
 * and can be used in combination with other token unique IDs that are part of
 * the same Subject.
 *
 * This method should return null if you want the accessID of the user to represent
 * uniqueness.  This is the typical scenario.
 *
 * @return String
 */
 public String getUniqueID()
 {
     // If you do not want to affect the cache lookup, just return NULL here.
     return null;

  String cacheKeyForThisToken = "dynamic attributes";

     // If you do want to affect the cache lookup, return a string of
     // attributes that you want factored into the lookup.
  return cacheKeyForThisToken;
 }

/**
 * Gets the bytes to be sent across the wire. The information in the byte[]
 * needs to be enough to recreate the token object at the target server.
 * @return byte[]
 */
 public byte[] getBytes ()
 {
  if (hashtable != null)
  {
   try
   {
          // Do this if the object is set read-only during login commit
          // because this ensures that new data is not set.
    if (isReadOnly() && tokenBytes == null)
     tokenBytes = custom_encryption_algorithm (hashtable);

    return tokenBytes;
   }
   catch (Exception e)
   {
    e.printStackTrace();
    return null;
   }
  }

  System.out.println("getBytes: returning null");
  return null;
 }

/**
 * Gets the name of the token, which is used to identify the byte[] in the
 * protocol message.
 * @return String
 */
 public String getName()
 {
  return oidName;
```

```
 }

/**
 * Gets the version of the token as an short type.  This also is used
 * to identify the byte[] in the protocol message.
 * @return short
 */
 public short getVersion()
 {
  String[] version = getAttributes("version");

  if (version != null && version[0] != null)
   return new Short(version[0]).shortValue();

  System.out.println("getVersion: returning default of 1");
  return 1;
    }

/**
 * When called, the token becomes irreversibly read-only.  The implementation
 * needs to ensure that any set methods check that this state has been set.
 */
 public void setReadOnly()
 {
  addAttribute("readonly", "true");
 }

/**
 * Called internally to see if the token is read-only
 */
 private boolean isReadOnly()
 {
  String[] readonly = getAttributes("readonly");

  if (readonly != null && readonly[0] != null)
   return new Boolean(readonly[0]).booleanValue();

  System.out.println("isReadOnly: returning default of false");
  return false;
 }

/**
 * Gets the attribute value based on the named value.
 * @param String key
 * @return String[]
 */
 public String[] getAttributes(String key)
 {
  ArrayList array = (ArrayList) hashtable.get(key);

  if (array != null && array.size() > 0)
  {
   return (String[]) array.toArray(new String[0]);
  }

  return null;
 }

/**
 * Sets the attribute name/value pair.  Returns the previous values set for key,
 * or null if not previously set.
 * @param String key
 * @param String value
 * @returns String[];
 */
 public String[] addAttribute(String key, String value)
 {
```

```
  // Gets the current value for the key
  ArrayList array = (ArrayList) hashtable.get(key);

  if (!isReadOnly())
  {
   // Copies the ArrayList to a String[] as it currently exists
   String[] old_array = null;
   if (array != null && array.size() > 0)
    old_array = (String[]) array.toArray(new String[0]);

   // Allocates a new ArrayList if one was not found
   if (array == null)
    array = new ArrayList();

   // Adds the String to the current array list
   array.add(value);

   // Adds the current ArrayList to the Hashtable
   hashtable.put(key, array);

   // Returns the old array
   return old_array;
  }

  return (String[]) array.toArray(new String[0]);
 }


/**
 * Gets the list of all attribute names present in the token.
 * @return java.util.Enumeration
 */
 public java.util.Enumeration getAttributeNames()
 {
  return hashtable.keys();
 }

/**
 * Returns a deep copying of this token, if necessary.
 * @return Object
 */
 public Object clone()
 {
  com.ibm.wsspi.security.token.AuthenticationToken deep_clone =
   new com.ibm.websphere.security.token.CustomAuthenticationTokenImpl();

  java.util.Enumeration keys = getAttributeNames();

  while (keys.hasMoreElements())
  {
   String key = (String) keys.nextElement();

   String[] list = (String[]) getAttributes(key);

   for (int i=0; i<list.length; i++)
    deep_clone.addAttribute(key, list[i]);
  }

      return deep_clone;
 }

/**
 * This method returns true if this token is storing a user ID and password
 * instead of a token.
 * @return boolean
 */
 public boolean isBasicAuth()
```

```
 {
  return false;
 }
}
```

## Example: custom AuthenticationToken login module

This file shows how to determine if the login is an initial login or a propagation login.

```
public customLoginModule()
{
 public void initialize(Subject subject, CallbackHandler callbackHandler,
    Map sharedState, Map options)
 {
   // (For more information on what to do during initialization, see
   // "Custom login module development for a system login configuration" on page 78.)
  _sharedState = sharedState;
 }

 public boolean login() throws LoginException
 {
   // (For information on what to do during login, see
   // "Custom login module development for a system login configuration" on page 78.)

  // Handles the WSTokenHolderCallback to see if this is an initial or
     // propagation login.
  Callback callbacks[] = new Callback[1];
  callbacks[0] = new WSTokenHolderCallback("Authz Token List: ");

  try
  {
   callbackHandler.handle(callbacks);
  }
  catch (Exception e)
  {
   // Handles exception
  }

  // Receives the ArrayList of TokenHolder objects (the serialized tokens)
  List authzTokenList = ((WSTokenHolderCallback) callbacks[0]).getTokenHolderList();

  if (authzTokenList != null)
  {
   // Iterates through the list looking for your custom token
   for (int i=0; i<authzTokenList.size(); i++)
   {
    TokenHolder tokenHolder = (TokenHolder)authzTokenList.get(i);

          // Looks for the name and version of your custom AuthenticationToken
          // implementation
    if (tokenHolder.getName().equals("your_oid_name") && tokenHolder.getVersion() == 1)
    {
            // Passes the bytes into your custom AuthenticationToken constructor
            // to deserialize
     customAuthzToken = new
      com.ibm.websphere.security.token.
               CustomAuthenticationTokenImpl(tokenHolder.getBytes());
```

```
   }
  }
 }
 else
        // This is not a propagation login. Create a new instance of your
        // AuthenticationToken implementation
 {
      //  Gets the principal from the default AuthenticationToken. This principal
      //  should match all default tokens.
      //  Note: WebSphere Application Server run time only enforces this for
      //  default tokens. Thus, you can choose
      //  to do this for custom tokens, but it is not required.
  defaultAuthToken = (com.ibm.wsspi.security.token.AuthenticationToken)
   sharedState.get(com.ibm.wsspi.security.auth.callback.Constants.WSAUTHTOKEN_KEY);
  String principal = defaultAuthToken.getPrincipal();

      // Adds a new custom authentication token. This is an initial login. Pass
      // the principal into the constructor
  customAuthToken = new com.ibm.websphere.security.token.
         CustomAuthenticationTokenImpl(principal);

  // Adds any initial attributes
  if (customAuthToken != null)
  {
   customAuthToken.addAttribute("key1", "value1");
   customAuthToken.addAttribute("key1", "value2");
   customAuthToken.addAttribute("key2", "value1");
   customAuthToken.addAttribute("key3", "something different");
  }
 }

    // Note: You can add the token to the Subject during commit in case
    // something happens during the login.
}

public boolean commit() throws LoginException
{
  // (For more information on what do during commit, see
  // "Custom login module development for a system login configuration" on page 78.)

 if (customAuthToken != null)
 {
  // Sets the customAuthToken token into the Subject
  try
  {
   private final AuthenticationToken customAuthTokenPriv = customAuthToken;
         // Do this in a doPrivileged code block so that application code does
         // not need to add additional permissions
   java.security.AccessController.doPrivileged(new java.security.PrivilegedAction()
   {
    public Object run()
    {
     try
     {
                // Adds the custom Authentication token if it is not
```

```
                // null and not already in the Subject
                              if ((customAuthTokenPriv != null) &&
        (!subject.getPrivateCredentials().
                        contains(customAuthTokenPriv)))
       {
        subject.getPrivateCredentials().add(customAuthTokenPriv);
       }
      }
      catch (Exception e)
      {
       throw new WSLoginFailedException (e.getMessage(), e);
      }

      return null;
     }
    });
   }
   catch (Exception e)
   {
    throw new WSLoginFailedException (e.getMessage(), e);
   }
  }
 }

 // Defines your login module variables
 com.ibm.wsspi.security.token.AuthenticationToken customAuthToken = null;
 com.ibm.wsspi.security.token.AuthenticationToken defaultAuthToken = null;
 java.util.Map _sharedState = null;
}
```

## Propagating a custom Java serializable object

Prior to completing this task, verify that security propagation is enabled in the administrative console.

With security attribute propagation enabled, you can propagate data either horizontally with single signon (SSO) enabled or downstream using Common Secure Interoperability version 2 (CSIv2). When a login occurs, either through an application login configuration or a system login configuration, a custom login module can be plugged in to add Java serializable objects into the Subject during login. This document describes how to add an object into the Subject from a login module and describes other infrastructure considerations to make sure that the Java object gets propagated.

1. Add your custom Java object into the Subject from a custom login module. There is a two-phase process for each Java Authentication and Authorization Service (JAAS) login module. WebSphere Application Server completes the following processes for each login module present in the configuration:

   **login() method**
      In this step, the login configuration callbacks are analyzed, if necessary, and the new objects or credentials are created.

   **commit() method**
      In this step, the objects or credentials that are created during login are added into the Subject.

   After a custom Java object is added into the Subject, WebSphere Application Server serializes the object, deserializes the object, and adds the object back into the Subject downstream. However, there are some requirements for this process to occur successfully. For more information on the JAAS programming model, see the JAAS information provided in “Security: Resources for learning” on page 21.

**Important:** Whenever you plug a custom login module into the login infrastructure of WebSphere Application Server, make sure that the code is trusted. When you add the login module into the *install_root*/classes directory, the login module has Java 2 Security AllPermissions. It is recommended that you add your login module and other infrastructure classes into any private directory. However, you must modify the *install_root*/properties/server.policy file to make sure that your private directory, Java archive (JAR) file, or both have the permissions needed to execute the application programming interfaces (API) that are called from the login module. Because the login module might be executed after the application code on the call stack, you might add doPrivileged code so that you do not need to add additional properties to your applications.

The following code sample shows how to add doPrivileged:

```
public customLoginModule()
{
 public void initialize(Subject subject, CallbackHandler callbackHandler,
     Map sharedState, Map options)
 {
  // (For more information on what to do during initialization, see
  // "Custom login module development for a system login configuration" on page 78.)
 }

 public boolean login() throws LoginException
 {
  // (For more information on what to do during login phase, see
  // "Custom login module development for a system login configuration" on page 78.)

     // Construct callback for the WSTokenHolderCallback so that you
     // can determine if
     // your custom object has propagated
    Callback callbacks[] = new Callback[1];
    callbacks[0] = new WSTokenHolderCallback(″Authz Token List: ″);

    try
    {
         _callbackHandler.handle(callbacks);
    }
    catch (Exception e)
    {
    throw new LoginException (e.getLocalizedMessage());
    }

    // Checks to see if any information is propagated into this login
    List authzTokenList = ((WSTokenHolderCallback) callbacks[1]).
            getTokenHolderList();

    if (authzTokenList != null)
    {
        for (int i = 0; i< authzTokenList.size(); i++)
             {
            TokenHolder tokenHolder = (TokenHolder)authzTokenList.get(i);

                  //  Look for your custom object. Make sure you use
                  //  ″startsWith″because there is some data appended
                  //  to the end of the name indicating in which Subject
                  //  Set it belongs. Example from getName():
```

```
                  //  ″com.acme.CustomObject (1)″. The class name is
                  //  generated at the sending side by calling the
                  //  object.getClass().getName() method. If this object
                  //  is deserialized by WebSphere Application Server,
                  //  then return it and you do not need to add it here.
                  //  Otherwise, you can add it below.
                  //  Note: If your class appears in this list and does
                  //  not use custom serialization (for example, an
                  //  implementation of the Token interface described in
                  //  the Propagation Token Framework), then WebSphere
                  //  Application  Server automatically deserializes the
                  //  Java object for you. You might just return here if
                  //  it is found in the list.

            if (tokenHolder.getName().startsWith(″com.acme.CustomObject″))
          return true;
       }
    }
       //  If you get to this point, then your custom object has not propagated
        myCustomObject = new com.acme.CustomObject();
        myCustomObject.put(″mykey″, ″mydata″);
}

public boolean commit() throws LoginException
{
 // (For more information on what to do during the commit phase, see
 // "Custom login module development for a system login configuration" on page 78.)

 try
 {
      // Assigns a reference to a final variable so it can be used in
      // the doPrivileged block
  final com.acme.CustomObject myCustomObjectFinal = myCustomObject;
  // Prevents your applications from needing a JAAS getPrivateCredential
      // permission.
  java.security.AccessController.doPrivileged(new java.security.
          PrivilegedExceptionAction()
  {
   public Object run() throws java.lang.Exception
   {
           // Try not to add a null object to the Subject or an object
           // that already exists.
    if (myCustomObjectFinal != null && !subject.getPrivateCredentials().
               contains(myCustomObjectFinal))
    {
              // This call requires a special Java 2 Security permission,
              // see the JAAS Javadoc.
     subject.getPrivateCredentials().add(myCustomObjectFinal);
    }
    return null;
   }
  });
 }
 catch (java.security.PrivilegedActionException e)
 {
  // Wraps the exception in a WSLoginFailedException
```

```
    java.lang.Throwable myException = e.getException();
    throw new WSLoginFailedException (myException.getMessage(), myException);
   }
  }

  // Defines your login module variables
  com.acme.CustomObject myCustomObject = null;
 }
```

2. Verify that your custom Java class implements the `java.io.Serializable` interface. An object that is added to the Subject must be serializable if you want the object to propagate. For example, the object must implement the `java.io.Serializable` interface. If the object is not serializable, the request does not fail, but the object does not propagate. To make sure that an object added to the Subject is propagated, implement one of the token interfaces defined in the "Security attribute propagation" on page 275 article or add attributes to one of the following existing default token implementations:

   **AuthorizationToken**
      Add attributes if they are user-specific. For more information, see "Default AuthorizationToken" on page 297.

   **PropagationToken**
      Add attributes that are specific to an invocation. For more information, see "Default PropagationToken" on page 282.

   If you are careful adding custom objects and follow all the steps to make sure that WebSphere Application Server can serialize and deserialize the object at each hop, then it is sufficient to use custom Java objects only.
3. Verify that your custom Java class exists on all of the systems that might receive the request. When you add a custom object into the Subject and expect WebSphere Application Server to propagate the object, make sure that the class definition for that custom object exists in the *install_root*/`classes` directory on all of the nodes where serialization or deserialization might occur. Also, verify that the Java class versions are the same.
4. Verify that your custom login module is configured in all of the login configurations used in your environment where you would need to add your custom object during a login. Any login configuration that interacts with WebSphere Application Server generates a Subject that might be propagated outbound for an EJB request. If you want WebSphere Application Server to propagate a custom object in all cases, make sure that the custom login module is added to every login configuration that is used in your environment. For more information, see "Custom login module development for a system login configuration" on page 78.
5. Verify that security attribute propagation is enabled on all of the downstream servers that receive the propagated information. When an EJB request is sent to a downstream server and security attribute propagation is disabled on that server, only the authentication token is sent for backwards compatibility. Therefore, you must review the configuration to verify that propagation is enabled in all of the cells that might receive requests. There are several places in the administrative console that you must check to make sure propagation is fully enabled. For more information, see "Enabling security attribute propagation" on page 279.
6. Add any custom objects to the propagation exclude list that you do not want to propagate. You can configure a property to exclude the propagation of objects that match specific class names, package names, or both. For example, you can have a custom object that is related to a specific process. If the object is propagated, it does not contain valid information. You must tell WebSphere Application Server not to propagate this object. Complete the following steps to specify the object in the propagation exclude list, using the administrative console:
   a. Click **Security > Global Security**.
   b. Under Additional Properties, click **Custom Properties > New**.
   c. Add `com.ibm.ws.security.propagationExcludeList` in the **Name** field.

d. Add the name of the custom object in the **Value** field. You can add a list of custom objects to the propagation exclude list separated by a colon. For example, you might enter `com.acme.CustomLocalObject:com.acme.private.*`. You can enter a class name such as `com.acme.CustomLocalObject` or a package name such as `com.acme.private.*`. In this example, WebSphere Application Server does not propagate any class that equals `com.acme.CustomLocalObject` or begins with `com.acme.private.`.

   Although you can add custom objects to the propagation exclude list, you must be aware of a side effect. WebSphere Application Server stores the opaque token, or the serialized Subject contents, in a local cache for the life of the single signon (SSO) token. The life of the SSO token, which has a default of two hours, is configured in the SSO properties on the administrative console. The information that is added to the opaque token includes only the objects not in the exclude list. If your authentication cache does not match your SSO token timeout, you might get a Subject on the local server that is regenerated from the opaque token but does not contain the objects on the exclude list. The authentication cache, which has a default of ten minutes, is configured on the Global Security panel on the administrative console. It is recommended that you make your authentication cache timeout value equal to the SSO token timeout so that the Subject contents are consistent locally.

As a result of this task, custom Java serializable objects are propagated horizontally or downstream. For more information on the differences between horizontal and downstream propagation, see "Security attribute propagation" on page 275.

## Authorization in WebSphere Application Server

WebSphere Application Server supports authorization based on the Java Authorization Contract for Containers (JACC) specification in addition to the default authorization. JACC is a new specification in Java 2 Platform, Enterprise Edition (J2EE) 1.4. It enables third-party security providers to manage authorization in the application server. The default JACC provider that is provided by WebSphere Application Server uses the Tivoli Access Manager as the authorization provider.

When security is enabled in the WebSphere Application Server, the default authorization is used unless a JACC provider is specified. The default authorization does not require special setup, and the default authorization engine makes all of the authorization decisions. However, if a JACC provider is configured and setup to be is used by WebSphere Application Server, all of the Enterprise JavaBeans (EJB) and Web authorization decisions are then delegated to the JACC provider.

WebSphere Application Server supports security for J2EE applications and also for its administrative components. J2EE applications such as Web and EJB components are protected and authorized per the J2EE specification. The administrative components are internal to WebSphere Application Server, and are protected by the RoleBasedAuthorizer. The administrative components include the adminConsole application, MBeans, and other components such as naming and security. For more information on administrative security, see "Role-based authorization" on page 120.

When a JACC provider is used for authorization in WebSphere Application Server, all of the J2EE application-based authorization decisions are delegated to the provider per the JACC specification. However, all administrative security authorization decisions are made by the WebSphere Application Server default authorization engine. The JACC provider is not called to make the authorization decisions for administrative security.

When a protected J2EE resource is accessed, the authorization decision to give access to the principal is the same whether using the default authorization engine or a JACC provider. Both of the authorization models satisfy the J2EE specification, so there should be no differences in function. Choose a JACC provider only when you want to work with an external security provider such as the Tivoli Access Manager. In this instance, the security provider must support the JACC specification and be set up to work with the WebSphere Application Server. Setting up and configuring a JACC provider requires additional

configuration steps, depending on the provider. Unless you have an external security provider that you can use with WebSphere Application Server, use the default authorization.

## JACC providers

The Java Authorization Contract for Containers (JACC) is a new specification introduced in Java 2 Platform, Enterprise Edition (J2EE) 1.4 through the Java Specifications Request (JSR) 115 process. This specification defines a contract between J2EE containers and authorization providers.

The contract enables third-party authorization providers to plug into J2EE 1.4 application servers (such as WebSphere Application Server) to make the authorization decisions when a J2EE resource is accessed. The access decisions are made through the standard java.security.Policy object.

In WebSphere Application Server, two authorization contracts are supported using both a native and a third-party JACC provider implementation. The default (out-of-box) solution is the WebSphere Application Server default J2EE role based authorization implementation, which does not implement the JACC Policy provider interface.

To plug-in to WebSphere Application Server, the third-party JACC provider must implement the policy class, policy configuration factory class, and policy configuration interface. All are required by the JACC specification.

The JACC specification does not specify how to handle the authorization table (user or group to role) information between the container and the provider. It is the responsibility of the provider to provide some management facilities to handle this information. It does not require the container to provide the authorization table information in the binding file to the provider.

WebSphere Application Server provides two role configuration interfaces (RoleConfigurationFactory and RoleConfiguration) to help the provider obtain information from the binding file, as well as an initialization interface (InitializeJACCProvider). The implementation of these interfaces is optional. See “Interfaces used to support JACC” on page 348 for more information about these interfaces.

**Tivoli Access Manager as the default JACC provider for WebSphere Application Server**

The JACC provider in WebSphere Application Server is implemented by both the client and the server pieces of the Tivoli Access Manager server. The client piece of Tivoli Access Manager is embedded in WebSphere Application Server. The server piece is located on a separate installable CD that is shipped as part of the WebSphere network deployment (ND) package.

The JACC provider is not an out-of-box solution. You must configure WebSphere Application Server to use the JACC provider.

## Authorization providers settings

Use this page to enable a Java Authorization Contract for Containers (JACC) provider for authorization decisions.

To view this administrative console page, click **Security** > **Global security**. Under Authorization, click **Authorization providers**.

WebSphere Application Server provides a default authorization engine that performs all of the authorization decisions. In addition, WebSphere Application Server also supports an external authorization provider using the JACC specification to replace the default authorization engine for Java 2 Platform, Enterprise Edition (J2EE) applications.

JACC is part of the J2EE specification, which enables third-party security providers such as Tivoli Access Manager to plug into WebSphere Application Server and make authorization decisions.

**Important:** Unless you have an external JACC provider or want to use a JACC provider for Tivoli Access Manager that can handle J2EE authorizations based on JACC, and it is configured and set up to be used with WebSphere Application Server, do not enable **External authorization using JACC**.

### Default authorization

This option should be used all the time unless you want an external security provider such as the Tivoli Access Manager to perform the authorization decision for J2EE applications based on the JACC specification.

**Default:** Enabled

### External authorization using a JACC provider

Enable this option only when you plan to use an external security provider such as the Tivoli Access Manager for performing authorization decisions for J2EE applications using the JACC specification.

To use an external provider, you must complete the following steps:

1. Configure your JACC provider.
2. Verify that the required provider implementation classes are in the class path for each WebSphere Application Server process.

   **Attention:** This step is not required when you use Tivoli Access Manager because the application server already contains the implementation classes.
3. Enable the **External authorization using a JACC provider** option
4. Enter the appropriate properties for the provider under the External JACC provider link, which is located under Related Items.

**Default:** Disabled

### External JACC provider

Use this link to configure WebSphere Application Server to use an external JACC provider. For example to configure an external JACC provider, the policy class name and the policy configuration factory class name are required by the JACC specification.

The default settings contained in this link are used by Tivoli Access Manager for authorization decisions. If you intend to use another provider, modify the settings as appropriate.

## JACC support in WebSphere Application Server

WebSphere Application Server supports the Java Contract for Containers (JACC) specification, which enables third-party security providers to handle the Java 2 Platform, Enterprise Edition (J2EE) authorization.

The specification requires that both the containers in the application server and the provider satisfy some requirements. Specifically, the containers are required to propagate the security policy information to the provider during the application deployment and to call the provider for all authorization decisions. The providers are required to include the storing of the policy information in their repository during application deployment. The providers then use this information to make authorization access decisions when called by the container.

### JACC access decisions

When security is enabled and an enterprise bean or Web resource is accessed, the Enterprise JavaBean (EJB) container or Web container calls the security run time to make an authorization decision on whether to permit access. When using an external provider, the access decision is delegated to that provider.

According to the Java Contract for Containers (JACC) specification, the appropriate permission object is created, the appropriate policy context handlers are registered, and the appropriate policy context identifier (contextID) is set. A call is made to the java.security.Policy object method implemented by the provider to make the access decision.

The following sections describe how the provider is called for both the EJB and the Web resources.

Access decisions for enterprise beans:

When security is enabled, and an EJB method is accessed, the EJB container delegates the authorization check to the security runtime. If JACC is enabled, the security runtime uses the following process to perform the authorization check:

1. It creates the EJBMethodPermission object using the bean name, method name, interface name and the method signature.
2. It creates the contextID and sets it on the thread by using the PolicyContext.setContextID(contextID) method.
3. It registers the required policy context handlers, including the Subject policy context handler.
4. It creates the ProtectionDomain object with principal in the Subject. If there is no principal, null is passed for the principal name.
5. The access decision is delegated to the JACC provider by calling the implies() method of the Policy object, which is implemented by the provider. The EJBMethodPermission and the ProtectionDomain objects are passed to this method.
6. The isCallerInRole( ) access check also follows the same process, except that an EJBRoleRefPermission object is created instead of an EJBMethodPermission.

Access decisions for Web Resources:

When security is enabled and configured to use a JACC provider, and when a Web resource such as a servlet or a JavaServer pages (JSP) is accessed, the security runtime delegates the authorization decision to the JACC provider by using the following process:

1. A WebResourcePermission is created to see if the URI is unchecked. If the provider honors the Everyone subject it should also be checked here.
   a. The WebResourcePermission is constructed with urlPattern and the HTTP method accessed.
   b. A ProtectionDomain with a null principal name is created.
   c. The JACC provider‘s Policy.implies( ) method is called with the permission and the protection domain. If the URI access is unchecked (or given access to Everyone subject), the provider should permit access (return true) in the implies() method. Access is then granted without further checks.
2. If the access was not granted in Step 1, a WebUserDataPermission is created and used to see if the Uniform Resource Identifier (URI) is precluded or excluded or must be redirected using HTTPS protocol.
   a. The WebUserDataPermission is constructed with the urlPattern accessed, along with the HTTP method invoked and the transport type of the request. If the request is over HTTPS, the transport type is set to CONFIDENTIAL; otherwise, null is passed.
   b. ProtectionDomain with a null principal name is created.
   c. The JACC provider‘s Policy.implies( ) method is called with the permission and the protection domain. If the request is using the HTTPS protocol and the implies returns false, the HTTP 403

error is returned to imply excluded/precluded permission and no further checks are performed. If the request is not using the HTTPS protocol, and the implies returns false, the request is redirected over HTTPS.

3. The security runtime attempts to authenticate the user. If the authentication information already exists (for example, LTPAToken), it is used. Otherwise, the user's credentials must be entered.
4. After the user credentials are validated, a final authorization check is performed to see if the user has been granted access privileges to the URI.
   a. As in Step 1, the WebResourcePermission is created. The ProtectionDomain now contains the Principal that is attempting to access the URI. The Subject policy context handler also contains the user‘s information, which can be used for the access check.
   b. The provider‘s implies() method is called using the Permission object and the ProtectionDomain created above. If the user is granted permission to access the resource, the implies( ) method should return true. If the user is not granted access, the implies() method should return false.

**Note:** Even if the order listed above is changed later (for example, to improve performance) the end result should be the same. For example, if the resource is precluded or excluded the end result is that the resource cannot be accessed.

Using information from the Subject for Access Decision:

If the provider relies on the WebSphere Application Server generated Subject for access decision, the provider can query the public credentials in the Subject to obtain the credential of type WSCredential . The WSCredential API is used to obtain information about the user, including the name and the groups that the user belongs to. This information is then used to make the access decision.

If the provider adds information to the Subject (for example, by using the Trust Association Interface feature or by plugging login modules into the Application Server), that information is available in the Subject. The provider can then make use of the information added in the Subject to make the access decision.

The security attribute propagation has more information on how to add information to the Subject. See "Enabling security attribute propagation" on page 279 for more information.

**Dynamic module updates in JACC**

WebSphere Application Server supports dynamic updates to Web modules under certain conditions. If a Web module is updated, deleted or added to an application, only that module is stopped and/or started as appropriate. The other existing modules in the application are not impacted, and the application itself is not stopped and then restarted.

When any security policies are modified in the Web modules, the application is stopped and then restarted when using the default authorization engine. When using the Java Contract for Containers (JACC) based authorization, the behavior depends on the functionality that a provider supports. If a provider can handle dynamic changes to the Web modules, then only the Web modules are impacted. Otherwise, the entire application is stopped and restarted for the new changes in the Web modules to take effect.

A provider can indicate if they will support the dynamic updates by configuring the **supports dynamic module updates** option in the JACC configuration model (see "Configuring a JACC provider" on page 346 for more information). This option can be enabled or disabled using the administrative console or by scripting. It is expected that most providers will store the policy information in their external repository, which makes it possible for them to support these dynamic updates. This option is **enabled** by default for most providers.

When the **supports dynamic module updates** option is enabled, if a Web module that contains security roles is dynamically added, modified, or deleted, only the specific Web modules are impacted and

restarted. If the option is disabled, the entire application is restarted. When dynamic updates are performed, the security policy information of the modules impacted are propagated to the provider. For more information about security policy propagation, see "JACC policy propagation" on page 341.

**Initialization of the JACC provider**

If a Java Contract for Containers (JACC) provider requires initialization during server startup (for example, to enable the client code to communicate to the server code), they can implement the com.ibm.wsspi.security.authorization.InitializeJACCProvider interface. See "Interfaces used to support JACC" on page 348 for more information.

When this interface is implemented, it is called during server startup. Any custom properties in the JACC configuration model are propagated to the initialize method of this implementation. The custom properties can be entered using either the administrative console or by scripting.

During server shutdown, the cleanup method is called for any clean-up work that a provider requires. Implementation of this interface is strictly optional, and should be used only if the provider requires initialization during server startup.

**Mixed node environment and JACC**

Authorization using Java Contract for Containers (JACC) is a new feature in WebSphere Application Server Version 6. Previous versions of the WebSphere Application Server do not support this feature. Also, the JACC configuration is set up at the cell level and is applicable for all the nodes and servers in that cell..

If you are planning to use the JACC-based authorization the cell only contains 6.0 nodes. This implies that a mixed node environment containing a set of 5.x nodes in a 6.0 cell is not supported.

## JACC policy context handlers

WebSphere Application Server supports all of the policy context handlers that are required by the Java Contract for Containers (JACC) specification. However, due to performance impacts, the Enterprise JavaBeans (EJB) arguments policy context handler is not activated unless it is specifically required by the provider. Performance impacts result if objects must be created for each of the arguments for each EJB method.

If the provider supports and requires this context handler, enable the **Requires the EJB arguments policy context handler for access decisions** check box in the External Jacc provider link under the Authorization providers panel or by using scripting. Any changes to this are effective after the servers has been restarted . By default this is disabled. When using the Tivoli Access Manager as the JACC provider, this option should be disabled, since the argument values are not required for access decisions.

## JACC policy context identifiers (ContextID) format

A policy context identifier is defined as a unique string that represents a policy context. A policy context contains all of the security policy statements as defined by the Java Contract for Containers (JACC) specification that affect access to the resources in a Web or Enterprise JavaBeans (EJB) module. During policy propagation to the JACC provider, a PolicyConfiguration object is created for each policy context. The object is populated with the policy statements (represented by the JACC permission objects) that correspond to the context. The object is then propagated to the JACC provider using the JACC specification APIs.

WebSphere Application Server makes the contextID unique by using the string `href:cellName/appName/moduleName` as the contextID format for the modules. The `href` part of the string indicates that a hierarchical name is passed as the contextID.

The cellName represents the name of the deployment manager cell or the base cell where the application is installed. After an application is installed in one cell (for example, in a base application server where the cell name is base1) and is added to a deployment manager cell whose name is cell1 by using addNode, the contextID for the modules in the application contain base1 (not cell1 ) as the cell name since the application was initially installed in base1.

The appName part of the string in the contextID represents the application name containing the module. The moduleName refers to the name of the module.

As an example, the contextID for the module Increment.jar in an application named DefaultApplication that is installed in cell1 is href:cell1/DefaultApplication/Increment.jar.

## JACC policy propagation

When an application is installed or deployed in the WebSphere Application Server, the security policy information in the application is propagated to the provider when the configuration is saved. The contextID for that application is saved in its application.xml file, used for propagating the policy to the JACC provider, and also for access decisions for J2EE resources.

When an application is uninstalled, the security policy information in the application is removed from the provider when the configuration is saved.

If the provider has implemented the RoleConfiguration interface, the security policy information propagated to the policy provider also contains the authorization table information. See “Interfaces used to support JACC” on page 348 for more information about this interface.

If an application does not contain security policy information, the PolicyConfiguration (and the RoleConfiguration, if implemented) objects do not contain any information. The existence of empty PolicyConfiguration and RoleConfiguration objects indicates that security policy information for the module does not exist.

Once an application is installed, it can be updated without first being uninstalled and reinstalled. For example, a new module can be added to an existing application, or an existing module can be modified. In this instance, the information in the impacted modules is propagated to the provider by default. A module is impacted when the deployment descriptor of the module changed as part of the update. If the provider supports the RoleConfiguration interfaces, the entire authorization table for that application is propagated to the provider.

If for some reason, the security information should not be propagated to the provider during application updates, you can set the JVM property **com.ibm.websphere.security.jacc.propagateonappupdate** to false in the deployment manager (in ND) or the unmanaged base application server. If this property is set to false, then any updates to an existing application in the server are not propagated to the provider. You also can set this property on a per-application basis using the custom properties of an application. The wsadmin tool can be used to set the custom property of an application. If this property is set at the application level, any updates to that application are not propagated to the provider. If the update to an application is a full update, for example a new application ear file is used to replace the existing one, the provider is then refreshed with the entire application security policy information.

In the network deployment (ND) environment, when an application is installed and saved, the security policy information in that application is updated in the provider from the deployment manager (dmgr or cell). However, the application is not propagated to its respective nodes until the sychronization command is issued and completed. Also, in the ND setup when an application is uninstalled and saved at the deployment manager, the policy for that application is removed from the JACC provider. However, unless the synchronization command is issued and completed from the deployment manager to the nodes hosting the application, the applications are still running in the respective nodes. In this instance, any access to this application should be denied since the JACC provider does not contain the required information to

make the access decision for that application. Note that any updates to the application already installed as described above are also propagated to the provider from the deployment manager. The changes in the provider are not in sync with the applications in the nodes until the synchronization is completed.

## JACC registration of the provider implementation classes

The JACC specification states that providers can plug in their provider using the system properties `javax.security.jacc.policy.provider` and `javax.security.jacc.PolicyConfigurationFactory.provider`.

The `javax.security.jacc.policy.provider` property is used to set the policy object of the provider, while the `javax.security.jacc.PolicyConfigurationFactory.provider` property is used to set the provider's PolicyConfigurationFactory implementation.

Although both system properties are supported in WebSphere Application Server, it is highly recommended that you use the configuration model provided. You can set these values using either the JACC configuration panel (see "Configuring a JACC provider" on page 346 for more information) or by using wsadmin scripting. One of the advantages of using the configuration model instead of the system properties is that the information is entered in one place at the cell level, and is propagated to all nodes during synchronization. Also, as part of the configuration model, additional properties can be entered as described in the JACC configuration panel.

# Enabling an external JACC provider

The Java Contract for Containers (JACC) defines a contract between Java 2 Platform, Enterprise Edition (J2EE) containers and authorization providers. This contract enables any third-party authorization providers to plug into a J2EE 1.4 application server such as WebSphere Application Server to make the authorization decisions when a J2EE resource is accessed.

To enable an external JACC provider using the administrative console:

1. From the WebSphere Application Server administrative console, click **Security** > **Global Security** from the left navigation menu.
2. Under Authorization, click **Authorization Providers**.
3. Under Related Items, click **External JACC provider**.
4. The fields are set for Tivoli Access Manager by default. Unless you want to use Tivoli Access Manager as the JACC provider, replace these fields with the details for your own external JACC provider.
5. If any custom properties are required by the JACC provider, use the **Custom properties** link to enter the properties. When using the Tivoli Access Manager, use the **Tivoli Access Manager properties** link instead of the **Custom properties** link.
6. Select the **External authorization using a JACC provider** option under **Security > Global Security > Authorization Providers** and then click **OK**.
7. Complete the remaining steps to enable global security. If you are using the Tivoli Access Manager you must select LDAP as the user registry. This same LDAP server should be used by the Tivoli Access Manager. For more information on configuring LDAP registries, see "Configuring Lightweight Directory Access Protocol user registries" on page 198.
8. In a multinode environment, start the deployment manager configuration by issuing the following commands:
   ```
   install_dir\profiles\profile_name\bin\stopManager.bat -username user_name -password password
   install_dir\profiles\profile_name\bin\startManager.bat
   ```
9. Restart all servers to make these changes effective.

## External Java Authorization Contract for Containers provider settings

Use this page to configure WebSphere Application Server to use an external Java Authorization Contract for Containers (JACC) provider. For example, the policy class name and the policy configuration factory class name are required by the JACC specification.

For more information on JACC support in WebSphere Application Server, refer to the information center documentation.

Use these settings when you have set up an external security provider to work with WebSphere Application Server that can support Java 2 Platform, Enterprise Edition (J2EE) authorization based on the JACC specification. The setup process involves installing and configuring the provider server and configuring the client of the provider in the application server to communicate with the server. If the JACC provider is not enabled, which implies the default authorization, these settings are not used.

To view this administrative console page, complete the following steps:

1. Click **Security > Global security**.
2. Under Authorization, click **Authorization providers**.
3. Under Related items, click **External JACC provider**.

Use the default settings when you use Tivoli Access Manager as the JACC provider. Install and configure the Tivoli Access Manager server prior to using it with WebSphere Application Server. Using the Tivoli Access Manager properties link under Additional properties, configure the Tivoli Access Manager client in the application server to use the Tivoli Access Manager server. If you intend to use another provider, modify the settings as appropriate.

### Name

Specifies the name used to identify the external JACC provider.

This field is required.

| | |
|---|---|
| **Data type:** | String |

### Description

Provides an optional description for the provider.

| | |
|---|---|
| **Data type:** | String |

### Policy class name

Specifies a fully qualified class name that represents the javax.security.jacc.policy.provider property as per the JACC specification. The class represents the provider-specific implementation of the java.security.Policy abstract methods.

The class file must reside in the class path of each WebSphere Application Server process. This class is used during authorization decisions. The default class name is for Tivoli Access Manager implementation of the policy file.

This field is required.

| | |
|---|---|
| **Data type:** | String |
| **Default:** | com.tivoli.pd.as.jacc.TAMPolicy |

## Policy configuration factory class name

Specifies a fully qualified class name that represents the javax.security.jacc.PolicyConfigurationFactory.provider property as per the JACC specification. The class represents the provider-specific implementation of the javax.security.jacc.PolicyConfigurationFactory abstract methods.

This class represents the provider-specific implementation of the PolicyConfigurationFactory abstract class. The class file must reside in the class path of each WebSphere Application Server process. This class is used to propagate the security policy information to the JACC provider during the installation of the J2EE application. The default class name is for the Tivoli Access Manager implementation of the policy configuration factory class name.

This field is required.

**Data type:** String
**Default:** com.tivoli.pd.as.jacc.TAMPolicyConfigurationFactory

## Role configuration factory class name

Specifies a fully qualified class name that implements the com.ibm.wsspi.security.authorization.RoleConfigurationFactory interface.

The class file must reside in the class path of each WebSphere Application Server process. When you implement this class, the authorization table information in the binding file is propagated to the provider during the installation of the J2EE application. The default class name is for the Tivoli Access Manager implementation of the role configuration factory class name.

This field is optional.

**Data type:** String
**Default:** com.tivoli.pd.as.jacc.TAMRoleConfigurationFactory

## Provider initialization class name

Specifies a fully qualified class name that implements the com.ibm.wsspi.security.authorization.InitializeJACCProvider interface.

The class file must reside in the class path of each WebSphere Application Server process. When implemented, this class is called at the start and the stop of all the application server processes. You can use this class for any required initialization that is needed by the provider client code to communicate with the provider server. The properties entered in the custom properties link are passed to the provider when the process starts up. The default class name is for the Tivoli Access Manager implementation of the provider initialization class name.

This field is optional.

**Data type:** String
**Default:** com.tivoli.pd.as.jacc.cfg.TAMConfigInitialize

## Requires the EJB arguments policy context handler for access decisions

Specifies whether the JACC provider requires the EJBArgumentsPolicyContextHandler to make access decisions.

Because this option has an impact on performance, do not set it unless it is required by the provider. Normally, this handler is required only when the provider supports instance-based authorization. Tivoli Access Manager does not support this option for J2EE applications.

**Default:** Disabled

### Supports dynamic module updates

Specifies whether you can apply changes, made to security policies of Web modules in a running application, dynamically without affecting the rest of the application.

If this option is enabled, the security policies of the added or modified Web modules are propagated to the JACC provider and only the affected Web modules are started.

If this option is disabled, then the security policies of the entire application are propagated to the JACC provider for any module-level changes. The entire application is restarted for the changes to take effect.

Typically, this option is enabled for an external JACC provider.

**Default:** Enabled

### Custom properties

Specifies the properties required by the provider.

These properties are propagated to the provider during the start up process when the provider initialization class name is initialized. If the provider does not implement the provider initialization class name as described previously, the properties are not used.

Tivoli Access Manager implementation does not require you to enter any properties in this link.

### Tivoli Access Manager properties

Specifies properties required by the Tivoli Access Manager implementation.

These properties are used to set up the communication between the application server and the Tivoli Access Manager server. You must install and configure the Tivoli Access Manager server before entering these properties.

## Propagating security policy of installed applications to a JACC provider using wsadmin

It is possible that you have applications installed prior to enabling the JACC-based authorization. You can start with default authorization and then move to an external provider based authorization using JACC later on. In this case, the security policy of the previously installed applications would not exist in the JACC provider to make the access decisions. You can reinstall all of the applications once JACC is enabled so that the JACC provider is updated with this information. However, since reinstallation might not be an option in some cases, the wsadmin tool can be used to propagate information to the JACC provider independent of the application install process. The tool eliminates the need for reinstalling the applications.

The tool uses the SecurityAdmin MBean to propagate the policy information in the deployment descriptor of any installed application to the JACC provider. The wsadmin tool can be used to invoke this method at the deployment manager level.

Use `propagatePolicyToJACCProvider(String appNames)` to propagate the policy information in the deployment descriptor of the enterprise archive (EAR) files to the JACC provider. If the RoleConfigurationFactory and the RoleConfiguration interfaces are implemented by the JACC provider, the authorization table information in the binding file of the EAR files is also propagated to the provider. See "Interfaces used to support JACC" on page 348 for more information about these interfaces.

The appNames contains the list of application names, delimited by a colon (:), whose policy information must be stored in the provider. If a null value is passed, the policy information of the deployed applications is propagated to the provider.

Also, be aware of the following items:

- Before migrating application(s) to the Tivoli Access Manager JACC provider, please create or import the users and groups that are in the application(s) to Tivoli Access Manager.
- Depending on the application or the number of applications propagated you might have to increase the request time-out period either in the soap.client.props (if using SOAP) or the sas.client.props (if using RMI) for the command to complete. You can set the request time-out value to 0 to avoid the timeout problem, and change it back to the original value after the command is run.

1. Configure your JACC provider in WebSphere Application Server. See "Configuring a JACC provider" for more information.
2. Restart the server.
3. Enter the following commands:

```
// use the SecurityAdmin Mbean at the Deployment Manager or the unmanaged base application server
wsadmin -user serverID -password serverPWD
set secadm [lindex [$AdminControl queryNames type=SecurityAdmin,*] 0]

// to propagate specific applications security policy information
wsadmin>set appNames [list app1:app2]
// or to propagate all applications installed
wsadmin>set appNames [list null]

// Run the command to propagate
wsadmin>$AdminControl invoke $secadm propagatePolicyToJACCProvider $appNames
```

## Configuring a JACC provider

The Java Contract for Containers (JACC) defines a contract between Java 2 Platform, Enterprise Edition (J2EE) containers and authorization providers. It enables any third party authorization providers to plug into a J2EE 1.4 application server such as the WebSphere Application Server to make the authorization decisions when a J2EE resource is accessed. The JACC provider is implemented using the Tivoli Access Manager.

Read the following articles for more detailed information about JACC before you attempt to configure the WebSphere Application Server to use a JACC provider:

- "JACC support in WebSphere Application Server" on page 337
- "JACC providers" on page 336
- "Tivoli Access Manager integration as the JACC provider" on page 351
-

1. Start the WebSphere Application Server administrative console by clicking `http://yourhost.domain:9060/ibm/console` after starting the WebSphere Application Server. If security is currently disabled, log in with any user ID. If security is currently enabled, log in with a predefined administrative ID and password (this is typically the server user ID specified when you configured the user registry).
2. Click **Security** > **Global Security** from the left navigation menu.
3. Under Authorization, click **Authorization Providers**.
4. Under General Properties, click **External JACC provider**.
5. Under Additional Properties, click **Tivoli Access Manager properties**.
6. Enter the following information:

**Enable embedded Tivoli Access Manager**
Select this option to enable the Tivoli Access Manager.

**Ignore errors during embedded Tivoli Access Manager disablement**
Select this option when you want to unconfigure the JACC provider. Do not select this option during configuration.

**Client listening point set**
WebSphere Application Server must listen using a TCP/IP port for authorization database updates from the policy server. More than one process can run on a particular node or machine

Enter the listening ports used by Tivoli Access Manager clients, separated by a comma. If a range of ports is specified, separate the lower and higher values by a colon (for example, 7999, 9990:999)..

**Policy server**
Enter the name of the Tivoli Access Manager policy server and the connection port. Use the form `policy_server:port`. The policy communication port is set at the time of the Tivoli Access Manager configuration, and the default is 7135.

**Authorization servers**
Enter the name of the Tivoli Access Manager authorization server. Use the form `auth_server:port:priority`. The authorization server communication port is set at the time of the Tivoli Access Manager configuration, and the default is 7136.

More than one authorization server can be specified by separating the entries with commas. Specifying more than one authorization server at a time is useful for reasons of failover and performance.

The priority value is determined by the order of the authorization server use (for example, `auth_server1:7136:1`, and `auth_server2:7137:2`). A priority value of `1` is required when configuring against a single authorization server.

**Administrator user name**
Enter the Tivoli Access Manager administrator user name that was created when Tivoli Access Manager was configured (it is usually `sec_master`).

**Administrator user password**
Enter the Tivoli Access Manager administrator password.

**User registry distinguished name suffix**
Enter the distinguished name suffix for the user registry that is shared between Tivoli Access Manager and WebSphere (for example, `o=inm, c=us`).

**Security domain**
You can create more than one security domain in Tivoli Access Manager, each with its own administrative user. Users, groups and other objects are created within a specific domain, and are not permitted to access resource in another domain.

Enter the name of the Tivoli Access Manager security domain that is used to store WebSphere Application Server users and groups.

If a security domain has not been established at the time of the Tivoli Access Manager configuration, leave the value as `Default`.

**Administrator user distinguished name**
Enter the full distinguished name of the WebSphere security administrator ID (for example, `cn=wasdmin, o=organization, c=country`). The ID name must match the Server user ID on the LDAP User Registry panel in the administrative console. To access the LDAP User Registry panel, click **Security** > **Global Security**. Under User registries, click **LDAP**.

After you have configured a JACC provider, you must enable it in the WebSphere Application Server administrative console. See "Enabling an external JACC provider" on page 342 for more information.

## Interfaces used to support JACC

WebSphere Application Server provides interfaces similar to PolicyConfigurationFactory and PolicyConfiguration so that the information that is stored in the bindings file can be propagated to the provider during installation. The interfaces are called RoleConfigurationFactory and RoleConfiguration . The implementation of these interfaces is optional.

### RoleConfiguration

The RoleConfiguration interface is used to propagate the authorization information to the provider. This interface is similar to the PolicyConfiguration interface found in Java Authorization Contact for Containers (JACC).

```
RoleConfiguration
      - com.ibm.wsspi.security.authorization.RoleConfiguration

/**
 * This interface is used to propagate the authorization table information
 * in the binding file during application install. Implementation of this interface is
 * optional. When a JACC provider implements this interface during an application, both
 * the policy and the authorization table information are propagated to the provider.
* If this is not implemented, only the policy information is propagated as per the JACC specification.
 *
 * @ibm-spi
 * @ibm-support-class-A1
 */


public interface RoleConfiguration

/**
 * Add the users to the role in RoleConfiguration.
 * The role is created, if it doesn't exist in RoleConfiguration.
 * @param role the role name.
 * @param users the list of the user names.
 * @exception RoleConfigurationException if the users cannot be added.
 */
 public void addUsersToRole(String role, List users)
 throws RoleConfigurationException

/**
 * Remove the users to the role in RoleConfiguration.
 * @param role the role name.
 * @param users the list of the user names.
 * @exception RoleConfigurationException if the users cannot be removed.
 */
 public void removeUsersFromRole(String role, List users)
 throws RoleConfigurationException


/**
 * Add the groups to the role in RoleConfiguration.
 * The role is created if it doesn't exist in RoleConfiguration.
 * @param role the role name.
 * @param groups the list of the group names.
 * @exception RoleConfigurationException if the groups cannot be added.
 */
 public void addGroupsToRole(String role, List groups)
 throws RoleConfigurationException

/**
```

```
 * Remove the groups to the role in RoleConfiguration.
 * @param role the role name.
 * @param groups the list of the group names.
 * @exception RoleConfigurationException if the groups cannot be removed.
 */
 public void removeGroupsFromRole( String role, List groups)
 throws RoleConfigurationException


/**
 * Add the everyone to the role in RoleConfiguration.
 * The role is created if it doesn't exist in RoleConfiguration.
 * @param role the role name.
 * @exception RoleConfigurationException if the everyone cannot be added.
 */
 public void addEveryoneToRole(String role)
 throws RoleConfigurationException

/**
 * Remove the everyone to the role in RoleConfiguration.
 * @param role the role name.
 * @exception RoleConfigurationException if the everyone cannot be removed.
 */
 public void removeEveryoneFromRole( String role)
 throws RoleConfigurationException

/**
 * Add the all authenticated users to the role in RoleConfiguration.
 * The role is created if it doesn't exist in RoleConfiguration.
 * @param role the role name.
 * @exception RoleConfigurationException if the authentication users cannot
 *  be added.
 */
 public void addAuthenticatedUsersToRole(String role)
 throws RoleConfigurationException

/**
 * Remove the all authenticated users to the role in RoleConfiguration.
 * @param role the role name.
 * @exception RoleConfigurationException if the authentication users cannot
 * be removed.
 */
 public void removeAuthenticatedUsersFromRole( String role)
 throws RoleConfigurationException

/**
 * This commits the changes in Roleconfiguration.
 * @exception RoleConfigurationException if the changes cannot be
 * committed.
 */
 public void commit( )
 throws RoleConfigurationException

/**
 * This deletes the RoleConfiguration from the RoleConfiguration Factory.
 * @exception RoleConfigurationException if the RoleConfiguration cannot
 * be deleted.
 */
 public void delete( )
 throws RoleConfigurationException

/**
 * This returns the contextID of the RoleConfiguration.
 * @exception RoleConfigurationException if the contextID cannot be
```

```
 * obtained.
 */
 public String getContextID( )
 throws RoleConfigurationException
```

### RoleConfigurationFactory

The RoleConfigurationFactory interface is similar to the PolicyConfigurationFactory interface introduced by JACC, and is used to obtain RoleConfiguration objects based on the contextIDs.

```
RoleConfigurationFactory
 - com.ibm.wsspi.security.authorization.RoleConfigurationFactory

/**
 * This interface is used to instantiate the com.ibm.wsspi.security.authorization.RoleConfiguration
 * objects based on the context identifier similar to the policy context identifier.
 * Implementation of this interface is required only if the RoleConfiguration interface is implemented.
 *
 * @ibm-spi
 * @ibm-support-class-A1
 */

public interface RoleConfigurationFactory
/**
 * This gets a RoleConfiguration with contextID from the
 * RoleConfigurationfactory. If the RoleConfiguration doesn't exist
 * for the contextID in the RoleConfigurationFactory, a new
 * RoleConfiguration with contextID is created in the
 * RoleConfigurationFactory. The contextID is similar to
 * PolicyContextID, but it doesn't contain the module name.
 * If remove is true, the old RoleConfiguration is removed and a new
 * RoleConfiguration is created, and returns with the contextID.
 * @return the RoleConfiguration object for this contextID
 * @param contextID the context ID of RoleConfiguration
 * @param remove true or false
 * @exception RoleConfigurationException if RoleConfiguration
 * can't be obtained.
 **/
public abstract com.ibm.ws.security.policy.RoleConfiguration
         getRoleConfiguration(String contextID, boolean remove)
      throws RoleConfigurationException
```

### InitializeJACCProvider

When implemented by the provider, this interface is called for every process start. All additional properties that are entered during the authorization check are passed to the provider. For example, the provider can use this information to initialize their client code to communicate with their server or repository. The cleanup method is called during server shutdown to clean up the configuration.

### Declaration

public interface **InitializeJACCProvider**

### Description

This interface has two methods. The JACC provider can implement it, and WebSphere Application Server calls it to initialize the JACC provider. The name of the implementation class is obtained from the value of the initializeJACCProviderClassName system property.

This class must reside in a Java archive (JAR) file on the class path of each server that uses this provider.

```
InitializeJACCProvider
   - com.ibm.wsspi.security.authorization.InitializeJACCProvider
```

```
/**
 * Initializes the JACC provider
    * @return 0 for success.
 * @param props the custom properties that are included for this provider will
 * pass to the implementation class.
 * @exception Exception for any problems encountered.
 **/
public int initialize(java.util.Properties props)
throws Exception

/**
 * This method is for the JACC provider cleanup and will be called during a process stop.
 **/
public void cleanup()
```

## Tivoli Access Manager integration as the JACC provider

Tivoli Access Manager uses the Java Authorization Contract for Container (JACC) model in WebSphere Application Server to perform access checks. It consists of the following components.

- Run time
- Client configuration
- Authorization table support
- Access check
- Authentication using the PDLoginModule module

### Tivoli Access Manager run-time changes that are used to support JACC

For the run-time changes, Tivoli Access Manager implements the PolicyConfigurationFactory and the PolicyConfiguration interfaces, as required by JACC. During the application installation, the security policy information in the deployment descriptor and the authorization table information in the binding files is propagated to the Tivoli provider using these interfaces. The Tivoli provider stores the policy and the authorization table information in the Tivoli Access Manager policy server by calling the respective Tivoli Access Manager APIs. The information is stored in the security policy database in the Tivoli Access Manager policy server.

Tivoli Access Manager also implements the RoleConfigurationFactory and the RoleConfiguration interfaces. These interfaces are used to ensure that the authorization table information is passed to the provider with the policy information. See “Interfaces used to support JACC” on page 348 for more information about these interfaces.

### Tivoli Access Manager client configuration

The Tivoli Access Manager client can be configured using either the administrative console or wsadmin scripting. The administrative console panels for the Tivoli Access Manager client configuration are located under the Security center panel. The Tivoli client must be set up to use the Tivoli JACC provider. The setup can be done either before (using wsadmin) or during the time of WebSphere Application Server configuration.

For more information about how to configure the Tivoli Access Manager client, see “Tivoli Access Manager JACC provider configuration” on page 355.

### Authorization table support

Tivoli Access Manager uses the RoleConfiguration interface to ensure that the authorization table information is passed to the Tivoli Access Manager provider when the application is installed or deployed. When an application is deployed or edited, the set of users and groups for the user or group-to-role mapping are obtained from the Tivoli Access Manager server, which shares the same Lightweight Directory

Access Protocol (LDAP) server as WebSphere Application Server. This sharing is accomplished by plugging into the application management users or groups-to-role administrative console panels. The management APIs are called to obtain users and groups rather than relying on the WebSphere Application Server-configured LDAP registry.

**Access check**

When WebSphere Application Server is configured to use the JACC provider for Tivoli Access Manager , it passes the information to Tivoli Access Manager to make the access decision. The Tivoli Access Manager policy implementation queries the local replica of the access control list (ACL) database for the access decision.

**Authentication using the PDLoginModule module**

The custom login module in WebSphere Application Server can do the authentication. This login module is plugged in before the WebSphere Application Server-provided login modules. The custom login modules can provide information that can be stored in the Subject. If the required information is stored, no additional registry calls are made to obtain that information.

As part of the JACC integration, the Tivoli Access Manager-provided PDLoginModule module is also used to plug into WebSphere Application Server for both Lightweight Third Party Authentication (LTPA) and Simple WebSphere Authentication Mechanism (SWAM) authentication. The PDLoginModule module is modified to authenticate with the user ID or password. The module is also used to fill in the required attributes in the Subject so that no registry calls are made by the login modules in WebSphere Application Server. The information that is placed in the Subject is available for the Tivoli Access Manager policy object to use for access checking.

# Tivoli Access Manager security for WebSphere Application Server

WebSphere Application Server Version 6.0 provides embedded IBM Tivoli Access Manager client technology to secure your WebSphere Application Server managed resources.

The benefits of using Tivoli Access Manager described here are only applicable when Tivoli Access Manager client code is used with the Tivoli Access Manager server:

**Note:** Tivoli Access Manager code is not imbedded but bundled in some versions of WebSphere Application Server.

- Robust container-based authorization
- Centralized policy management
- Management of common identities, user profiles, and authorization mechanisms
- Single-point security management for Java 2 Platform, Enterprise Edition (J2EE) compliant and non-compliant J2EE resources using the Tivoli Access Manager Web Portal Manager GUI
- No requirements for coding or deployment changes to applications
- Easy management of users, groups, and roles using the WebSphere Application Server administrative console

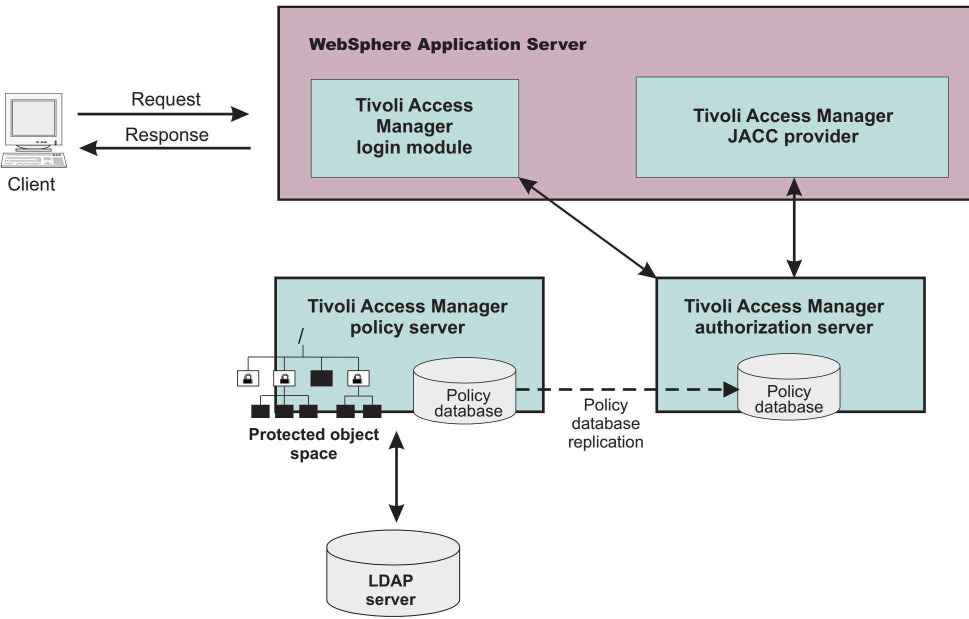
WebSphere Application Server Version 6.0 supports the Java Authorization Contract for Containers (JACC) specification. JACC details the contract requirements for J2EE containers and authorization providers. With this detail, authorization providers can perform the access decisions for resources in J2EE 1.4 application servers such as WebSphere Application Server. The Tivoli Access Manager security utility that is embedded within WebSphere Application Server Version 6.0 is JACC-compliant and is used to:

- Add security policy information when applications are deployed
- Authenticate users
- Authorize access to WebSphere Application Server-secured resources.

When applications are deployed, the embedded Tivoli Access Manger client takes any policy and or user and role information that is stored within the application deployment descriptor and stores it within the Tivoli Access Manager Policy Server.

The Tivoli Access Manager JACC provider is also called when a user requests access to a resource that is managed by WebSphere Application Server.

**Embedded Tivoli Access Manager client architecture**




The previous figure illustrates the following sequence of events:

1. Users that access protected resources are authenticated using the Tivoli Access Manager login module that is configured for use when the embedded Tivoli Access Manager client is enabled.
2. The WebSphere Application Server container uses information from the J2EE application deployment descriptor to determine the required role membership.
3. WebSphere Application Server uses the embedded Tivoli Access Manager client to request an authorization decision (granted or denied) from the Tivoli Access Manager authorization server. Additional context information, when present, is also passed to the authorization server. This context information is comprised of the cell name, J2EE application name, and J2EE module name. If the Tivoli Access Manager policy database has policies that are specified for any of the context information, the authorization server uses this information to make the authorization decision.
4. The authorization server consults the permissions that are defined for the specified user within the Tivoli Access Manager-protected object space. The protected object space is part of the policy database.
5. The Tivoli Access Manager authorization server returns the access decision to the embedded Tivoli Access Manager client.
6. WebSphere Application Server either grants or denies access to the protected method or resource, based on the decision returned from the Tivoli Access Manager Authorization Server.

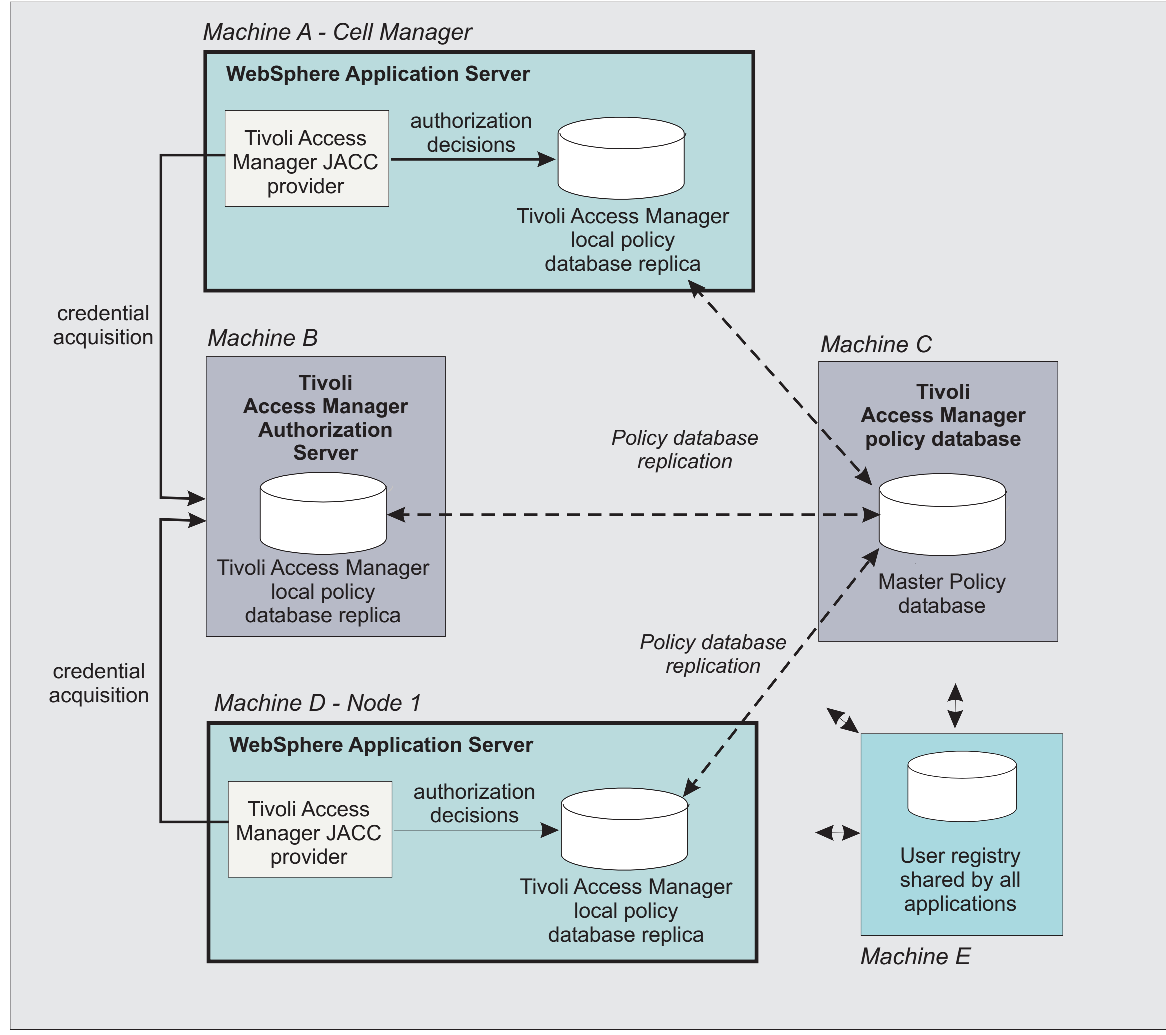
At its core, Tivoli Access Manager provides an authentication and authorization framework. You can learn more about Tivoli Access Manager, including information that is necessary to make deployment decisions, by reviewing the product documentation. Start with the following guides, available at http://publib.boulder.ibm.com/tividd/td/tdprodlist.html:

- *IBM Tivoli Access Manager Base Installation Guide*

  This guide describes how to plan, install, and configure a Tivoli Access Manager secure domain. Using a series of easy installation scripts, you can quickly deploy a fully functional secure domain. These scripts are very useful when prototyping the deployment of a secure domain.
- *IBM Tivoli Access Manager Base Administration Guide*

  This document presents an overview of the Tivoli Access Manager security model for managing protected resources. This guide describes how to configure the Tivoli Access Manager servers that make access control decisions. In addition, detailed instructions describe how to perform important tasks such as declaring security policies, defining protected object spaces, and administering user and group profiles.

**Tivoli Access Manager provides centralized administration of multiple servers.**




The previous figure is an example architecture showing WebSphere Application Servers secured by Tivoli Access Manager.

The participating WebSphere Application Servers use a local replica of the Tivoli Access Manager policy database to make authorization decisions for incoming requests. The local policy databases are replicas of the master policy database that are installed as part of the Tivoli Access Manager installation. Having policy database replicas on each participating WebSphere Application Server optimizes performance when making authorization decisions and provides failover capability.

The authorization server can also be installed on the same system as WebSphere Application Server, although this configuration is not illustrated in the diagram.

All instances of Tivoli Access Manager and WebSphere Application Server in the example architecture share the Lightweight Directory Access Protocol (LDAP) user registry on *Machine E*.

The LDAP registries that are supported by WebSphere Application Server are also supported by Tivoli Access Manager.

**Note:** It is possible to have separate WebSphere Application Server profiles on the same host configured against different Tivoli Access Manager servers. Such an architecture requires the profiles to be configured against separate Java Runtime Environments (JRE) and therefore multiple JREs need to be installed on the same host.

## Creating the security administrative user

Enabling security requires the creation of a WebSphere Application Server administrative user. Use either the Tivoli Access Manager command-line pdadmin utility (available on the policy server host box) to create the Tivoli Access Manager administrative user for WebSphere Application Server. To use the pdadmin utility:

1. From a command line, start the pdadmin utility as the Tivoli Access Manager administrative user, sec_master:

   ```
   pdadmin -a sec_master -p sec_master_password
   ```

2. Create a WebSphere Application Server security user. For example, the following instructions create a new user, *wasadmin*. The command is entered as one continuous line:

   ```
   pdadmin> user create wasadmin cn=wasadmin,o=organization,
   c=country wasadmin wasadmin myPassword
   ```

   Substitute values for organization and country that are valid for your Lightweight Directory Access Protocol (LDAP) user registry.

3. Enable the account for the WebSphere Application Server security administrative user by issuing the following command:

   ```
   pdadmin> user modify wasadmin account-valid yes
   ```

Configure the Java Authorization Contract for Container (JACC) provider for Tivoli Access Manager- “Tivoli Access Manager JACC provider configuration.”

## Tivoli Access Manager JACC provider configuration

The Tivoli Access Manager JACC provider can be configured to deliver authentication and authorization protection for your applications or authentication only. Most deployments using the Tivoli Access Manager JACC provider will configure Tivoli Access Manager to provide both authentication and authorization functionality.

If you want Tivoli Access Manager to provide authentication but leave authorization as part of WebSphere Application Server’s native security, add the following property to the `amwas.amjacc.template.properties` file located on the directory `profiles/`*`profileName`*`/cells/`*`cellName`*.

```
com.tivoli.pd.as.amwas.DisableAddAuthorizationTableEntry=true
```

Once this property is set, perform the tasks for setting Tivoli Access Manager Security as documented.

You can configure the Tivoli Access Manager JACC provider using either the WebSphere Application Server administrative console or the **wsadmin** command line utility.

- For details on configuring the Tivoli Access Manager JACC provider using the administration console, refer to “Configuring the JACC provider for Tivoli Access Manager using the administrative console” on page 358
- For details on configuring the Tivoli Access Manager JACC provider using the **wsadmin** command line utility, refer to “Configuring the JACC provider for Tivoli Access Manager using the wsadmin utility”

**Note:**

Tivoli Access Manager JACC configuration files that are common across multiple WebSphere Application Server profiles are created by default under the `java/jre` directory. The user installing WebSphere Application Server will be given permissions to read and write to the files in this directory. On UNIX platforms, profiles created by users who are different to the user that installed the application will have read-only permissions for this directory. In addition, all users on the iSeries platform will have read-only access to this directory. This is not ideal as configuration of the Tivoli Access Manager JACC provider will fail in these situations.

To avoid this problem read and write permissions can be manually applied to the `java/jre` directory. For iSeries installations however, the permissions for this directory cannot be changed. To avoid this situation the following property can be added to the `etc/amwas.amjacc.template.properties` file.

```
com.tivoli.pd.as.jacc.CommonFileLocation=new location
```

Where *new location* is a fully qualified directory name. This property sets the location of the Tivoli Access Manager JACC provider properties files that are common across profiles.

**Note:** The **wsadmin** command is available to reconfigure the Tivoli Access Manager Java Authorization Contract for Containers (JACC) interface:

```
$AdminTask reconfigureTAM -interactive
```

This command effectively prompts you through the process of unconfiguring the interface and then reconfiguring it.

## Configuring the JACC provider for Tivoli Access Manager using the wsadmin utility

In a network deployment architecture, verify that all the managed servers, including node agents, are started. The following configuration is performed once on the deployment manager server. The configuration parameters are forwarded to managed servers, including node agents, when a synchronization is performed. The managed servers then require their own restart for the configuration changes to take effect.

You can use the wsadmin utility to configure Tivoli Access Manager security for WebSphere Application Server:

1. Start WebSphere Application Server.
2. Start the command-line utility by running the **wsadmin** command from the `install_dir/bin` directory.
3. At the **wsadmin** prompt, enter the following command:

   ```
   $AdminTask configureTAM -interactive
   ```

You are prompted to enter the following information:

| Option | Description |
| --- | --- |
| **WebSphere Application Server node name** | Specify a single node or enter an asterisk (*) to choose all nodes. |
| **Tivoli Access Manager Policy Server** | Enter the name of the Tivoli Access Manager policy server and the connection port. Use the format, *policy_server* **:** *port*. The policy server communication port is set at the time of Tivoli Access Manager configuration – the default port is 7135. |
| **Tivoli Access Manager Authorization Server** | Enter the name of the Tivoli Access Manager authorization server. Use the format *auth_server* **:** *port* **:** *priority*. The authorization server communication port is set at the time of Tivoli Access Manager configuration – the default port is 7136. More than one authorization server can be specified by separating the entries with commas. Having more than one authorization server configured is useful for failover and performance. The priority value is the order of authorization server use. For example: auth_server1**:**7136**:**1,auth_server2**:**7137**:**2. A priority (of 1) is still required when configuring against a single authorization server. |
| **WebSphere Application Server administrator's distinguished name** | Enter the full distinguished name of the WebSphere Application Server security administrator ID as created in "Creating the security administrative user" on page 355. For example: cn=wasadmin,o=*organization*,c=*country* |
| **Tivoli Access Manager user registry distinguished name suffix** | For example: o=*organization*,c=*country* |
| **Tivoli Access Manager administrator's user name** | Enter the Tivoli Access Manager administration user ID, as created at the time of Tivoli Access Manager configuration. This ID is usually, sec_master. |
| **Tivoli Access Manager administrator's user password** | Enter the password for the Tivoli Access Manager administrator. |
| **Tivoli Access Manager security domain** | Enter the name of the Tivoli Access Manager security domain that is used to store users and groups. If a security domain is not already established at the time of Tivoli Access Manager configuration, click **Return** to accept the default. |
| **Embedded Tivoli Access Manager listening port set** | WebSphere Application Server needs to listen on a TCP/IP port for authorization database updates from the policy server. More than one process can run on a particular node and machine so a list of ports is required for the processes. Enter the ports that are used as listening ports by Tivoli Access Manager clients, separated by a comma. If you specify a range of ports, separate the lower and higher values by a colon. For example, 7999, 9990:9999. |
| **Defer** | Set to *yes*, this option defers the configuration of the management server until the next restart. Set to *no*, configuration of the management server occurs immediately. Managed servers are configured on their next restart. |

4. When all information is entered, select **F** to save the configuration properties or **C** to cancel from the configuration process and discard entered information.

Now enable the JACC provider for Tivoli Access Manager- "Enabling the JACC provider for Tivoli Access Manager" on page 361.

## Configuring the JACC provider for Tivoli Access Manager using the administrative console

In a Network Deployment architecture, verify that all the managed servers, including node agents, are started. The following configuration is performed on the management server. When either **Apply** or **OK** is clicked, configuration information is checked for consistency, saved, and applied if successful. In Network Deployment environments, this configuration information is propagated to nodes when a synchronization is performed. Restart the nodes for the configuration changes to take effect.

To configure the Java Authorization Contract for Containers (JACC) provider for Tivoli Access Manager using the administrative console:

1. Click **Security** > **Global security**.
2. Under Authorization, click **Authorization Providers**.
3. Under General properties, select **External authorization using a JACC provider**.
4. Under Related items, click **External JACC provider**.
5. Under Additional properties, click **Tivoli Access Manager Properties**. The Tivoli Access Manager JACC provider configuration screen is displayed.
6. Enter the following information:

| Option | Description |
|---|---|
| **Enable embedded Tivoli Access Manager** | enable |
| **Ignore errors during embedded Tivoli Access Manager disablement** | This option is applicable only when reconfiguring an embedded Tivoli Access Manager client or when disabling an embedded Tivoli Access Manager client. When selected, errors are ignored during disablement of an embedded Tivoli Access Manager client. |
| **Client listening port set** | WebSphere Application Server needs to listen on a TCP/IP port for authorization database updates from the policy server. More than one process can run on a particular node and machine so a list of ports is required for the processes. Enter the ports that are used as listening ports by Tivoli Access Manager clients, with each entry on a new line. If you specify a range of ports, separate the lower and higher values by a colon (:), as shown in the following example:<br><br>`7999`<br>`9990:9999` |
| **Policy Server** | Enter the name, the fully-qualified domain name, or the IP address of the Tivoli Access Manager policy server. Include the connection port. Use the form *policy_server* **:** *port*. The policy server communication port is set at the time of Tivoli Access Manager configuration – the default is 7135. |

| Option | Description |
| --- | --- |
| **Authorization Servers** | Enter the name, the fully-qualified domain name, or the IP address of the Tivoli Access Manager authorization server. Use the form *auth_server* **:** *port* **:** *priority*. The authorization server communication port is set at the time of Tivoli Access Manager configuration – the default is 7136. More than one authorization server can be specified by entering each server on a new line. Having more than one authorization server configured is useful for failover and performance. The priority value is the order of authorization server use. For example:<br><br>`auth_server1:7136:1`<br>`auth_server2:7137:2`<br><br>A priority (of 1) is still required when configuring against a single authorization server. |
| **Administrator user name** | Enter the Tivoli Access Manager administration user ID as created at the time of Tivoli Access Manager configuration. This ID is usually, sec_master. |
| **Administrator user password** | Enter the Tivoli Access Manager administration password for the user ID identified previously. |
| **User registry distinguished name suffix** | Enter the distinguished name suffix for the user registry for Tivoli Access Manager and WebSphere Application Server to share. For example: o=*organization*,c=*country* |
| **Security domain** | More than one security domain can be created in Tivoli Access Manager with its own administrative user. Users, groups, and other objects are created within a specific domain and are not permitted to access resources in another domain. Enter the name of the Tivoli Access Manager security domain that is used to store WebSphere Application Server users and groups. If a security domain is not yet established at the time of Tivoli Access Manager configuration, leave the value as *Default*. |
| **Administrator user distinguished name** | Enter the full distinguished name of the WebSphere Application Server user ID, as created for Tivoli Access Manager in "Creating the security administrative user" on page 355. For example, cn=wasadmin,o=*organization*,c=*country*. The name specified in this field must match the server user ID that is specified on the Lightweight Directory Access Protocol setting panel in the WebSphere Application Server administrative console. To access this panel, click **Security > Global security**. Under User registries, click **LDAP**. |

7. When all information is entered, click **OK** to save the configuration properties. The configuration parameters are checked for validity and the configuration is attempted at the host server or cell manager.

After you click **OK**, WebSphere Application Server completes the following actions:

- Validate the configuration parameters.
- Configure the host server or cell manager.

These processes might take some time depending on network traffic or the speed of your machine.

If the configuration is successful, the parameters are copied to all subordinate servers, including the node agents. To complete the embedded Tivoli Access Manager client configuration, you must restart all of the servers, including the host server, and enable WebSphere Application Server security.

## Tivoli Access Manager JACC provider settings

Use this page to configure the Java Authorization Contract for Container (JACC) provider for Tivoli Access Manager.

To view the JACC provider settings for Tivoli Access Manager, complete the following steps:

1. Click **Security** > **Global security**.
2. Under Authorization, click **Authorization Providers**.
3. Under Related items, click **External JACC provider**.
4. Under Additional properties, click **Tivoli Access Manager Properties**.

### Enable embedded Tivoli Access Manager

Enables or disables the embedded Tivoli Access Manager client configuration.

| | |
|---|---|
| **Default:** | Disabled |
| **Range:** | Enabled or Disabled |

### Ignore errors during embedded Tivoli Access Manager disablement

When selected, errors are ignored during disablement of the embedded Tivoli Access Manager client.

This option is applicable only when reconfiguring an embedded Tivoli Access Manager client or disabling an embedded Tivoli Access Manager.

| | |
|---|---|
| **Default:** | Disabled |
| **Range:** | Enabled or Disabled |

### Client listening port set

Enter the ports that are used as listening ports by Tivoli Access Manager clients.

WebSphere Application Server needs to listen on a TCP/IP port for authorization database updates from the policy server. More than one process can run on a particular node and machine so a list of ports is required for use by the processes. If a range of ports is to be specified, separate the lower and higher values by a colon (:). Single ports and port ranges are specified on separate lines. An example list might look like the following example:

```
7999
 9990:9999
```

### Policy server

Enter the name, fully-qualified domain name, or IP address of the Tivoli Access Manager policy server and the connection port.

Use the form *policy_server*:*port*. The policy server communication port was set at the time of Tivoli Access Manager configuration – the default is 7135.

### Authorization servers

Enter the name, fully-qualified domain name, or IP address of the Tivoli Access Manager authorization server. Use the form *auth_server*:*port*:*priority*.

The authorization server communication port was set at the time of Tivoli Access Manager configuration – the default is 7136. More than one authorization server can be specified by entering each server on a new

line. Having more than one authorization server configured is useful for failover and performance. The priority value is the order of authorization server use. For example:

```
auth_server1.mycompany.com:7136:1
auth_server2.mycompany.com:7137:2
```

A priority (of 1) is still required when configuring against a single authorization server.

### Administrator user name

Enter the Tivoli Access Manager administration user ID, as created at the time of Tivoli Access Manager configuration. This ID is usually, sec_master.

### Administrator user password

Enter the Tivoli Access Manager administration password for the user ID entered in the *Administrator user name* field.

### User registry distinguished name suffix

Enter the distinguished name suffix for the user registry to share between Tivoli Access Manager and WebSphere Application Server. For example: o=*organization*,c=*country*

### Security domain

Enter the name of the Tivoli Access Manager security domain that is used to store WebSphere Application Server users and groups.

Specification of the Tivoli Access Manager domain is required as more than one security domain can be created in Tivoli Access Manager with its own administrative user. Users, groups, and other objects are created within a specific domain and are not permitted to access resources in another domain. If a security domain is not established at the time of Tivoli Access Manager configuration, leave the value as *Default*.

**Default:** Default

### Administrator user distinguished name

Enter the full, distinguished name of the WebSphere Application Server security administrator ID. For example, cn=wasadmin,o=*organization*,c=*country*

## Enabling the JACC provider for Tivoli Access Manager

**Note:** Do not perform this task if you are configuring the Java Authorization Contract for Container (JACC) provider for Tivoli Access Manager to supply authentication services only. Only perform this task for installations that require both Tivoli Access Manager authentication and authorization protection.

The JACC provider for Tivoli Access Manager is configured by default. The following list shows the JACC provider configuration settings for Tivoli Access Manager .

| Field | Value |
|---|---|
| **Name** | Tivoli Access Manager |
| **Description** | This field is optional and used as a reference. |
| **J2EE policy class name** | `com.tivoli.pd.as.jacc.TAMPolicy` |
| **Policy configuration factory class name** | `com.tivoli.pd.as.jacc.TAMPolicyConfigurationFactory` |
| **Role configuration factory class name** | `com.tivoli.pd.as.jacc.TAMRoleConfigurationFactory` |
| **JACC provider initialization class name** | `com.tivoli.pd.as.jacc.cfg.TAMConfigInitialize` |
| **Requires the EJB arguments policy context handler for access decisions** | false |

| Field | Value |
|---|---|
| **Supports dynamic module updates** | true |

To enable the JACC provider for Tivoli Access Manager, use the previous settings and complete the following steps:

1. Click **Security > Global security**.
2. Under Authorization, click **Authorization providers**.
3. Select the **External JACC provider** option.
4. The JACC provider settings for Tivoli Access Manager are displayed. Click **OK**.
5. Save the settings by clicking **Save** at the top of the page; click the **Save** button.
6. Log out of the WebSphere Application Server administrative console.
7. Restart the WebSphere Application Server. The security configuration is now replicated to managed servers and node agents. These other servers within a cell also require restarting before the security changes take effect.

## Configuring additional authorization servers

Tivoli Access Manager secure domains can contain more than one authorization server. Having multiple authorization servers is useful for providing a failover capability as well as improving performance when the volume of access requests is large.

1. Refer to the *Tivoli Access Manager Base Administration Guide* for details on installing and configuring authorization servers. This document is available from http://publib.boulder.ibm.com/tividd/td/tdprodlist.html.
2. Reconfigure the Java Authorization Contract for Containers (JACC) provider using the `$AdminTask reconfigureTAM interactive` wsadmin command. Enter all new and existing options.

## Role-based security with embedded Tivoli Access Manager

The Java 2 Platform, Enterprise Edition (J2EE) role-based authorization model uses the concepts of roles and resources. An example is provided here.

| | **Methods** | | |
|---|---|---|---|
| **Roles** | **getBalance** | **deposit** | **closeAccount** |
| Teller | granted | granted | |
| Cashier | granted | | |
| Supervisor | | | granted |

In the example of the banking application that is conceptualized in the previous table, three roles are defined: teller, cashier, and supervisor. Permission to perform the getBalance, deposit, and closeAccount application methods are mapped to these roles. From the example, you can see that users assigned the role, Supervisor, can run the closeAccount method, whereas the other two roles are unable to run this method.

The term, principal, within WebSphere Application Sever security refers to a person or a process that performs activities. Groups are logical collections of principals that are configured in WebSphere Application Server to promote the ease of applying security. Roles can be mapped to principals, groups, or both. The entry invoked in the following table indicates that the principal or group can invoke any methods that are granted to that role.

| Principal/Group | Roles | | |
|---|---|---|---|
| | Teller | Cashier | Supervisor |
| TellerGroup | Invoke | | |
| CashierGroup | | Invoke | |
| SupervisorGroup | | | |
| Frank - a principal who is not a member of any of the previous groups | | Invoke | Invoke |

In the previous example, the principal Frank, can invoke the getBalance and the closeAccount methods, but cannot invoke the deposit method because this method is not granted either the Cashier or the Supervisor role.

At the time of application deployment, the Java Authorization Contract for Container (JACC) provider of Tivoli Access Manager populates the Tivoli Access Manager-protected object space with any security policy information that is contained in the application deployment descriptor. This security information is used to determine access whenever the WebSphere resource is requested.

By default, the Tivoli Access Manager access check is performed using the role name, the cell name, the application name, and the module name.

Tivoli Access Manager access control lists (ACLs) determine which application roles are assigned to a principal. ACLs are attached to the applications in the Tivoli Access Manager-protected object space at the time of application deployment.

**Note:** Principal-to-role mappings are managed from the WebSphere Application Server administrative console and are never modified using Tivoli Access Manager. Direct updates to ACLs are performed for administrative security users only.

The following sequence of events occur:

1. During application deployment, policy information is sent to the Tivoli Access Manager JACC provider. This policy information contains permission-to-role mappings and role-to-principal and role-to-group mapping information.
2. The Tivoli Access Manager JACC provider converts the information into the required format, and passes this information to the Tivoli Access Manager policy server.
3. The policy server adds entries to the Tivoli Access Manager-protected object space to represent the roles that are defined for the application and the permission-to-role mappings. A permission is represented as a Tivoli Access Manager-protected object and the role granted to this object is attached as an extended attribute.

## Administering security users and roles with Tivoli Access Manager

User-to-role mapping and user-to-group mapping for the Tivoli Access Manager JACC provider are performed using the WebSphere Application Server administrative console. To manage user-to-role mappings and user-to-group mappings for applications:

1. Click **Applications > Enterprise applications >***application_name*.
2. Under Additional properties, click **Map security roles to Tivoli Access Manager users/groups**. The user and groups management screen is displayed.

3. Select the role which requires user or group management and use **Lookup users** or **Lookup groups** to manage the users or groups for the selected role. The native role mapping uses the MapRolesToUsers administrative task. If you are using Tivoli Access Manager, use the TAMMapRolesToUsers administrative task instead. The syntax and options for the Tivoli version are the same as those used in the native version.

## Configuring Tivoli Access Manager groups

The WebSphere Application Server administrative console can be used to specify security policies for applications that run in the WebSphere Application Server environment. The WebSphere Application Server administrative console can also specify security policies for other Web resources, based on the entities that are stored in the registry.

Tivoli Access Manager adds the accessGroup object class to the registry. Tivoli Access Manager administrators can use the pdadmin utility (available only on the policy server host in the PD.RTE fileset) to create new groups. These new groups are added to the registry as the accessGroup object class.

The WebSphere Application Server administrative console is not configured by default to recognize objects of the accessGroup class as user registry groups. You can configure the WebSphere Application Server administrative console to add this object class to the list of object classes that represent user registry groups. To do this configuration, complete the following instructions:

1. From the WebSphere Application Server administrative console, access the advanced settings for configuring security by clicking **Security > Global security**.
2. Under User registries, click **LDAP**.
3. Under Additional properties, click **Advanced Lightweight Directory Access Protocol (LDAP) user registry settings**
4. Modify the **Group Filter** field. Add the following entry: `(objectclass=accessGroup)`

   The Group Filter field then looks like the following example:

   ```
   (&(cn=%w)(|(objectclass=groupOfNames)
   (objectclass=groupOfUniqueNames)(objectclass=accessGroup)))
   ```
5. Modify the **Group Member ID Map** field. Add the following entry: `accessGroup:member` The Group Member ID Map field then looks like the following example:

   ```
   groupOfNames:member;groupOfUniqueNames:uniqueMember;
   accessGroup:member
   ```
6. Stop and restart WebSphere Application Server.

## Tivoli Access Manager JACC provider configuration properties

The Java property files are created in the WebSphere Application Server `install_dir/profiles/profiles_name/etc/tam` directory.

There are two properties files that may require configuration:

- **amwas.*node_server*.amjacc.properties** – contains properties used by the Tivoli Access Manager JACC provider.
- **amwas.*node_server*.pdjlog.properties** – contains logging properties created from the amwas.pdjlog.template.properties file for the specific node and server combination at the time of configuration.

Use **amwas.*node_server*.amjacc.properties** to configure static role caching, dynamic role caching, object caching, and role-based policy framework properties.

## Static role caching properties

The static role cache holds role memberships that do not expire. These properties are in the file, `amwas.`*`node_server`*`.amjacc.properties`, located in the WebSphere Application Server *`install_dir`*`/profiles/`*`profile_name`*`/etc/tam` directory.

### Enabling static role caching

```
com.tivoli.pd.as.cache.EnableStaticRoleCaching=true
```

Enables or disables static role caching. Static role caching is enabled by default.

### Setting the static role cache

```
com.tivoli.pd.as.cache.StaticRoleCache=com.tivoli.pd.as.cache.StaticRoleCacheImpl
```

This property holds the implementation class of the static role cache. You should not need to change this though the opportunity exists to implement your own cache if considered necessary.

### Define static roles

```
com.tivoli.pd.as.cache.StaticRoleCache.Roles=Administrator,Operator,Monitor,Deployer
```

Defines the administration roles for WebSphere Application Server.

**Note:** Application performance can be enhanced by adding the static roles: **CosNamingRead**, **CosNamingWrite**, **CosNamingCreate**, **CosNamingDelete**. These roles allow for improved lookup performance within the application naming service.

## Dynamic role caching properties

The dynamic role cache holds role memberships that expire. These properties are in the file, `amwas.`*`node_server`*`.amjacc.properties`, located in the WebSphere Application Server *`install_dir`*`/profiles/`*`profile_name`*`/etc/tam` directory.

### Enabling dynamic role caching

```
com.tivoli.pd.as.cache.EnableDynamicRoleCaching=true
```

Enables or disables dynamic role caching. Dynamic role caching is enabled by default.

### Setting the dynamic role cache

```
com.tivoli.pd.as.cache.DynamicRoleCache=com.tivoli.pd.as.cache.DynamicRoleCacheImpl
```

This property holds the implementation class of the dynamic role cache. You should not need to change this though the opportunity exists to implement your own cache if considered necessary.

### Specifying the maximum number of users

```
com.tivoli.pd.as.cache.DynamicRoleCache.MaxUsers=100000
```

The maximum number of users that the cache supports before a cache cleanup is performed. The default number of users is 100000.

### Specifying the number of cache tables

```
com.tivoli.pd.as.cache.DynamicRoleCache.NumBuckets=20
```

The number of tables used internally by the dynamic role cache. The default is 20. When a large number of threads use the cache, increase the value to tune and optimize cache performance.

### Specifying the principal lifetime

```
com.tivoli.pd.as.cache.DynamicRoleCache.PrincipalLifeTime=10
```

The period of time in minutes that a principal entry is stored in the cache. The default time is 10 minutes. The term *principal* here refers to the Tivoli Access Manager credential returned from a unique LDAP user.

### Specifying the role lifetime

```
com.tivoli.pd.as.cache.DynamicRoleCache.RoleLifetime=20
```

The period of time in seconds that a role is stored in the role list for a user before it is discarded. The default is 20 seconds.

## Object caching properties

The object cache is used to cache all Tivoli Access Manager objects, including their extended attributes. This bypasses the need to query the Tivoli Access Manager authorization server for each resource request.

These properties are in the file, `amwas.`*`node_server`*`.amjacc.properties`, located in the WebSphere Application Server *`install_dir`*`/profiles/`*`profile_name`*`/etc/tam` directory.

### Enabling object caching

```
com.tivoli.pd.as.cache.EnableObjectCaching=true
```

This property enables or disables object caching. The default value is true.

### Setting the object cache

```
com.tivoli.pd.as.cache.ObjectCache=com.tivoli.pd.as.cache.ObjectCacheImpl
```

This property is the class used to perform object caching. You can implement your own object cache if required. This can be done by implementing the *com.tivoli.pd.as.cache.IObjectCache* interface. The default is *com.tivoli.pd.as.cache.ObjectCacheImpl*.

### Setting the number of cache buckets

```
com.tivoli.pd.as.cache.ObjectCache.NumBuckets=20
```

This property specifies the number of buckets used to store object cache entries in the underlying hash table. The default is 20.

### Setting the number of cache bucket entries

```
com.tivoli.pd.as.cache.ObjectCache.MaxResources=10000
```

This property specifies the total number of entries for all buckets in the cache. This figure, divided by NumBuckets determines the maximum size of each bucket. The default is 10000.

**Setting the resource lifetime**

```
com.tivoli.pd.as.cache.ObjectCache.ResourceLifeTime=20
```

This property specifies the length of time in minutes that objects are kept in the object cache. The default is 20.

These object cache properties cannot be changed after configuration. If any require changing, it should be done before configuration of the nodes in the cell. Changes need to be made in the template properties file before any configuration actions are performed. Properties changed after configuration might cause access decisions to fail.

## Role-based policy framework properties

The role-based policy framework parameters are located in the JACC configuration file and in the authorization configuration file. They are set at the time of JACC provider configuration and authorization server configuration. The role-based policy framework settings for the authorization table and the JACC provider can be modified separately for each WebSphere Application Server instance. The name of the configuration file generated from the authorization table is, `amwas.`*`node_server`*`.authztable.properties`. The name of the configuration file generated from the JACC provider is, `amwas.`*`node_server`*`.amjacc.properties`. Both files are stored on the WebSphere Application Server *`install_dir`*`/profiles/`*`profile_name`*`/etc/tam` directory. It is very unlikely that you will need to change these properties. They are described here for reference:

Supported properties include :

**com.tivoli.pd.as.rbpf.AMAction=i**
: This property is used to signify that a user is granted access to a role. This value is added to a Tivoli Access Manager access control list (ACL). It places invoke access on roles for users and groups.

**com.tivoli.pd.as.rbpf.AMActionGroup=WebAppServer**
: This property sets the Tivoli Access Manager action group that serves as a container for the action specified by the `com.tivoli.pd.as.rbpf.AMAction` property. The permission set in `com.tivoli.pd.as.rbpf.AMAction` goes into this action group.

**com.tivoli.pd.as.rbpf.PosRoot=WebAppServer**
: This property is used to determine where roles are stored in the protected object space.

**com.tivoli.pd.as.rbpf.ProductId=deployedResources**
: This property specifies the location under the root location (specified in the posroot property) to separate other products in the protected object space. Thus, embedded Tivoli Access Manager objects are found in the `/WebAppServer/deployedResources` directory and say AMWLS is in the `/WebAppServer/WLS` directory. The default value is **deployedResources**.

**com.tivoli.pd.as.rbpf.ResourceContainerName=Resources**
: This property specifies the Tivoli Access Manager object space container name for the protected resources. The default location is the `/WebAppServer/deployedResources/Resources` directory.

**com.tivoli.pd.as.rbpf.RoleContainerName=Roles**
: This property specifies the Tivoli Access Manager protected object space container name for the security roles. The default location is the `/WebAppServer/deployedResources/Roles` directory.

The previous settings cannot be changed after configuration. If any of these properties require changing it should be done before configuration of the nodes in the cell. Changes need to be made in the template properties file before any configuration actions are performed. Properties changed after configuration will cause access decisions to fail.

## System-dependent configuration properties

These properties are in the amwas.*node_server*.amjacc.properties file on the *install_dir*/etc directory. They should not be changed and are included here for reference only.

The supported arguments include :

**com.tivoli.pd.as.rbpf.AmasSession.CfgURL=$WAS_HOME/profiles\profile_Name\etc\tam Files\\IBM\\WebSphere\\AppServer\\etc\\amwas.node_server.pdperm.properties**
: This entry is generated by the Java Authorization Contract for Containers (JACC) provider configuration. It specifies the location of the file containing information about the Tivoli Access Manager JACC provider. This entry should not be changed nor the properties in the file it points to.

**com.tivoli.pd.as.rbpf.AmasSession.LoggingURL=file\:/C\:\\Program Files\\IBM\\WebSphere\\AppServer\\etc\\amwas.node_server.pdjlog.properties**
: This entry contains the location of the logging configuration file for the Tivoli Access Manager JACC provider. The file referenced is generated by the Tivoli Access Manager JACC provider configuration. This entry should not be changed.

## Logging Tivoli Access Manager security

Tivoli Access Manager JACC provider messages are logged to the WebSphere Application Server file, SystemOut.log . Trace logging is sent to the WebSphere Application Server file, trace.log. When trace is enabled, all logging, both trace and messaging, is sent to trace.log.

The Tivoli Access Manager JACC provider uses the JLog logging framework as does the Tivoli Access Manager Java runtime environment. Tracing and messaging can be enabled selectively for specific Tivoli Access Manager JACC provider components.

Tracing and message logging for the Tivoli Access Manager JACC provider is configured in the properties file, amwas.*node_server*.pdjlog.properties, located on the etc directory. This file contains logging properties taken from the template file, amwas.pdjlog.template.properties, for the specific node and server combination at the time of Tivoli Access Manager JACC provider configuration.

The contents of this file lets the user control:

- Whether tracing is enabled or disabled for Tivoli Access Manager JACC provider components.
- Whether message logging is enabled or disabled for Tivoli Access Manager JACC provider components.

The amwas.*node_server*.pdjlog.properties file defines several *loggers*, each of which is associated with one Tivoli Access Manager JACC provider component. These loggers include:

| | |
|---|---|
| AmasRBPFTraceLogger AmasRBPFMessageLogger | Used to log messages and trace for the role-based policy framework. This is an underlying framework used by embedded Tivoli Access Manager to make access decisions. |
| AmasCacheTraceLogger AmasCacheMessageLogger | Used to log messages and trace for the policy caches used by the role-based policy framework. |
| AMWASWebTraceLogger AMWASWebMessageLogger | Used to log messages and trace for the WebSphere Application Server authorization plug-in. |
| AMWASConfigTraceLogger AMWASConfigMessageLogger | Used to log messages and trace for the configuration actions for the Tivoli Access Manager JACC provider. |
| JACCTraceLogger JACCMessageLogger | Used to log messages and trace for Tivoli Access Manager JACC provider activity. |

**Note:** Tracing can have a significant impact on system performance and should only be enabled when diagnosing the cause of a problem.

The implementation of these loggers routes messages to the WebSphere Application Server logging sub-system. All messages are written to the WebSphere Application Server's `trace.log` file.

For each logger, the `amwas.`*`node_server`*`.pdjlog.properties` file defines an **isLogging** attribute which, when set to *true*, enables logging for the specific component. A value of *false* disables logging for that component.

`amwas.node_server.pdjlog.properties` defines parent loggers called **MessageLogger** and **TraceLogger** that also have an **isLogging** attribute. If the child loggers do not specify this **isLogging** attribute, they inherit the value of their respective parent. When the Tivoli Access Manager JACC provider is enabled, the **isLogging** attribute is set to *true* for the **MessageLogger** and *false* for the **TraceLogger**. Message logging is therefore enabled for all components and tracing is disabled for all components by default.

To turn on tracing for a Tivoli Access Manager JACC provider component, two operations must occur:

1. The `amwas.`*`node_server`*`.pdjlog.properties` file must be updated and the **isLogging** attribute set to *true* for the required component. For example, to enable tracing for the Tivoli Access Manager JACC provider, the following line must be set to *true* in the `amwas.`*`node_server`*`.pdjlog.properties:baseGroup.AMWASWebTraceLogger.isLogging=true`
2. Enable tracing for the Tivoli Access Manager JACC provider components in the WebSphere Application Server administrative console by completing the following steps:
   a. Click **Troubleshooting** > **Logs and Trace** > *server_name*.
   b. Under Logs and Trace tasks, click **Diagnostic trace**.
   c. Select the **Enable Log** check box.
   d. Click **Apply**.
   e. Click **Troubleshooting** > **Logs and Trace** > *server name*.
   f. Under Logs and Trace tasks, click **Change Log Detail Levels**.
   g. Click **Components.** Tracing for all components can be enabled using **com.tivoli.pd.as.*** or tracing for separate components can be enabled using:
      - **com.tivoli.pd.as.rbpf.*** for role-based policy framework tracing
      - **com.tivoli.pd.as.jacc.*** for JACC provider tracing
      - **com.tivoli.pd.as.pdwas.*** for the authorization table
      - **com.tivoli.pd.as.cfg.*** for configuration
      - **com.tivoli.pd.as.cache.*** for caching
   h. Click **Apply**.

The trace specification should now indicate that tracing is enabled at the required level. Save the configuration, and restart the server for the changes to take effect.

## Enabling embedded Tivoli Access Manager

Embedded Tivoli Access Manager is not enabled by default but needs to be configured for use.

Enabling Tivoli Access Manager security within WebSphere Application Server requires:

- A supported Lightweight Directory Access Protocol (LDAP) installed somewhere on your network. This is the user registry containing the user and group information for both Tivoli Access Manager and WebSphere Application Server.

- A Tivoli Access Manager Version 5.1 domain exists and is configured to use the user registry. For details on the installation and configuration of Tivoli Access Manager refer to the: *Tivoli Access Manager Base installation Guide* and the *Tivoli Access Manager Base Administrator's Guide* available from http://publib.boulder.ibm.com/tividd/td/tdprodlist.html.
- WebSphere Application Server Version 6 is installed either in a single server model or as a network deployment.

Complete the following steps to enable the embedded Tivoli Access Manager security:

1. Create the security administrative user. For more information, see "Creating the security administrative user" on page 355.
2. Configure the Tivoli Access Manager Java Authorization Contract for Containers (JACC) provider. For more information, see "Tivoli Access Manager JACC provider configuration" on page 355.
3. Enable WebSphere Application Server security. When you are using Tivoli Access Manager you must configure LDAP as the user registry. For more information, see "Configuring Lightweight Directory Access Protocol user registries" on page 198.
4. Enable the Tivoli Access Manager JACC provider. For more information, see "Enabling the JACC provider for Tivoli Access Manager" on page 361.

## Disabling embedded Tivoli Access Manager client

You can unconfigure Tivoli Access Manager Security in WebSphere Application Server using either the **wsadmin** command line utility or the WebSphere Application Server Administrative Console.

- For details on unconfiguring embedded Tivoli Access Manager client using the WebSphere Application Server Administration console, refer to "Disabling embedded Tivoli Access Manager client using the Administration Console."
- For details on unconfiguring embedded Tivoli Access Manager client using the **wsadmin** command line utility, refer to "Disabling embedded Tivoli Access Manager client using wsadmin" on page 371.

## Disabling embedded Tivoli Access Manager client using the Administration Console

In a network deployment architecture ensure all managed servers, including node agents, are started then perform the following process once on the deployment management server. Information from the unconfigure operation is forwarded to managed servers, including node agents, when the server is restarted. The managed servers then require their own restart for changes to take effect.

To unconfigure the Tivoli Access Manager JACC provider using the WebSphere Application Server administration console, complete the following steps:

1. Disable global security by clicking **Security** > **Global security** and deselect the **Enable global security** option.
2. Restart the server or, in a network deployment architecture, restart the deployment manager process.
3. Select **Security** > **Global security**.
4. Under Authorization, click **Authorization Providers**.
5. Under Related items, click **External JACC provider**.
6. Under Additional properties, click **Tivoli Access Manager Properties**. The configuration screen for the Tivoli Access Manager JACC provider is displayed.
7. Deselect the **Enable embedded Tivoli Access Manager** option. If you want to ignore errors when unconfiguring, select the **Ignore errors during embedded Tivoli Access Manager disablement** option. Select this option only when the Tivoli Access Manager domain is in an irreparable state.
8. Click **OK**.
9. **Optional:** If you want security enabled, without Tivoli Access Manager, re-enable global security.

10. **Optional:** In network deployment environments, synchronize all nodes.
11. Restart all WebSphere Application Server instances for the changes to take effect.

## Disabling embedded Tivoli Access Manager client using wsadmin

In a network deployment architecture ensure all managed servers, including node agents, are started then perform the following process once on the deployment management server. Details of the unconfiguration are forwarded to managed servers, including node agents, when a synchronization is performed. The managed servers then require their own reboot for the configuration changes to take effect.

To unconfigure the Tivoli Access Manager JACC provider:

1. Disable global security by clicking **Security** > **Global security** and deselect the **Enable global security** option.
2. Restart the server or, in a network deployment architecture, restart the deployment manager process.
3. Start the **wsadmin** command line utility. The **wsadmin** command is found in install_dir\AppServer\bin.
4. From the **wsadmin** prompt, enter the following command:

```
WSADMIN>$AdminTask unconfigureTAM -interactive
```

You are prompted to enter the following information:

| Option | Description |
|---|---|
| **WebSphere Application Server node name** | Enter * to select all nodes. |
| **Tivoli Access Manager administrator's user name** | Enter the Tivoli Access Manager administration user ID as created at the time of Tivoli Access Manager configuration. This name is usually, sec_master. |
| **Tivoli Access Manager administrator's user password** | Enter the password for the Tivoli Access Manager administrator. |
| **Force** | Enter *yes* if you want to ignore errors when unconfiguring the Tivoli Access Manager Java Authorization Contract for Containers (JACC) provider. Enter this option as *yes* only when the Tivoli Access Manager domain is in an irreparable state. |
| **Defer** | Enter `no` to force the unconfiguration of the connected server. Enter `No` for the unconfiguration to proceed correctly. |

5. When all information is entered, enter `F` to save the properties or `C` to cancel from the unconfiguration process and discard entered information.
6. **Optional:** If you want security enabled, not using Tivoli Access Manager, re-enable global security.
7. **Optional:** In network deployment environments, synchronize all nodes.
8. Restart all WebSphere Application Server instances for the changes to take effect.

## Forcing the unconfiguration of the Tivoli Access Manager JACC provider

If you find you cannot restart WebSphere Application Server after configuring the Tivoli Access Manager JACC provider a utility is available to clear the security configuration and return WebSphere Application Server to an operable state.

The utility removes all of the **PDLoginModuleWrapper** entries as well as the Tivoli Access Manager authorization table from `security.xml` and `wsjaas.conf`. This effectively removes the Tivoli Access Manager JACC provider.

1. Back-up `security.xml` and `wsjaas.conf`.
2. Enter the following command as one continuous line:

```
WAS_HOME/java/jre/bin/java -classpath ″WAS_HOME/lib/AMJACCProvider.jar:CLASSPATH″ com.tivoli.pd.as.jacc.cfg.
```

## Updating console users and groups

Additions and changes to console users and groups are not automatically added to the Tivoli Access Manager object space once the Tivoli Access Manager JACC provider is configured. Changes to console users and groups are saved in the `admin-authz.xml` file and this file will require migration before any changes take effect. The Tivoli Access Manger JACC provider includes the migration utility, migrateEAR, for incorporating console user and group changes into the Tivoli Access Manager object space.

**Note:** The migrateEAR utility is used to migrate the changes made to console users and groups *after* the Tivoli Access Manager JACC provider has been configured. The utility will not need to be run for changes and additions to console user and groups made prior to the Tivoli Access Manager JACC provider being configured as the changes (made to `admin-authzn.xml`) are automatically migrated at configuration time. Furthermore, the migration tool does not need to be run before deploying standard J2EE applications, J2EE application policy deployment is also performed automatically.

To migrate `admin-authzn.xml`:

1. Before executing the `migrateEAR` utility, setup the environment by running `setupCmdLine.bat` or `setupCmdLine.sh` located in the *installation*/`bin` directory.
2. Make sure that the *WAS_HOME* environment variable is set to the WebSphere Application Server installation directory.
3. Change to the directory where the `migrateEAR` utility is located: `${WAS_HOME}/bin/`
4. Run the `migrateEAR` utility to migrate the data contained in `admin-authzn.xml`. Use the parameter descriptions listed in The Tivoli Access Manager migrateEAR5 utility. For example:

```
migrateEAR
\ -j "c:\Program Files\IBM\WebSphere\AppServer\profiles\profileName\config\cells\cellName\admin-authz.xml"
\ -a sec_master
\ -p password
\ -w wsadmin
\ -d o=ibm,c=us
\ -c file:/"c:\Program Files\IBM\WebSphere\AppServer\java\jre\PdPerm.properties"
```

   A status message is displayed when the migration completes. Output of the utility is logged to the file, `pdwas_migrate.log`, created on the directory where the utility is run. Check the log file after each migration. If the log file displays errors, check the last recorded transaction, correct the source of the error, and rerun the migration utility. If the migration is unsuccessful, verify that you supplied the correct values for the `-c` and `-j` options.
5. WebSphere Application Server does **not** require a restart for the changes to take effect.

## The Tivoli Access Manager migrateEAR utility

### Purpose

Migrates changes made to console users and groups in the `admin-authzn.xml` file into the Tivoli Access Manager object space.

### Syntax

```
migrateEAR
-j path
-c URI
-a admin_ID
-p admin_pwd
-w Websphere_admin_user
-d user_registry_domain_suffix
[-r root_objectspace_name]
[-t ssl_timeout]
```

### Parameters

**-a admin_ID**
Specifies the administrative user identifier. The administrative user must have the privileges required to create users, objects, and access control lists (ACLs). For example, `-a sec_master`.

This parameter is optional. When the parameter is not specified, you are prompted to supply it at run time.

**-c URI**
Specifies the Uniform Resource Indicator (URI) location of the `PdPerm.properties` file that is configured by the pdwascfg utility. When WebSphere Application Server is installed in the default location, the URI is:

Solaris, Linux and HP-UX -
`file:/opt/IBM/WebSphere/AppServer/java/jre/PdPerm.properties`AIX –
`file:/usr/IBM/WebSphere/AppServer/java/jre/PdPerm.properties`
Windows –
`file:/"c:\Program Files\IBM\WebSphere\AppServer\java\jre\PdPerm.properties"`

**-d user_registry_domain_suffix**
Specifies the domain suffix for the user registry to use. For example, for Lightweight Directory Access Protocol (LDAP) user registries, this value is the domain suffix, such as: `"o=ibm,c=us"`

Windows platforms require that the domain suffix is enclosed within quotes.

You can use the **pdadmin user show** command to display the distinguished name (DN) for a user.

**-j path**
Specifies the fully qualified path and file name of the Java 2 Platform Enterprise Edition application archive file. Optionally, this path can also be a directory of an expanded enterprise application. When WebSphere Application Server is installed in the default location, the paths to data files to migrate include:

Solaris, Linux and HP-UX -
file:/opt/IBM/WebSphere/AppServer/profiles/*profileName*/config/cells/*cellName*/admin-authz.xml
AIX –
file:/usr/IBM/WebSphere/AppServer/profiles/*profileName*/config/cells/*cellName*/admin-authz.xml
Windows –
"c:\Program Files\IBM\WebSphere\AppServer\profiles*profileName*\config\cells*cellName*\admin-authz.xml"

**-p admin_pwd**
Specifies the password for the Tivoli Access Manager administrative user. The administrative user must have the privileges required to create users, objects, and access control lists (ACLs). For example, you can specify the password for the `-a sec_master` administrative user as `-p myPassword`.

This parameter is optional. When it is not specified, the user is prompted to supply the password for the administrative user name.

**-r root_objectspace_name**
Specifies the space name of the root object. The value is the name of the root of the protected object namespace hierarchy that is created for WebSphere Application Server policy data. This parameter is optional.

The default value for the root object space is `WebAppServer`.

The Tivoli Access Manager root object space name is set by modifying the `amwas.amjacc.template.properties` prior to configuring the Tivoli Access Manager JACC provider for the first time. This option should be used if the default object space value is not used in the configuration of the Tivoli Access Manager JACC provider.

The Tivoli Access Manager object space name should never be changed after the Tivoli Access Manager JACC provider has been configured.

**-t ssl_timeout**
Specifies the number of minutes for the Secure Sockets Layer (SSL) timeout. This parameter is used to disconnect and reconnect the SSL context between the Tivoli Access Manager authorization server and policy server before the default connection times out.

The default is 60 minutes. The minimum is 10 minutes. The maximum value cannot exceed the Tivoli Access Manager ssl-v3-timeout value. The default value for ssl-v3-timeout is 120 minutes.

This parameter is optional. If you are not familiar with the administration of this value, you can safely use the default value.

**-w WebSphere_admin_user**
Specifies the user name that is configured in the WebSphere Application Server security user registry field as the administrator. This value matches the account that you created or imported in Creating a Tivoli Access Manager administrative user for WebSphere Application Server. Access permission for this user is needed to create or update the Tivoli Access Manager protected object space.

When the WebSphere Application Server administrative user does not already exist in the protected object space, it is created or imported. In this case, a random password is generated for the user and the account is set to `not valid`. Change this password to a known value and set the account to `valid`.

A protected object and access control list (ACL) are created. The administrative user is added to the `pdwas-admin` group with the following ACL attributes:

**T**
Traverse permission

**i**
Invoke permission

**WebAppServer**
Specifies the action group name. `WebAppServer` is the default name. This action group name (and the matching root object space) can be overwritten when the migration utility is run with the -r option.

### Comments

This utility migrates security policy information from deployment descriptors (enterprise archive files) to Tivoli Access Manager for WebSphere Application Server. The script calls the Java class: `com.tivoli.pdwas.migrate.Migrate`.

Before invoking the script you must run `setupCmdLine.bat` or `setupCmdLine.sh`. These files can be found in the `%WAS_HOME%/bin` directory.

The script is dependent on finding the correct environment variables for the location of prerequisite software. The script calls Java code with the following options:

**-Dpdwas.lang.home**
The directory containing the native language support libraries that are provided with the Tivoli

Access Manager JACC provider. These libraries are located in a subdirectory under the Tivoli Access Manager JACC provider installation directory. For example: `-Dpdwas.lang.home=%PDWAS_HOME%\java\nls`

**-cp %CLASSPATH% com.tivoli.pdwas.migrate.Migrate**
The CLASSPATH variable must be set correctly for your Java installation.

On Windows platforms, both the -j option and the -c option can reference the %WAS_HOME% variable to determine where WebSphere Application Server is installed. This information is used to:

- Build the full path name of the enterprise archive file.
- Build the full URI path name to the location of the `PdPerm.properties` file.

### Return codes

The following exit status codes can be returned:

**0** The command completed successfully.

**1** The command failed.

## Troubleshooting authorization providers

This article describes the issues you might encounter using a Java Contract for Containers (JACC) authorization provider. Tivoli Access Manager is bundled with WebSphere Application Server as an authorization provider. However, you also can plug in your own authorization provider.

### Using Tivoli Access Manager as a Java Contract for Containers authorization provider

You might encounter the following issues when using Tivoli Access Manager as a JACC authorization provider:

- The configuration of JACC might fail.
- The server might fail to start after configuring JACC.
- The application might not deploy properly.
- The startServer command might fail after you have configured Tivoli Access Manager or a clean uninstall did not take place after unconfiguring JACC.
- An ″HPDIA0202w An unknown user name was presented to Access Manager″ error might occur.
- An ″HPDAC0778E The specified user's account is set to invalid″ error might occur.
- An WASX7017E: Exception received while running file ″InsuranceServicesSingle.jacl″ error might occur.

### Using an external provider for Java Contract for Containers authorization

You might encounter the following issues when you use an external provider for JACC authorization:

- An ″HPDJA0506E Invalid argument: Null or zero-length user name field for the ACL entry″ error might occur.

### The configuration of JACC might fail

If you are having problems configuring JACC, check the following:

- Ensure that the parameters are correct. For example, there should not be a number after TAM_Policy_server_hostname:7135, but there should be a number after TAM_Authorization_server_hostname:7136 (for example, TAM_Authorization_server_hostname:7136:1).
- If a message such as "server can't be contacted" appears, it is possible that the host names or port numbers of the Tivoli Access Manager servers are incorrect, or that the Tivoli Access Manager servers have not been started.

- Ensure that the password for sec_master is correct.
- Check the SystemOut.log and search for the string AMAS to see if any error messages are present.

**The server might fail to start after configuring JACC**

If the server does not start after JACC has been configured, check the following:

- Ensure that the WebSphere Application Server and Tivoli Access Manager use the same Lightweight Directory Access Protocol (LDAP) server.
- If the message "Policy Director Authentication failed" appears, ensure that the:
  - WebSphere Application Server LDAP serverID is the same as the "Administrator user" in the Tivoli Access Manager JACC configuration panel.
  - Tivoli Access Manager Administrator distinguished name (DN) is correct.
  - Password of the Tivoli Access Manager administrator has not expired and is valid.
  - Account is valid for the Tivoli Access Manager administrator.
- If a message such as "socket can't be opened for *xxxx*" (where *xxxx* is a number) appears, do the following:
  1. Go to $WAS_HOME/profiles/profile_name/etc/tam.
  2. Change *xxxx* to an available port number in amwas.commomconfig.properties, and amwas*cellName_dmgr.properties if dmgr failed to start. If Node failed to start, change *xxx* to an available port number in amwas*cellName_nodeName_.properties. If appSever failed to start, change *xxxx* in Amwas*cellname_nodeName_serverName.properties.

**The application might not deploy properly**

When you click **Save**, the policy and role information is propagated to the Tivoli Access Manager policy. It might take some time to finish. If the save fails, you must uninstall the application and then reinstall it.

To access an application after it is installed, you must wait 30 seconds (by default) to start the application after you save.

**The startServer command might fail after you have configured Tivoli Access Manager or a clean uninstall did not take place after unconfiguring JACC.**

If the cleanup for JACC unconfiguration or start server fails after JACC has been configured, do the following:

- Remove Tivoli Access Manager properties files from WebSphere Application Server. For each application server in a network deployment (ND) environment with N servers defined (for example, server1, server2), the following files must be removed:

  ```
  $WAS_INSTALL/java/jre/PdPerm.properties
  $WAS_INSTALL/java/jre/PdPerm.ks
  $WAS_INSTALL/profiles/profile_name/etc/tam/*
  ```
- Use a utility to clear the security configuration and return the system to the state it was in before Tivoli Access Manager JACC was configured. The utility removes all of the PDLoginModuleWrapper entries as well as the Tivoli Access Manager authorization table entry from the security.xml file, effectively removing the Tivoli Access Manager JACC provider. Backup security.xml before running this utility.

  Enter the following commands:

  ```
  $WAS_HOME/java/jre/bin/java -classpath
   "$WAS_HOME/lib/AMJACCProvider.jar:CLASSPATH"
  com.tivoli.pd.as.jacc.cfg.CleanSecXML fully_qualified_path/security.xml
  ```

### An "HPDIA0202w An unknown user name was presented to Access Manager" error might occur

You might encounter the following error message if you are attempting to use an existing user in a Local Directory Access Protocol (LDAP) user registry with Tivoli Access Manager:

```
AWXJR0008E   Failed to create a PDPrincipal for principal mgr1.:
AWXJR0007E   A Tivoli Access Manager exception was caught. Details are:
"HPDIA0202W   An unknown user name was presented to Access Manager."
```

To correct this error, complete the following steps:

1. On the command line, type the following information to get a Tivoli Access Manager command prompt:

   ```
   pdadmin -a administrator_name -p administrator_password
   ```

   The pdadmin *administrator_name* prompt is displayed. For example:

   ```
   pdadmin -a administrator1 -p password
   ```
2. At the pdadmin command prompt, import the user from the LDAP user registry to Tivoli Access Manager by typing the following information:

   ```
   user import user_name cn=user_name,o=organization_name,c=country
   ```

   For example:

   ```
   user import jstar cn=jstar,o=ibm,c=us
   ```

After importing the user to Tivoli Access Manager, you must use the `user modify` command to set the user account to valid. The following syntax shows how to use this command:

```
user modify user_name account-valid yes
```

For example:

```
user modify jstar account-valid yes
```

For information on how to import a group from LDAP to Tivoli Access Manager, see the Tivoli Access Manager documentation.

### An "HPDAC0778E The specified user's account is set to invalid" error might occur

You might encounter the following error message after you import a user to Tivoli Access Manager and restart the client:

```
AWXJR0008E   Failed to create a PDPrincipal for principal mgr1.:
AWXJR0007E   A Tivoli Access Manager exception was caught.
Details are: "HPDAC0778E   The specified user's account is set to invalid."
```

To correct this error, use the `user modify` command to set the user account to valid. The following syntax shows how to use this command:

```
user modify user_name account-valid yes
```

For example:

```
user modify jstar account-valid yes
```

### An "HPDJA0506E Invalid argument: Null or zero-length user name field for the ACL entry" error might occur

You might encounter an error similar to the following message when you propagate the security policy information from the application to the provider using the wsadmin command propagatePolicyToJACCProvider:

```
AWXJR0035E   An error occurred while  attempting to add member, cn=agent3,o=ibm,c=us, to role AgentRole
HPDJA0506E   Invalid argument: Null or zero-length user name field for the ACL entry
```

To correct this error, create or import the user, which is mapped to the security role to the Tivoli Access Manager. For more information on propagating the security policy information, see the documentation for your authorization provider.

### An WASX7017E: Exception received while running file "InsuranceServicesSingle.jacl" error might occur

After the JACC provider and Tivoli Access Manager are enabled, when attempting to install the application (which is configured with security roles using the wsadmin command), the following error might occur:

```
WASX7017E: Exception received while running file "InsuranceServicesSingle.jacl"; exception information:
 com.ibm.ws.scripting.ScriptingException: WASX7111E: Cannot find a match for supplied option:
"[RuleManager, , , cn=mgr3,o=ibm,c=us|cn=agent3,o=ibm,c=us, cn=ManagerGro
up,o=ibm,c=us|cn=AgentGroup,o=ibm,c=us]" for task "MapRolesToUsers
```

The $AdminApp task option MapRolesToUsers becomes invalid when Tivoli Access Manager is used as the authorization server. To correct the error, change MapRolesToUsers to TAMMapRolesToUsers.

## Authentication protocol for EJB security

In WebSphere Application Server Version 6, two authentication protocols are available to choose from: Secure Authentication Service (SAS) and Common Secure Interoperability Version 2 (CSIv2). SAS is the authentication protocol used by all previous releases of WebSphere Application Server and is maintained for backwards compatibility. The Object Management Group (OMG) has defined the authentication protocol called CSIv2 so that vendors can interoperate securely. CSIv2 is implemented in WebSphere Application Server with more features than SAS and it is considered the strategic protocol.

Invoking EJB methods in a secure WebSphere Application Server environment requires an authentication protocol to determine the level of security and the type of authentication, which occur between any given client and server for each request. It is the job of the authentication protocol during a method invocation to merge the server authentication requirements (determined by the object Interoperable Object Reference (IOR)) with the client authentication requirements (determined by the client configuration) and come up with an authentication policy specific to that client and server pair.

The authentication policy makes the following decisions, among others, which are all based on the client and server configurations:
- What kind of connection can you make to this server--SSL or TCP/IP?
- If Secure Sockets Layer (SSL) is chosen, how strong is the encryption of the data?
- If SSL is chosen, do you authenticate the client using client certificates?
- Do you authenticate the client with a user ID and password? Does an existing credential exist?
- Do you assert the client identity to downstream servers?
- Given the configuration of the client and server, can a secure request proceed?

You can configure both protocols (SAS and CSIv2) to work simultaneously. If a server supports both protocols, it exports an IOR containing tagged components describing the configuration for SAS and CSIv2. If a client supports both protocols, it reads tagged components for both CSIv2 and SAS. If the client supports both and the server supports both, CSIv2 is used. However, if the server supports SAS (for

example, it is a previous WebSphere Application Server release) and the client supports both, the client chooses SAS for this request because the SAS protocol is what both have in common.

Choose a protocol by specifying the `com.ibm.CSI.protocol` property on the client side and configuring through the administrative console on the server side. More details are included in the SAS and CSIv2 properties articles.

**Common Secure Interoperability Specification, Version 2**

The Common Secure Interoperability Specification, Version 2 (CSIv2) defines the Security Attribute Service (SAS) that enables interoperable authentication, delegation and privileges. The CSIv2 SAS and SAS protocols are entirely different. The CSIv2 SAS is a subcomponent of CSIv2 that supports SSL and interoperability with the EJB Specification, Version 2.1.

**Security Attribute Service**

The Common Secure Interoperability Specification, Version 2 Security Attribute Service (CSIv2 SAS) protocol is designed to exchange its protocol elements in the service context of a General Inter-ORB Protocol (GIOP) request and reply messages that are communicated over a connection-based transport. The protocol is intended for use in environments where transport layer security, such as that available through Secure Sockets Layer (SSL) and Transport Layer Security (TLS), is used to provide message protection (that is, integrity and or confidentiality) and server-to-client authentication. The protocol provides client authentication, delegation, and privilege functionality that might be applied to overcome corresponding deficiencies in an underlying transport. The CSIv2 SAS protocol facilitates interoperability by serving as the higher-level protocol under which secure transports can be unified.

**Connection and request interceptors**

The authentication protocols used by WebSphere Application Server are add-on Interoperable Inter-ORB Protocol (IIOP) services. IIOP is a request-and-reply communications protocol used to send messages between two Object Request Brokers (ORBs). For each request made by a client ORB to a server ORB, an associated reply is made by the server ORB back to the client ORB. Prior to any request flowing, a connection between the client ORB and the server ORB must be established over the TCP/IP transport (SSL is a secure version of TCP/IP). The client ORB invokes the authentication protocol client connection interceptor, which is used to read the tagged components in the IOR of the object located on the server. As mentioned previously, this is where the authentication policy is established for the request. Given the authentication policy (a coalescing of the server configuration with the client configuration), the strength of the connection is returned to the ORB. The ORB makes the appropriate connection, usually over SSL.

After the connection is established, the client ORB invokes the authentication protocol client request interceptor, which is used to send security information other than what is established by the transport. The security information includes the user ID and password token (authenticated by the server), an authentication mechanism-specific token (validated by the server), or an identity assertion token. Identity assertion is a way for one server to trust another server without the need to reauthenticate or revalidate the originating client. However, some work is required for the server to trust the upstream server. This additional security information is sent with the message in a *service context*. A service context has a registered identifier so that the server ORB can identify which protocol is sending the information. The fact that a service context contains a unique identity is another way for WebSphere Application Server to support both SAS or z/SAS and CSIv2 simultaneously because both protocols have different service context IDs. After the client request interceptor finishes adding the service context to the message, the message is sent to the server ORB.

When the message is received by the server ORB, the ORB invokes the authentication protocol server request interceptor. This interceptor looks for the service context ID known by the protocol. When both SAS or z/SAS and CSIv2 are supported by a server, two different server request interceptors are invoked and both interceptors look for different service context IDs. However, only one finds a service context for

any given request. When the server request interceptor finds a service context, it reads the information in the service context. A method is invoked to the security server to authenticate or validate client identity. The security server either rejects the information or returns a credential. A credential contains additional information about the client, retrieved from the user registry so that authorization can make the appropriate decision. Authorization is the process of determining if the user can invoke the request based on the roles applied to the method and the roles given to the user. If the request is rejected by the security server, a reply is sent back to the client without ever invoking the business method.

If a service context is not found by the CSIv2 server request interceptor, the interceptor then looks at the transport connection to see if a client certificate chain was sent. This is done when SSL client authentication is configured between the client and server.

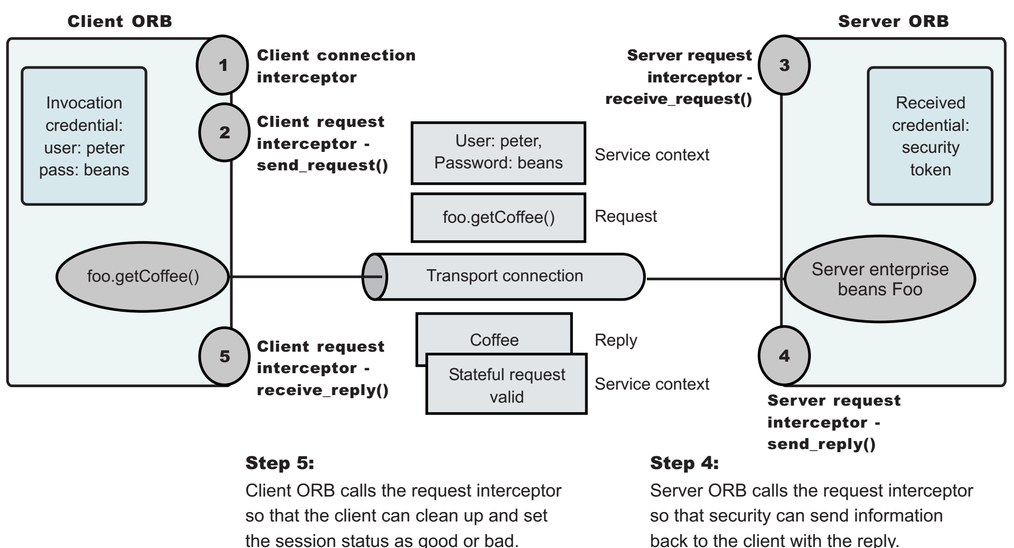
If a client certificate chain is found, the distinguished name (DN) is extracted from the certificate and is used to map to an identity in the user registry. If the user registry is Lightweight Directory Access Protocol (LDAP), the search filters defined in the LDAP registry configuration determine how the certificate maps to an entry in the registry. If the user registry is local OS, the first attribute of the distinguished name (DN) maps to the user ID of the registry. This attribute is typically the common name.

If the certificate does not map, no credential is created and the request is rejected. When invalid security information is presented, the method request is rejected and a NO_PERMISSION exception is sent back with the reply. However, when no security information is presented, an unauthenticated credential is created for the request and the authorization engine determines if the method gets invoked or not. For an unauthenticated credential to invoke an Enterprise JavaBean (EJB) method, either no security roles are defined for the method or a special **Everyone** role is defined for the method.

When the method invocation is completed in the EJB container, the server request interceptor is invoked again to complete server authentication and a new reply service context is created to inform the client request interceptor of the outcome. This process is typically for making the request *stateful*. When a stateful request is made, only the first request between a client and server requires that security information is sent. All subsequent method requests need to send a unique context ID only so that the server can look up the credential stored in a session table. The context ID is unique within the connection between a client and server.

Finally, the method request cycle is completed by the client request interceptor receiving a reply from the server with a reply service context providing information so the client side stateful context ID can be confirmed and reused.

Specifying a stateful client is done through the property com.ibm.CSI.performStateful (true/false). Specifying a stateful server is done through the administrative console configuration.




. *Authentication protocol flow*

## Authentication policy for each request

The authentication policy of a given request determines the security protection between a client and a server. A client or server authentication protocol configuration can describe required features, supported features and non-supported features. When a client requires a feature, it can only talk to servers that either require or support that feature. When a server requires a feature, it can only talk to clients that either require or support that feature. When a client supports a feature, it can talk to a server that supports or requires that feature, but can also talk to servers that do not support the feature. When a server supports a feature, it can talk to a client that supports or requires the feature, but can also talk to clients that do not support the feature (or chose not to support the feature).

For example, for a client to support client certificate authentication, some setup is required to either generate a self-signed certificate or to get one from a certificate authority (CA). Some clients might not need to complete these actions, therefore, you can configure this feature as `not supported`. By making this decision, the client cannot communicate with a secure server requiring client certificate authentication. Instead, this client can choose to use the user ID and password as the method of authenticating itself to the server.

Typically, supporting a feature is the most common way of configuring features. It is also the most successful during run time because it is more forgiving than requiring a feature. Knowing how secure servers are configured in your domain, you can choose the right combination for the client to ensure successful method invocations and still get the most security. If you know that all of your servers support both client certificate and user ID and password authentication for the client, you might want to require one and not support the other. If both the user ID and password and the client certificate are supported on the client and server, both are performed but user ID and password take precedence at the server. This action is based on the CSIv2 specification requirements.

## Common Secure Interoperability Version 2 features

The following Common Secure Interoperability Version 2 (CSIv2) features are available in IBM WebSphere Application Server: Secure Sockets Layer (SSL) client certificate authentication, message layer authentication, identity assertion, and security attribute propagation.

- SSL Client Certificate authentication.

  An additional way to authenticate a client to a server using SSL client authentication.
- Message Layer Authentication.

  Authenticates credential information and sends that information across the network so that a receiving server can interpret it.
- Identity Assertion.

  Supports a downstream server in accepting the client identity that is established on an upstream server, without having to authenticate again. The downstream server trusts the upstream server.
- Security attribute propagation

  Supports the use of the authorization token to propagate serialized Subject contents and PropagationToken contents with the request. You can propagate these objects using a pure client or a server login that adds custom objects to the Subject. Propagating security attributes prevents downstream logins from having to make UserRegistry calls to look up these attributes.

  Propagating security attributes is also useful when the security attributes contain information that is only available at the time of authentication (meaning this information cannot be located using the UserRegistry on downstream servers).

## Identity assertion

*Identity assertion* is the invocation credential that is asserted to the downstream server.

When a client authenticates to a server, the received credential is set. When authorization checks the credential to determine whether access is permitted, it also sets the *invocation* credential so that if the Enterprise JavaBeans (EJB) method calls another EJB method that is located on other servers, the invocation credential can be the identity used to invoke the downstream method. Depending on the RunAs mode for the enterprise beans, the invocation credential is set as the originating client identity, the server identity, or a specified different identity. Regardless of the identity that is set, when identity assertion is enabled, it is the invocation credential that is asserted to the downstream server.

The invocation credential identity is sent to the downstream server in an identity token. In addition, the sending server identity, including the password or token, is sent in the client authentication token when basic authentication is enabled. The sending server identity is sent through a Secure Sockets Layer (SSL) client certification authentication when client certificate authentication is enabled. Basic authentication takes precedence over client certificate authentication.

Both tokens are needed by the receiving server to accept the asserted identity. The receiving server completes the following actions to accept the asserted identity:

- The server determines whether the sending server identity, sent with a basic authentication token or with an SSL client certificate, is on the trusted principal list of the receiving server. The server determines whether the sending server can send an identity token to the receiving server.
- After it is determined that the sending server is on the trusted list, the server authenticates the sending server to verify its identity.
- The server is authenticated by comparing the user ID and password from the sending server to the receiving server, or it might require a real authenticated call. If the credentials of the sending server are authenticated and on the trusted principal list, then the server proceeds to evaluate the identity token.

Evaluation of the identity token consists of the following four identity formats that exist in an identity token:

- Principal name
- Distinguished name
- Certificate chain
- Anonymous identity

The product servers that receive authentication information typically support all four identity types. The sending server decides which one is chosen, based on how the original client authenticated. The existing type depends on how the client originally authenticates to the sending server. For example, if the client uses Secure Sockets Layer (SSL) client authentication to authenticate to the sending server, then the identity token sent to the downstream server contains the certificate chain. With this information, the receiving server can perform its own certificate chain mapping and interoperability is increased with other vendors and platforms.

After the identity format is understood and parsed, the identity maps to a credential. For an ITTPrincipal identity token, this identity maps one-to-one with the user ID fields.

For an ITTDistinguishedName identity token, the mapping depends on the user registry. For Lightweight Directory Access Protocol (LDAP), the configured search filter determines how the mapping occurs. For LocalOS, the first attribute of the distinguished name (DN), which is typically the same as the common name, maps to the user ID of the registry. For an ITTCertChain identity token, see the Map certificates to users section for details on how this action is performed for the LDAP user registry. For LocalOS, the first attribute of the DN in the certificate is used to map to the user ID in the registry.

Some user registry methods are called to gather additional credential information that is used by authorization. In a stateful server, this action completes once for the sending server and the receiving server pair where the identity tokens are the same. Subsequent requests are made through a session ID.

Identity assertion is only available using the Common Secure Interoperability Version 2 (CSIv2) protocol.

## Message layer authentication

Defines the credential information and sends that information across the network so that a receiving server can interpret it.

When you send authentication information across the network using a token (whether the token is a user ID and password token, that is, Generic Security Services Username Password (GSSUP), or a mechanism-specific format token, Lightweight Third Party Authentication (LTPA), for example on non-z/OS platforms), the transmission is considered message layer authentication because the data is sent with the message inside a service context.

A pure Java client uses basic authentication (GSSUP) as the authentication mechanism to establish client identity.

However, a servlet can use either basic authentication (GSSUP) or the authentication mechanism of the server (LTPA) to send security information in the message layer. Use LTPA by authenticating or by mapping the basic authentication credentials to the security mechanism of the server.

The security token that is contained in a token-based credential is authentication mechanism-specific. The way that the token is interpreted is only known by the authentication mechanism. Therefore, each authentication mechanism has an object ID (OID) representing it. The OID and the client token are sent to the server, so that the server knows which mechanism to use when reading and validating the token. The following list contains the OIDs for each mechanism:

```
BasicAuth (GSSUP):   oid:2.23.130.1.1.1
LTPA:      oid:1.3.18.0.2.30.2
SWAM:      No OID because it is not forwardable
```

On the server, the authentication mechanisms can interpret the token and create a credential, or they can authenticate basic authentication data from the client, and create a credential. Either way, the created

credential is the *received* credential that the authorization check uses to determine if the user has access to invoke the method. You can specify the authentication mechanism by using the following property on the client side:

- com.ibm.CORBA.authenticationTarget

Basic authentication is currently the only valid value. You can configure the server through the administrative console.

While this property tells you which authentication mechanism to use, you also need to specify whether you want to perform authentication over the message layer, that is get a BasicAuth or a token-based credential. To complete this task, specify the com.ibm.CSI.performClientAuthenticationRequired (`True` or `False`) and com.ibm.CSI.performClientAuthenticationSupported (`True` or `False`) properties. Indicating that client authentication is required implies that it must be done for every request. Indicating that the authentication mechanism is supported implies that it might be done, but is not required. For some servers, this option is appropriate if no resources are protected. In most cases, it is a best practice to indicate that this mechanism is supported so that client authentication is performed if both the client and server support it. Client authentication is not performed when communicating with certain servers that do not want security, yet the method requests still succeed.

**Configuring authentication retries**

Situations occur where you want a prompt to display again if you entered your user ID and password incorrectly or you want a method to retry when a particular error occurs back at the client. If you can correct the error by information at the client side, the system automatically performs a retry without the client seeing the failure, if the system is configured appropriately.

Some of these errors include:
- Entering a user ID and password that are not valid
- Having an expired credential on the server
- Failing to find the stateful session on the server

By default, authentication retries are enabled and perform three retries before returning the error to the client. Use the com.ibm.CORBA.authenticationRetryEnabled property (`True` or `False`) to enable or disable authentication retries. Use the com.ibm.CORBA.authenticationRetryCount property to specify the number of retry attempts.

**Immediate validating of a basic authentication login**

In WebSphere Application Server Version 6, a behavior is defined during request_login for a BasicAuth login. In releases prior to Version 5, a BasicAuth login takes the user ID and password entered through the loginSource method and creates a BasicAuth credential. If either the user ID or the password is not valid, the client program does not find out until the first method request is attempted. When the user ID or password is specified during a prompt or programmatic login, the user ID and password are authenticated by default with the security server, with a `True` or `False` returned as the result. If `False`, an `org.omg.SecurityLevel2.LoginFailed` exception is returned to the client indicating that the user ID and password are not valid. If `True`, then the BasicAuth credential is returned to the caller of the request_login. To disable this feature on the pure client, specify `com.ibm.CORBA.validateBasicAuth=false`. By default, this feature is set to `True`. On the server side, specify this property in the security dynamic properties.

## Secure Sockets Layer client certificate authentication

An additional way to authenticate a client to a server is using Secure Sockets Layer (SSL) client authentication.

Using SSL client authentication is another way of authenticating a client to a server. This form of authentication does not occur at the message layer using a user ID and password or tokens. This authentication occurs during the connection handshake using SSL certificates.

When the client is configured with a personal certificate in the SSL keystore file, which indicates that SSL client authentication is required and the server supports SSL client authentication, the following actions occur to establish the identity on the client side.

- When a method request is invoked in the client code to a remote enterprise bean, the Object Request Broker (ORB) invokes the client connection interceptor to establish a connection with the server.

  Because the configuration specifies SSL and SSL client authentication, the connection type is SSL and the SSL handshake sends the client certificate to the server to validate. If the client certificate does not validate, the connection is not established and an exception is sent back to the client code where the method is invoked, which indicates the failure. If the client certificate is validated, then a connection opens between the client and the server.
- The ORB proceeds to call the client request interceptor, which might be busy.

  If basic authentication is also configured, for example, then the user might be prompted for a user ID and password. Because this action is not necessary, disable this option in the configuration if the SSL certificate is the identity against which to invoke the method. If no message layer security exists, then no security context is created and associated with the request.
- After the server receives the request, the server-side request interceptor checks for a security context.

  Because the server does not find a service context, it checks the server socket for a client certificate chain that contains the client identity. In this case, the server finds the certificate chain from the client. The identity in the certificate chain is valid because the connection is made. To create a credential, map the identity from the certificate to the user registry. This action is done differently based on the type of authentication mechanism. Mapping a certificate to a credential is done differently based on the user registry type.

  See the “Map certificates to users” on page 451 article, for details on how this mapping is performed for the Lightweight Directory Access Protocol (LDAP) user registry. For local OS, the first attribute of the distinguished name (DN) in the certificate is used to map to the user ID in the registry.

One benefit of SSL client certificate authentication is that it optimizes authentication performance, because an SSL connection is typically created anyway. The extra overhead of sending the client certificate is minimal. While the client-side request interceptor performs no activity, the server-side request interceptor maps the certificate to a credential.

One disadvantage to this type of authentication is the complexity of setting up the keystore file on each client system.

To enable SSL client certificate authentication on the client side, you must enable the properties, such as SSL. This action is completed using the following two properties:
- com.ibm.CSI.performTransportAssocSSLTLSRequired (true or false)
- com.ibm.CSI.performTransportAssocSSLTLSSupported (true or false)

Indicating that SSL is required implies that every request must generate an SSL connection key. If a server does not support SSL, then the request fails. After you enable SSL by either supporting it or requiring it, you can enable some of the SSL features.

To enable SSL client authentication, you can specify the following two properties:
- com.ibm.CSI.performTLClientAuthenticationRequired (true or false)
- com.ibm.CSI.performTLClientAuthenticationSupported (true or false)

The `TL` means *transport layer*. If you indicate that SSL client authentication is required, then you only limit the ability to communicate with servers that support SSL client authentication. For a server to support SSL client authentication, that server must have similarly configured properties through the administrative

console, and have an SSL listener port that is open to handle mutual authentication handshakes. Configuration of server properties are done through the administrative console.

SSL client certificate authentication from a Java client is only available using the Common Secure Interoperability Version 2 (CSIv2) protocol.

## Supported authentication protocols

Two authentication protocols are supported. Secure Authentication Service (SAS) is the authentication protocol used by all previous releases of the WebSphere Application Server product. Common Secure Interoperability Version 2 (CSIv2), which is considered the strategic protocol, is implemented in WebSphere Application Server, Version 5 and later.

You can configure both protocols to work simultaneously. If a server supports both protocols, it exports an interoperable object reference (IOR) that contains tagged components describing the configuration for SAS and CSIv2. If a client supports both protocols, it reads tagged components for both CSIv2 and SAS. If the client and the server support both protocols, CSIv2 is used. However, if the server supports SAS (for example, it is a previous WebSphere Application Server release) and the client supports both protocols, the client chooses SAS for this request.

Choose a protocol using the com.ibm.CSI.protocol property on the client side and configure this protocol through the administrative console on the server side.

# Configuring Common Secure Interoperability Version 2 and Security Authentication Service authentication protocols

1. Determine how to configure security inbound and outbound at each point in your infrastructure.

   For example, you might have a Java client communicating with an Enterprise JavaBeans (EJB) application server, which in turn communicates to a downstream EJB application server.

   The Java client utilizes the `sas.client.props` file to configure outbound security. Pure clients need to configure outbound security only.

   The upstream EJB application server configures inbound security to handle the right type of authentication from the Java client. The upstream EJB application server utilizes the outbound security configuration when going to the downstream EJB application server.

   This type of authentication might be different than what you expect from the Java client into the upstream EJB application server. Security might be tighter between the pure client and the first EJB server, depending on your infrastructure. The downstream EJB server utilizes the inbound security configuration to accept requests from the upstream EJB server. These two servers require similar configuration options as well. If the downstream EJB application server communicates to other downstream servers, then the outbound security might require a special configuration.
2. Specify the type of authentication.

   By default, authentication by a user ID and password is performed.

   Both Java client certificate authentication and identity assertion are disabled by default. If you want this type of basic authentication performed at every tier, use the CSIv2 authentication protocol configuration as is. However, if you have any special requirements where some servers authenticate differently from other servers, then consider how to configure CSIv2 to its best advantage.
3. Configure clients and servers.

   Configuring a pure Java client is done through the `sas.client.props` file, where properties are modified.

   Configuring servers is always done from the administrative console, either from the security navigation for cell-level configurations or from the server security of the application server for server-level configurations. If you want some servers to authenticate differently from others, modify some of the server-level configurations. When you modify the server-level configurations, you are overriding the cell-level configurations.

## Common Secure Interoperability Version 2 and Security Authentication Service client configuration

A secure Java client requires configuration properties to determine how to perform security with a server. These configuration properties are typically put into a properties file somewhere on the client system and referenced by specifying the following system property on the command line of the Java client. The syntax of this property accepts a valid Web address with the protocol type, `file`.

```
-Dcom.ibm.CORBA.ConfigURL=file:/C:/WebSphere/AppServer/properties/sas.client.props
```

When this file is processed by the Object Request Broker (ORB), security can be enabled between the Java client and the target server.

If any syntax problems exist with the ConfigURL property and the `sas.client.props` file is not found, the Java client proceeds to connect insecurely. Errors display indicating the failure to read the ConfigURL property. Typically the problem is related to having two slashes after `file`, which is not valid.

Use the following properties to configure the SAS and CSIv2 authentication protocols:
- “CSIv2 authentication protocol client settings”
- “Security Authentication Service authentication protocol client settings” on page 390

### CSIv2 authentication protocol client settings

In addition to the properties that are valid for both Security Authentication Service (SAS) and Common Secure Interoperability Version 2 (CSIv2), this page documents the properties that are valid for the CSIv2 protocol only.

***com.ibm.CSI.performStateful:***

Used to determine if the CSIv2 protocol maintains stateful sessions between a client and server after the initial secure association (authentication between a particular client and server).

For performance reasons, it is beneficial to enable this property. Considerations for disabling this property include troubleshooting an authentication protocol session-related problem.

| | |
|---|---|
| **Data type:** | Boolean |
| **Default:** | True |
| **Range:** | True or False |

***com.ibm.CSI.performClientAuthenticationSupported:***

Use to determine if message layer client authentication is supported.

When supported, message layer client authentication is performed when communicating with any server that supports or requires the authentication. Message layer client authentication involves transmitting either a user ID and password or a token from an already authenticated credential. If the authenticationTarget property is BasicAuth, the user ID and password are transmitted to the target server. If the authenticationTarget password is a token-based mechanism such as Lightweight Third Party Authentication (LTPA) or Kerberos, then the credential token is transmitted to the server after authenticating the user ID and password directly to the security server.

| | |
|---|---|
| **Data type:** | Boolean |
| **Default:** | True |
| **Range:** | True or False |

***com.ibm.CSI.performClientAuthenticationRequired:***

Use to determine if message layer client authentication is required.

When required, message layer client authentication must occur when communicating with any server. If transport layer client authentication is also enabled, both authentications are performed, but message layer client authentication takes precedence at the server.

| | |
|---|---|
| **Data type:** | Boolean |
| **Default:** | True |
| **Range:** | True or False |

***com.ibm.CSI.performTransportAssocSSLTLSSupported:***

Use to determine if Secure Sockets Layer (SSL) is supported.

When SSL is supported, this client causes either SSL or TCP/IP to communicate with a server. If SSL is not supported, then the client must communicate over TCP/IP to the server. Supporting SSL is recommended so that any sensitive information is encrypted and digitally signed. When the associated com.ibm.CSI.performTransportAssocSSLTLSRequired property is enabled (set to `true`), this property is ignored. In this case, SSL is always required.

| | |
|---|---|
| **Data type:** | Boolean |
| **Default:** | True |
| **Range:** | True or False |

***com.ibm.CSI.performTransportAssocSSLTLSRequired:***

Use to determine if SSL is required.

When SSL is required, this client must use SSL to communicate to a server. If SSL is not supported by a server, this client does not attempt a connection to that server. When this property is enabled, the associated com.ibm.CSI.performTransportAssocSSLTLSSupported property is ignored.

| | |
|---|---|
| **Data type:** | Boolean |
| **Default:** | True |
| **Range:** | True or False |

***com.ibm.CSI.performTLClientAuthenticationSupported:***

Use to determine if transport layer client authentication is supported.

When performing client authentication using SSL, the client key file must have a personal certificate configured. Without a personal certificate, the client cannot authenticate to the server over SSL. If the personal certificate is a self-signed certificate, the server must contain the public key of the client in the server trust file. If the personal certificate is granted from a certificate authority (CA), the server must contain the root public key of the CA in the server trust file. This property is only valid when SSL is supported or required. If the associated com.ibm.CSI.performTLClientAuthenticationRequired property is enabled, this property is ignored.

| | |
|---|---|
| **Data type:** | Boolean |
| **Default:** | True |
| **Range:** | True or False |

***com.ibm.CSI.performTLClientAuthenticationRequired:***

Use to determine if transport layer client authentication is required.

If required, every secure socket that is opened between a client and server authenticates using SSL mutual authentication. When performing client authentication using SSL, the client key file must have a personal certificate configured. Without a personal certificate, the client cannot authenticate to the server over SSL.

If the personal certificate is a self-signed certificate, the server must contain the public key of the client in the server trust file. If the personal certificate is granted by a certificate authority (CA), the server must contain the root public key of the CA in the server trust file. When this property is specified, the associated com.ibm.CSI.performTLClientAuthenticationSupported property is ignored.

**Data type:** Boolean
**Default:** True
**Range:** True or False

***com.ibm.CSI.performMessageConfidentialitySupported:***

Use to determine if 128-bit ciphers are supported to make SSL connections.

If a target server does not support 128-bit ciphers, you can make a connection at a lower encryption strength. This property is only valid when SSL is enabled.

**Data type:** Boolean
**Default:** True
**Range:** True or False

***com.ibm.CSI.performMessageConfidentialityRequired:***

Use to determine if 128-bit ciphers must be used to make SSL connections.

If a target server does not support 128-bit ciphers, a connection to that server fails. This property is only valid when SSL is enabled. When this property is enabled, the associated com.ibm.CSI.performMessageConfidentialitySupported property is ignored.

**Data type:** Boolean
**Default:** True
**Range:** True or False

***com.ibm.CSI.performMessageIntegritySupported:***

Use to determine if 40-bit ciphers are supported to make SSL connections.

If a target server does not support 40-bit ciphers, you can make a connection using only digital-signing ciphers. This property is only valid when SSL is enabled. This property is ignored if the associated com.ibm.CSI.performMessageIntegrityRequired property is enabled.

**Data type:** Boolean
**Default:** True
**Range:** True or False

***com.ibm.CSI.performMessageIntegrityRequired:***

Use to determine if 40-bit ciphers must be used to make SSL connections.

If a target server does not support 40-bit ciphers, a connection to that server fails. This property is only valid when SSL is enabled. When this property is enabled, the associated com.ibm.CSI.performMessageIntegritySupported property is ignored.

| | |
|---|---|
| **Data type:** | Boolean |
| **Default:** | True |
| **Range:** | True or False |

***com.ibm.CSI.rmiOutboundPropagationEnabled:*** Enables the propagation of custom objects that are added to the Subject. On a pure client, add this property to the `sas.client.props` file. For more information, see Security Attribute Propagation.

### Security Authentication Service authentication protocol client settings

In addition to those properties which are valid for both Security Authentication Service (SAS) and Common Secure Interoperability Version 2 (CSIv2), this article documents properties which are valid only for the SAS authentication protocol.

***com.ibm.CORBA.standardPerformQOPModels:***

Specifies the strength of the ciphers when making an Secure Sockets Layer (SSL) connection.

| | |
|---|---|
| **Data type:** | String |
| **Default:** | High |
| **Range** | Low, Medium, High |

## Configuring Common Secure Interoperability Version 2 inbound authentication

*Inbound authentication* refers to the configuration that determines the type of accepted authentication for inbound requests. This authentication is advertised in the interoperable object reference (IOR) that the client retrieves from the name server.

1. Start the administrative console.
2. Click **Security > Global security**.
3. Under Authentication, click **Authentication Protocol > CSI inbound authentication**
4. Consider the following three layers of security:
   - Identity assertion (attribute layer).

     When selected, this server accepts identity tokens from upstream servers. If the server receives an identity token, the identity is taken from an originating client. For example, the identity is in the same form that the originating client presented to the first server. An upstream server sends the identity of the originating client. The format of the identity can be either a principal name, a distinguished name, or a certificate chain. In some cases, the identity is anonymous. It is important to trust the upstream server that sends the identity token because the identity is authenticating on this server. Trust of the upstream server is established either using Secure Sockets Layer (SSL) client certificate authentication or basic authentication. You must select one of the two layers of authentication in both inbound and outbound authentication when you choose identity assertion.

     The server ID is sent in the client authentication token with the identity token. The server ID is checked against the trusted server ID list. If the server ID is on the trusted server list, the server ID is authenticated. If the server ID is valid, the identity token is put into a credential and used for authorization of the request.
   - User ID and password (message layer).

This type of authentication is the most typical. The user ID and password or authenticated token is sent from a pure client or from an upstream server. However, the upstream server cannot be a z/OS server because z/OS does not support a user ID or password from a server acting as a client. When a user ID and password are received at the server, they are authenticated with the user registry.

Usually, a token is sent from an upstream server and a user ID and password are sent from a client, including a servlet. When a token is received at the server level, the token is validated to determine whether tampering has occurred or whether it is expired.

- Secure Sockets Layer client certificate authentication (transport layer).

  This type of authentication typically occurs from pure clients using the certificate identity and from servers trusting the upstream server. Usually, when a server delegates an identity to a downstream server, the identity comes from either the message layer (a client authentication token) or the attribute layer (an identity token), not from the transport layer, through the client certificate authentication.

  A client has an SSL client certificate that is stored in the keystore file (or in the key ring file on the z/OS platform) of the client configuration. When SSL client authentication is enabled on this server, the server requests that the client send the SSL client certificate when the connection is established. The certificate chain is available on the socket whenever a request is sent to the server. The server request interceptor gets the certificate chain from the socket and maps this certificate chain to a user in the registry. This type of authentication is optimal for communicating directly from a client to a server. However, when you have to go downstream, the identity typically flows over the message layer or through identity assertion.

5. Consider the following points when deciding what type of authentication to accept:
   - A server can receive multiple layers simultaneously, so an order of precedence rule decides which identity to use. The identity assertion layer has the highest priority, the message layer follows, and the transport layer has the lowest priority. The SSL client certificate authentication is used when it is the only layer provided. If the message layer and the transport layer are provided, the message layer is used to establish the identity for authorization. The identity assertion layer is used to establish precedence when provided.
   - Does this server usually receive requests from a client, from a server or both? If the server always receives requests from a client, identity assertion is not needed. You can then choose either the message layer, the transport layer, or both. You also can decide when authentication is required or just supported. To select a layer as required, the sending client must supply this layer, or the request is rejected. However, if the layer is only supported, the layer might not be supplied.
   - What kind of client identity is supplied? If the client identity is client certificates authentication and you want the certificate chain to flow downstream so that it maps to the downstream server user registries, then identity assertion is the appropriate choice. Identity assertion preserves the format of the originating client. If the originating client authenticated with a user ID and password, then a principal identity is sent. If authentication is done with a certificate, then the certificate chain is sent.

     In some cases, if the client authenticated with a token and a Lightweight Directory Access Protocol (LDAP) server is the user registry, then a distinguished name (DN) is sent.
6. Configure a trusted server list. When identity assertion is selected for inbound requests, insert a pipe-separated (|) list of server administrator IDs to which this server can support identity token submission. For backwards compatibility, you can still use a comma-delimited list. However, if the server ID is a distinguished name (DN), then you must use a pipe-delimited (|) list because a comma delimiter does not work. If you choose to support any server sending an identity token, you can enter an asterisk (*) in this field. This action is called *presumed trust*. In this case, use SSL client certificate authentication between servers to establish the trust.
7. Configure session management. You can choose either *stateful* or *stateless* security. Performance is optimum when choosing stateful sessions. The first method request between a client and server is authenticated. All subsequent requests (or until the credential token expires) reuse the session information, including the credential. A client sends a context ID for subsequent requests. The context ID is scoped to the connection for uniqueness.

When you finish configuring this panel, you have configured most of the information that a client coalesces when determining what to send to this server. A client or server outbound configuration with this server inbound configuration, determines the security that is applied. When you know what clients send, the configuration is simple. However, if you have a diverse set of clients with differing security requirements, your server considers various layers of authentication.

For an enterprise bean server, the authentication choice is usually either identity assertion or message layer because you want the identity of the originating client delegated downstream. You cannot easily delegate a client certificate using an SSL connection. It is acceptable to enable the transport layer because additional server security, as the additional client certificate portion of the SSL handshake, adds some overhead to the overall SSL connection establishment.

After you determine which type of authentication data this server might receive, you can determine what to select for outbound security. Refer to the article, Configuring Common Secure Interoperability Version 2 outbound authentication.

## Common Secure Interoperability inbound authentication settings

Use this page to specify the features that a server supports for a client accessing its resources.

To view this administrative console page, click **Security > Global security**. Under Authentication, click **Authentication protocols > CSIv2 inbound authentication**.

You can also view this administrative console page, by clicking **Servers > Application servers >***server_name*. Under Security, click **Server security**. Under Additional properties, click **CSIv2 inbound authentication**.

Use common secure interoperability (CSI) inbound authentication settings for configuring the type of authentication information that is contained in an incoming request or transport.

Authentication features include three layers of authentication that you can use simultaneously:
- **Transport layer**. The transport layer, which is the lowest layer, might contain a Secure Sockets Layer (SSL) client certificate as the identity.
- **Message layer**. The message layer might contain a user ID and password or an authenticated token with an expiration.
- **Attribute layer**. The attribute layer might contain an identity token, which is an identity from an upstream server that already is authenticated. The identity layer has the highest priority, followed by the message layer, and then the transport layer. If a client sends all three, only the identity layer is used. The only way to use the SSL client certificate as the identity is if it is the only information that is presented during the request. The client picks up the interoperable object reference (IOR) from the namespace and reads the values from the tagged component to determine what the server needs for security.

***Basic authentication:***

Specifies that basic authentication occurs over the message layer.

In the message layer, basic authentication (user ID and password) takes place. This type of authentication typically involves sending a user ID and a password from the client to the server for authentication.

This authentication also involves delegating a credential token from an already authenticated credential, provided the credential type is forwardable (for example, Lightweight Third Party Authentication (LTPA)).

If you select **Basic Authentication** and LTPA is the configured authentication protocol, user name, password, and LTPA tokens are accepted.

The following options are available for **Basic Authentication**:

**Never** This option indicates that this server cannot accept user ID and password authentication.

**Supported**
This option indicates that a client communicating with this server can specify a user ID and password. However, a method might be invoked without this type of authentication. For example, an anonymous or client certificate might be used instead.

**Required**
This option indicates that clients communicating with this server must specify a user ID and password for any method request.

Basic authentication takes precedence over client certificate authentication, if both are performed.

***Client certificate authentication:***

Specifies that authentication occurs when the initial connection is made between the client and the server during a method request.

In the transport layer, Secure Sockets Layer (SSL) client certificate authentication occurs. In the message layer, basic authentication (user ID and password) is performed. Client certificate authentication typically performs better than message layer authentication, but requires some additional setup. These additional steps involve verifying that the server has the signer certificate of each client to which it is connected. If the client uses a certificate authority (CA) to create its personal certificate, you only need the CA root certificate in the server signer section of the SSL trust file.

When the certificate is authenticated to a Lightweight Directory Access Protocol (LDAP) user registry, the distinguished name (DN) is mapped based on the filter that is specified when configuring LDAP. When the certificate is authenticated to a LocalOS user registry, the first attribute of the distinguished name (DN) in the certificate, which is typically the common name, is mapped to the user ID in the registry.

The identity from client certificates is used only if no other layer of authentication is presented to the server.

The following options are available for Client certificate authentication:

**Never** This option indicates that clients cannot attempt Secure Sockets Layer (SSL) client certificate authentication with this server.

**Supported**
This option indicates that clients connecting to this server can authenticate using SSL client certificates. However, the server can invoke a method without this type of authentication. For example, anonymous or basic authentication can be used instead.

**Required**
This option indicates that clients connecting to this server must authenticate using SSL client certificates before invoking the method.

***Identity assertion:***

Specifies that identity assertion is a way to assert identities from one server to another during a downstream Enterprise JavaBeans (EJB) invocation.

This server does not authenticate the asserted identity again because it trusts the upstream server. Identity assertion takes precedence over all other types of authentication.

Identity assertion is performed in the attribute layer and is only applicable on servers. The principal determined at the server is based on precedence rules. If identity assertion is performed, the identity is always derived from the attribute. If basic authentication is performed without identity assertion, the identity

is always derived from the message layer. Finally, if SSL client certificate authentication is performed without either basic authentication, or identity assertion, then the identity is derived from the transport layer.

The identity asserted is the invocation credential that is determined by the RunAs mode for the enterprise bean. If the RunAs mode is Client, the identity is the client identity. If the RunAs mode is System, the identity is the server identity. If the RunAs mode is Specified, the identity is the one specified. The receiving server receives the identity in an identity token and also receives the sending server identity in a client authentication token. The receiving server validates the sending server identity as a trusted identity through the Trusted Server IDs entry box. Enter a list of pipe-separated (|) principal names, for example, `serverid1|serverid2|serverid3`.

When authenticating to a LocalOS user registry, all identity token types map to the user ID field of the active user registry. For an ITTPrincipal identity token, this token maps one-to-one with the user ID fields. For an ITTDistinguishedName identity token, the value from the first equal sign is mapped to the user ID field. For an ITTCertChain identity token, the value from the first equal sign of the distinguished name is mapped to the user ID field.

When authenticating to an LDAP user registry, the LDAP filters determine how an identity of type ITTCertChain and ITTDistinguishedName get mapped to the registry. If the token type is ITTPrincipal, then the principal gets mapped to the UID field in the LDAP registry.

| | |
|---|---|
| **Data type:** | String |

***Trusted servers:***

Specifies a pipe-separated (|) list of trusted server IDs, which are trusted to perform identity assertion to this server. For example, `serverid1|serverid2|serverid3`. WebSphere Application Server supports the comma (,) character as the list delimiter for backwards compatibility. WebSphere Application Server checks the comma character when the pipe character (|) fails to find a valid trusted server ID.

Use this list to decide whether a server is trusted. Even if the server is on the list, the sending server must still authenticate with the receiving server to accept the identity token of the sending server.

| | |
|---|---|
| **Data type** | String |

***Stateful sessions:***

Specifies stateful sessions that are used mostly for performance improvements.

The first contact between a client and server must fully authenticate. However, all subsequent contacts with valid sessions reuse the security information. The client passes a context ID to the server, and the ID is used to look up the session. The context ID is scoped to the connection, which guarantees uniqueness. Whenever the security session is not valid and the authentication retry is enabled, which is the default, the client-side security interceptor invalidates the client-side session and submits the request again without user awareness. This situation might occur if the session does not exist on the server (the server failed and resumed operation). When this value is disabled, every method invocation must authenticate again.

| | |
|---|---|
| **Data type** | String |

***Login configuration:***

Specifies the type of system login configuration to use for inbound authentication.

You can add custom login modules by clicking **Security > Global security**. Under Authentication, click **JAAS configuration > System logins**.

***Security attribute propagation:***

Specifies whether to support security attribute propagation during login requests. When you select this option, WebSphere Application Server retains additional information about the login request, such as the authentication strength used, and retains the identity and location of the request originator.

Verify that you are using Lightweight Third Party Authentication (LTPA) as your authentication mechanism. LTPA is the only authentication mechanism supported when you enable the security attribute propagation feature. To configure LTPA, click **Security > Global security**. Under Authentication, click **Authentication mechanisms > LTPA**.

If you do not select this option, WebSphere Application Server does not accept any additional login information to propagate to downstream servers.

## Configuring Common Secure Interoperability Version 2 outbound authentication

*Outbound authentication* refers to the configuration that determines the type of authentication performed for outbound requests to downstream servers. Several *layers* or *methods* of authentication can occur. The downstream server inbound authentication configuration must support at least one choice made in this server outbound authentication configuration. If nothing is supported, the request might go outbound as `unauthenticated`. This situation does not create a security problem because the authorization run time is responsible for preventing access to protected resources. However, if you choose to prevent an unauthenticated credential to go outbound, you might want to designate one of the authentication layers as `required`, rather than `supported`. If a downstream server does not support authentication, then when authentication is required, the method request fails to go outbound.

The following choices are available in the Common Secure Interoperability Version 2 (CSIv2) Outbound Authentication panel. Remember that you are not required to complete these steps in the displayed order. Rather, these steps are provided to help you understand your choices for configuring outbound authentication.

1. Select **Identity Assertion** (attribute layer). When selected, this server submits an identity token to a downstream server, if the downstream server supports identity assertion. When an originating client authenticates to this server, the authentication information supplied is preserved in the outbound identity token. If the client authenticating to this server uses client certificate authentication, then the identity token format is a certificate chain, containing the exact client certificate chain on the socket. The same scenario is true for other mechanisms of authentication. Read theIdentity Assertion article for more information.
2. Select **User ID** and **Password** (message layer). This type of authentication is the most typical. The user ID and password (if BasicAuth credential) or authenticated token (if authenticated credential) are sent outbound to the downstream server if the downstream server supports message layer authentication in the inbound authentication panel. Refer to the Message Layer Authentication article for more information.
3. Select **SSL Client certificate authentication** (transport layer). The main reason to enable outbound Secure Sockets Layer (SSL) client authentication from one server to a downstream server is to create a trusted environment between those servers. For delegating client credentials, use one of the two layers mentioned previously. However, you might want to create SSL personal certificates for all the servers in your domain, and only trust those servers in your SSL truststore file. No other servers or clients can connect to the servers in your domain, except at the tiers where you want them. This process can protect your enterprise bean servers from access by anything other than your servlet servers. Refer to the SSL Client Certificate Authentication article for more information.

A server can send multiple layers simultaneously, therefore, an order of precedence rule decides which identity to use. The identity assertion layer has the highest priority, the message layer follows, and the transport layer has the lowest priority. SSL client certificates are only used as the identity for invoking method requests, when that is the only layer provided. SSL client certificates are useful for trust purposes, even if the identity is not used for the request. If only the message layer and transport layer are provided, the message layer is used to establish the identity for authorization. If the identity assertion layer is provided (regardless of what is provided), then the identity from the identity token is always used by the authorization engine as the identity for that request.

## Configuring session management

You can choose either *stateful* or *stateless* security. Performance is optimum when choosing stateful sessions. The first method request between this server and the downstream server is authenticated. All subsequent requests reuse the session information, including the credential. A *unique session entry* is defined as the combination of a unique client authentication token and an identity token, scoped to the connection.

When you finish configuring this panel, you configured the information that this server uses to make decisions about the type of authentication to perform with downstream servers. If the downstream server is configured not to support the outbound configuration of the server, the following exception likely occurs:

```
Exception received: org.omg.CORBA.INITIALIZE:
CWWSA1477W: SECURITY CLIENT/SERVER CONFIG MISMATCH:  The client security
configuration (sas.client.props or outbound settings in GUI) does not
support the server security configuration for the following reasons:
ERROR 1: CWWSA0607E: The client requires SSL Confidentiality but the server
does not support it.
ERROR 2: CWWSA0610E: The server requires SSL Integrity but the client does
not support it.
ERROR 3: CWWSA0612E: The client requires client (e.g., userid/password or token),
but the server does not support it.
minor code: 0  completed: No
        at com.ibm.ISecurityLocalObjectBaseL13Impl.SecurityConnectionInterceptor.
getConnectionKey(SecurityConnectionInterceptor.java:1770)
        at com.ibm.ws.orbimpl.transport.WSTransport.getConnection(Unknown Source)
        at com.ibm.rmi.iiop.TransportManager.get(TransportManager.java:79)
        at com.ibm.rmi.iiop.GIOPImpl.locate(GIOPImpl.java:167)
        at com.ibm.CORBA.iiop.ClientDelegate._createRequest(ClientDelegate.java:2088)
        at com.ibm.CORBA.iiop.ClientDelegate.createRequest(ClientDelegate.java:1264)
        at com.ibm.CORBA.iiop.ClientDelegate.createRequest(ClientDelegate.java:1177)
        at com.ibm.CORBA.iiop.ClientDelegate.request(ClientDelegate.java:1726)
        at org.omg.CORBA.portable.ObjectImpl._request(ObjectImpl.java:245)
        at com.ibm.WsnOptimizedNaming._NamingContextStub.get_compatibility_level
(Unknown Source)
        at com.ibm.websphere.naming.DumpNameSpace.getIdlLevel(DumpNameSpace.java:300)
        at com.ibm.websphere.naming.DumpNameSpace.getStartingContext
(DumpNameSpace.java:329)
        at com.ibm.websphere.naming.DumpNameSpace.main(DumpNameSpace.java:268)
        at java.lang.reflect.Method.invoke(Native Method)
        at com.ibm.ws.bootstrap.WSLauncher.main(WSLauncher.java:163)
```

The reasons for the mismatch are explained in the exception. You can make the corrections when you configure the outbound configuration for this server, or when you configure the inbound configuration of the downstream server. If multiple reasons exist for a failure, the reasons are explained as message text in the exception.

Typically, the outbound authentication configuration is for an upstream server to communicate with a downstream server. Most likely, the upstream server is a servlet server and the downstream server is an Enterprise JavaBeans (EJB) server. On a servlet server, the client authentication that is performed to access the servlet can be one of many different types of authentication, including client certificate and basic authentication. When receiving basic authentication data, whether through a prompt login or a form-based login, the basic authentication information is typically authenticated to from a credential of the mechanism type that is supported by the server, such as Lightweight Third Party Authentication (LTPA) or LocalOS. When LTPA is the mechanism, a forwardable token exists in the credential. Choose the message layer (BasicAuth) authentication to propagate the client credentials. If the credential is created using a certificate login and you want to preserve sending the certificate downstream, you might decide to go outbound with identity assertion.

Save the configuration and restart the server for the changes to take effect.

## Common Secure Interoperability Version 2 outbound authentication settings

Use this page to specify the features that a server supports when acting as a client to another downstream server.

To view this administrative console page, click **Security** > **Global Security**. Under Authentication, click **Authentication protocols > CSIv2 outbound authentication**.

You can also view this administrative console page by clicking **Servers** > **Application Servers** > *server_name*. Under Security, click **Server Security**. Under Additional properties, click **CSIv2 Outbound Authentication**.

Authentication features include the following layers of authentication that you can use simultaneously:

**Transport layer**
: The transport layer, the lowest layer, might contain a Secure Sockets Layer (SSL) client certificate as the identity.

**Message layer**
: The message layer might contain a user ID and password or authenticated token.

**Attribute layer**
: The attribute layer might contain an identity token, which is an identity from an upstream server that is already authenticated. The attribute layer has the highest priority, followed by the message layer and then the transport layer. If this server sends all three, only the attribute layer is used by the downstream server. The only way to use the SSL client certificate as the identity is if it is the only information presented during the outbound request.

***Basic authentication:***

Specifies whether to send a user ID and a password from the client to the server for authentication.

This type of authentication occurs over the message layer. Basic authentication also involves delegating a credential token from an already authenticated credential, provided the credential type is forwardable (for example, Lightweight Third Party Authentication (LTPA)). Basic authentication refers to any authentication over the message layer and indicates user ID and password as well as token-based authentication.

If you select **Basic Authentication**, the following options are available:

**Never**
: This option indicates that this server does not send user ID and password authentication information to downstream servers. By selecting never, requests to downstream servers that require basic authentication fail.

**Supported**
: This option indicates that this server can specify a user ID and password to authenticate with downstream servers. However, a method might be invoked without this type of authentication. For example, the server can use anonymous or client certificate instead.

**Required**
This option indicates that this server must specify a user ID and password to authenticate with downstream servers for any method request. This server cannot initiate requests with servers that do not support or require basic authentication for inbound requests.

***Client certificate authentication:***

Specifies whether a client certificate from the configured keystore file is used to authenticate to the server when the SSL connection is made between this server and a downstream server (provided that the downstream server supports client certificate authentication).

Typically, client certificate authentication has a higher performance than message layer authentication, but requires some additional setup. These additional steps include verifying that this server has a personal certificate and that the downstream server has the signer certificate of this server.

If you select client certificate authentication, the following options are available:

**Never**
This option indicates that this server does not attempt Secure Sockets Layer (SSL) client certificate authentication with downstream servers.

**Supported**
This option indicates that this server can use SSL client certificates to authenticate to downstream servers. However, a method can be invoked without this type of authentication. For example, the server can use anonymous or basic authentication instead.

**Required**
This option indicates that this server must use SSL client certificates to authenticate to downstream servers.

***Identity assertion:***

Specifies whether to assert identities from one server to another during a downstream enterprise bean invocation.

The identity asserted is the invocation credential that is determined by the RunAs mode for the enterprise bean. If the RunAs mode is Client, the identity is the client identity. If the RunAs mode is System, the identity is the server identity. If the RunAs mode is Specified, the identity is the identity specified. The receiving server receives the identity in an identity token and also receives the sending server identity in a client authentication token. The receiving server validates the identity of the sending server to ensure a trusted identity.

When specifying identity assertion on the CSIv2 Authentication Outbound panel, you must also select basic authentication as supported or required on the CSIv2 Authentication Outbound panel. The server identity can then be submitted with the identity token, so that the receiving server can *trust* the sending server. Without specifying basic authentication as supported or required, trust is not established and the identity assertion fails.

***Stateful sessions:***

Specifies whether to reuse security information during authentication. This option is usually used to increase performance.

The first contact between a client and server must fully authenticate. However, all subsequent contacts with valid sessions reuse the security information. The client passes a context ID to the server, and that ID is used to look up the session. The context ID is scoped to the connection, which guarantees uniqueness. When the security session is not valid and if authentication retry is enabled, which is the default, the client-side security interceptor invalidates the client-side session and resubmits the request transparently. For example, if the session does not exist on the server; the server fails and resumes operation.

When this value is disabled, every method invocation must authenticate again.

***Login configuration:***

Specifies the type of system login configuration that is used for outbound authentication.

You can add custom login modules before or after this login module by clicking **Security > Global security**. Under Authentication, click **JAAS configuration > System login**.

***Custom outbound mapping:***

Enables the use of custom Remote Method Invocation (RMI) outbound login modules.

The custom login module maps or performs other functions before the predefined RMI outbound call. To declare a custom outbound mapping, click **Security > Global security**. Under Authentication, click **JAAS configuration > System logins > New**.

***Security attribute propagation:***

Enables WebSphere Application Server to propagate the Subject and the security content token to other application servers using the Remote Method Invocation (RMI) protocol.

Verify that you are using Lightweight Third Party Authentication (LTPA) as your authentication mechanism. LTPA is the only authentication mechanism that is supported when you enable the security attribute propagation feature. To configure LTPA, click **Security > Global security**. Under Authentication, click**Authentication mechanisms > LTPA**.

By default, the Security attribute propagation option is enabled and outbound login configuration is invoked. If you clear this option, WebSphere Application Server does not propagate any additional login information to downstream servers.

***Trusted target realms:***

Specifies a list of trusted target realms, separated by a pipe character (|), that differ from the current realm.

Prior to WebSphere Application Server, Version 5.1.1, if the current realm does not match the target realm, the authentication request is not sent outbound to other application servers.

***Additional Common Secure Interoperability outbound authentication settings:***

Use this page to configure additional authentication settings for requests that are received by this server using the Object Management Group (OMG) Common Secure Interoperability authentication protocol.

To view this administrative console page, click **Global security** > **CSIv2 Outbound Authentication** > **Additional Settings**. You can also view this administrative console page by clicking **Servers** > **Application Servers** > *server_name* > **Server Security** > **CSIv2 Outbound Authentication** > **Additional Settings**.

*Client authentication type:*

Specifies the type of client authentication that is supported for outbound requests.

| | |
|---|---|
| **Data type** | String |
| **Default** | SAFUSERIDPASSWORD |

## Configuring inbound transports

*Inbound transports* refer to the types of listener ports and their attributes that are opened to receive requests for this server. Both Common Secure Interoperability Specification, Version 2 (CSIv2) and Secure Authentication Service (SAS) have the ability to configure the transport.

However, the following differences between the two protocols exist:
- CSIv2 is much more flexible than SAS, which requires Secure Sockets Layer (SSL); CSIv2 does not require SSL.
- SAS does not support SSL client certificate authentication, while CSIv2 does.
- CSIv2 can require SSL connections, while SAS only supports SSL connections.
- SAS always has two listener ports open: TCP/IP and SSL.
- CSIv2 can have as few as one listener port and as many as three listener ports. You can open one port for just TCP/IP or when SSL is required. You can open two ports when SSL is supported, and open three ports when SSL and SSL client certificate authentication is supported.

Complete the following steps to configure the Inbound transport panels in the administrative console:
1. Click **Security > Global security**.
2. Under Authentication, click **Authentication Protocol > CSIv2 Inbound Transport** to select the type of transport and the SSL settings. By selecting the type of transport, as noted previously, you choose which listener ports you want to open. In addition, you disable the SSL client certificate authentication feature if you choose TCP/IP as the transport.
3. Select the SSL settings that correspond to an SSL transport. These SSL settings are defined in the **Security** > **SSL** panel and define the SSL configuration including the key ring, security level, ciphers, and so on.
4. Consider fixing the listener ports that you configured.

   You complete this action in a different panel, but think about this action now. Most end points are managed at a single location, which is why they do not display in the Inbound transport panels. Managing end points at a single location helps you decrease the number of conflicts in your configuration when you assign the endpoints. The location for SSL end points is at each server. The following port names are defined in the End points panel and are used for Object Request Broker (ORB) security:
   - CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS - CSIv2 Client Authentication SSL Port
   - CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS - CSIv2 SSL Port
   - SAS_SSL_SERVERAUTH_LISTENER_ADDRESS - SAS SSL Port
   - ORB_LISTENER_PORT - TCP/IP Port

   For an application server, click **Servers > Application servers >***server_name* . Under Communications, click **Ports**. The Ports panel is displayed for the specified server.

   The Object Request Broker (ORB) on WebSphere Application Server uses a listener port for Remote Method Invocation over the Internet Inter-ORB Protocol (RMI/IIOP) communications, which is generally not specified and selected dynamically during run time. If you are working with a firewall, you must specify a static port for the ORB listener and open that port on the firewall so that communication can pass through the specified port. The endPoint property for setting the ORB listener port is: ORB_LISTENER_ADDRESS.

   Complete the following steps using the administrative console to specify the ORB_LISTENER_ADDRESS port or ports.
   a. Click **Servers > Application Servers >** *server_name*. Under Communications, click **Ports > New**.
   b. Select **ORB_LISTENER_ADDRESS** from the Port name field in the Configuration panel.
   c. Enter the IP address, the fully qualified Domain Name System (DNS) host name, or the DNS host name by itself in the Host field. For example, if the host name is `myhost`, the fully qualified DNS name can be `myhost.myco.com` and the IP address can be `155.123.88.201`.

d. Enter the port number in the Port field. The port number specifies the port for which the service is configured to accept client requests. The port value is used with the host name. Using the previous example, the port number might be 9000.

5. Click **Security > Global security**. Under Authentication, click **Authentication Protocol > SAS inbound transport** to select the SSL settings used for inbound requests from SAS clients. Remember that the SAS protocol is used to interoperate with previous releases. When configuring the keystore and truststore files in the SSL configuration, these files need the right information for interoperating with previous releases of WebSphere Application Server. For example, a previous release has a different truststore file than the Version 5 release. If you use the Version 5 keystore file, add the signer to the truststore file of the previous release for those clients connecting to this server.

The inbound transport configuration is complete. With this configuration, you can configure a different transport for inbound security versus outbound security. For example, if the application server is the first server that is used by users, the security configuration might be more secure. When requests go to back-end enterprise bean servers, you might lessen the security for performance reasons when you go outbound. With this flexibility you can design the right transport infrastructure to meet your needs.

When you finish configuring security, perform the following steps to save, synchronize, and restart the servers:
1. Click **Save** in the administrative console to save any modifications to the configuration.
2. Stop and restart all servers, when synchronized.

## Common Secure Interoperability Version 2 transport inbound settings

Use this page to specify which listener ports to open and which Secure Sockets Layer (SSL) settings to use. These specifications determine which transport a client or upstream server uses to communicate with this server for incoming requests.

To view this administrative console page, click **Security > Global security**. Under Authentication, click **CSIv2 inbound transport**.

***Transport:***

Specifies whether client processes connect to the server using one of its connected transports.

You can choose to use either Secure Sockets Layer (SSL), TCP/IP or both as the inbound transport that a server supports. If you specify TCP/IP, the server only supports TCP/IP and cannot accept SSL connections. If you specify SSL-supported, this server can support either TCP/IP or SSL connections. If you specify SSL-required, then any server communicating with this one must use SSL.

If you specify SSL-supported or SSL-required, decide which set of SSL configuration settings you want to use for the inbound configuration. This decision determines which key file and trust file are used for inbound connections to this server.

**TCP/IP**

If you select **TCP/IP**, then the server opens a TCP/IP listener port only and all inbound requests do not have SSL protection.

**SSL-required**

If you select **SSL-required**, then the server opens an SSL listener port only and all inbound requests are received using SSL.

**Important:** If you set the active authentication protocol to **CSI and SAS**, then the server opens a TCP/IP listener port for the Secure Authentication Service (SAS) protocol regardless of this setting.

Only an SSL listener port is opened, and all requests come through SSL connections. If you choose **SSL-required**, you must also choose **CSI** as the active authentication protocol. If you

choose **CSI and SAS**, SAS requires an open TCP/IP socket for some special requests. You can select either **CSI** or **CSI and SAS** from the Global security panel, which is accessible from **Security > Global Security**.

**SSL-supported**
If you select **SSL-supported**, then the server opens both a TCP/IP and an SSL listener port and most inbound requests are received using SSL.

By default, SSL ports for Common Secure Interoperability Version 2 (CSIv2) and Security Authentication Service (SAS) are dynamically generated. In cases where you need to fix the SSL ports on application servers, click **Servers** > **Application Servers** > *server_name* > **End Points**.

Provide a fixed port number for the following ports. A zero port number indicates that a dynamic assignment is made at run time.

CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS
CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS
SAS_SSL_SERVERAUTH_LISTENER_ADDRESS

| | |
|---|---|
| **Default:** | SSL-Supported |
| **Range:** | TCP/IP, SSL Required, SSL-Supported |

### Secure Authentication Service inbound transport settings

Use this page to specify transport settings for connections that are accepted by this server using the Secure Authentication Service (SAS) authentication protocol. The SAS protocol is used to communicate securely to enterprise beans with previous releases of the WebSphere Application Server.

To view this administrative console page, click **Security > Global security**. Under Authentication, click **Authentication protocol > SAS inbound transport**.

***SSL Settings:***

Specifies a list of predefined SSL settings to choose from for inbound connections. These settings are configured at the SSL Repertoire panel.

| | |
|---|---|
| **Data type:** | String |
| **Default:** | DefaultSSLSettings |

## Configuring outbound transports

*Outbound transports* refers to the transport used to connect to a downstream server. When you configure the outbound transport, consider the transports that the downstream servers support. If you are considering Secure Sockets Layer (SSL), also consider including the signers of the downstream servers in this server truststore file for the handshake to succeed.

When you select an SSL configuration, that configuration points to keystore and truststore files that contain the necessary signers.

If you configured client certificate authentication for this server by completing the following steps, then the downstream servers contain the signer certificate belonging to the server personal certificate:

1. Click **Security > Global security**.
2. Under Authentication, click **Authentication protocols > CSIv2 outbound authentication**.

Complete the following steps to configure the Outbound Transport panels.

1. Select the type of transport and the SSL settings by clicking **Security > Global security**. Under Authentication, click **Authentication Protocol > CSIv2 Outbound Transport**. By selecting the type of transport, you are choosing the transport to use when connecting to downstream servers. The downstream servers support the transport that you choose. If you choose **SSL-Supported**, the transport used is negotiated during the connection. If both the client and server support SSL, always select the **SSL-Supported** option unless the request is considered a special request that does not require SSL, such as if an object request broker (ORB) is a request.
2. Pick the SSL settings that correspond to an SSL transport. Click **Security > SSL** .

   This panel includes the SSL configuration of keystore files, truststore files, file formats, security levels, ciphers, cryptographic token selections, and so on. Verify that the truststore keyring file in the selected SSL configuration contains the signers for any downstream servers. Also, verify that the downstream servers contain the server signer certificates when outbound client certificate authentication is used.
3. Select the SSL settings used for outbound requests to downstream Secure Authentication Service (SAS) servers. Click **Security > Global security**. Under Authentication, click **Authentication Protocol > SAS Outbound transport**. Remember that the SAS protocol allows interoperability with previous releases. When configuring the keystore and truststore files in the SSL configuration, these files have the correct information for interoperating with previous releases of WebSphere Application Server. For example, a previous release has a different personal certificate than the Version 5 release. If you use the keystore file from the Version 5.0 release, you must add the signer to the truststore file of the previous release. Also, you must extract the signer for the Version 5.0 release and import that signer into the truststore file of the previous release.

The outbound transport configuration is complete. With this configuration you can configure a different transport for inbound security versus outbound security. For example, if the application server is the first server used by end users, the security configuration might be more secure. When requests go to back-end enterprise beans servers, you might consider less security for performance reasons when you go outbound. With this flexibility you can design a transport infrastructure that meets your needs.

When you finish configuring security, perform the following steps to save, synchronize, and restart the servers.
- Click **Save** in the administrative console to save any modifications to the configuration.
- Stop and restart all servers, after synchronization.

## Common Secure Interoperability Version 2 outbound transport settings

Use this page to specify which transports and Secure Sockets Layer (SSL) settings this server uses when communicating with downstream servers for outbound requests.

To view this administrative console page, click **Security > Global security**. Under Authentication, click **Authentication > Authentication protocol > CSIv2 outbound transport**.

***Transport:***

Specifies whether the client processes connect to the server using one of the server-connected transports.

You can choose to use either SSL, TCP/IP, or Both as the outbound transport that a server supports. If you specify `TCP/IP`, the server supports only TCP/IP and cannot initiate SSL connections with downstream servers. If you specify `SSL-supported`, this server can initiate either TCP/IP or SSL connections. If you specify `SSL-required`, this server must use SSL to initiate connections to downstream servers. When you do specify SSL, decide which set of SSL configuration settings you want to use for the outbound configuration.

This decision determines which keyfile and trustfile to use for outbound connections to downstream servers.

Consider the following options:

**TCP/IP**
: If you select this option, the server opens TCP/IP connections with downstream servers only.

**SSL-required**
: If you select this option, the server opens SSL connections with downstream servers.

**SSL-supported**
: If you select this option, the server opens SSL connections with any downstream server that supports them and opens TCP/IP connections with any downstream servers that do not support SSL.

| | |
|---|---|
| **Default:** | SSL-supported |
| **Range:** | TCP/IP, SSL-required, SSL-supported |

***SSL settings:***

Specifies a list of predefined SSL settings for outbound connections. These settings are configured at the SSL Configuration Repertoires panel. To access the panel, click **Security > SSL**.

| | |
|---|---|
| **Data type:** | String |
| **Default:** | DefaultSSLSettings |
| **Range:** | Any SSL settings that are configured in the SSL Configuration Repertoires panel |

### Secure Authentication Service outbound transport settings

Use this page to specify transport settings for connections that are accepted by this server using the Secure Authentication Service (SAS) authentication protocol. Use the SAS protocol to communicate securely to enterprise beans with previous releases of WebSphere Application Server.

To view this administrative console page, click **Security > Global security**. Under Authentication, click **Authentication protocol > SAS outbound transport**.

***SSL settings:***

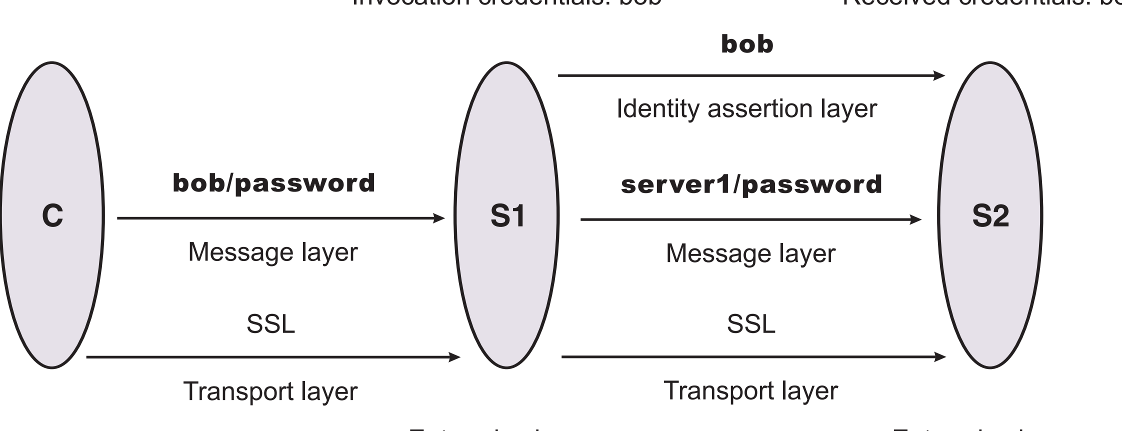
Specifies a list of predefined Secure Sockets Layer (SSL) settings to choose from for outbound connections. These settings are configured at the SSL Repertoire panel.

| | |
|---|---|
| **Data type:** | String |
| **Default:** | DefaultSSLSettings |

## Example: Common Secure Interoperability Version 2 scenarios

The articles included in this section are intended to demonstrate how to configure specific Common Secure Interoperability Version 2 (CSIv2) configuration examples.

### Scenario 1: Basic authentication and identity assertion




This example presents a pure Java client, C, that accesses a secure enterprise bean on server, S1, through user "bob." The enterprise bean code on S1 accesses another enterprise bean on server, S2. This configuration uses identity assertion to propagate the identity of "bob" to the downstream server, S2. S2 trusts that "bob" already is authenticated by S1 because it trusts S1. To gain this trust, the identity of S1 also flows to S2 simultaneously and S2 validates the identity by checking the trustedPrincipalList list to verify that it is a valid server principal. S2 also authenticates S1. The following steps take you through the configuration of C, S1, and S2.

**Configuring client, C**

Client C requires message layer authentication with a Secure Sockets Layer (SSL) transport. To accomplish this task:

1. Point the client to the `sas.client.props` file using the `com.ibm.CORBA.ConfigURL=file:/C:/was/properties/sas.client.props` property.

   All further configuration involves setting properties within this file.
2. Enable SSL.

   In this case, SSL is supported but not required:
   `com.ibm.CSI.performTransportAssocSSLTLSSupported=true,`
   `com.ibm.CSI.performTransportAssocSSLTLSRequired=false`
3. Enable client authentication at the message layer.

   In this case, client authentication is supported but not required:
   `com.ibm.CSI.performClientAuthenticationRequired=false,`
   `com.ibm.CSI.performClientAuthenticationSupported=true`
4. Use all of the remaining defaults in the `sas.client.props` file.

**Configuring server, S1**

In the administrative console, server S1 is configured for incoming requests to support message-layer client authentication and incoming connections to support SSL without client certificate authentication. Server S1 is configured for outgoing requests to support identity assertion.

1. Configure S1 for incoming connections.
   a. Disable identity assertion.
   b. Enable user ID and password authentication.
   c. Enable SSL.
   d. Disable SSL client certificate authentication.
2. Configure S1 for outgoing connections.

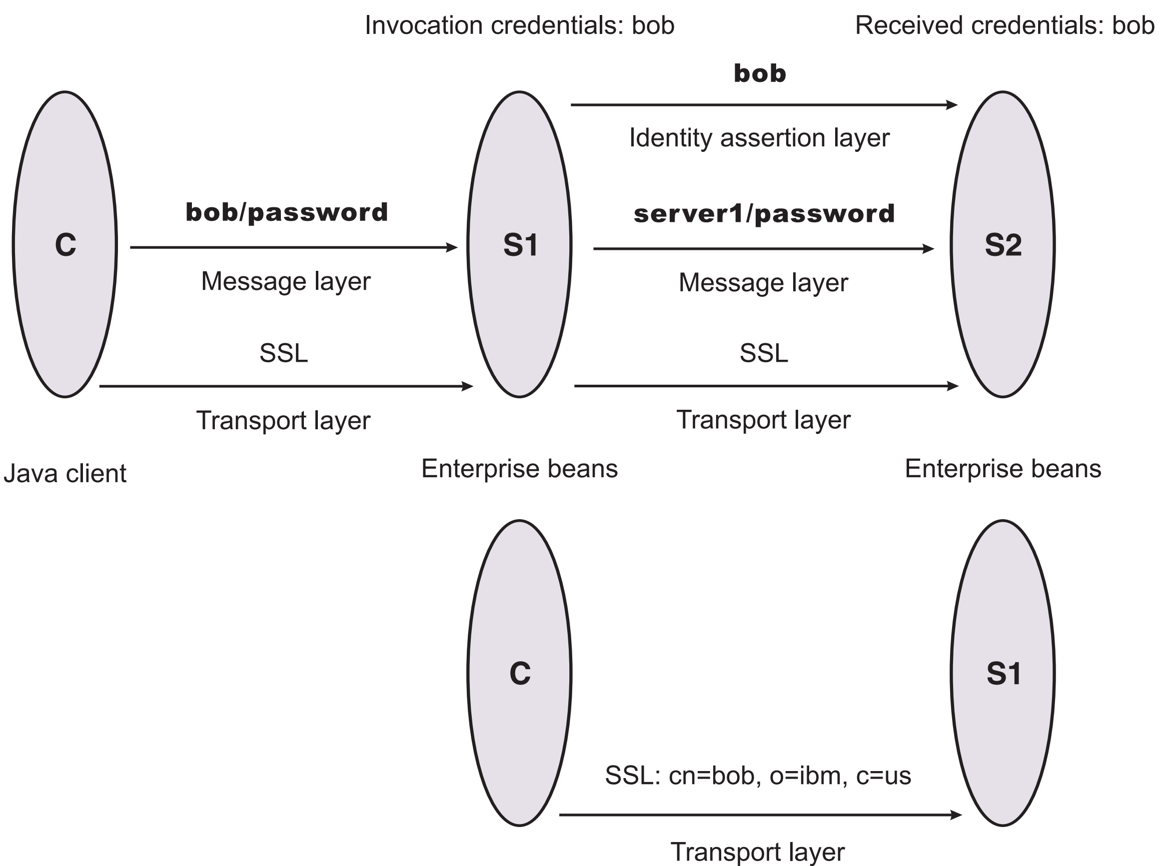
a. Enable identity assertion.
b. Disable user ID and password authentication.
c. Enable SSL.
d. Disable SSL client certificate authentication.

**Configuring server, S2**

In the administrative console, server S2 is configured for incoming requests to support identity assertion and to accept SSL connections. Complete the following steps to configure incoming connections. Configuration for outgoing requests and connections are not relevant for this scenario.

1. Enable identity assertion.
2. Disable user ID and password authentication.
3. Enable SSL.
4. Disable SSL client authentication.

## Scenario 2: Basic authentication, identity assertion, and client certificates




This scenario is the same as Scenario 1, except for the interaction from client C2 to server S2. Therefore, the configuration of Scenario 1 still is valid, but you have to modify server S2 slightly and add a configuration for client C2. The configuration is not modified for C1 or S1.

**Configuring client C2**

Client C2 requires transport layer authentication (Secure Sockets Layer (SSL) client certificates). To configure transport layer authentication:

1. Point the client to the `sas.client.props` file using the `com.ibm.CORBA.ConfigURL=file:/C:/was/properties/sas.client.props` property.

   All further configuration involves setting properties within this file.
2. Enable SSL.

   In this case, SSL is supported but not required:

   ```
   com.ibm.CSI.performTransportAssocSSLTLSSupported=true,
   com.ibm.CSI.performTransportAssocSSLTLSRequired=false
   ```
3. Disable client authentication at the message layer.

   ```
   com.ibm.CSI.performClientAuthenticationRequired=false,
   com.ibm.CSI.performClientAuthenticationSupported=false
   ```
4. Enable client authentication at the transport layer where it is supported, but not required:

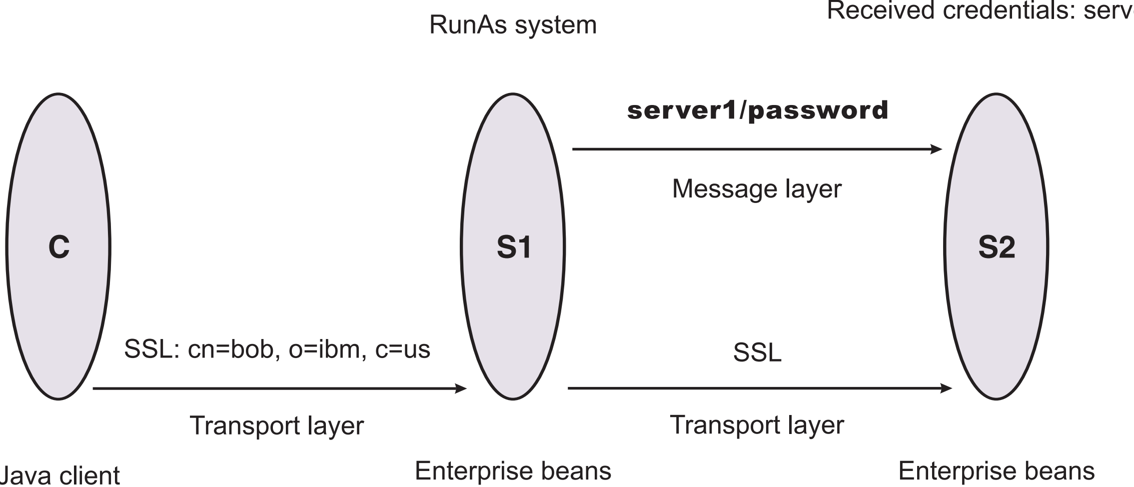
   ```
   com.ibm.CSI.performTLClientAuthenticationRequired=false,
   com.ibm.CSI.performTLClientAuthenticationSupported=true
   ```

**Configuring server, S2**

In the administrative console, server S2 is configured for incoming requests to SSL client authentication and identity assertion. Configuration for outgoing requests is not relevant for this scenario.
1. Enable identity assertion.
2. Disable user ID and password authentication.
3. Enable SSL.
4. Enable SSL client authentication.

You can mix and match these configuration options. However, a precedence exists as to which authentication features become the identity in the received credential:
1. Identity assertion
2. Message-layer client authentication (basic authentication or token)
3. Transport-layer client authentication (SSL certificates)

## Scenario 3: Client certificate authentication and RunAs system




This example presents a pure Java client, C, accessing a secure enterprise bean on S1. C authenticates to S1 using Secure Sockets Layer (SSL) client certificates. S1 maps the common name of the

distinguished name (DN) in the certificate to a user in the local registry. The user in this case is `bob`. The enterprise bean code on S1 accesses another enterprise bean on S2. Because the RunAs mode is `system`, the invocation credential is set as server1 for any outbound requests.

**Configuring C**

C requires transport layer authentication (SSL client certificates):

1. Point the client to the `sas.client.props` file using the `com.ibm.CORBA.ConfigURL=file:/C:/was/properties/sas.client.props` property.

   All further configuration involves setting properties within this file.
2. Enable SSL.

   In this case, SSL is supported but not required: `com.ibm.CSI.performTransportAssocSSLTLSSupported=true`, `com.ibm.CSI.performTransportAssocSSLTLSRequired=false`
3. Disable client authentication at the message layer:`com.ibm.CSI.performClientAuthenticationRequired=false`, `com.ibm.CSI.performClientAuthenticationSupported=false`
4. Enable client authentication at the transport layer. It is supported, but not required: `com.ibm.CSI.performTLClientAuthenticationRequired=false`, `com.ibm.CSI.performTLClientAuthenticationSupported=true`

**Configuring S1**

In the administrative console, S1 is configured for incoming connections to support SSL with client certificate authentication. The S1 server is configured for outgoing requests to support message layer client authentication.

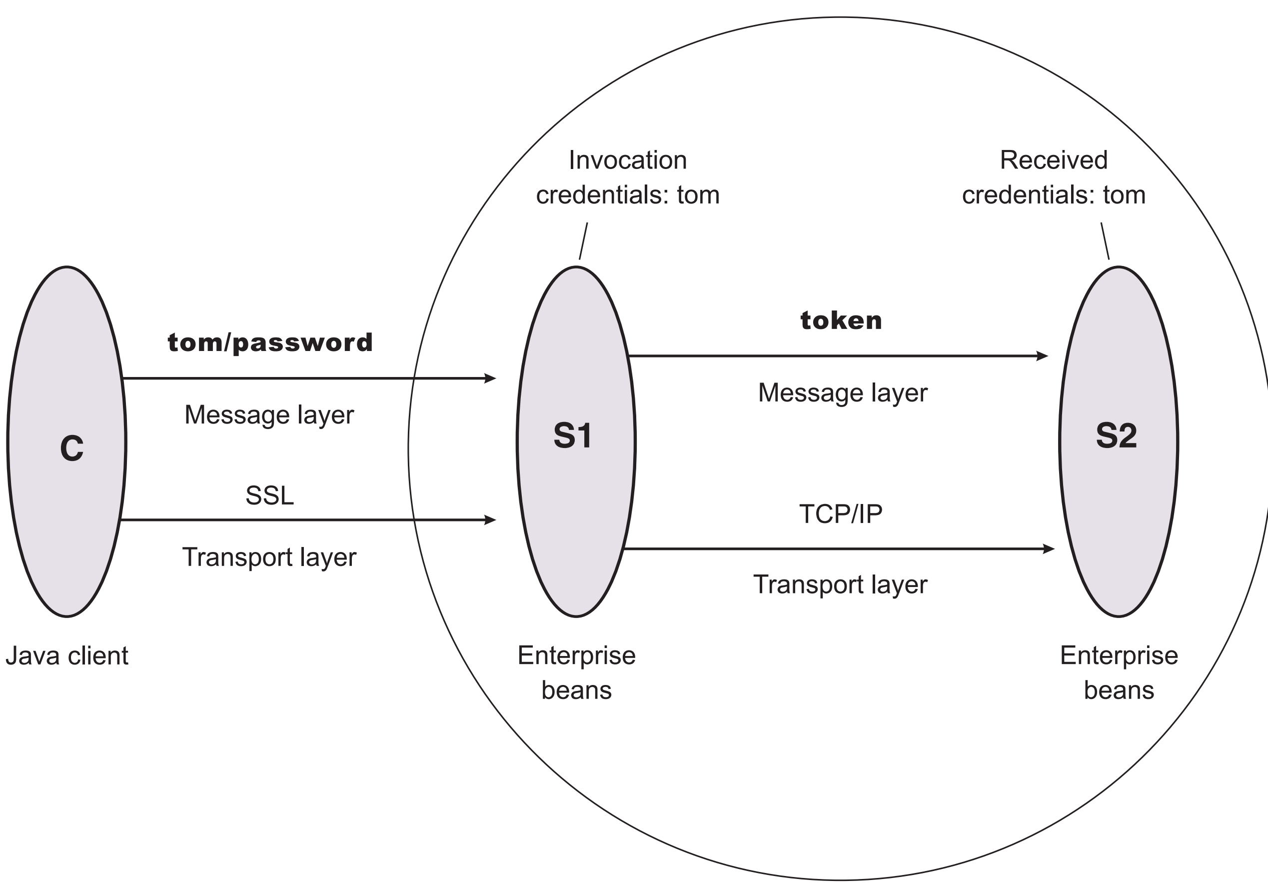
1. Configure S1 for incoming connections:
   a. Disable identity assertion.
   b. Disable user ID and password authentication.
   c. Enable SSL.
   d. Enable SSL client certificate authentication.
2. Configure S1 for outgoing connections:
   a. Disable identity assertion.
   b. Disable user ID and password authentication.
   c. Enable SSL.
   d. Enable SSL client certificate authentication.

**Configuring S2**

In the administrative console, the S2 server is configured for incoming requests to support message layer authentication over SSL. Configuration for outgoing requests is not relevant for this scenario.

1. Disable identity assertion.
2. Enable user ID and password authentication.
3. Enable SSL.
4. Disable SSL client authentication.

## Scenario 4: TCP/IP transport using a virtual private network




This scenario illustrates the ability to choose TCP/IP as the transport when it is appropriate. In some cases, when two servers are on the same virtual private network (VPN), it can be appropriate to select TCP/IP as the transport for performance reasons because the VPN already encrypts the message.

**Configuring C**

C requires message layer authentication with an SSL transport:

1. Point the client to the `sas.client.props` file using the `com.ibm.CORBA.ConfigURL=file:/C:/was/properties/sas.client.props` property. All further configuration involves setting properties within this file.
2. Enable SSL. In this case, SSL is supported but not required: `com.ibm.CSI.performTransportAssocSSLTLSSupported=true`, `com.ibm.CSI.performTransportAssocSSLTLSRequired=false`
3. Enable client authentication at the message layer. In this case, client authentication is supported but not required: `com.ibm.CSI.performClientAuthenticationRequired=false`, `com.ibm.CSI.performClientAuthenticationSupported=true`
4. Use the remaining defaults in the `sas.client.props` file.

**Configuring the S1 server**

In the administrative console, the S1 server is configured for incoming requests to support message-layer client authentication and incoming connections to support SSL without client certificate authentication. The S1 server is configured for outgoing requests to support identity assertion.

1. Configure S1 for incoming connections:

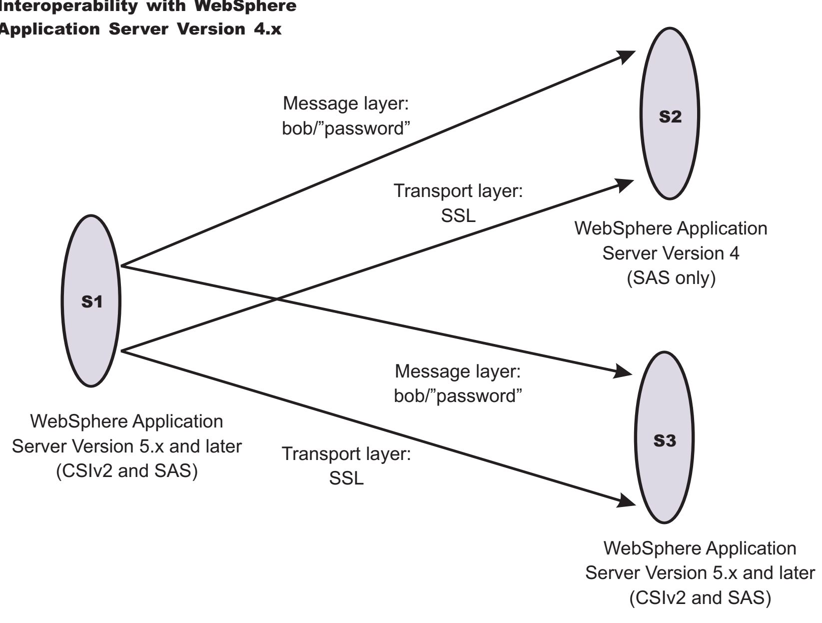
a. Disable identity assertion.
b. Enable user ID and password authentication.
c. Enable SSL.
d. Disable SSL client certificate authentication.
2. Configure S1 for outgoing connections:
a. Disable identity assertion.
b. Enable user ID and password authentication.
c. Disable SSL.

It is possible to enable SSL for inbound connections and disable SSL for outbound connections. The same is true in reverse.

**Configuring the S2 server**

In the administrative console, the S2 server is configured for incoming requests to support identity assertion and to accept SSL connections. Configuration for outgoing requests and connections are not relevant for this scenario.
1. Disable identity assertion.
2. Enable user ID and password authentication.
3. Disable SSL.

## Scenario 5: Interoperability with WebSphere Application Server Version 4.x




The purpose of this scenario is to show how secure interoperability can occur between different releases simultaneously while using multiple authentication protocols (Security Authentication Service (SAS) and Common Secure Interoperability Version 2 (CSIv2)). For WebSphere Application Server Version 5.x or later to communicate with a WebSphere Application Server Version 4, Version 5.x or later server must support either SAS or SAS and CSIv2 as the protocol choice. By choosing SAS and CSIv2, the Version 5.x or later server also can communicate with other Version 5.x or later servers that support CSI. If the only servers in your security domain are version 5.x or later, it is recommended that you choose CSI as the protocol because this prevents the SAS interceptors from loading. However, a chance exists that any server has to communicate with a previous release of WebSphere Application Server, select the protocol choice of SAS and CSIv2.

### Configuring the S1 server

The S1 server requires message layer authentication with an SSL transport. The protocol for the S1 server must be SAS and CSIv2. Configuration for incoming requests for the S1 server is not relevant for this scenario. To configure the S1 server for outgoing connections:
1. Disable identity assertion.
2. Enable user ID and password authentication.
3. Enable Secure Sockets Layer (SSL).
4. Disable SSL client certificate authentication.
5. Set authentication protocol to SAS and CSIv2 in the global security settings.

### Configuring the S2 server

All previous releases of WebSphere Application Server support the SAS authentication protocol only. No special configuration steps are needed other than enabling global security on the server (S2).

### Configuring the S3 server

In the administrative console, the S3 server is configured for incoming requests to support message layer authentication and to accept SSL connections. Configuration for outgoing requests and connections are not relevant for this scenario.
1. Enable identity assertion.
2. Disable user ID and password authentication.
3. Enable SSL.
4. Disable SSL client authentication.
5. Set authentication protocol to either CSI or SAS and CSIv2.

## Secure Sockets Layer

The Secure Sockets Layer (SSL) protocol provides transport layer security with authenticity, integrity, and confidentiality, for a secure connection between a client and server in WebSphere Application Server. The protocol runs above TCP/IP and below application protocols such as Hypertext Transfer Protocol (HTTP), Lightweight Directory Access Protocol (LDAP), and Internet Inter-ORB Protocol (IIOP), and provides trust and privacy for the transport data.

Depending upon the SSL configurations of both the client and server, various levels of trust, data integrity, and privacy can be established. Understanding the basic operation of SSL is very important to proper configuration and to achieve the required protection level for both client and application data.

Some of the security features that are provided by SSL are data encryption to prevent the exposure of sensitive information while data flows. Data signing prevents unauthorized modification of data while data flows. Client and server authentication ensures that you talk to the appropriate person or machine. SSL can be effective in securing an enterprise environment.

SSL is used by multiple components within WebSphere Application Server to provide trust and privacy. These components are the built-in HTTP transport, the Object Request Broker (ORB), and the secure LDAP client.

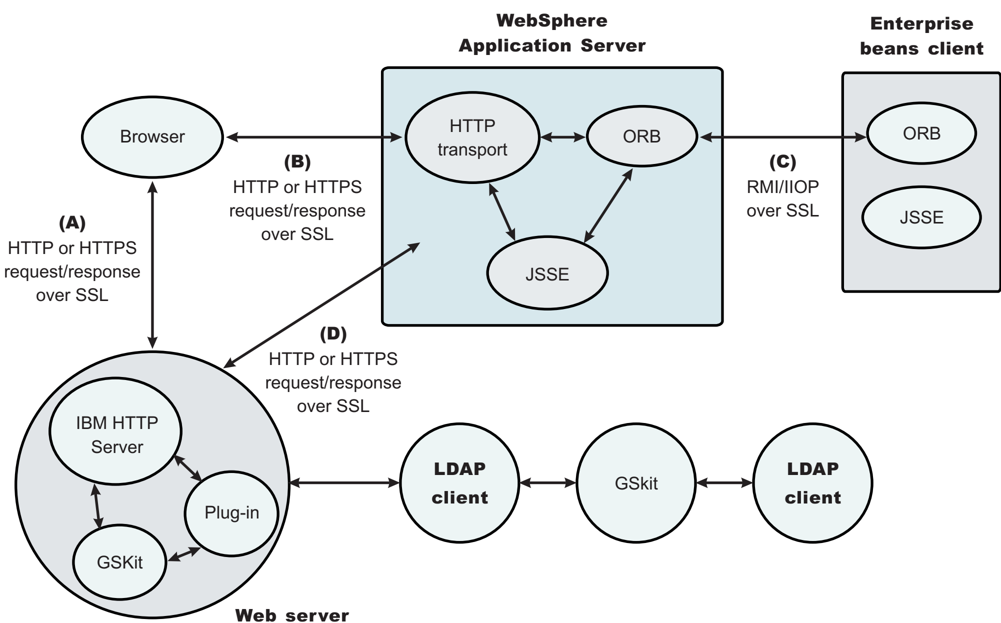



*Figure 2. SSL and WebSphere Application Server*

In this figure:
- The built-in HTTP transport in WebSphere Application Server accepts HTTP requests over SSL from a Web client like a browser.
- The Object Request Broker used in WebSphere Application Server can perform Internet Inter-ORB Protocol (IIOP) over SSL to secure the message.
- The secure LDAP client uses LDAP over SSL to securely connect to an LDAP user registry and is present only when LDAP is configured as the user registry.

## WebSphere Application Server and the IBM Java Secure Socket Extension (IBMJSSE and IBMJSSE2) providers

The SSL implementations used by WebSphere Application Server are the IBM Java Secure Sockets Extension (IBMJSSE) and the IBM Java Secure Sockets Extension 2 (IBMJSSE2). The IBMJSSE and IBMJSSE2 providers contain a reference implementation supporting SSL and Transport Layer Security (TLS) protocols and an application programming interface (API) framework. The IBMJSSE and IBMJSSE2 providers also come with a standard provider, which supplies Rivest Shamir Adleman (RSA) support for the signature-related J2EE Connector Architecture (JCA) features of the Java 2 platform, common SSL and TLS cipher suites, hardware cryptographic token device, X.509-based key and trust managers, and PKCS12 implementation for a JCA *keystore*. A graphical tool called Key Management Tool (iKeyman) also is provided to manage digital certificates. With this tool, you can create a new key database or a test digital certificate, add certificate authority (CA) roots to the database, copy certificates from one database to another as well as request and receive a digital certificate from a CA.

**Note:** The HTTP and JMS transports utilize the transport channel service for asynchronous I/O. This framework requires the use of IBMJSSE2 provider for SSL. Any provider you specify other than the IBMJSSE2 provider in the SSL repertoire is ignored and the IBMJSSE2 provider is used. Other SSL transports such as IIOP over SSL and LDAP over SSL utilize the provider you specify in the SSL repertoire configuration.

Configuring the JSSE provider is very similar to configuring most other SSL implementations (for example, GSKit); however, a couple of differences are worth noting.

- The JSSE provider supports both signer and personal certificate storage in an SSL key file, but it also supports a separate file called a *trust file*. A trust file can contain only signer certificates. You can put all of your personal certificates in an SSL keyfile and your signer certificates in a trustfile. This support might be helpful, for example, if you have an inexpensive hardware cryptographic device with only enough memory to hold a personal certificate. In this case, the keyfile refers to the hardware device and the trustfile refers to a file on a disk that contains all of the signer certificates.
- The JSSE provider does not recognize the proprietary SSL keyfile format, which is used by the plug-in (*.kdb* files). Instead, the JSSE provider recognizes standard file formats such as Java Key Standard (JKS). SSL keyfiles might not be shared between the plug-in and application server. Furthermore, a different implementation of the key management utility must be used to manage application server key and trustfiles.

Certain limitations exist with the Java Secure Socket Extension (JSSE) provider:
- Customer code using JSSE and Java Cryptography Extension (JCE) APIs must reside within WebSphere Application Server environment. This restriction includes applications that are deployed in WebSphere Application Server and client applications in the J2EE application client environment.
- Only `com.ibm.crypto.provider.IBMJCE`, `com.ibm.jsse.IBMJSSEProvider`, `com.ibm.security.cert.IBMCertPath`, and `com.ibm.crypto.pkcs11.provider.IBMPKCS11` are provided as the cryptography package providers.
- Interoperability of the IBMJSSE implementation with other SSL implementations by vendors is limited to tested implementations. The tested implementations include Microsoft Internet Information Services (IIS), BEA WebLogic Server, IBM AIX, and IBM AS/400.
- Hardware token support is limited to supported cryptographic token devices. .

| Tested for SSL clients | Tested for SSL clients or servers |
| --- | --- |
| IBM Security Kit Smartcard | IBM 4758-23 |
| GemPlus Smartcards | IBM 4758-23 |
| Rainbow iKey 1000/2000(USB ″Smartcard″ device) | IBM 4758-23 |

- The SSL protocol of Version 2.0 is not supported. In addition, the JSSE and JCE APIs are not supported with Java applet applications.

**WebSphere Application Server and the Federal Information Processing Standards for Java Secure Socket Extension and Java Cryptography Extension providers**

The Federal Information Processing Standards (FIPS)-approved Java Secure Socket Extension (JSSE) and Java Cryptography Extension (JCE) providers are optional in WebSphere Application Server. By default, the FIPS-approved JSSE and JCE providers are disabled. When these providers are enabled, WebSphere Application Server uses FIPS-approved cryptographic algorithms in the IBMJSSEFIPS and IBMJCEFIPS provider packages only.

**Important:** The IBMJSSEFIPS and IBMJCEFIPS modules are undergoing FIPS 140-2 certification. For more information on the FIPS certification process and to check the status of the IBM submission, see the Cryptographic Module Validation Program FIPS 140-1 and FIPS 140-2 Pre-validation List Web site.

## Authenticity

Authenticity of client and server identities during a Secure Sockets Layer (SSL) connection is validated by both communicating parties using public key cryptography or asymmetric cryptography, to prove the claimed identity from each other.

*Public key cryptography* is a cryptographic method that uses public and private keys to encrypt and decrypt messages. The public key is distributed as a public key certificate while the private key is kept private. The public key is also a cryptographic inverse of the private key. Well known public key

cryptographic algorithms such as the Rivest Shamir Adleman (RSA) algorithm and Diffie-Hellman (DH) algorithm are supported in WebSphere Application Server.

Public key certificates are either issued by a trusted organization like a certificate authority (CA) or extracted from a self-signed personal certificate by using the IBM Key Management Tool (iKeyman). A self-signed certificate is less secure and is not recommended for use in a production environment.

The public key certificate includes the following information:
- Issuer of the certificate
- Expiration date
- Subject that the certificate represents
- Public key belonging to the subject
- Signature by the issuer

You can link multiple key certificates into a certificate chain. In a certificate chain, the client is always first, while the certificate for a root CA is last. In between, each certificate belongs to the authority that issued the previous one.

During the Secure Sockets Layer (SSL) connection, a digital signature is also applied to avoid forged keys. The digital signature is an encrypted hash and cannot be reversed. It is very useful for validating the public keys.

SSL supports reciprocal authentication between the client and the server. This process is optional during the handshake. By default, a WebSphere Application Server client always authenticates its server during the SSL connection. For further protection, you can configure a WebSphere Application Server for client authentication.

Refer to the Transport Layer Security (TLS) specification at http://www.ietf.org/rfc/rfc2246.txt for further information.

## Confidentiality

Secure Sockets Layer (SSL) uses private or secret key cryptography or symmetric cryptography to support message confidentiality or privacy. After an initial handshake (a negotiation process by message exchange), the client and server decide on a secret key and a cipher suite. Between the communicating parties, each message encryption and decryption using the secret key occurs based on the cipher suite.

Private key cryptography requires the two communicating parties to use the same key for encryption and decryption. Both parties must have the key and keep the key private. Well known secret key cryptographic algorithms include the Data Encryption Standard (DES), triple-strength DES (3DES), and Rivest Cipher 4 (RC4), which are all supported in WebSphere Application Server. These algorithms provide excellent security and quick encryption.

A cryptographic algorithm is a *cipher*, while a set of ciphers is a *cipher suite*. A cipher suite is a combination of cryptographic parameters that define the security algorithms and the key sizes used for authentication, key agreement, encryption strength, and integrity protection.
- SSL_RSA_WITH_RC4_128_MD5
- SSL_RSA_WITH_RC4_128_SHA
- SSL_RSA_WITH_AES_128_CBC_SHA
- SSL_RSA_WITH_AES_256_CBC_SHA
- SSL_RSA_FIPS_WITH_DES_CBC_SHA
- SSL_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_RSA_WITH_AES_128_CBC_SHA
- SSL_DHE_RSA_WITH_AES_256_CBC_SHA
- SSL_DHE_RSA_WITH_DES_CBC_SHA

- SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_DSS_WITH_AES_128_CBC_SHA
- SSL_DHE_DSS_WITH_AES_256_CBC_SHA
- SSL_DHE_DSS_WITH_RC4_128_SHA
- SSL_DHE_DSS_WITH_DES_CBC_SHA
- SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_EXPORT_WITH_RC4_40_MD5
- SSL_RSA_EXPORT_WITH_DES40_CBC_SHA
- SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5
- SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
- SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
- SSL_RSA_WITH_NULL_MD5
- SSL_RSA_WITH_NULL_SHA
- SSL_DH_anon_WITH_AES_128_CBC_SHA *
- SSL_DH_anon_WITH_AES_256_CBC_SHA *
- SSL_DH_anon_WITH_RC4_128_MD5 *
- SSL_DH_anon_WITH_DES_CBC_SHA *
- SSL_DH_anon_WITH_3DES_EDE_CBC_SHA *
- SSL_DH_anon_EXPORT_WITH_RC4_40_MD5 *
- SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA *

**Important:** Although anonymous cipher suites are enabled, the IBM version of the Java Secure Sockets Extension (JSSE) client trust manager does not support anonymous cipher suites. The default implementation can be overwritten by providing your own trust manager that does support anonymous cipher suites.

All of the previously mentioned cipher suites provide data integrity protection by using hash algorithms like `MD5` and `SHA-1`. The cipher suite names ending with `_SHA` indicate that the `SHA-1` algorithm is used. SHA-1 is considered a stronger hash, while MD5 provides better performance.

The `SSL_DH_anon_xxx` cipher suites (for example, those cipher suites that begin with `SSL_DH_anon_`, where, `anon` is *anonymous*) are not enabled on the product client side. Because the Java Secure Socket Extension (JSSE) client trust manager does not support anonymous connections, the JSSE client must always establish trust in the server. However, the `SSL_DH_anon_xxx` cipher suites are enabled on the server side to support another type of client connection. That client might not require trust in the server. These cipher suites are vulnerable to *man-in-the-middle* attacks and are strongly discouraged. In a *man-in-the-middle* attack, an attacker can intercept and potentially modify communications between two parties without either party being aware of the attack.

Where:

| Name | Description |
| --- | --- |
| SSL | Secure Sockets Layer |
| RSA | • Public key algorithm developed by Rivest, Shamir and Adleman<br>• Requires RSA or DSS key exchange |
| DH | • Diffie-Hellman public key algorithm<br>• Server certificate contains the Diffie-Hellman parameters that are signed by the certificate authority (CA) |
| DHE | • Ephemeral Diffie-Hellman public key algorithm<br>• Diffie-Hellman parameters are signed by a DSS or an RSA certificate, which is signed by the certificate authority (CA) |

| Name | Description |
| --- | --- |
| DSS | Digital Signature Standard, using the Digital Signature Algorithm for digital signatures |
| DES | • Data Encryption Standard, an symmetric encryption algorithm<br>• Block cipher<br>• Performance cost is high when using software without the support of a hardware cryptographic device |
| 3DES | • Triple DES, increasing the security of DES by encrypting three times with different keys<br>• Strongest of the ciphers<br>• Performance cost is very high when using software without the support of a hardware cryptographic device support |
| RC4 | • A stream cipher designed for RSA<br>• Variable key-size stream cipher with key length from 40 bits to 128 bits |
| EDE | Encrypt-decrypt-encrypt for the triple DES algorithm |
| CBC | • Cipher block chaining<br>• A mode in which every plain text block that is encrypted with the block cipher is first exclusive-ORed with the previous ciphertext block |
| 128 | 128-bit key size |
| 40 | 40-bit key size |
| EXPORT | Exportable |
| MD5 | • Secure hashing function that converts an arbitrarily long data stream into a digest of fixed size<br>• Produces 128-bit hash |
| SHA | • Secure Hash Algorithm, same as SHA-1<br>• Produces 160-bit hash |
| anon | For anonymous connections |
| NULL | No encryption |
| WITH | The cryptographic algorithm is defined after this key word |

Refer to the Transport Layer Security (TLS) specification at http://www.ietf.org/rfc/rfc2246.txt for further information.

## Integrity

Secure Sockets Layer (SSL) uses a cryptographic hash function similar to checksum, to ensure data integrity in transit. Use the cryptographic hash function to detect accidental alterations in the data. This function does not require a cryptographic key. After a cryptographic hash is created, the hash is encrypted with a secret key. The private key belonging to the sender encrypts the hash for the digital signature of the message.

When secret key information is included with the cryptographic hash, the resulting hash is known as a *Key-Hashing Message Authentication Code* (HMAC) value. HMAC is a mechanism for message authentication that uses cryptographic hash functions. Use this mechanism with any iterative cryptographic hash function, in combination with a secret shared key.

In the product, both well known *one-way* hash algorithms, MD5 and SHA-1, are supported. One-way hash is an algorithm that converts processing data into a string of bits known as a *hash value* or a *message*

*digest*. *One-way* means that it is extremely difficult to turn the fixed string back into the original data. The following explanation includes both the MD5 and SHA-1 *one-way* hash algorithms:

- MD5 is a hash algorithm designed for a 32-bit machine. It takes a message of arbitrary length as input and produces a 128-bit hash value as output. Although this process is less secure than SHA-1, MD5 provides better performance.
- SHA-1 is a secure hash algorithm specified in the Secure Hash Standard. It is designed to produce a 160-bit hash. Although it is slightly slower than MD5, the larger message digest makes it more secure against attacks like *brute-force collision*.

Refer to the Transport Layer Security (TLS) specification at http://www.ietf.org/rfc/rfc2246.txt for further information.

## Configuring Secure Sockets Layer

Secure Sockets Layer (SSL) is used by multiple components within WebSphere Application Server to provide trust and privacy. The following is a listing of these components:

- Built-in HTTP Transport
- Object Request Broker (ORB) for client and server
- Secure Lightweight Directory Access Protocol (LDAP) client.

Configuring SSL is different between client and server with WebSphere Application Server

1. Configure the client (JSSE). Use the `sas.client.props` file located, by default, in the *install_root*/profiles/*profile_name*/properties directory. The `sas.client.props` file is a configuration file that contains lists of property-value pairs, using the syntax *<property> = <value>*. The property names are case sensitive, but the values are not; the values are converted to lowercase when the file is read. Specify the following properties for an SSL connection:
   - com.ibm.ssl.protocol
   - com.ibm.ssl.keyStoreType
   - com.ibm.ssl.keyStore
   - com.ibm.ssl.keyStorePassword
   - com.ibm.ssl.trustStoreType
   - com.ibm.ssl.trustStore
   - com.ibm.ssl.trustStorePassword
   - com.ibm.ssl.enabledCipherSuites
   - com.ibm.ssl.contextProvider
   - com.ibm.ssl.keyStoreServerAlias
   - com.ibm.ssl.keyStoreClientAlias
   - For the Secure Authentication Services (SAS) authentication protocol only: com.ibm.CORBA.standardPerformQOPModels
   - For the cryptographic token device:
     - com.ibm.ssl.tokenType
     - com.ibm.ssl.tokenLibraryFile
     - com.ibm.ssl.tokenPassword
     - com.ibm.ssl.tokenSlot (added as a custom property)

   **Note:** Although WebSphere Application Server supports the IBM Federal Information Processing Standard-approved Java Secure Socket Extension (IBMJSSEFIPS), IBMJSSEFIPS is not supported for the HTTP and JMS transports due to their use of the channel framework which requires the IBMJSSE2 provider. This provider itself does not need to be FIPS compliant because it uses IBMJCE for encryption.
2. Configure the server. Use the administrative console to configure an application server that makes SSL connections. To start the administrative console, specify the following Web address: http://*server_hostname*:9060/ibm/console.

3. Create an SSL configuration repertoires alias or entry. You can select the alias later when a component is configured for SSL support. An SSL configuration repertoires entry contains the following fields:
   - Typical configuration settings:
     - Alias
     - Key file name
     - Key file password
     - Key file format
     - Trust file name
     - Trust file password
     - Trust file format
     - Client authentication
     - Security level
     - Cipher suites
   - For the cryptographic token device:
     - Cryptographic token (Create the alias first so you can configure these fields).
       - Token type
       - Library file
       - Password
   - For additional Java properties:
     - Custom properties (Create the alias first so you can configure these fields).
       - `com.ibm.ssl.contextProvider`
       - `com.ibm.ssl.protocol`
       - `com.ibm.ssl.tokenSlot` (for crypto slot)
       - `com.ibm.ssl.keyStoreClientAlias` (alias selection for client authentication to servers)
       - `com.ibm.ssl.keyStoreServerAlias` (alias selection for server authentication to clients)

   **Note:** WebSphere Application Server contains IBM Developer Kit for Java Technology Edition Version 1.4.x , which includes changes from IBM Developer Kit for Java Technology Edition Version 1.3. See Changes to IBM Developer Kit for Java Technology Edition Version 1.4.x for more information.

## Configuring Secure Sockets Layer for Web client authentication

To enable client-side certificate-based authentication, you must modify the authentication method that is defined on the Java 2 Platform, Enterprise Edition (J2EE) Web module that you want to manage. The Web module might already be configured to use the basic challenge authentication method. In this case, modify the challenge type to `client certificate`. This functionality is delivered to the WebSphere Application Server administrator in assembly tools. However, developers can use the Rational Web Developer environment to achieve the same result.

1. Launch the assembly tools. This step can be done either before an enterprise application archive `.ear` file is deployed into WebSphere Application Server or after deployment into the product. The latter option is discouraged in a production environment because it involves opening the expanded archive correlating to the enterprise application archive, found in the `installedApps` directory.
2. Locate and expand the Web module package under an application to enable the client-side certificate authentication method.
3. Select the appropriate Web application, and switch to the **Advanced** tab. Modify the authentication method to client certificate. The realm name is the scope of the login operation and is the same for all participating resources.
4. Click **OK**, and save the changes you made with the assembly tools.
5. Stop and restart the associated application server containing the resource, so that the security modification is included in the run time. Complete this action if the modification is made to a resource that already is deployed in WebSphere Application Server.

Now your enterprise application prompts the user for proof of identity with a certificate.

**Note:**

The Web server must also be configured to request a client certificate. If the Web server is external, refer to the appropriate configuration documentation. If the Web server is the Web container transport (for example, 9043) within WebSphere Application Server, verify that the **client authentication** flag is selected in the referenced SSL configuration.

Also, add the browser's signer certificate to the application server's keystore. For a self-signed personal certificate, the signer certificate is the is the public key of the personal certificate. For a certificate authority-signed server personal certificate, the signer certificate is the root certificate authority certificate of the certificate authority that signed the personal certificate.

Refer to the Map certificates to users article to determine how a certificate is authenticated within the product.

## Configuring Secure Sockets Layer for the Lightweight Directory Access Protocol client

This topic describes how to establish a Secure Sockets Layer (SSL) connection between WebSphere Application Server and a Lightweight Directory Access Protocol (LDAP) server. This page provides an overview. Refer to the linked pages for more details. To understand SSL concepts, refer to "Secure Sockets Layer" on page 411.

Setting up an SSL connection between WebSphere Application Server and an LDAP server requires the following steps:

1. Set up an LDAP server with users. The server configured in this example is IBM Directory Server. Other servers are configured differently. Refer to the documentation of the directory server you are using for details on SSL enablement. For a product-supported LDAP directory server, see the "Supported directory services" on page 208 article.
2. Configure certificates for the LDAP server using the key management utility (iKeyman) that is located in the `install_dir\java\jre\bin` directory.
3. Click **Key Database File > New**.
4. Type `LDAPkey.kdb` as the file name and a proper path and click **OK**.
5. Specify a password, confirm the password, and click **OK**.
6. Under Key database content, select **Personal Certificates**.
7. Click **New Self-signed**. The **Create New Self-Signed Certificate** panel is displayed. Type the following required information in the fields and click **OK**:

   **Key Label**
   
   `LDAP_Cert`

   **Version**
   
   Select the version of the X.509 certificate.

   **Key size**
   
   Select either a 512 or a 1024 bit size for your key.

   **Common Name**
   
   droplet.austin.ibm.com

   This common name is the host name where the WebSphere Application Server plug-in runs.

   **Organization**
   
   `ibm`

   **Country**
   
   `US`

   **Validity period**
   
   Specify the number days in which your certificate is valid.
8. Return to the Personal Certificates panel and click **Extract Certificate**.

9. Click the **Base64-encoded ASCII data** data type. Type `LDAP_cert.arm` as the file name and a proper path. Click **OK**.
10. Enable SSL on the LDAP server:
    a. Copy the `LDAPkey.kdb`, `LDAPkey.sth`, `LDAPkey.rdb`, and `LDAPkey.crl` files created previously to the LDAP server system, for example, the `\Program Files\IBM\LDAP\ssl\` directory.
    b. Open the LDAP Web administrator from a browser (http://secnt3.austin.ibm.com/ldap, for example). IBM HTTP Server is running on `secnt3`.
    c. Click **SSL properties** to open the SSL Settings window.
    d. Click **SSL On** > **Server Authentication** and type an SSL port (`636`, for example) and a full path to the `LDAPkey.kdb` file.
    e. Click **Apply**, and restart the LDAP server.
11. Manage certificates for WebSphere Application Server using the default SSL key files.
    a. Open the *install_root*`\etc\DummyServerTrustFile.jks` file using the key management utility that shipped with WebSphere Application Server. The password is `WebAS`.
    b. Click **Personal Certificates > Import**. The **Import Key** panel is displayed. Specify `LDAP_cert.arm` for the file name. Complete this step for all the servers including the deployment manager.
12. Establish a connection between the WebSphere Application Server and the LDAP server using the WebSphere Application Server administrative console.
    a. Click **Security > Global security**.
    b. Under User registries, click **LDAP**.
    c. Enter the **Server ID**, **Server Password**, **Type**, **Host**, **Port**, and **Base Distinguished Name** fields.
    d. Select the **SSL Enabled** option. The port is the same port number that the LDAP server is using for SSL (636, for example).
    e. Click **Apply**.
    f. Return to the Global security panel and click **Authentication Mechanisms > LTPA > Single SignOn (SSO)**.
    g. Under Additional properties, click **Single signon (SSO)**.
    h. Type in a domain name (`austin.ibm.com`, for example).
    i. Click **Apply**.
13. Enable global security.
    a. Click **Security > Global Security**.
    b. Select the **Enable global security** option.
    c. Select the **Lightweight Third Party Authentication (LTPA)** option as the active authentication mechanism and the **Lightweight Directory Access Protocol (LDAP) user registry** option as the active user registry.

       **Note:** Verify that the security level for the LDAP server is set to HIGH. The default security level is HIGH (128-bit).
    d. Click **Apply** and **Save**.
    e. Verify that the ibm-slapdSSLCipherSpecs parameter in the *LDAP_install_root*`\etc\slapd32.conf` file has the value, 15360, instead of 12288.
    f. Restart the servers.

       Restarting the servers ensures that the security settings are synchronized between the deployment manager and the application servers.

You can test the configuration by accessing https://*fully_qualified_host_name*:9443/snoop. You are presented with a login challenge. This test can be beneficial when using LDAP as your user registry. Sensitive information can flow between the WebSphere Application Server and the LDAP server, including passwords. Using SSL to encrypt the data protects this sensitive information.

1. If you are enabling security, make sure that you complete the remaining steps. As the final step, validate this configuration by clicking **OK** or **Apply** in the Global Security panel. Refer to the "Configuring global security" on page 142 article for detailed steps on enabling global security.
2. For changes in this panel to become effective, save, stop, and start all WebSphere Application Servers (cells, nodes and all the application servers).
3. After the server starts up, go through all the security-related tasks (getting users, getting groups, and so on) to make sure that the changes to the filters are functioning.

## Configuring IBM HTTP Server for Secure Sockets Layer mutual authentication

IBM HTTP Server supports Secure Sockets Layer (SSL) Version 2 and Version 3 and Transport Layer Security (TLS) Version 1. IBM HTTP Server is based on the Apache Web server, but for SSL configuration it requires the IBM-supplied SSL modules, rather than the OpenSSL modules. This document describes configuration of IBM HTTP Server, although it is possible to use another supported Web server.

SSL is disabled by default and it is necessary to modify a configuration file and generate a server-side certificate using the key management utility (iKeyman) provided with IBM HTTP Server to enable SSL.

1. For a single server, enable SSL on IBM HTTP Server (port 443,for example).
2. To set up certificates complete the following steps: Start the key management utility by clicking **Start > Programs > IBM HTTP Server > Start Key Management Utility**. Refer to Requesting a CA-signed personal certificate, Creating a certificate signing request (CSR), Receiving a CA-signed personal certificate, and Extracting a public certificate for use in a truststore file
3. Create a key database and click **Key Database File > New**.
4. Type a file name, `serverkey.kdb`, for example, and the location path. Click **OK**.
5. Type a password, select the **Stash the password to a file** check box and click **OK**.
6. Obtain a personal certificate for IBM HTTP Server: Click **Personal Certificate** in the key management utility menu. Click **Create > New Certificate Request**. The **Create New Key and Certificate Request** panel is displayed. Complete the following information:

   **Key label**
      `Server_Cert`

   **Key size**
      Select either a 512 or a 1024 bit size for your key.

   **Common name**
      droplet.austin.ibm.com

   **Organization**
      `IBM`

   **Organization unit**
      WebSphere

   **Locality**
      Austin

   **State**
      Texas

   **Zip code**
      76758

   **Country**
      `US`

   **File name**
      `Server_certreq.arm`

   The Verisign Test CA Root Certificate is in the set of signer certificates that ship with the IKeyMan utility for IBM HTTP Server.
7. Go to `http://www.verisign.com`, click **Free SSL Trial**. Complete the profile information, click **Submit**, and click **Continue** twice.

8. Use your favorite text editor to edit the request file `Server_certreq.arm`, and copy the entire contents of the file into the browser request panel. Click **Continue**. VeriSign sends the signed personal certificate to your e-mail.
9. Copy and paste this certificate into a file, for example `Server_Cert.arm`. Click **Personal Certificate** from the menu in the key management utility. Click **Receive**. Specify the file name, `Server_Cert.arm`, and click **OK**. You might need to add VeriSign test root certificate to the signer certificates for the receive to be successful. Close the `serverkey.kdb` file.
10. To allow IBM HTTP Server to support HTTPS, port 443, for example, enable SSL on IBM HTTP Server. Modify the configuration file of IBM HTTP Server, `IHS_HOME/conf/httpd.conf`. You also can enable SSL through the IBM HTTP Server administrative console. Open the `IHS_HOME/conf/httpd.conf`file and add the following lines to the bottom of the file:

```
LoadModule  ibm_ssl_module   modules/mod_ibm_ssl.so
Listen 443
<VirtualHost  droplet.austin.ibm.com:443>
ServerName  droplet.austin.ibm.com
DocumentRoot <install_root>\htdocs
SSLEnable
#SSLClientAuth  required
</VirtualHost>
SSLDisable
Keyfile <IHS_HOME>/serverkey.kdb
```

**Note:** Change the host name and the path for the key file accordingly. Modify the Web server to support client certificates by uncommenting the SSLClientAuth directive shown in the `httpd.conf` file.

SSLClientAuth  required

11. Restart IBM HTTP Server.
12. Test SSL between a browser and IBM HTTP Server. For more information on the default IBM HTTP Server port number, see Port number settings in WebSphere Application Server versions.
13. Follow the prompts to select a personal certificate if the SSLClientAuth directive is set to required.
14. To enable the application server to communicate with IBM HTTP Server using port 443, add the host alias on the default_host. In the administrative console, click **Environment > Virtual Hosts >***default_host*. Under Additional properties, click **Host Aliases > New**. Enter the following information in the appropriate fields:

    **Host name**
        *

    **Port** `443`
15. Click **Apply** and **Save** When you click **Save**, the information is written to the `security.xml` file and the Web server plug-in is automatically updated.
16. Restart WebSphere Application Server.
17. Test your connection.

You can connect to the Snoop servlet.

Enable Secure Sockets Layer communication between IBM HTTP Server and WebSphere Application Server.

## Configuring the Web server plug-in for Secure Sockets Layer

WebSphere has an internal HTTP transport which accepts HTTP requests. If you install an external HTTP server, the Web server plug-in must forward requests from the external HTTP server to WebSphere's

internal HTTP transport. You should follow HTTP vendor's instruction to install and configure your HTTP server. Test your HTTP server by accessing http://your-host-URL and https://your-host-URL. You should also have Web server plugin installed. See ″Installing IBM HTTP Server″ for instructions on installing HTTP Server and Web server plugin. The connection between external HTTP server and WebSphere is by default not secured, even when global security is enabled.

This section documents the configuration necessary to instantiate a secure connection between the Web server plug-in and the internal HTTP transport in the WebSphere Application Server Web container on a distributed platform. By default, this connection is not secure, even when global security is enabled. This document discusses the configuration for IBM HTTP Server; however, the Web server-related configuration in this situation is not specific to any distributed platform Web server.

1. "Creating self-signed personal certificates" on page 447. The Web server plug-in requires a key ring file to store its own private and public key files and to store the public certificate from the Web container key file. The following steps are required to generate a self-signed certificate for the Web server plug-in.

   When you install Web server plugin, a default key ring, plugin-key.kdb, is installed in plugin_root\etc. Use this file instead of creating a new one. In the following steps, a new file is created, but the steps are similar if you use an existing file. Create a directory on the Web server host for storing the key ring file that is referenced by the plug-in and associated files (for example, IHS_install_root\conf\keys).

   a. Create a directory on the Web server host for storing the key ring file that is referenced by the plug-in and associated files, for example: `IHS_install_root\conf\keys`.
   b. Launch the key management utility (iKeyman), which is available in the WebSphere Application Server *install_root*`\bin` installation directory.
   c. From the iKeyman menu, click **Key Database File** > **New**.
   d. Enter the following settings:

      **Key database type**
      : `CMS Key Database File`

      **File name**
      : `WASplugin.kdb`

      **Location**
      : `C:\http1324\conf\keys\` or the file of your choice

   e. Click **OK**.
   f. Set the password of your choice at the password prompt and confirm the password.
   g. Click the **Stash the password to a file?** option.
   h. Click **OK**.
   i. From the iKeyman menu, click **Create > New Self-Signed Certificate** to create a new self-signed certificate key pair. Specify the following options. Optionally, you can choose to complete all of the remaining fields.

      **Key label**
      : `WASplugin`

      **Version**
      : `X509 V3`

      **Key size**
      : `1024`

      **Common name**
      : `droplet.austin.ibm.com`

      **Organization**
      : `IBM`

      **Country**
      : `US`

      **Validity period**
      : `365`

j. Click **OK**.

k. Extract the public self-signed certificate key. This key is used later by the embedded HTTP server peer to authenticate connections that originate from the plug-in.

l. Click **Personal Certificates** in the menu and select the `WASplugin` certificate that you just created.

m. Click **Extract Certificate**. Extract the certificate to a file:

**Data type**
`Base64-encoded ASCII data`

**Certificate file name**
`WASpluginPubCert.arm`

**Location**
`C:\http1324\conf\keys` , or a directory of your choice

n. Click **OK**.

o. Close the key database and exit the iKeyman utility when you finish.

2. Generate a self-signed certificate for the Web container.

a. Launch the JKS-capable iKeyman version that is located the product `/bin` directory.

b. Click **Key Database File > New** from the iKeyman menu.

c. Enter the following settings:

**Key database type**
`JKS`

**File name**
`WASWebContainer.jks`

**Location**
`C:\WebSphere\AppServer\etc\` or the directory of your choice

d. Click **OK**.

e. Set the password of your choice at the password prompt and confirm the password.

f. Click **Create > New Self-Signed Certificate** from the iKeyman menu. The following values are used in this example:

**Key Label**
`WASWebContainer`

**Version**
`X509 V3`

**Key size**
`1024`

**Common name**
`droplet.austin.ibm.com`

**Organization**
`IBM`

**Country**
`US`

**Validity Period**
`365`

g. Click **OK**.

h. Extract the public self-signed certificate key. This key is used later by the Web server plug-in peer to authenticate connections that originate from the embedded HTTP server in the product.

i. Click **Personal Certificates** from the list. Select the **WASWebContainer** certificate that you just created. Click **Extract Certificate**. Extract the certificate to a file:

**Data type**
`Base64-encoded ASCII data`

**Certificate file name**
`WASWebContainerPubCert.arm`

**Location**
`C:\WebSphere\AppServer\etc\`

j. Click **OK**.
k. Close the database and exit the key management utility.

3. Exchange the public certificates.
   a. Copy the `WASpluginPubCert.arm` file from the Web server machine to the WebSphere Application Server machine. The source directory in this case is `C:\http1324\conf\keys`, while the destination is `C:\WebSphere\Appserver\etc`.
   b. Copy the `WASWebContainerPubCert.arm` file from the product machine to the Web server machine. The source directory in this case is `C:\WebSphere\Appserver\etc`, while the destination is `C:\http1324\conf\keys`.
4. Import the certificate into the Web server plug-in key file.
   a. On the Web server machine, launch the iKeyman utility, which supports the CMS key database format.
   b. From the iKeyman menu, click **Key Database File > Open** and select the previously created key database file: `WASplugin.kdb`.
   c. In the password prompt window, enter the password. Click **OK**.
   d. Click **Signer Certificates** from the list and click **Add**. This action imports the public certificate previously extracted from the embedded HTTP server (Web container) keystore file.

      **Data type**
      `Base64-encoded ASCII data`

      **Certificate file name**
      `WASWebContainerPubCert.arm`

      **Location**
      `C:\WebSphere\Appserver\etc\`
   e. Click **OK**. You are prompted for a label name that represents the trusted signer public certificate.
   f. Enter a label for the certificate: `WASWebContainer`.
   g. Close the key database and exit IKeyman when you finish.
5. Import the certificate into the Web container keystore file.
   a. On the WebSphere Application Server machine, launch the JKS-capable iKeyman version, which is located in the product `/bin` directory.
   b. From the iKeyman menu, click **Key Database File > Open**. Select the previously created `WASWebContainer.jks` file.
   c. In the password prompt window, enter the password. Click **OK**.
   d. Click **Signer Certificates** from the list. Click **Add**. This action imports the public certificate previously extracted from the embedded HTTP server (Web container) keystore file.

      **Data type**
      `Base64-encoded ASCII data`

      **Certificate file name**
      `WASpluginPubCert.arm`

      **Location**
      `C:\WebSphere\Appserver\etc\`
   e. Click **OK**. You are prompted for a label name that represents the trusted signer public certificate.
   f. Enter a label for the certificate: `WASplugin`.
   g. Close the key database and exit iKeyman when you finish.
6. Modify the Web server plug-in file. In a production environment, add the secure transport definition, port 9443, to the `plugin-cfg.xml` file. For example, your modified `plugin-key.kdb` file contains the following lines:

```
<Transport Hostname="hpws07" Port="9080" Protocol="http"/>
<Transport Hostname="hpws07" Port="9443" Protocol="https"/>
```

After you verify that the proper `plugin-key.kdb` and `plugin-key.sth` files exist on the Web server, modify the `plugin-cfg.xml` file that resides on the Web server. You must specify the local path to both the `plugin-key.kdb` and `plugin-key.sth` files in the `plugin-cfg.xml` file.

**Important:** If you manually edit the `plugin-cfg.xml` file and an automatic regeneration of the file occurs, you must replace your manual edits.

7. Modify the Web container to support SSL. To complete the configuration between Web server plug-in and Web container, modify the WebSphere Application Server Web container to use the previously created self-signed certificates.
   a. Start the WebSphere Application Server administrative console.
   b. Click **Security > SSL**.
   c. Click **New JSSE repertoire** to create a new entry in the repertoire. Provide the following values to complete the form:
      **Alias** `WebContainerSSLSettings`
      **Security level**
         `HIGH`
      **Key file name**
         `C:\WebSphere\Appserver\etc\WASWebContainer.jks`
      **Key file password**
         *<key_file_password>*
      **Key file format**
         `JKS`
      **Trust file name**
         `C:\WebSphere\Appserver\etc\WASWebContainer.jks`
      **Trust file password**
         *<trust_file_password>*
      **Trust file format**
         `JKS`
   d. Click **OK**.
   e. If you want mutual SSL between the two parties, select the **Client authentication** option.
   f. Save the configuration in the administrative console.
   g. Click **Servers > Clusters >** *<cluster_name>* **> Cluster Members >** *<server_name>*.
   h. Under Container settings, click **Web container settings > Web container transport chains**.

      You can either modify the WCInboundDefaultSecure transport chain or click **New** and create a new transport chain.

      If you are modifying the WCInboundDefaultSecure transport chain, click **TCP Inbound Channel (TCP 4)**. Under related items, click **Ports**. Click **WC defaulthost secure** and modify the information in the Host and Port fields. Click **OK** and then **Save**.

      If you create a new transport chain, use the transport chain wizard, and specify a secure port number. You must add the same port number to the virtual hosts.
   i. Add a new virtual host entry by clicking **Environment > Virtual hosts > default_host**.
   j. Under Additional properties, click **Host aliases > New**.
   k. Enter a host name and specify the same port number that you specified for the transport chain.
   l. Click **OK**.
   m. Click **Save** at the top of the panel.
8. **Optional:** If you want to access the Web server plug-in from the Web server, click **Servers > Web servers**, and then click the **Generate Plug-in** option.
9. Test the secure connection. Test the secure connection by accessing a Web application on the WebSphere Application Server using port 9443. For example, `https://droplet.austin.ibm.com:9443/snoop`.

10. Import the correct certificate with public and private keys into the browser to test the secured connection, when client-side certification is required.
    a. Launch the iKeyman utility that supports the CMS key database file, on the Web server machine. The iKeyman utility is also bundled with IBM HTTP Server.
    b. Open the key file for the plug-in, `C:\http1324\conf\keys\WASplugin.kdb`. Provide the password when prompted.
    c. Click **WASplugin certificate**, located under the personal certificates. Click **Export**.
    d. Save the certificate in PKCS12 format to a file, for example `C:\http1324\conf\keys\WASplugin.p12` . Provide a password to secure the PKCS12 certificate file.
    e. Close the key file and exit iKeyman.
    f. Copy the saved `WASplugin.p12` file to the client machine from where you access the product server.
    g. Import the PKCS12 file into your browser. Then, access `https://`*`your_server_address`*`:9443/snoop`.
    h. The browser asks which personal certificate to use for the connection. Select the certificate, and continue connecting.
    i. After the browser test with direct product access is successful, test the connection through the Web server using port 9443. For example, `https://`*`your_server_address`*`:9443/snoop`.

The IBM HTTP Server plug-in and the internal Web server are configured for SSL.

## Configuring Secure Sockets Layer for Java client authentication

WebSphere Application Server supports Java client authentication using a digital certificate when the client attempts to make a Secure Sockets Layer (SSL) connection. The authentication occurs during an SSL handshake. The SSL handshake is a series of messages exchanged over the SSL protocol to negotiate for connection-specific protection. During the handshake, the secure server requests that the client to send back a certificate or certificate chain for the authentication.

To configure SSL for Java client authentication, consider the following questions:
- Have you enabled security with your WebSphere Application Server?
- Have you configured Common Secure Interoperability (CSI) authentication protocol for your target application server? Refer to “Configuring global security” on page 142 for more details.

  **Note:** The Security Authentication Service (SAS) authentication protocol does not support Java client authentication with SSL transport.
- Have you configured your server to support secure transport for the CSIv2 inbound authentication protocol?
- Have you configured your server to support client authentication at the transport layer for the inbound CSI authentication protocol?
- If you are using a self-signed personal certificate, have you exported the public certificate from your client application Java keystore file or cryptographic token device?
- If you are using a certificate authority (CA)-signed personal certificate, have you received the root certificate of the CA?
- If you are using a self-signed personal certificate, have you imported the public certificate into your target Java truststore file as a signer certificate?
- If you are using a CA-signed (certificate authority) personal certificate, have you imported the CA root certificate into your target Java truststore file as a signer certificate?
- Does the common name (CN) specified in your personal certificate name exist in your configured user registry or is there a SAF mapping for the certificate?

If you answer yes to all of these questions that are appropriate to your product and platform, you can configure SSL for Java client authentication.

**Note:** Java client authentication using digital certificates is supported only by the Common Secure Interoperability Version 2 (CSIv2) authentication protocol.

1. "Configuring Common Secure Interoperability Version 2 for Secure Sockets Layer client authentication."
2. "Adding keystore files" on page 429.
3. "Adding truststore files" on page 430.
4. Save changes.
5. Restart the server if you configured the server.

A secure client connects to a secure Internet InterORB Protocol (IIOP) server that requires client authentication at the transport layer. If a connection problem occurs, you can set a Java property, `javax.net.debug=true`, before you run your client or your server to generate debugging information. See Chapter 16, "Troubleshooting security configurations," on page 955 for further information about how to debug an IBMJSSE problem.

## Configuring Common Secure Interoperability Version 2 for Secure Sockets Layer client authentication

Configure the Secure Sockets Layer (SSL) client authentication using the `sas.client.props` configuration file or the administrative console. To configure a Java client application, use the `sas.client.props` configuration file. By default, the `sas.client.props` file is located in the *install_root*`\profiles\`*profile_name*`\properties` directory of your WebSphere Application Server installation.

To configure a WebSphere Application Server, use the administrative console. To start the administrative console, specify URL: `http://`*server host_name*`:9060/ibm/console`.

To configure a Java client application, complete the following steps, which explain how to edit the `sas.client.props` file directly:

1. To require SSL client authentication, set property `com.ibm.CSI.performTLClientAuthenticationRequired=true`. Do not set this property unless you know your target server also supports SSL client authentication for the inbound CSI authentication protocol.
2. To support SSL client authentication, set the property `com.ibm.CSI.performTLClientAuthenticationSupported=true`.
3. To specify the CSI protocol, set the property `com.ibm.CSI.protocol=csiv2`.
4. To match the SSL protocol configured with your server, set the property, `com.ibm.ssl.protocol`, accordingly.
5. Specify the com.ibm.CORBA.ConfigURL property with the fully qualified path of your Java property file when you run your application. For example, `-Dcom.ibm.CORBA.ConfigURL=file:/c:/WebSphere/AppServer/profiles/`*profile_name*`/properties/sas.client.props`

***Using the WebSphere Application Server to edit the sas.client.props file:***

To edit the `sas.client.props` file using the administrative console, complete the following steps:

1. Start the administrative console.
2. Expand **Security > Global security**.
3. Under Authentication, click **Authentication protocol > CSIv2 inbound authentication**.
4. Select **Supported** or **Required** for Client certificate authentication.
5. Click **OK**.

6. If you selected **Required** in step 4, configure the CSIv2 outbound authentication as well to support the client certificate authentication. Otherwise, you can skip this step. Return to the Global security panel and under Authentication, click **CSIv2 Outbound Authentication**. Select either **Supported** or **Required** for Client certificate authentication.
7. Click **CSIv2 Outbound Transport**.
8. Select an SSL setting from the SSLSettings list for keystore, truststore, cryptographic token, SSL protocol, and ciphers use.
9. Create an alias from the SSL Configuration Repertoires panel for an SSL setting.
10. Update the SSL setting selected in CSIv2 Inbound Transport accordingly.
11. Save your configuration.
12. Restart the server for the changes to become effective.

Client authentication using digital certificates is performed during SSL connection. A secure client connects using SSL to a secure Internet InterORB Protocol (IIOP) server with client authentication at the transport layer.

Specify the keystore and truststore files in your configuration.

## Adding keystore files

A keystore file contains both public keys and private keys. Public keys are stored as signer certificates while private keys are stored in the personal certificates. In WebSphere Application Server, adding keystore files to the configuration is different between client and server. For the client, a keystore file is added to a property file like `sas.client.props`. For the server, a keystore file is added through the WebSphere Application Server administrative console.

Before you add the keystore file to your configuration, consider the following questions:
- Is a self-signed or a certificate authority (CA)-signed personal certificate created in the keystore file?
- If you configure client authentication using digital certificates, is the public key of the signed personal certificate imported as a signer certificate into the server truststore file?

1. Add a keystore file into a client configuration by editing the `sas.client.props` file and setting the following properties:
   - **com.ibm.ssl.keyStoreType** for the keystore format. Range: JKS (default), PKCS12KS, JCEK
   - **com.ibm.ssl.keyStore** for a fully qualified path to the keystore file. The keystore file contains private keys and sometimes public keys.
   - **com.ibm.ssl.keyStorePassword** for the password to access the keystore file.
2. Add a keystore file into a server configuration:
   a. Start the administrative console by specifying: http://*server_hostname*:9060/ibm/console.
   b. Click **Security > SSL Configuration Repertoires**.
   c. Create a new Secure Sockets Layer (SSL) setting alias if one does not exist.
   d. Select the alias that you want to add into the keystore file.
   e. Type in the key file name for the path of the keystore file.
   f. Type in the key file password for the password to access the keystore file.
   g. Select the key file format for the keystore type. Range: JKS (default), PKCS12KS, or JCEK.
   h. Click **OK** and **Save** to save the configuration.

The SSL configuration alias now has a valid keystore file for an SSL connection.

**Note:** If the Cryptographic token field is selected and you only want to use cryptographic tokens for your keystore file, leave the Key file name field and the Key file password field blank.
- SSL connection for Internet InterORB Protocol (IIOP)
- SSL connection for Lightweight Directory Access Protocol (LDAP)
- SSL connection for Hypertext Transfer Protocol (HTTP)

## Adding truststore files

A *truststore file* is a key database file that contains public keys. The public key is stored as a signer certificate. The keys are used for a variety of purposes, including authentication and data integrity. In WebSphere Application Server, adding truststore files to the configuration is different between client and server. For the client, a truststore file is added to a property file, like `sas.client.props`. For the server, a truststore file is added through the WebSphere Application Server administrative console.

Before you add the truststore file to your configuration, ask the following questions:
- If you configure for client authentication using digital certificate, has the public key of the client personal certificate been imported as a signer certificate into the server truststore file?
- Does the truststore file contain all the required signer certificates with respect to the keystore files of the target servers?

1. Add a truststore file into a client configuration, by editing the `sas.client.props` file and setting the following properties:
   - **com.ibm.ssl.trustStoreType** for the truststore format. Range: JKS (default), PKCS12KS, JCEK, JCERACFKS.
   - **com.ibm.ssl.trustStore** for a fully qualified path to the truststore file. The truststore file contains the public keys.
   - **com.ibm.ssl.trustStorePassword** for the password to access the truststore file.
2. Add a truststore file into a server configuration:
   a. Start the administrative console by specifying : http://*server_host_name*:9060/ibm/console
   b. Click **Security > SSL**.
   c. Create a new Secure Sockets Layer (SSL) setting alias if one does not exist.
   d. Select the alias that you want to add into the truststore file.
   e. Type the trust file name for the path of the truststore file.
   f. Type the trust file password for the password to access the truststore file.
   g. Select the trust file format for the truststore type. JKS (Default), PKCS12KS, JCEK.
   h. Click **OK** and **Save** to save the configuration.

The SSL configuration alias now contains a valid truststore file for an SSL connection.
- SSL connection for Internet InterORB Protocol (IIOP)
- SSL connection for Lightweight Directory Access Protocol (LDAP)
- SSL connection for Hypertext Transfer Protocol (HTTP)

# Secure Sockets Layer configuration repertoire settings

Use this page to define a new Secure Sockets Layer (SSL) alias. Using the SSL configuration repertoire, administrators can define any number of SSL settings to use in configuring the Hypertext Transfer Protocol with SSL (HTTPS), Internet InterORB Protocol with SSL (IIOPS) or Lightweight Directory Access Protocol with SSL (LDAPS) connections. You can pick one of the SSL settings defined here from any location within the administrative console that supports SSL connections. This flexibility simplifies the SSL configuration process because you can reuse many of these SSL configurations by specifying the alias in multiple places.

To view this administrative console page, click **Security** > **SSL**.

Click **New** to create a new SSL Configuration Repertoire alias.

Click **Delete** to remove an SSL Configuration Repertoire alias. If an SSL configuration alias is referenced in the configuration and is deleted here, then an SSL connection fails when the deleted alias is accessed.

## Alias

Specifies the name of the specific SSL setting.

***Type:***

Specifies the type of repertoire configured for the alias listed.

The value is either SSSL for System Secure Sockets Layer repertoire or JSSE for Java Secure Sockets Extension repertoire.

## New Secure Sockets Layer repertoire

Use this page to specify the list of defined Secure Sockets Layer (SSL) configurations.

To view this administrative console page, click **Security** > **SSL** > **New JSSE repertoire**.

***Alias:***

Specifies the name of the specific SSL setting.

| | |
|---|---|
| **Data type:** | String |

This field is used on the System SSL Repertoire and Java Secure Sockets Extension (JSSE) Repertoire panels.

***Key file name:***

Specifies the fully qualified path to the SSL key file that contains public keys and private keys.

| | |
|---|---|
| **Data type:** | String |

For JSSE SSL, the key file specifies the keystore file. The key file might also specify the System Authorization Facility (SAF) key ring that contains certificates and keys. You can create a JSSE SSL keystore file by using the keytool utility found in the WebSphere Application Server `bin`directory. The key file contains certificates and keys.

For System SSL or JSSE, you can create an SSL key ring by using the Resource Access Control Facility (RACF) command, RACDCERT. Issue this command in your MVS environment, such as TSO READY or ISPF option 6. The key ring contains the private certificate of this server and certificates of trusted certificate authorities. The certificates for the trusted certificate authorities validate the client certificates and other server certificates that are exchanged with this server during the SSL handshake. The repertoires that you define for a server require identical key file names.

***Client authentication:***

Specifies whether to request a certificate from the client for authentication purposes when making a connection.

| | |
|---|---|
| **Data type:** | Boolean |
| **Default:** | Disabled |
| **Range:** | Enabled or Disabled |

When performing client authentication with the Internet InterORB Protocol (IIOP) for Enterprise JavaBeans (EJB) requests, click **Security** > **Global Security** > **CSIv2 Inbound or Outbound Authentication** from

the left navigation pane of the administrative console. (You can also click **Servers** > **Application Servers** > *server_name* > **CSIv2 Inbound or Outbound Authentication**.) Click **SSL Client Certificate Authentication** to enable it for these requests.

***Security level:***

Specifies whether the server selects from a preconfigured set of security levels.

**Data type:** Valid values include Low, Medium or High.

- Low specifies only digital signing ciphers (no encryption)
- Medium specifies only 40-bit ciphers (including digital signing)
- High specifies only 128-bit ciphers (including digital signing).

To specify all ciphers or any particular range, you can set the `com.ibm.ssl.enabledCipherSuites` property.

See the SSL documentation for more information.

**Default:** High

**Range:** Low, Medium, or High

***V3 timeout:***

Specifies the length of time that a browser can reuse a System SSL Version 3 session ID without renegotiating encryption keys with the server.

The repertoires that you define for a server require the same V3 timeout value.

| | |
|---|---|
| **Data type** | integer |
| **Default** | 100 |
| **Range** | 1 to 86400 |

***Cipher suites:***

Specifies a list of supported cipher suites that can be selected during the SSL handshake. If you select cipher suites individually here, you override the cipher suites set in the Security Level field.

**Data type:**
**Default:**
**Range:**

***Transport channel name:***

This name must be unique across all channels in a WebSphere Application Server environment. TCP transport channels and HTTP transport channels cannot have the same name if they reside within the same system.

**Data type:**
**Default:**
**Range:**

## Repertoire settings

Use this page to configure Secure Sockets Layer (SSL) or Java Secure Sockets Extension (JSSE) settings for the server. To configure Secure Sockets Layer (SSL), you need to define an SSL configuration repertoire. A repertoire contains the details necessary for building an SSL connection, such as the location of the key files, their type and the available ciphers. WebSphere Application Server provides a default repertoire called DefaultSSLSettings.

To view this administrative console page, click **Security > SSL >***alias_name*.

***Alias:***

Specifies the name of the specific SSL setting

| | |
|---|---|
| **Data type:** | String |

***Client authentication:***

Specifies whether to request a certificate from the client for authentication purposes when making a connection.

This attribute is only valid when it is used by the Web container HTTP transport.

When performing client authentication with the Internet InterORB Protocol (IIOP) for EJB requests, click **Security > Global security**. Under Authentication, click **Authentication protocol > CSIv2 inbound authentication** or **Authentication protocol > CSIv2 outbound authentication**. Select the appropriate option under Client certificate authentication.

| | |
|---|---|
| **Default:** | Disabled |
| **Range:** | Enabled or Disabled |

***Security level:***

Specifies whether the server selects from a preconfigured set of security levels.

| | |
|---|---|
| **Data type:** | Valid values include Low, Medium or High.<br>• Low specifies digital signing ciphers only without encryption.<br>• Medium specifies 40-bit ciphers only including digital signing.<br>• High specifies 128-bit ciphers only including digital signing.<br><br>To specify all ciphers or any particular range, you can set the com.ibm.ssl.enabledCipherSuites property.<br><br>See the SSL documentation for more information. |
| **Default:** | High |
| **Range:** | Low, Medium, or High |

***Cipher suites:***

Specifies a list of supported cipher suites that can be selected during the SSL handshake. If you select cipher suites individually here, you override the cipher suites set in the Security Level field.

***Cryptographic token:***

Specifies whether the server enables or disables cryptographic hardware and software support. The SOAP connector does not use hardware cryptography.

| | |
|---|---|
| **Data type:** | Boolean |
| **Default:** | Disabled |
| **Range:** | Enabled or Disabled |

***Provider:***

Refers to a package that implements a subset of the Java security application programming interface (API) cryptography aspects.

If you select **Predefined JSSE provider**, select a provider from the menu.

WebSphere Application Server has the IBMJSSE, IBMJSSE2, and the IBMJSSEFIPS predefined providers. IBMJSSEFIPS is the the IBMJSSE provider that is Federal Information Processing Standard (FIPS) certified. If you select **Custom JSSE provider** , enter a custom provider. For a custom provider, you first must enter the cipher suites through Custom properties under Additional Properties. Cipher suites and protocol values depend on the provider.

**Note:** You can only specify the IBMJSSE2 provider for transports using the channel framework, including HTTP and JMS. Any other provider specified causes the server to fail initialization. FIPS is not supported for these transports in WebSphere Application Server Version 6.

***Protocol:***

Specifies which SSL protocol to use.

If you are using a FIPS-approved JSSE such as IBMJSSEFIPS, you must select a TLS protocol. However, because the FIPS-approved JSSE providers are not backwards-compatible, a server that uses the TLS protocol cannot communicate with a client that uses an SSL protocol.

| | |
|---|---|
| **Default** | SSLv3 |
| **Range** | SSL, SSLv2, SSLv3, TLS, TLSv1 |

***Key file name:***

Specifies the fully qualified path to the SSL key file that contains public keys and might contain private keys.

You can create an SSL key file with the key management utility, or this file can correspond to a hardware device if one is available. In either case, this option indicates the source for personal certificates and for signer certificates unless a trust file is specified. The default SSL key files, `DummyClientKeyFile.jks` and `DummyServerKeyFile.jks`, contais a self-signed personal test certificate expiring on March 17, 2005. The test certificate is only intended for use in a test environment. The default SSL key files should never be used in a production environment because the private keys are the same on all the WebSphere Application Server installations. Refer to the Managing certificates article for information about creating and managing digital certificates for your WebSphere Application Server domain.

| | |
|---|---|
| **Data type:** | String |

***Key file password:***

Specifies the password for accessing the SSL key file.

| | |
|---|---|
| **Data type:** | String |

#### *Key file format:*

Specifies the format of the SSL key file.

You can choose from the following key file formats: JKS, JCEK, PKCS12. The JKS format does not store a shared key. For more secure key files, use the JCEK format. PKCS12 is the standard file format.

| | |
|---|---|
| **Data type:** | String |
| **Default:** | JKS |
| **Range:** | JKS, PKCS12, JCEK |

#### *Trust file name:*

Specifies the fully qualified path to a trust file containing the public keys.

You can create a trust file with the key management utility included in the WebSphere *bin* directory. Using the key management utility from Global Security Kit (GSKit) (another SSL implementation) does not work with the Java Secure Socket Extension (JSSE) implementation.

Unlike the SSL key file, no personal certificates are referenced; only signer certificates are retrieved. The default SSL trust files, `DummyClientTrustFile.jks` and `DummyServerTrustFile.jks`, contain multiple test public keys as signer certificates that can expire. The public key for the WebSphere Application Server Version 4.0 test certificates expires on January 15, 2004, and the public key for the WebSphere Application Server Version 5 test certificates and WebSphere Application Server CORBA C++ client expires on March 17, 2005. The test certificate is only intended for use in a test environment.

The public key for the WebSphere Application Server Version 6 test certificates expires on October 13, 2021.

If a trust file is not specified but the SSL key file is specified, then the SSL key file is used for retrieval of signer certificates as well as personal certificates.

| | |
|---|---|
| **Data type:** | String |

#### *Trust file password:*

Specifies the password for accessing the SSL trust file.

| | |
|---|---|
| **Data type:** | String |

#### *Trust file format:*

Specifies the format of the SSL trust file.

You can choose from the following trust file formats: JKS, JCEK, PKCS12. The JKS format does not store a shared key. For more secure key files, use the JCEK format. PKCS12 is the standard file format.

| | |
|---|---|
| **Data type:** | String |
| **Default:** | JKS |
| **Range:** | JKS, JCEK, PKCS12 |

## Secure Sockets Layer settings for custom properties

Use this page to configure additional Secure Sockets Layer (SSL) settings for a defined alias.

To view this administrative console page, click **Security** > **SSL** > *alias_name* > **Custom properties**.

***Custom Properties:***

Specifies the name-value pairs that you can use to configure additional SSL settings beyond those available in the `com.ibm.ssl.protocol` administrative interface.

This value is the SSL protocol used (including its version). The possible values are SSL, SSLv2, SSLv3, TLS, or TLSv1. The default value, SSL, is backward-compatible with the other SSL protocols.

**com.ibm.ssl.keyStoreProvider**
The name of the key store provider to use. Specify one of the security providers listed in your `java.security` file, which has a keystore implementation. The default value is IBMJCE.

**com.ibm.ssl.keyManager**
The name of the key management algorithm to use. Specify any key management algorithm that is implemented by one of the security providers listed in your `java.security` file. The default value is IbmX509.

**com.ibm.ssl.trustStoreProvider**
The name of the trust store provider to use. Specify one of the security providers listed in your `java.security` file, which has a truststore implementation. The default value is IBMJCE.

**com.ibm.ssl.trustManager**
The name of the trust management algorithm to use. Specify any trust management algorithm that is implemented by one of the security providers listed in your `java.security` file. The default value is IbmX509.

**com.ibm.ssl.trustStoreType**
The type or format of the truststore file. The possible values are JKS, PKCS12, JCEK. The default value is JKS.

**com.ibm.ssl.enabledCipherSuites**
The list of cipher suites to enable. By default, this is not set and the set of cipher suites used is determined by the value of the security level (high, medium, or low). A cipher suite is a combination of cryptographic algorithms used for an SSL connection. Enter a space-separated list of any of the following cipher suites:

- SSL_RSA_WITH_RC4_128_MD5
- SSL_RSA_WITH_RC4_128_SHA
- SSL_RSA_WITH_DES_CBC_SHA
- SSL_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_RSA_WITH_DES_CBC_SHA
- SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_DSS_WITH_DES_CBC_SHA
- SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_EXPORT_WITH_RC4_40_MD5
- SSL_RSA_EXPORT_WITH_DES40_CBC_SHA
- SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5
- SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
- SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
- SSL_RSA_WITH_NULL_MD5
- SSL_RSA_WITH_NULL_SHA
- SSL_DH_anon_WITH_RC4_128_MD5
- SSL_DH_anon_WITH_DES_CBC_SHA
- SSL_DH_anon_WITH_3DES_EDE_CBC_SHA
- SSL_DH_anon_EXPORT_WITH_RC4_40_MD5
- SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA

| **Data type:** | String |
|---|---|

***Cryptographic token:***

Specifies information about the cryptographic tokens related to SSL support.

A cryptographic token is a hardware or software device that has a built-in keystore implementation. Document the exact values for the following fields found in the literature of your supported cryptographic device.

## Creating a Secure Sockets Layer repertoire configuration entry

The first step in configuring Secure Sockets Layer (SSL) is to define an SSL configuration repertoire. A *repertoire* contains the details necessary for building an SSL connection, such as the location of the key files, their type and the available ciphers. WebSphere Application Server provides a default repertoire called DefaultSSLSettings. To view this page in the administrative console, click **Security > SSL** to see the list of SSL repertoire settings.

The appropriate repertoire is referenced during the configuration of a service that sends and receives requests encrypted using SSL, such as the Web and enterprise beans containers. If an SSL configuration alias is referenced elsewhere, but the alias is deleted from the SSL Configuration Repertoires panel, the SSL connection fails if the deleted alias is accessed.

With the SSL configuration repertoire, administrators can define SSL settings to use for making Hypertext Transfer Protocol with SSL (HTTPS), Internet InterORB Protocol with SSL (IIOPS) or Lightweight Directory Access Protocol with SSL (LDAPS) connections. You can pick one of the SSL settings defined here from any location within the administrative console, which supports SSL connections. This selection simplifies the SSL configuration process because you can reuse many of these SSL configurations by specifying the alias in multiple places.

1. From the SSL Configuration Repertoire window, click **New**.
2. Enter the information needed to access the key file.
   a. Type the name of the key file, which must include the fully qualified path to the key file, in the Key File Name field. Type `safkeyring:///` if you are using a RACF key ring for the key file.
   b. Type the password needed to access the key file in the Key File Password field. Type `password` if you are using a RACF key ring for the key store.
   c. Select the format of the key file from the Key File Format menu.
3. Enter the information needed to access the trust file.
   a. Type the name of the trust file, which must include the fully qualified path to the trust file, in the Trust File Name field. Type `safkeyring:///` if you are using a RACF key ring as the trust store.
   b. Type the password needed to access the trust file in the Trust File Password field. Type `password` if you are using a RACF key ring as the trust store.
   c. Select the format of the trust file from the Trust File Format menu.
4. Select the **Client Authentication** option if this configuration supports client authentication. This selection only affects HTTP and LDAP requests.
5. Select the appropriate security level from the Security Level menu. Valid values are low, medium, and high. Low specifies digital signing ciphers only (no encryption), medium specifies 40-bit ciphers only (including digital signing), high specifies 128-bit ciphers only (including digital signing).

   If you are using a Federal Information Processing Standards (FIPS)-supported Java Secure Socket Extension (JSSE), you must select **High** from the Security Level menu.
6. Select a cipher suite from the Cipher Suites menu. Manually add the cipher suite if the preset security level does not define the required cipher. Select the **Cryptographic Token** check box if the RACF key ring contains keys or certificates that were created using the `RACDCERT` command with the ICSF keyword specified.

7. Select the **Cryptographic Token** check box if hardware or software cryptographic support is available.

   See "Configuring to use cryptographic tokens" on page 455 for details regarding cryptographic support.
8. Indicate which JSSE provider you are using by either selecting **IBMJSSE**, **IBMJSSE2** (recommended) or **IBMJSSEFIPS** from the menu, or by typing the name of the provider. WebSphere Application Server includes the IBMJSSE, IBMJSSE2 and IBMJSSEFIPS JSSE providers. Use IBMJSSEFIPS only if you are using the Transport Layer Security (TLS) protocol and not the Secure Sockets Layer (SSL) protocol. See "Configuring Federal Information Processing Standard Java Secure Socket Extension files" for more information.

   If you are not using the predefined providers, configure the custom provider by clicking **Apply**, then **Custom Properties > New** in the Additional Properties section. After the custom provider is configured, return to the SSL Configuration Repertoires window and continue with these instructions.
9. Select an SSL or TLS protocol version.

   If you are using a FIPS-approved JSSE, you must select a TLS protocol version.
10. Click **Apply** to apply the changes.
11. If no errors occur, save the changes to the master configuration and restart the WebSphere Application Server.

    For more information on the FIPS certification process and to check the status of the IBM submission, see the Cryptographic Module Validation Program FIPS 140-1 and FIPS 140-2 Pre-validation List Web site.

You included additional SSL configuration repertoires with the default DefaultSSLSettings repertoire.

The appropriate repertoire is referenced during the configuration of a service that sends and receives requests encrypted using SSL, such as the Web and enterprise bean containers, and Lightweight Directory Access Protocol (LDAP) servers.

For the changes to take effect, restart the server after saving the configuration.

## Configuring Federal Information Processing Standard Java Secure Socket Extension files

In WebSphere Application Server Version 6, the Java Secure Socket Extension (JSSE) provider used is the IBMJSSE2 provider. This provider delegates encryption and signature functions to the Java Cryptography Extension (JCE) provider. Consequently, IBMJSSE2 does not need to be Federal Information Processing Standard (FIPS)-approved because it does not perform cryptography. However, the JCE provider requires FIPS-approval.

WebSphere Application Server provides a FIPS-approved IBMJCEFIPS provider that IBMJSSE2 can utilize. The IBMJCEFIPS provider shipped in WebSphere Application Server Version 6 supports the following SSL ciphers:
- SSL_RSA_WITH_AES_128_CBC_SHA
- SSL_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_RSA_WITH_AES_128_CBC_SHA
- SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_DSS_WITH_AES_128_CBC_SHA
- SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA
- SSL_DH_anon_WITH_AES_128_CBC_SHA
- SSL_DH_anon_WITH_3DES_EDE_CBC_SHA

Even though the IBMJSSEFIPS provider is still present, the runtime does not use this provider. If IBMJSSEFIPS is specified as a contextProvider, WebSphere Application Server automatically defaults to the IBMJSSE2 provider (with the IBMJCEFIPS provider) for supporting FIPS in Version 6. When enabling FIPS in the server Global Security Panel, the runtime always uses IBMJSSE2. despite whatever contextProvider you specify for SSL (IBMJSSE, IBMJSSE2 or IBMJSSEFIPS). Also, because FIPS requires the SSL protocol be TLS, the runtime always uses TLS when FIPS is enabled regardless of the SSL protocol setting in the SSL repertoire. This simplifies the FIPS configuration in Version 6 because an administrator only needs to enable the FIPS flag in the Global Security Panel to enable all transports using SSL.

1. Click **Security > Global Security**. Select the Use the Federal Information Processing Standard (FIPS) option and click **OK**. IBMJSSE2 and IBMJCEFIPS is enabled.
2. If you have a Java client that must access enterprise beans, modify the `install_dir/profiles/profile_name/properties/sas.client.props` file and set the property:
   ```
   #com.ibm.security.useFIPS=false
   #com.ibm.security.useFIPS=true
   ```
3. If you have an administrative client using the Simple Object Access Protocol (SOAP) connector, modify the `install_dir/profiles/profile_name/properties/soap.client.props` file on the administrative client and set the following property:
   ```
   #com.ibm.ssl.contextProvider=IBMJSSE2
   com.ibm.ssl.contextProvider=IBMJSSEFIPS
   ```

   **Note:** Note: Specifying IBMJSSEFIPS indicates that the client wants to be in FIPS mode, and the runtime uses the IBMJSSE2 provider in combination with the IBMJCEFIPS provider.

After completing these steps, a FIPS-approved JSSE or JCE provider offers increased encryption capabilities. However, when you use FIPS-approved providers:

- By default, Microsoft Internet Explorer Version 5.5 might not have Transport Layer Security (TLS) enabled. To enable TLS, open the Internet Explorer browser and click **Tools > Internet Options**. On the Advanced tab, select the Use TLS 1.0 option.

  **Note:** Netscape Version 4.7.x and earlier versions might not support TLS.
- IBM Directory Server Version 5.1 (and earlier versions) do not support TLS.
- If you have an administrative client that uses a SOAP connector, and you enable FIPS, add the following lines to the `install_dir/profiles/profile_name/properties/soap.client.props` file:
  ```
  com.ibm.ssl.contextProvider=IBMJSSEFIPS
  ```
- When you select the Use the Federal Information Processing Standard (FIPS) option on the Global Security panel, the Lightweight Third-Party Authentication (LTPA) token format is not backwards-compatible with previous releases of WebSphere Application Server. However, you can continue to use the LTPA keys configured using a previous version of WebSphere Application Server.

**Note:** When enabling FIPS, you cannot configure cryptographic token devices in the SSL repertoires. IBMJSSE2 must use IBMJCEFIPS when utilizing crytpographic services for FIPS.

## Digital certificates

Certificates provide a way of authenticating users. Instead of requiring each participant in an application to authenticate every user, third-party authentication relies on the use of digital certificates.

A digital certificate is equivalent to an electronic ID card. It serves two purposes:
- Establishes the identity of the owner of the certificate
- Distributes the owner's public key

Certificates are issued by trusted parties, called *certificate authorities* (CAs). These authorities can be commercial ventures or they can be local entities, depending on the requirements of your application. Regardless, the CA is trusted to adequately authenticate users before issuing certificates. A CA issues

certificates with digital signatures. When a user presents a certificate, the recipient of the certificate validates it by using the digital signature. If the digital signature validates the certificate, the certificate is recognized as intact and authentic. Participants in an application only need to validate certificates; they do not need to authenticate users. The fact that a user can present a valid certificate proves that the CA has authenticated the user. The descriptor, *trusted third-party*, indicates that the system relies on the trustworthiness of the CAs.

**Contents of a digital certificate**

A certificate contains several pieces of information, including information about the owner of the certificate and the issuing CA. Specifically, a certificate includes:
- The distinguished name (DN) of the owner. A DN is a unique identifier, a fully qualified name including not only the common name (CN) of the owner but the owner's organization and other distinguishing information.
- The public key of the owner.
- The date on which the certificate was issued.
- The date on which the certificate expires.
- The distinguished name of the issuing CA.
- The digital signature of the issuing CA. (The message-digest function is run over all the preceding fields.)

The core idea of a certificate is that a CA takes the owner's public key, signs the public key with its own private key, and returns the information to the owner as a certificate. When the owner distributes the certificate to another party, it signs the certificate with its private key. The receiver can extract the certificate (containing the CA signature) with the owner's public key. By using the CA public key and the CA signature on the extracted certificate, the receiver can validate the CA signature. If it is valid, the public key used to extract the certificate is recognized as good. The owner signature is then validated, and if the validation succeeds, the owner is successfully authenticated to the receiver.

The additional information in a certificate helps an application decide whether to honor the certificate. With the expiration date, the application can determine if the certificate is still valid. With the name of the issuing CA, the application can check that the CA is considered trustworthy by the site.

A process that uses certificates must provide its personal certificate, the one containing its public key, and the certificate of the CA that signed its certificate, called a *signer certificate*. In cases where chains of trust are established, several signer certificates can be involved.

**Requesting certificates**

To get a certificate, send a certificate request to the CA. The certificate request includes:
- The distinguished name of the owner (the user for whom the certificate is requested).
- The public key of the owner.
- The digital signature of the owner.

The message-digest function is run over all these fields.

The CA verifies the signature with the public key in the request to ensure that the request is intact and authentic. The CA then authenticates the owner. Exactly what the authentication consists of depends on a prior agreement between the CA and the requesting organization. If the owner in the request is successfully authenticated, the CA issues a certificate for that owner.

**Using certificates: Chain of trust and self-signed certificate**

To verify the digital signature on a certificate, you must have the public key of the issuing CA. Because public keys are distributed in certificates, you must have a certificate for the issuing CA that is signed by the issuer. One CA can certify other CAs, so a chain of CAs can issue certificates for other CAs, all of

whose public keys you need. Eventually, you reach a root CA that issues itself a self-signed certificate. To validate a user's certificate, you need certificates for all intervening participants, back to the root CA. Then you have the public keys you need to validate each certificate, including the user's.

A self-signed certificate contains the public key of the issuer and is signed with the private key. The digital signature is validated like any other, and if the certificate is valid, the public key it contains is used to check the validity of other certificates issued by the CA. However, anyone can generate a self-signed certificate. In fact, you can probably generate self-signed certificates for testing purposes before installing production certificates. The fact that a self-signed certificate contains a valid public key does not mean that the issuer is really a trusted certificate authority. To ensure that self-signed certificates are generated by trusted CAs, such certificates must be distributed by secure means (hand-delivered on floppy disks, downloaded from secure sites, and so on).

Applications that use certificates store these certificates in a *keystore* file. This file typically contains the necessary personal certificates, its signing certificates, and its private key. The private key is used by the application to create digital signatures. Servers always have personal certificates in their keystore files. A client requires a personal certificate only if the client must authenticate to the server when mutual authentication is enabled.

To allow a client to authenticate to a server, a server keystore file contains the private key and the certificate of the server and the certificates of its CA. A client truststore file must contain the signer certificates of the CAs of each server to which the client must authenticate.

If mutual authentication is needed, the client keystore file must contain the client private key and certificate. The server truststore file requires a copy of the certificate of the client CA.

## Digital signatures

A *digital signature* is a number attached to a document. For example, in an authentication system that uses public-key encryption, digital signatures are used to sign certificates.

This signature establishes the following information:

- The integrity of the message: Is the message intact? That is, has the message been modified between the time it was digitally signed and now?
- The identity of the signer of the message: Is the message authentic? That is, was the message actually signed by the user who claims to have signed it?

A digital signature is created in two steps. The first step distills the document into a large number. This number is the *digest code* or *fingerprint*. The digest code is then encrypted, resulting in the digital signature. The digital signature is appended to the document from which the digest code was generated.

Several options are available for generating the digest code. WebSphere Application Server supports the MD5 message digest function and the SHA1 secure hash algorithm, but these procedures reduce a message to a number. This process is not encryption, but a sophisticated checksum. The message cannot regenerate from the resulting digest code. The crucial aspect of distilling the document to a number is that if the message changes, even in a trivial way, a different digest code results. When the recipient gets a message and verifies the digest code by recomputing it, any changes in the document result in a mismatch between the stated and the computed digest codes.

To stop someone from intercepting a message, changing it, recomputing the digest code, and retransmitting the modified message and code, you need a way to verify the digest code as well. To verify the digest code, reverse the use of the public and private keys. For private communication, it makes no sense to encrypt messages with your private key; these keys can be decrypted by anyone with your public key. This technique can be useful for proving that a message came from you. No one can create it because no one else has your private key. If some meaningful message results from decrypting a document by using someone's public key, the decryption process verifies that the holder of the corresponding private key did encrypt the message.

The second step in creating a digital signature takes advantage of this reverse application of public and private keys. After a digest code is computed for a document, the digest code is encrypted with the sender's private key. The result is the digital signature, which is attached to the end of the message.

When the message is received, the recipient follows these steps to verify the signature:
1. Recomputes the digest code for the message.
2. Decrypts the signature by using the sender's public key. This decryption yields the original digest code for the message.
3. Compares the original and recomputed digest codes. If these codes match, the message is both intact and authentic. If not, something has changed and the message is not to be trusted.

## Public key cryptography

All encryption systems rely on the concept of a key. A key is the basis for a transformation, usually mathematical, of an ordinary message into an unreadable message. For centuries, most encryption systems have relied on what is called private key encryption. Only within the last 30 years has a challenge to private key encryption appeared - public key encryption.

### Private key encryption

Private-key encryption systems use a single key that is shared between the sender and the receiver. Both must have the key; the sender encrypts the message by using the key, and the receiver decrypts the message with the same key. Both must keep the key private to keep their communication private. This kind of encryption has characteristics that make it unsuitable for widespread, general use:
- Private key encryption requires a key for every pair of individuals who need to communicate privately. The necessary number of keys rises dramatically as the number of participants increases.
- The fact that keys must be shared between pairs of communicators means the keys must somehow be distributed to the participants. The need to transmit secret keys makes them vulnerable to theft.
- Participants can communicate only by prior arrangement. There is no way to send a usable encrypted message to someone spontaneously. You and the other participant must make arrangements to communicate by sharing keys.

Private-key encryption is also called *symmetric encryption*, because the same key is used to encrypt and decrypt the message.

### Public key encryption

Public key encryption uses a pair of mathematically related keys. A message encrypted with the first key must be decrypted with the second key, and a message encrypted with the second key must be decrypted with the first key.

Each participant in a public-key system has a pair of keys. The symmetric (private) key is kept secret. The other key is distributed to anyone who wants it; this key is the public key.

To send an encrypted message to you, the sender encrypts the message by using your public key. When you receive the message, you decrypt it by using your symmetric key. To send a message to someone, you encrypt the message by using the recipient's public key. The message can be decrypted with the recipient's symmetric key only. This kind of encryption has characteristics that make it very suitable for general use:
- Public-key encryption requires only two keys per participant. The increase in the total number of keys is less dramatic as the number of participants increases, compared to symmetric key encryption.
- The need for secrecy is more easily met. Only the symmetric key needs to be kept symmetric and because it does not need to be shared, the symmetric key is less vulnerable to theft in transmission than the shared key in a symmetric key system.
- Public keys can be published, which eliminates the need for prior sharing of a secret key before communication. Anyone who knows your public key can use it to send you a message that only you can read.

Public-key encryption is also called *asymmetric encryption*, because the same key cannot be used to encrypt and decrypt the message. Instead, one key of a pair is used to undo the work of the other. WebSphere Application Server uses the Rivest Shamir Adleman (RSA) public and symmetric key encryption algorithm.

With symmetric key encryption, you have to be careful of stolen or intercepted keys. In public-key encryption, where anyone can create a key pair and publish the public key, the challenge is in verifying that the owner of the public key is really the person you think it is. Nothing prevents a user from creating a key pair and publishing the public key under a false name. The listed owner of the public key cannot read messages encrypted with that key because the owner does not have the symmetric key. If the creator of the false public key can intercept these messages, that person can decrypt and read messages intended for someone else. To counteract the potential for forged keys, public-key systems provide mechanisms for validating public keys and other information with digital signatures and digital certificates.

## Managing digital certificates

Secure Sockets Layer (SSL) connections rely on the existence of *digital certificates*. A digital certificate reveals information about its owner, including their identity. During the initialization of an SSL connection, the server must present its certificate to the client for the client to determine the server identity. The client can also present the server with its own certificate for the server to determine the client identity. SSL is therefore, a means of propagating identity between components. Refer to “Configuring Secure Sockets Layer” on page 417 and “Creating a Secure Sockets Layer repertoire configuration entry” on page 437.

A client can trust the contents of a certificate if that certificate is digitally signed by a trusted third party. A Certificate Authority (CA) acts as a trusted third party and signs certificates on the basis of its knowledge of the certificate requestor. Complete the following steps to manage digital certificates using either the key management utility (iKeyman) or the keytool utility:

- Use the supplied key management utility. Refer to “Starting the key management utility (iKeyman)” on page 446. There are two options for creating a new certificate.
  - Request that a CA generates the certificates on your behalf. The CA creates a new certificate, digitally signs it, and delivers it to the requester. Popular Web browsers are preconfigured to trust certificates that are signed by certain CAs. No further client configuration is necessary for a client to connect to the server through an SSL connection. Therefore, CA signed certificates are useful where configuration for each and every client that accesses the server is impractical. Refer to “Requesting certificate authority-signed personal certificates” on page 447, “Creating certificate signing requests” on page 448, “Receiving certificate authority-signed personal certificates” on page 449, and “Extracting public certificates for truststore files” on page 449.
  - Generate a self-signed certificate. This option might be the quickest and require the fewest details to create the certificate. However, the certificate is not signed by a CA. Any client that connects to this server over an SSL connection needs configuration to trust the signer of this certificate. Therefore, self-signed certificates are only useful when you can configure each of the clients to trust the certificate. It is possible in some cases to present a self-signed certificate to an untrusting client. In some Web browsers, when the certificate is received and does not match any of those listed in the client trust file, a prompt appears asking if the certificate should be trusted for the connection and added to the trust file. Refer to “Creating a keystore file” on page 446, “Creating truststore files” on page 450, “Adding keystore files” on page 429, “Adding truststore files” on page 430, “Creating self-signed personal certificates” on page 447, and “Importing signer certificates” on page 451.

  You must configure server-side options. The WebSphere Application Server stores the keystore information in the repository and the keystore files are referred to in the `security.xml` file. Therefore, complete all server-side configuration through the administration console. For Java clients, refer to “Configuring Secure Sockets Layer for Java client authentication” on page 427.
- Use the command line Java utility called *keytool*. With keytool, you can create a private and public self-signed certificate key pair. For this example, the first user is *cn=rocaj*.

1. Specify **RSA** for the private key to ensure that the *MD5 with RSA* signature algorithm is used. Not all Web browsers support the *DSA* cryptograph algorithm, which is the default when RSA is not specified. Set a password of at least six characters to protect the private key. Finally, specify the keystore file and keystore password (the option is storepass):

   ```
   ${WAS_HOME}/java/jre/bin/keytool -genkey -keyalg RSA -dname "cn=rocaj, ou=users,
   u=uk, DC=internetchaos, DC=com" -alias rocaj -keypass websphere -keystore
   testkeyring.jks -storepass websphere
   ```

   The previous three lines of code belong on one line, but were split onto three lines due to the width of the page.
2. Create the second private and public self-signed certificate key pair in the same manner for the user *cn=amorv*.

   ```
   ${WAS_HOME}/java/jre/bin/keytool -genkey -keyalg RSA -dname "cn=amorv, ou=users,
   ou=uk, DC=internetchaos, DC=com" -alias amorv -keypass websphere -keystore
   testkeyring.jks -storepass websphere
   ```

   The previous three lines of code belong on one line, but were split onto three lines due to the width of the page.

   Now the keystore *testkeyring.jks* contains two self-signed certificates with the owner being the same as the issuer for each certificate.
3. Verify the integrity and authenticity of the certificates by getting each certificate signed by the certificate authority.
   a. Generate the Certificate Signing Request, CSR-1 (for the first user cn=rocaj).

      ```
      ${WAS_HOME}/java/jre/bin/keytool -v certreq -alias rocaj -file rocajReq.csr
      -keypass websphere -keystore testkeyring.jks -storepass websphere
      ```

      The previous two lines of code belong on one line, but were split onto two lines due to the width of the page.
   b. On UNIX-based platforms, remove the end of line characters (^M) from the certificate signing request. To remove the end of line characters, type the following command:

      ```
      cat rocajReq.csr |tr -d "\r"
      ```
   c. Generate the CSR-2 (for the second user cn=amorv).

      ```
      ${WAS_HOME}/java/jre/bin/keytool -v -certreq -alias amorv  -file amorvReq.csr
      -keypass websphere -keystore testkeyring.jks -storepass websphere
      ```

      The previous two lines of code belong on one line, but were split onto two lines due to the width of the page.
   d. On UNIX-based platforms, remove the end of line characters (^M) from the certificate signing request. To remove the end of line characters, type the following command:

      ```
      cat amoryReq.csr |tr -d "\r"
      ```
4. Use the free Test SSL certificate program offered by Thawte Consulting to sign the Certificate Signing Requests (CSRs) for this example. In each case, select the **Custom Cert** option and set the certificate format to use the default for your kind of certificate. The example also selects the **Generate an X.509v3 Certificate** option and saves the two resulting files as *rocajRes.arm* and *amorvRes.arm*, respectively.
5. Import the CA trusted root certificate into the keystore. Copy and paste the Thawte test root certificate in BASE64-encoded ASCII data format to a file called *ThawteTestCA.arm*. Add the test root CA certificate into the keystore file with the following command:

   ```
   ${WAS_HOME}/java/jre/bin/keytool -import -alias "Thawte Test CA Root" -file ThawteTestCA.arm
   -keystore testkeyring.jks -storepass websphere
   ```

The previous two lines of code belong on one line, but were split onto two lines due to the width of the page.

6. Import the two certificate responses from the CA into the keystore file using the same alias name that was first given to the self-signed certificates. In this example, these alias names are *rocaj* and *amorv* respectively. Using an alternative alias name generates a new signer certificate and not a personal certificate chain.
   – Import the certificate response -1 (for the first user cn=rocaj).

     ```
     ${WAS_HOME}/java/jre/bin/keytool -import -trustcacerts -alias rocaj -file rocajRec.arm
     -keystore testkeyring.jks  -storepass websphere.
     Certificate reply was installed in keystore
     ```

     The previous three lines of code belong on one line, but were split onto three lines due to the width of the page.
   – Import the certificate response -2 (for the second user cn=amorv).

     ```
     ${WAS_HOME}/java/jre/bin/keytool -import -trustcacerts -alias amorv -file amorvRec.arm
     -keystore testkeyring.jks -storepass websphere.
     Certificate  reply was installed in keystore
     ```

     The previous three lines of code belong on one line, but were split onto three lines due to the width of the page.
7. Launch the JSSE ikeyman utility, which supports the PKCS12 format and the private key exporting associated with any certificate (the public key is also exported).
8. Open the *testkeyring.jks* keystore file and select the first certificate from the **Personal Certificates** menu.
9. Click **Export** and name the file, *rocajprivate.p12*. Export the second personal certificate and name it *amorvprivate.p12*.
10. Verify that the same root certificate of the authenticating CA is installed as a trusted authority in the browser.
11. To install either of the personal certificates into Netscape Communicator, click **Communicator > Tools > Security Info > Certificates > Yours**. Use the **Import a Certificate** option.
12. Enter a password or PIN for the communicator certificate database, when you attempt to import the certificate. Enter the password used when first initializing your certificate database. Enter the password protecting the PKCS#12 certificate file, as set when you exported the personal private and public certificate key pair in iKeyman.
13. Click **Verify** to check integrity and validity of the certificate. If you did not install the root CA certificate, your certificate fails the verification.
14. Verify that you modified your Web server to support client side certificate requests.
15. Go to the following URL: `https://`*server_name*`/snoop`; the Web browser prompts you to select a personal certificate when accessing a resource protected by the *SSLClientAuth* directive.
16. Select the HTTPS information displayed by the snoop servlet; you see the certificate SubjectDN matching the following: **Subject: CN=amorv, OU=users, OU=uk, DC=internetchaos, DC=com**.

- Refer to “Creating a Secure Sockets Layer repertoire configuration entry” on page 437 to create a new SSL definition entry for WebSphere Application Server using the administrative console. Once a keystore file is configured, either by creating a self-signed certificate or by creating a certificate request and importing the reply, you can configure WebSphere Application Server to use the certificates. The product uses the certificates to establish a secure connection with a client through SSL.
- Set up the appropriate components to use the newly-defined SSL configuration. To ensure a secure connection, configure some non-WebSphere components, such as a Web server. A digital certificate is created for each component. The WebSphere Application Server owns a certificate and the Web server owns another certificate.

  Refer to “Configuring IBM HTTP Server for Secure Sockets Layer mutual authentication” on page 421.

Setting up SSL communication between the Web browser and WebSphere Application Server. Using digital signatures, you can communicate securely from the Web browser through the Web server to WebSphere Application Server. Once you finish configuring security, perform the following steps to save, synchronize, and restart the servers:
1. Click **Save** in the administrative console to save any modifications to the configuration.
2. Synchronize the configuration with all node agents (Network Deployment only).
3. Once synchronized, stop all servers and restart them.

## Starting the key management utility (iKeyman)

It is recommended to read the documentation located in the http://www.ibm.com/developerworks/java/jdk/security/iKeymanDocs.zip file for further information.

WebSphere Application Server provides a graphical tool, the key management utility (iKeyman), for managing keys and certificates. With the key management utility, you can:
- Create a new key database
- Create a self-signed digital certificate
- Add certificate authority (CA) roots to the key database as a signer certificate
- Request and receive a digital certificate from a CA

To start the key management utility, complete the following steps:
1. Move to the `install_root/bin directory`.
2. Issue one of the following commands:
   - On Windows systems, `ikeyman.bat`
   - On UNIX systems, `ikeyman.sh`

A graphical user interface of the key management utility appears.

## Creating a keystore file

The keystore file is a key database file that contains both public keys and private keys. Public keys are stored as signer certificates while private keys are stored in the personal certificates. The keys are used for a variety of purposes, including authentication and data integrity. You can use both the key management utility (iKeyman) and the keytool utility to create keystore files.

Read the documentation located at http://www.ibm.com/developerworks/java/jdk/security/iKeymanDocs.zip for further information.
1. Start the iKeyman utility, if it is not already running.
2. Open a new key database file by clicking **Key Database File > New** from the menu bar.
3. Select the Key Database Type: JKS (default), PKCS12, and JCEKS. This is the *key file format* (or the value of `com.ibm.ssl.keyStoreType` property in the `sas.client.props` file) when you configure the SSL setting for your application.
4. Type in the file name and location. The full path of this key database file is used as the *key file name* (or the value of the `com.ibm.ssl.keyStore` property in the `sas.client.props` file) when you configure the SSL setting for your application.
5. Click **OK** to continue.
6. Then, type in password to restrict access to the file. This password is used as the *key file password* (or the value of `com.ibm.ssl.keyStorePassword` property in the `sas.client.props` file) when you configure the SSL setting for your application. Do not set an expiration date on the password or save the password to a file; you must then reset the password when it expires or protect the password file. This password is used only to release the information stored by the key management utility during run time.
7. Click **OK** to continue. The tool displays all of the available default signer certificates. These certificates are the public keys of the most common certificate authorities (CAs). You can add, view or delete signer certificates from this panel.

A new SSL keystore file is created.

Prepare keystore files for an SSL connection.

Specify the keystore file in the configuration of WebSphere Application Server. Create a truststore if one does not yet exist.

***Creating self-signed personal certificates:***

A self-signed personal certificate is a temporary digital certificate you issue to yourself, acting as the certificate authority (CA). Creating a self-signed certificate creates a private key and a public key within the key database file. The self-signed certificate is created in a keystore file and it is useful when you develop and test your application. You can also create a self-signed personal certificate from your cryptographic token device.

If you want to create a self-signed certificate for a keystore, you must have already created the keystore file. (Refer to “Creating a keystore file” on page 446 for more information.) You can later extract the public key and add the key as a signer certificate to other truststore files.

Read the documentation in the http://www.ibm.com/developerworks/java/jdk/security/iKeymanDocs.zip file for further information about how to create a self-signed personal certificate within a key database file.

1. Start the key management utility, if it is not already running.
2. Click **Key Database file** > **Open** to select an existing file, or click **Key Database file** > **New** to select a new file. Select ″CMS″ for Key database type, select ″key.kdb″ for file name, and enter a directory for the file location.
3. Click **New Self-Signed** from the tool bar or click **Create** > **New Self-Signed Certificate**.
4. Select the **X509** version and the key size that suits your application.
5. Enter the appropriate information for your self-signed certificate:

   **Key Label**
   Give the certificate a key label, which is used to uniquely identify the certificate within the keystore file. If you have only one certificate in each keystore file, you can assign any value to the label. However, it is good practice to use a unique label related to the server name.

   **Common Name**
   Enter the common name. This name is the primary, universal identity for the certificate; it should uniquely identify the principal that it represents. In a WebSphere environment, certificates frequently represent server principals, and the common convention is to use common names of the form *host_name* and *server_name*. The common name must be valid in the configured user registry for the secured WebSphere environment.

   **Organization**
   Enter the name of your organization.

   **Optional fields**
   Enter the organization unit (a department or division), location (city), state and province (if applicable), zip code (if applicable), and select the two-letter identifier of the country in which the server belongs. For a self-signed certificate, these fields are optional. However, commercial CAs might require them.

   **Validity period**
   Specify the lifetime of the certificate in days, or accept the default.
6. Click **OK**.

Your key database file now contains a self-signed personal certificate.

Create a self-signed test certificate for testing purposes. If you need a test certificate signed by a certificate authority, follow the procedure in Creating a certification request.

***Requesting certificate authority-signed personal certificates:***

In a production environment, use a personal certificate signed by a certificate authority (CA). The principal or the owner of the CA-signed personal certificate is authenticated by a CA when the CA signs the principal certificate. Since the certificate authorities (CAs) keep their private keys secure, the signed certificate is more trustworthy than a self-signed certificate. Certificate authorities are entities that issue valid certificates for other entities. Well-known CAs include VeriSign, Entrust, and GTE CyberTrust. You can request a test certificate or a production certificate from some of the CAs like VeriSign.

The authentication process by a CA can take time. Commercial CAs often require up to a week to complete their authentication process. Even on-site CAs can take several minutes, if not hours, or even days, to complete their authentication process. Therefore, you must plan for the certificates that you need.

Considering the following points when you plan for the CA-signed certificate:
- On the certificate signing request that you send to the CA, specify the common name for the certificate. The common name is the primary, universal identity for the certificate. It should uniquely identify the principal that it represents. Verify that the common name is valid in the configured user registry for the WebSphere domain.
- Check the formatting of the address fields that your CA requires when planning the address for a certificate request.

1. Create and send a certificate signing request (CSR) to the CA.
2. Visit the CA Web site and follow the instructions to request a test or production certificate.

Once the request is accepted, the certificate authority verifies your identity and finally issues a signed certificate to you. The certificate is usually sent through e-mail.

Request a production certificate from a trusted CA for the production WebSphere Application Server environment. Once you receive the e-mail from the CA, follow the instructions to store your signed certificate as a file. Receive or store the certificate into the keystore file as a personal certificate.

***Creating certificate signing requests:***

To obtain a certificate from a certificate authority, submit a certificate signing request (CSR) using the key management utility (iKeyman). You can request either production or test certificates from a CA with a CSR. With the key management utility, generating a certificate signing request also generates a private key for the application for which the certificate is requested. The private key remains in the application keystore file, so it stays private. The public key is included in the certificate requested.

For information on how to create a certificate signing request from a key database file, see the http://www.ibm.com/developerworks/java/jdk/security/iKeymanDocs.zip file.

1. Start the key management utility, if it is not already running.
2. Open the key database file from which you want to generate the request.
3. Type the password and click **OK**.
4. Click **Create > New Certificate Request**. The Create New Key and Certificate Request window displays.
5. Type a **Key Label**, a **Common Name**, and **Organization**; and select a Country. For the remaining fields, accept the default value, type a value, or select new values. The common name must be valid in the configured user registry for the secured WebSphere environment.
6. Type in a name for the file, such as `certreq.arm`.
7. Click **OK** to complete.
8. **Optional:** On UNIX-based platforms, remove the end of line characters (^M) from the certificate signing request. To remove the end of line characters, type the following command:

   ```
   cat certreq.arm |tr -d "\r" > new_certreq.arm
   ```

9. Send the certreq.arm file to the certificate authority (CA) following the instructions from the CA Web site for requesting a new certificate.

The Personal Certificate Requests list shows the key label of the new digital certificate request you just created. Send the file to a CA to request a new digital certificate, or cut and paste the request into the request forms of the CA Web site.

You need to request a certificate authority-signed digital certificate for your secure WebSphere domain. Once you submit the certificate signing request, wait for the CA to accept the request. After the CA has verified your identity, it sends back the signed certificate usually through e-mail. Receive the signed certificate back to the keystore file from which you generated the CSR.

***Receiving certificate authority-signed personal certificates:***

Once the certificate signing request (CSR) is accepted, a certificate authority (CA) processes the request and verifies your identity. Once approved, the CA sends the signed certificate back through e-mail. Store the signed certificate in a keystore database file. This procedure describes how to receive the CA-signed certificate into a keystore file using the key management utility (iKeyman). You use this utility the same way for both test certificates and production certificates. The primary difference between the two certificate types is the amount of time it takes for the CA to authenticate the principal your certificate represents. Test certificates are authenticated automatically based on some simple edit checks and returned to you within a few hours. Production certificates may take several days or a week to authenticate and return to you. If the CSR request is made for the cryptographic token, the certificate must be received into that token. If the request is made for the secondary key database of the token, the certificate must be received into that database.

Receive the signed certificate from the CA through e-mail. Follow the instructions from the CA to store the certificate into a file.

Read the http://www.ibm.com/developerworks/java/jdk/security/iKeymanDocs.zip file for further information about how to receive a personal certificate into a key database file from the CA.

1. Start IKeyman, if it is not already running.
2. Open the key database file from which you generated the request.
3. Type the password and click **OK**.
4. Select **Personal Certificates** from the pull-down list.
5. Click **Receive**.
6. Click **Data type** and select the data type of the new digital certificate, such as Base64-encoded ASCII data. Select the data type that matches the CA-signed certificate. If the CA sends the certificate as part of an E-mail message, you may first need to cut and paste the certificate into a separate file.
7. Type the certificate file name and location for the new digital certificate, or click **Browse** to locate the CA-signed certificate.
8. Click **OK**.
9. Type a label for the new digital certificate and click **OK**.

The personal certificate list now displays the label you just gave for the new CA-signed certificate.

Once the CA-signed certificate is successfully received, you can extract or export the public key of the certificate to a file for distribution to the network.

***Extracting public certificates for truststore files:***

Use this procedure to extract a public certificate, which includes its public key, from a keystore file. If a target truststore file already contains the signer certificate of the certificate authority (CA) that signed the

certificate, you do not need to extract and add the certificate to the target truststore file. However, in general, you need to complete this procedure for a self-signed certificate.

Extracting a certificate from one keystore file and adding it to a truststore file is not the same as exporting the certificate and then importing it. Exporting a certificate copies all the certificate information, including its private key, and is normally only used if you want to copy a personal certificate into another keystore file as a personal certificate.

If a certificate is self-signed, extract the certificate and its public key from the keystore file and add it to the target truststore file.

If a certificate is CA-signed, verify that the CA certificate used to sign the certificate is listed as a signer certificate in the target truststore file. The keystore file must already exist and contain the certificate to be extracted.

Read the http://www.ibm.com/developerworks/java/jdk/security/iKeymanDocs.zip file for further information about how to extract a public certificate from a key database file.

1. Start the key management utility (iKeyman), if it is not already running.
2. Open the keystore file from which the public certificate will be extracted.
3. Select **Personal Certificates**.
4. Click **Extract Certificate**.
5. Click **Base64-encoded ASCII data** under Data type.
6. Enter the **Certificate File Name** and **Location**.
7. Click **OK** to export the public certificate into the specified file.

A certificate file that contains the public key of the signed personal certificate is now available for the target truststore file.

Prepare truststore files for distributing the public keys to support the secure WebSphere domain using Secure Sockets Layer (SSL). Once the keystore and truststore files are ready, make them accessible by specifying them in your client and server configurations.

## Creating truststore files

A truststore file is a key database file that contains the public keys for target servers. The public key is stored as a signer certificate. If the target uses a self-signed certificate, extract the public certificate from the server keystore file. Add the extracted certificate into the truststore file as a signer certificate. For a commercial certificate authority (CA), the CA root certificate is added. The truststore file can be a more publicly accessible key database file that contains all the trusted certificates.

Read the documentation located at http://www.ibm.com/developerworks/java/jdk/security/iKeymanDocs.zip for further information.

1. Start the key management utility (iKeyman), if it is not already running.
2. Open a new key database file by clicking **Key Database File > New** from the menu bar.
3. Click the **Key Database Type**: `JKS(Default)`, `PKCS12`, and `JCEKS`. The key database type is the *trust file format* (or the value of the `com.ibm.ssl.trustStoreType` property in the `sas.client.props` file) when you configure the SSL setting for your application.
4. Type in the file name and location. The full path of this key database file is used as the *trust file name* (or the value of `com.ibm.ssl.trustStore` property in the `sas.client.props`) when you configure the SSL setting for your application.
5. Click **OK** to continue.
6. Type in a password to restrict access to the file. This password is used as the *trust file password* (or the value of the `com.ibm.ssl.trustStorePassword` property in the `sas.client.props` file) when you configure the SSL setting for your application. Do not set an expiration date on the password or save

the password to a file. You must reset the password when it expires or protect the password file. This password is used only to release the information stored by the key management utility during run time.

7. Click **OK** to continue. The tool now displays all of the available default signer certificates. These are the public keys of the most common CAs. You can add, view or delete signer certificates from this screen.

A new SSL truststore file is created.

Prepare truststore files for an SSL connection. Specify the truststore file in the configuration of WebSphere Application Server. Create a keystore file if one does not exist.

***Importing signer certificates:***

A *signer certificate* is the trusted certificate entry that is usually in a truststore file. You can import a certificate authority (CA) root certificate from the CA, or a public certificate from the self-signed personal certificate of the target into your truststore file, as a signer certificate.

Read the documentation located in http://www.ibm.com/developerworks/java/jdk/security/iKeymanDocs.zip file for further information.

1. Start the key management utility (iKeyman), if it is not already running.
2. Open the truststore file. The Password Prompt window displays.
3. Type the password and click **OK**.
4. Select **Signer Certificates** from the menu.
5. Click **Add**.
6. Click **Data type** and select a data type, such as Base64-encoded ASCII data. This data type must match the data type of the importing certificate.
7. Type a certificate file name and location for the CA root digital certificate or click **Browse** to select the name and location.
8. Click **OK**.
9. Type a label for the importing certificate.
10. Click **OK**.

The **Signer Certificates** field now displays the label of the signer certificate you just added. Receive a CA root certificate or the public key from your secure target.

## Map certificates to users

Client-side certificates support access to secured resources from Web or Java clients. A client presents an X.509-compliant digital certificate to perform mutual authentication with a single sockets layer-enabled server. The product security run time attempts to map the certificate to a known user in the associated Lightweight Directory Access Protocol (LDAP) directory or the custom registry. If the certificate successfully maps to a user, then the holder of the certificate is regarded as the user in the registry and is authorized as this user. In the case of LocalOS registry, the DN is parsed and the name between the first equals (=) and comma (,) is used as the mapped name. If the DN does not contain the ″=″, the complete name is used. If there is no ″,″ in the DN, everything after the ″=″ is used as the name.

After the single sockets layer-enabled server gets the client certificate, the server needs to map the certificate to a user. WebSphere Application Server supports two techniques for mapping certificates to entries in LDAP registries:

- By exact distinguished name
- By matching attributes in the certificate to attributes of LDAP entries

1. Map by exact distinguished name (DN).

This approach attempts to map the distinguished name (DN) associated with the **Subject** field in the certificate to an entry in the LDAP directory. If the mapping is successful, the user is authenticated and is authorized according to the privileges granted to the identity in the LDAP directory.

The mapping is case insensitive. For example, the following two DNs match on a case-insensitive comparison:

```
"cn=Smith, ou=NewUnit, o=NewCompany, c=us"
"cn=smith, ou=newunit, o=NewCompany, c=US"
```

If a match is found, authentication succeeds; if no match is found, authentication fails.

2. Map by filtering certificate attributes.

   This approach maps certificate attributes to attributes of entries in an LDAP directory. For example, you can specify that the common name (CN) attribute of the **Subject** field in the certificate must match the uid attribute of your LDAP entry. If the mapping is successful, the user is authenticated and is authorized according to the privileges granted to the identity in the LDAP directory.

   If you are matching the Subject CN field in the certificate to the uid attribute of the LDAP entry, a certificate with the Subject DN ″`cn=Smith, ou=NewUnit, o=NewCompany, c=us`″ matches an LDAP user entry with uid=Smith.

   To use this mapping technique, you must request certificate mapping and set up the certificate filter in the administrative console.

This specification extracts the CN field from the Subject attribute in the certificate (`Smith`) and creates a filter (user ID = `Smith`) from it. The LDAP directory is searched for a user entry that matches the filter. If an entry matches the filter, authentication succeeds.

**Note:** The search and match of the LDAP directory are based in part on how your LDAP directory is configured.

## Changes to IBM Developer Kit for Java Technology Edition Version 1.4.x

WebSphere Application Server, Version 5.1 includes the IBM Developer Kit, Java Technology Edition Version 1.4.x, which contains changes to the IBM Developer Kit, Java Technology Edition Version 1.3.x. This document is intended to assist application developers and system administrators in understanding the changes.

### Security packaging changes in IBM Developer Kit, Java Technology Edition Version 1.4.x

In IBM Developer Kit, Java Technology Edition Version 1.4.x, many of the security technologies have been included in the core of the IBM Developer Kit, Java Technology Edition Version 1.4.x. Because of the packaging changes, we are supporting specific `java.security` configurations for each platform. This document discusses the impact these `java.security` configuration changes have on each platform.

### Security providers for the Windows, Linux, and AIX platforms

The Windows, Linux, and AIX platforms use all of the IBM security provider implementations, which is similar to how IBM Developer Kit, Java Technology Edition Version 1.3.x shipped. Because the security technologies in IBM Developer Kit, Java Technology Edition Version 1.3.x, were not part of the core, these technologies were shipped in the `java/jre/lib/ext` directory and provided more flexibility in implementing the technologies. Only those JSSE providers configured by WebSphere Application Server are supported.

The following list shows the providers and sequence of how these providers are supported on the Windows, Linux, and AIX platforms. Add any additional providers at the end of this list of providers. The IBMJSSE, IBMJSSE2 and IBMJSSEFIPS providers are the only SSL providers supported on these platforms. You must configure HTTP and JMS transports to use the IBMJSSE2 providers because they use the channel framework (asynchronous network I/O (NIO) APIs from Java SDK 1.4.2). The NIO APIs only work with the IBMJSSE2 provider and the channel framework.

```
security.provider.1=com.ibm.crypto.provider.IBMJCE
security.provider.2=com.ibm.jsse.IBMJSSEProvider
security.provider.3=com.ibm.security.jgss.IBMJGSSProvider
security.provider.4=com.ibm.security.cert.IBMCertPath
security.provider.5=com.ibm.crypto.pkcs11.provider.IBMPKCS11
```

**Security providers for the Sun Solaris environment**

In the Sun Solaris environment, by default, we are using the IBM JSSE framework classes. These classes enable you to plug-in the IBMJSSE, IBMJSSE2, and IBMJSSEFIPS providers. You must configure HTTP and JMS transports to use the IBMJSSE2 providers because they use the channel framework (asynchronous network I/O (NIO) APIs from Java SDK 1.4.2). The NIO APIs only work with the IBMJSSE2 provider and the channel framework.

The following list shows thee default provider lists for the Sun Solaris environment. Add any additional providers to the end of this list.

```
security.provider.1=com.ibm.security.jgss.IBMJGSSProvider
security.provider.2=sun.security.provider.Sun
security.provider.3=com.ibm.crypto.provider.IBMJCE
security.provider.4=com.ibm.jsse.IBMJSSEProvider
security.provider.5=com.ibm.security.cert.IBMCertPath
# security.provider.6=com.ibm.crypto.pkcs11.provider.IBMPKCS11
```

**Note:** You only need to uncomment the IBMPKCS11 provider when using IKeyMan to access a cryptographic token device. The WAS runtime now uses the IBMPKCS11Impl provider for cryptographic token access, instead of the IBMPKCS11 provider. To get more information on this provider, please see ″Security: Resources for Learning″ for information on this provider.

**Security providers for the HP-UX platform**

In the HP-UX environment, by default, IBM JSSE framework classes are used. These classes enable you to plug-in the IBMJSSE, IBMJSSE2 and IBMJSSEFIPS providers. You must configure HTTP and JMS transports to use the IBMJSSE2 providers because they use the channel framework (asynchronous network I/O (NIO) APIs from Java SDK 1.4.2). The NIO APIs only work with the IBMJSSE2 provider and the channel framework.

```
security.provider.1=com.ibm.security.jgss.IBMJGSSProvider
security.provider.2=sun.security.provider.Sun
security.provider.3=com.ibm.crypto.provider.IBMJCE
security.provider.4=com.ibm.jsse.IBMJSSEProvider
security.provider.5=com.ibm.security.cert.IBMCertPath
# security.provider.6=com.ibm.crypto.pkcs11.provider.IBMPKCS11
```

**Note:** You must uncomment the IBMPKCS11 provider when using IKeyMan to access a cryptographic token device. The WAS runtime now uses the IBMPKCS11Impl provider for cryptographic token access, instead of the IBMPKCS11 provider. To obtain more information about this provider, see the ″Security: Resources for Learning″ article.

**Changes to the CertPath API package name**

In IBM Developer Kit, Java Technology Edition Version 1.3.x, the package for CertPath APIs was javax.security.cert.*. However, in IBM Developer Kit, Java Technology Edition Version 1.4.x, the package has changed to java.security.cert.*. While your applications might still work using javax.security.cert.* due to the oldcertpath.jar packaged in `${WAS_INSTALL_ROOT}/java/jre/lib/ext/oldcertpath.jar` file, change your applications to use the new package name for CertPath from this point forward. In this release, either

package name should work, but it is recommended that you use the correct package, which is java.security.cert.*.

**Known problems with IBM Developer Kit, Java Technology Edition Version 1.4.x**

For a list of known problems with the various platforms related to the IBM Developer Kit, Java Technology Edition Version 1.4.x changes, please review the release notes for WebSphere Application Server, Version 5.1.

There are some known issues with the the IBMJSSE2 provider:

- When configuring a cryptographic token device, you must use the IBMJSSE2 provider. There is a dependency on the new IBMPKCS11Impl provider for cryptographic token support. This provider can only be initialized once in a JVM, and is done programmatically by the WebSphere Application Server runtime when a cryptographic token device is configured. The user of the IBMPKCS11Impl provider in applications is not supported unless the cryptographic token device is not configured for use by WebSphere Application Server.
- FIPS support does not exist for the IBMJSSE2 provider because there currently is not an IBMJCEFIPS that can be used with IBMJSSE2.
- Any transport using the channel framework, including HTTP and JMS, must use the IBMJSSE2 provider.
- Any transport using the channel framework, including HTTP and JMS, must use the IBMJSSE2 provider.
- To use AES_256 ciphers for IBMJSSE2, you must download the JCE Unlimited Strength Jurisdiction Policy.
- IBMJSSE2 provider's HTTPS protocol handler is ″com.ibm.net.ssl.www2.protocol.Handler″. The package to add to the package handler property is ″com.ibm.net.ssl.www2.protocol″.

## Cryptographic token support

A *cryptographic token* is a hardware or software device with a built-in keystore implementation. The cryptographic device is used to manage certificates stored on the cryptographic tokens (also known as *smartcards*).

Both cryptographic accelerators, where the cryptographic hardware device has no persistent key storage, and secure cryptographic hardware, where a cryptographic token generates and securely stores the private key used for Secure Sockets Layer (SSL) key exchange, are supported in the product.

Hardware cryptographic token support has changed providers in Version 6. In Version 5 and before, WebSphere Application Server used com.ibm.crypto.pkcs11.provider.IBMPKCS11 provider for hardware crypto support along with the old IBMJSSE provider for SSL. The IBMPKCS11 provider is still used when accessing hardware using IKeyMan. The IBMJSSE provider can still be used, if necessary, for SSL.

**Note:** To use cryptographic token devices in the Solaris Operating Environment, you must edit the `${WAS_INSTALL_ROOT}/java/jre/lib/security/java.security` file. Uncomment the line containing `com.ibm.crypto.pkcs11.provider.IBMPKCS11`. By default, the line is commented out because the algorithm MD4 is not present in the IBMPKCS11 provider.

The WebSphere Application Server runtime in Version 6 now uses the com.ibm.crypto.pkcs11impl.provider.IBMPKCS11Impl provider for hardware crypto support and the IBMJSSE2 provider for SSL. Both the IBMPKCS11Impl and IBMJSSE2 providers are initialized programmatically. The IBMPKCS11Impl provider is only initialized when hardware crypto is configured in one of the SSL repertoire configurations. Once IBMPKCS11Impl provider is configured, the IBMPKCS11 provider cannot be used in the system since only one provider can initialize a hardware crypto card in the same process.

Please see the following document for more information on the IBMPKCS11Impl provider:
`http://www.ibm.com/developerworks/java/jdk/security/142/pkcs11implDocs.zip`

Please see the following document for more information on the IBMJSSE2 provider:
`http://www.ibm.com/developerworks/java/jdk/security/142/jsse2docs.zip`

## Opening a cryptographic token using the key management utility (iKeyman)

Verify that your cryptographic token device is installed and functions properly. Create a cryptographic token, following the instructions provided by the manual of the cryptographic device.

From your cryptographic token device documentation, identify the token library. For example, the IBM 4758 PCI Cryptographic Card uses CRYPTOKI.DLL as the PKCS#11-type token library (see `http://www.ibm.com/security/cryptocards/html/library.shtml` for details).

Read the documentation located in the http://www.ibm.com/developerworks/java/jdk/security/142/ikmuserguide.pdf file for further information about using the key management utility (iKeyman).

**Important:** To use iKeyMan for key management with a cryptographic token device, you must edit the ${WAS_INSTALL_ROOT}/java/jre/lib/security/java.security file. Uncomment the line containing com.ibm.crypto.pkcs11.provider.IBMPKCS11.

You can use the key management utility to open a cryptographic token. Once opened, you can manage your keys and certificates just like you do with keystore and truststore files:

- Create a self-signed digital certificate
- Extract or add both certificate authority (CA) roots and personal certificate signer certificates
- Request and receive a digital certificate from a CA

1. Start the key management utility, if it is not already running.
2. Click **Key DataBase File > Open**.
3. Click **Cryptographic Token** from the list of key database types.
4. Fill in the information for **File Name** and **Location**, or browse for the cryptographic device library.
5. Click **OK** to open the library.
6. Type in the slot number in the next panel. This is the number of the slot in which you previously created the cryptographic token.
7. Enter the password. This is the password configured for the cryptographic token that you created.

All of the personal and signer certificates are stored on the cryptographic token card. With the token open, you can create or request digital certificates and receive CA-signed certificates.

Use a cryptographic token device as a key database to manage keys and certificates for an SSL connection. Once the cryptographic token is open, you can add or delete keys and certificates. Configure the cryptographic token settings in WebSphere Application Server.

## Configuring to use cryptographic tokens

You can configure cryptographic token support in both client and server configuration. To configure a Java client application, use the `sas.client.props` configuration file. By default, the `sas.client.props` is located in the *install_root*/profiles/*profile_name*/properties/ directory of your WebSphere Application Server installation. To configure WebSphere Application Server, start the administrative console by specifying the following URL: `http://server_hostname:9060/ibm/console`.

To understand how to make WebSphere Application Server (both the run time and the key management utility) work correctly with any cryptographic token device, become familiar with the Java Secure Socket Extension (JSSE) documentation available in the http://www.ibm.com/developerworks/java/jdk/security/142/jsse2docs.zip. and http://www.ibm.com/developerworks/java/jdk/security/142/ikmuserguide.pdf files.

Follow the documentation that accompanies your device to install your cryptographic device. Installation instructions for IBM cryptographic hardware devices can be found in the Administration section of "Security: Resources for learning" on page 21.

**Note:** You cannot use cryptographic token devices when you have Federal Information Processing Standard (FIPS) enabled.

**Important:** To use iKeyMan for key management with a cryptographic token device, you must edit the `${WAS_INSTALL_ROOT}/java/jre/lib/security/java.security` file. Uncomment the line containing `com.ibm.crypto.pkcs11.provider.IBMPKCS11`.

WebSphere Application Server Version 6 and later runtime uses the IBMPKCS11Impl provider instead of the IBMPKCS11 provider for hardware crypto support. See http://www.ibm.com/developerworks/java/jdk/security/142/pkcs11implDocs.zip for more information. Refer to the "IBM Java PKCS 11 Implementation Provider.htm" document located in this zip file.

**Note:** To use cryptographic token devices in the Solaris Operating Environment, you must edit the `${WAS_INSTALL_ROOT}/java/jre/lib/security/java.security` file. Uncomment the line containing `com.ibm.crypto.pkcs11.provider.IBMPKCS11`. By default, the line is commented out because the algorithm MD4 is not present in the IBMPKCS11 provider.

1. To configure a client to use a cryptographic token, edit the `sas.client.props` file and set the following properties. Leave the **KeyStore File Name**, **KeyStore File Password**, **TrustStore File Name**, **TrustStore File Password** fields in a Secure Sockets Layer (SSL) configuration blank (or comment out the properties com.ibm.ssl.trustStore, com.ibm.ssl.trustStorePassword, com.ibm.ssl.keyStore, and com.ibm.ssl.keyStorePassword, using a # in front of the property name) , if you want to use only cryptographic tokens as your keystore.

   **com.ibm.ssl.tokenType**
      Specifies the type of built-in keystore file that is implemented in the cryptographic token. (For example, `com.ibm.ssl.tokenType=PKCS\#11`). The valid values are: **PKCS\#7**, **PKCS\#11**, **PKCS\#12**, and **MSCAPI**.

   **com.ibm.ssl.tokenLibraryFile**
      Specifies the token file name for **PKCS#7** tokens, **PKCS#12** tokens, and the library name for **PKCS#11**, MSCAPI tokens. Make sure the cryptographic token device is installed and functions properly with a cryptographic token created.

   **com.ibm.ssl.tokenPassword**
      Specifies the password to unlock the cryptographic token.

2. Configure your server to use the cryptographic device. Leave the **KeyStore File Name**, **KeyStore File Password**, **TrustStore File Name**, **TrustStore File Password** fields in an SSL configuration blank, if you want to use only cryptographic tokens as your keystore. You can modify an existing configuration if you click **Security > SSL >** *alias*. You must specify an alias and select the **Cryptographic token** option. The following directions explain how to configure WebSphere Application Server for a new cryptographic device.

   a. Specify `http://server_hostname:9060/ibm/console` to start the administrative console.

   b. Click **Security > SSL** to open the SSL Configuration Repertoires panel. You must decide if you want to modify existing SSL repertoire entries to convert them to use hardware cryptographic devices, or create new SSL repertoire entries for the new configuration. The former is easiest as this does not require you to change any of the alias references elsewhere in the configuration. Each protocol picks up the new configuration since it's already referencing these existing aliases.

The latter is a little more difficult as you might not change every location that needs to be referenced by the new aliases. However, you have more control over which protocols actually use the cryptographic token device. If you want a specific protocol to use the cryptographic token device, it is best to create a new SSL repertoire for the cryptographic token device, then associate the alias of the new SSL repertoire with the specific protocol's SSL configuration.

c. Click **New JSSE Repertoire** to create a new SSL setting alias if you do not want to use the default.
d. Specify an alias name in the **alias** field for the new cryptographic device. After you configure the cryptographic device, the alias appears on the Secure Sockets Layer (SSL) configuration repertoires panel. To access the panel, click **Security > SSL**.
e. Select **Cryptographic token** check box and click **OK**. This opens the **Cryptographic token - General Properties** panel.
f. Complete the information for **Token Type** to specify the type of built-in keystore file that is implemented in the cryptographic token. The valid values are: **PKCS#7**, **PKCS#11**, **PKCS#12**,or **MSCAPI**.
g. Complete the information for **Library File** to specify the path to the cryptographic device driver. Make sure the cryptographic token device is installed and functions properly with a new cryptographic token.
h. Complete the information for **Password** to specify the password for unlocking the cryptographic device.
i. Click **OK**. This returns you to the **SSL configuration repertoires - General Properties** panel for this alias.
j. Optionally, to configure a specific Token Slot for the cryptographic token device, click **Custom Properties** from the SSL configuration repertoires - General Properties panel. Add a new property name, **com.ibm.ssl.tokenSlot**, and a property value with the slot number, for example: 0. Optionally, to configure the selection of a specific inbound certificate alias (the alias selected for server transports) within the configured slot, add a new property name, **com.ibm.ssl.keyStoreServerAlias**, with a property value equal to the certificate alias name as it appears when viewing the slot through iKeyMan. Optionally, to configure the selection of a specific outbound certificate alias (the alias selected for client transports) within the configured slot, add a new property name, **com.ibm.ssl.keyStoreClientAlias**, with a property value equal to the certificate alias name as it appears when viewing the slot through iKeyMan, for example. Click **OK** to exit the Custom Properties panel and return to the SSL configuration repertoires - General Properties panel.
k. Make sure the SSL configurations when associated with a transport have the appropriate signers added to the truststore or cryptographic token device so that they can contact all servers for which they are configured. For example, any CSIv2 outbound transport should have signers for all CSIv2 inbound transports that they are connecting to. This means that all CSIv2 inbound keystores (or cryptographic token devices) must have the public key of personal certificates extracted, and added as signers to the CSIv2 outbound truststores (or cryptographic token devices).
l. The following lists the locations of where SSL configuration repertoire aliases are used in the WebSphere Application Server configuration:

   For any transports that use the new NIO channel chains, including HTTP and JMS, you can modify the aliases from the following location for each server:

   - Click **Server > Application server >***server_name*. Under Communications, click **Ports**. Locate a transport chain where SSL is enabled and click **View associated transports**. Click *transport_channel_name*. Under Transport Channels, click **SSL Inbound Channel (SSL_2)**.

   For the Object Request Broker (ORB) SSL transports, you can modify the SSL configuration repertoire aliases in the following locations. These configurations are for the server-level for WebSphere Application Server and WebSphere Application Server Express and the cell level for WebSphere Application Server Network Deployment.

   - Click **Security > Global security**. Under Authentication, click **Authentication protocol > CSIv2 Inbound Transport**.

- Click **Security > Global security**. Under Authentication, click **Authentication protocol > CSIv2 Outbound Transport**.
- Click **Security > Global security**. Under Authentication, click **Authentication protocol > SAS Inbound Transport**.
- Click **Security > Global security**. Under Authentication, click **Authentication protocol > SAS Outbound Transport**.

For the Simple Object Access Protocol (SOAP) Java Management Extensions (JMX) administrative transports, you can modify the SSL configurations repertoire aliases by clicking **Servers > Application servers >***server_name*. Under Server infrastructure, click **Administration > Administration services**. Under Additional properties, click **JMX connectors > SOAPConnector**. Under Additional properties, click **Custom properties**. If you want to point the sslConfig property to a new alias, click **sslConfig** and select an alias in the Value field.

For the Lightweight Directory Access Protocol (LDAP) SSL transport, you can modify the SSL configuration repertoire aliases by clicking **Security > Global security**. Under User registries, click **LDAP**.

m. Finish configuring the SSL settings for this alias. When using hardware cryptographic tokens, you must use a JSSE provider of type IBMJSSE2. The IBMPKCS11Impl provider only works with the IBMJSSE2 provider.

3. Now that you have the aliases configured in the **SSL configuration repertoires** panel, you must associate the aliases with each protocol that needs to use them. If you edited existing aliases, you do not need to make any changes since they are already associated with SSL protocols. However, if you created new aliases and want to rearrange this existing alias association, then proceed to the next step.
4. Repeat steps a. through l. to edit existing or create new SSL configuration repertoires for creating a cryptographic token configuration for use by the IBMJSSE2 provider.
5. Click **OK** to complete the editing of the SSL configuration repertoire for this alias.

The WebSphere Application Server configuration is configured to take advantage of a cryptographic token device for cryptographic functions used by SSL This can improve the system performance over software encryption when SSL is used to protect your data transferred over the network.

WebSphere Application Server uses the cryptographic token as a keystore file for and SSL connection.

If the server configuration has changed, restart the configured server.

## Cryptographic token settings

Use this page to configure cryptographic token settings. A cryptographic token is a hardware or software device with a built-in key store implementation. The cryptographic device is used to manage certificates stored on the cryptographic tokens. These devices are also known as smartcards.

The following types of cryptographic accelerators are supported by WebSphere Application Server:

- A cryptographic hardware device that is without a persistent key storage.
- Secure cryptographic hardware, where a cryptographic token generates and securely stores the private key used for Secure Sockets Layer (SSL) key exchange.

To view this administrative console page, click **Security > SSL >** *alias_name*. Under Additional Properties, click **Cryptographic token**.

### Token type

Specifies the type of built-in keystore file that is implemented in the cryptographic token, such as PKCS#11.

The WebSphere Application Server uses an implementation of Java Secure Socket Extension (JSSE) to support cryptographic token with Secure Sockets Layer (SSL). Different cryptographic devices are supported. For an SSL server, the following devices are supported:
- IBM 4758-23
- nCipher nForce
- Rainbow Cryptoswift

For an SSL client, the following devices are supported:
- IBM 4758-23
- nCipher nForce
- Rainbow Cryptoswift
- IBM Security Kit Smartcard
- GemPlus Smartcards
- Rainbow iKey 1000/2000 (USB ″Smartcard″ device)
- Eracom CSA8000

Follow the documentation that accompanies your device to install your cryptographic token.

| **Data type:** | String |
|---|---|

### Library file

Specifies the dynamic link library (DLL) or shared object that implements the interface to the cryptographic token device.

| **Data type:** | String |
|---|---|

### Password

Specifies the password for the cryptographic token device.

| **Data type:** | String |
|---|---|

## Using Java Secure Socket Extension and Java Cryptography Extension with Servlets and enterprise bean files

**Java Secure Socket Extension**

Java Secure Socket Extension (JSSE) provides the transport security for WebSphere Application Server. It provides application programming interface (API) framework and the implementation of the APIs, for Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols, including functionality for data encryption, message integrity and authentication.

JSSE APIs are integrated into the Java 2 SDK, Standard Edition (J2SDK), Version 1.4. The API package for JSSE APIs is javax.net.ssl.*. Documentation for using JSSE APIs can be found in the J2SE 1.4.2 JavaDoc located at http://java.sun.com/j2se/1.4.2/docs/api/index.html.

Several JSSE providers ship with the J2SDK 1.4 that comes with WebSphere Application Server. The IBMJSSE provider is used in previous WebSphere releases. Associated with the IBMJSSE provider is the IBMJSSEFIPS provider, which is used when FIPS is enabled on the server. Both of these providers do not work with the JMS and HTTP transports in WebSphere Application Server Version 6. These transports take advantage of the J2SDK 1.4 network input/output (NIO) asynchronous channels.

The HTTP and JMS transports use a new IBMJSSE2 provider. All other transports in WebSphere Application Server Version 6 currently use the IBMJSSE2 provider, but can be switched to the old

IBMJSSE provider, if necessary (specified in the SSL repertoire configuraiton). You can specify the IBMJSSEFIPS provider for all transports with the exception of JMS and HTTP.

For more information on the new IBMJSSE2 provider, please review the documentation located in http://www.ibm.com/developerworks/java/jdk/security/142/jsse2docs.zip. Once unzipped, the JSSE2 Reference Guide can be found at jsse2Docs/JSSE2RefGuide.html, the JSSE2 API documentation can be found at jsse2Docs/api/index.html and finally, the JSSE2 samples can be found at jsse2Docs/samples.

### Customizing Java Secure Socket Extension

You can customize a number of aspects of JSSE by plugging in different implementations of Cryptography Package Provider, X509Certificate and HTTPS protocols, or specifying different default keystore files, key manager factories and trust manager factories. A provided table summarizes which aspects can be customized, what the defaults are, and which mechanisms are used to provide customization. Some of the key customizable aspects follow:

| Customizable item | Default | How to customize |
| --- | --- | --- |
| X509Certificate | X509Certificate implementation from IBM | cert.provider.x509v1 security property |
| HTTPS protocol | Implementation from IBM | java.protocol.handler.pkgs system property |
| Cryptography Package Provider | IBMJSSE | A security.provider.n= line in security properties file. See description. |
| Default keystore | None | * javax.net.ssl.keyStore system property |
| Default truststore | jssecacerts, if it exists. Otherwise, cacerts | * javax.net.ssl.trustStore system property |
| Default key manager factory | `IbmX509` | ssl.KeyManagerFactory.algorithm security property |
| Default trust manager factory | `IbmX509` | ssl.TrustManagerFactory.algorithm security property |

For aspects that you can customize by setting a system property, statically set the system property by using the `-D` option of the Java command (you can set the system property using the administrative console), or set the system property dynamically by calling the `java.lang.System.setProperty` method in your code: `System.setProperty(propertyName,"propertyValue")`.

For aspects that you can customize by setting a Java security property, statically specify a security property value in the `java.security` properties file located in the *install_root*`/java/jre/lib/security` directory. The security property is `propertyName=propertyValue`. Dynamically set the Java security property by calling the `java.security.Security.setProperty` method in your code.

### Application Programming Interface

The JSSE provides a standard application programming interface (API) available in packages of the `javax.net` file, `javax.net.ssl` file, and the `javax.security.cert` file. The APIs cover:
- Sockets and SSL sockets
- Factories to create the sockets and SSL sockets
- Secure socket context that acts as a factory for secure socket factories
- Key and trust manager interfaces
- Secure HTTP URL connection classes
- Public key certificate API

You can find more information documented for the JSSE APIs if you download and unzip the http://www.ibm.com/developerworks/java/jdk/security/142/jsse2Docs.zip and look at the jsse2Docs/api/index.html file.

### Samples using Java Secure Socket Extension

The Java Secure Socket Extension (JSSE) also provides samples to demonstrate its functionality. The Java Secure Socket Extension (JSSE) also provides samples to demonstrate its functionality. Download and unzip the samples included in the `http://www.ibm.com/developerworks/java/jdk/security/142/jsse2Docs.zip` file. Look in the `jsse2Docs/samples/` directory for the following files:

| Files | Description |
|---|---|
| `ClientJsse.java` | Demonstrates a simple client and server interaction using JSSE. All enabled cipher suites are used. |
| `OldServerJsse.java` | Back-level samples |
| `ServerPKCS12Jsse.java` | Demonstrates a simple client and server interaction using JSSE with the PKCS12 keystore file. All enabled cipher suites are used. |
| `ClientPKCS12Jsse.java` | Demonstrates a simple client and server interaction using JSSE with the PKCS12 keystore file. All enabled cipher suites are used. |
| `UseHttps.java` | Demonstrates accessing an SSL or non-SSL Web server using the Java protocol handler of the`com.ibm.net.ssl.www.protocol` class. The URL is specified with the `http` or `https` prefix. The HTML returned from this site displays. |

See more instructions in the source code. Follow these instructions before you run the samples.

#### Permissions for Java 2 security

You might need the following permissions to run an application with JSSE: (This is a reference list only.)
- java.util.PropertyPermission ″java.protocol.handler.pkgs″, ″write″
- java.lang.RuntimePermission ″writeFileDescriptor″
- java.lang.RuntimePermission ″readFileDescriptor″
- java.lang.RuntimePermission ″accessClassInPackage.sun.security.x509″
- java.io.FilePermission ″${user.install.root}${/}etc${/}.keystore″, ″read″
- java.io.FilePermission ″${user.install.root}${/}etc${/}.truststore″, ″read″

For the IBMJSSE provider:
- java.security.SecurityPermission ″putProviderProperty.IBMJSSE″
- java.security.SecurityPermission ″insertProvider.IBMJSSE″

For the SUNJSSE provider:
- java.security.SecurityPermission ″putProviderProperty.SunJSSE″
- java.security.SecurityPermission ″insertProvider.SunJSSE″

#### Debugging

By configuring through the `javax.net.debug` system property, JSSE provides the following dynamic debug tracing: `-Djavax.net.debug=true`.

A value of **true** turns on the trace facility, provided that the debug version of JSSE is installed.

#### Documentation

See the “Security: Resources for learning” on page 21 article for documentation references to JSSE.

### JCE

Java Cryptography Extension (JCE) provides cryptographic, key and hash algorithms for WebSphere Application Server. It provides a framework and implementations for encryption, key generation, key agreement, and Message Authentication Code (MAC) algorithms. Support for encryption includes symmetric, asymmetric, block and stream ciphers.

### IBMJCE

The IBM version of the Java Cryptography Extension (IBMJCE) is an implementation of the JCE cryptographic service provider that is used in WebSphere Application Server. The IBMJCE is similar to SunJCE, except that the IBMJCE offers more algorithms:

- Cipher algorithm (AES, DES, TripleDES, PBEs, Blowfish, and so on)
- Signature algorithm (SHA1withRSA, MD5withRSA, SHA1withDSA)
- Message digest algorithm (MD5, MD2, SHA1, SHA-256, SHA-384, SHA-512)
- Message authentication code (HmacSHA1, HmacMD5)
- Key agreement algorithm (DiffieHellman)
- Random number generation algorithm (IBMSecureRandom, SHA1PRNG)
- Key store (JKS, JCEKS, PKCS12)

The IBMJCE belongs to the com.ibm.crypto.provider.* packages.

For further information, see the `http://www.ibm.com/developerworks/java/jdk/security/142/jceDocs.zip` file.

### IBMJCEFIPS

The IBM version of the Java Cryptography Extension Federal Information Processing Standard (IBMJCEFIPS) is an implementation of the JCE cryptographic service provider that is used in WebSphere Application Server. The IBMJCEFIPS service provider implements the following:

- Signature algorithms (SHA1withDSA, SHA1withRSA)
- Cipher algorithms (AES, TripleDES, RSA)
- Key agreement algorithm (DiffieHellman)
- Key (pair) generator (DSA, AES, TripleDES, HmacSHA1, RSA, DiffieHellman)
- Message authentication code (MAC) (HmacSHA1)
- Message digest (MD5, SHA-1, SHA-256, SHA-384, SHA-512)
- Algorithm parameter generator (DiffieHellman, DSA)
- Algorithm parameter (AES, DiffieHellman, DES, TripleDES, DSA)
- Key factory (DiffieHellman, DSA, RSA)
- Secret key factory (AES, TripleDES)
- Certificate (X.509)
- Secure random (IBMSecureRandom)

### Application Programming Interface

Java Cryptography Extension (JCE) has a provider-based architecture. Providers can be plugged into the JCE framework by implementing the APIs defined by the JCE. The JCE APIs covers:

- Symmetric bulk encryption, such as DES, RC2, and IDEA
- Symmetric stream encryption, such as RC4
- Asymmetric encryption, such as RSA
- Password-based encryption (PBE)
- Key Agreement
- Message Authentication Codes

There is more information documented for the JCE APIs in the `http://www.ibm.com/developerworks/java/jdk/security/jceDocs.zip` file.

**Samples using Java Cryptography Extension**

There are samples located in `http://www.ibm.com/developerworks/java/jdk/security/142/jceDocs.zip` file. Unzip the file and locate the following samples in the`jceDocs/samples` directory:

| File | Description |
| --- | --- |
| `SampleDSASignature.java` | Demonstrates how to generate a pair of DSA keys (a public key and a private key) and use the key to digitally sign a message using the SHA1with DSA algorithm |
| `SampleMarsCrypto.java` | Demonstrates how to generate a Mars secret key, and how to do Mars encryption and decryption |
| `SampleMessageDigests.java` | Demonstrates how to use the message digest for MD2 and MD5 algorithms |
| `SampleRSACrypto.java` | Demonstrates how to generate an RSA key pair, and how to do RSA encryption and decryption |
| `SampleRSASignatures.java` | Demonstrates how to generate a pair of RSA keys (a public key and a private key) and use the key to digitally sign a message using the SHA1withRSA algorithm |
| `SampleX509Verification.java` | Demonstrates how to verify X509 Certificates |

**Documentation**

Refer to the “Security: Resources for learning” on page 21 for documentation on JCE.

## Java 2 security

Java 2 security provides a policy-based, fine-grain access control mechanism that increases overall system integrity by checking for permissions before allowing access to certain protected system resources. Java 2 security guards access to system resources such as file I/O, sockets, and properties. J2EE security guards access to Web resources such as servlets, JavaServer Pages (JSP) files and EJB methods. WebSphere global security includes J2EE role-based authorization, the Common Secure Interoperability Version 2 (CSIv2) authentication protocol, and Secure Sockets Layer (SSL) configuration.

Since Java 2 security is relatively new, many existing or even new applications might not be prepared for the very fine-grain access control programming model that Java 2 security is capable of enforcing. Administrators should understand the possible consequences of enabling Java 2 security if applications are not prepared for Java 2 security. Java 2 security places some new requirements on application developers and administrators.

**Java 2 security for deployers and administrators**

Although Java 2 security is supported in WebSphere Application Server Version 5, it is disabled by default. However, it is enabled automatically if you also enable global security when configuring security. Although it becomes enabled automatically when you enable WebSphere global security, you can choose to disable it. You can configure Java 2 security and global security independently of one another. Disabling global security does not disable Java 2 security automatically. You need to explicitly disable it.

If your applications, or third-party libraries are not ready, having Java 2 security enabled causes problems. You can identify these problems as Java 2 security AccessControlExceptions in the `SystemOut.log` file,

SystemError.log file, or the trace log files. If you are unsure about the Java 2 security readiness of your applications, disable Java 2 security initially to get your application installed and verify that it is working properly.

There are implications if Java 2 Security is enabled; deployers or administrators are required to make sure that all the applications are granted the required permissions, otherwise, applications might fail to run. By default, applications are granted the permissions recommended in the J2EE 1.3 Specification. For details of default permissions granted to applications in the product, refer to the following policy files:
- *install_root*/java/jre/lib/security/java.policy
- *install_root*/properties/server.policy
- *install_root*/config/cells/<*cell name*>/nodes/<*node_name*>/app.policy

**Note:** This policy embodied by these policy files cannot be made more restrictive because the product might not have the necessary Java 2 security doPrivileged APIs in place. The restrictive policy is the default policy. You can grant additional permissions, but you cannot make the default more restrictive because AccessControlExceptions is generated from within WebSphere Application Server. The product does not support a more restrictive policy than the default defined in the policy files previously mentioned.

There are several policy files used to define the security policy for the Java process. These policy files are static (code base is defined in the policy file) and they are in the default policy format provided by the IBM Developer Kit, Java Technology Edition. For enterprise application resources and utility libraries, WebSphere Application Server provides dynamic policy support. The code base is dynamically calculated based on deployment information and permissions are granted based on template policy files during run time. Refer to the article, Java 2 security policy files for more information.

**Note:** Syntax errors in the policy files cause the application server process to fail. Edit these policy files carefully using the Policy Tool provided by the IBM Developer Kit, Java Technology Edition for editing the policy files (*install_root*/java/jre/bin/policytool).

If an application is not prepared for Java 2 security, if the application provider does not provide a was.policy file as part of the application, or if the application provider does not communicate the expected permissions the application is likely to cause Java 2 security access control exceptions at run time. It might not be obvious that an application is not prepared for Java 2 security. Several run-time debugging aids help troubleshoot applications that might have access control exceptions. See the Java 2 security debugging aids for more details. See Handling applications that are not Java 2 security ready for information and strategies for dealing with such applications.

It is important to note that when Java 2 Security is enabled in the Global Security settings, the installed SecurityManager does not currently check modifyThread and modifyThreadGroup permissions for non-system threads. Allowing Web and EJB application code to create or modify a thread can have a negative impact on other components of the container and can affect the capability of the container to manage enterprise bean life cycles and transactions.

**Java 2 security for application developers**

Application developers must understand the permissions granted in the default WebSphere policy and the permission requirements of the SDK APIs that their application calls to know whether additional permissions are required. The ″Permissions in the Java 2 SDK″ reference in the resources section describes which APIs require which permission.

Application providers can assume that applications have the permissions granted in the default policy previously mentioned. Applications that access resources not covered by the default WebSphere policy are required to grant the additional Java 2 security permissions to the application.

While it is possible to grant the application additional permissions in one of the other dynamic WebSphere policy files or in one of the more traditional static policy files, such as `java.policy`, the `was.policy` (which is embedded in the EAR file) ensures the additional permissions are scoped to the exact application that requires them. Scoping the permission beyond the application code that requires it can permit code that normally does not have permission to access particular resources.

If an application component is being developed, like a library that might actually be included in more than one `.ear` file, then the library developer should document the required Java 2 permissions needed by the application assembler. There is no `was.policy` file for library type components. The developer must communicate the required permissions through application programming interface (API) documentation or some other external documentation.

If the component library is shared by multiple enterprise applications, the permissions can be granted to all enterprise applications on the node in the `app.policy` file.

If the permission is only used internally by the component library and the application should never be granted access to resources protected by the permission, then it might be necessary to mark the code as **privileged** (inserting doPrivileged). Refer to the article, AccessControlException, for more details. However, improperly inserting a doPrivileged might open up security holes. Understand the implication of doPrivileged to make a correct judgement whether a doPrivileged should be inserted or not.

The section on Dynamic Policy describes how the permissions in the `was.policy` files are granted at run time.

Developing an application with Java 2 security in mind might be a new skill and impose a security awareness not previously required of application developers. Describing the Java 2 security model and the implications on application development is beyond the scope of this section. The following URL can help you get started: `http://java.sun.com/j2se/1.3/docs/guide/security/index.html`.

**Debugging Aids**

There are two primary aids, the WebSphere `SystemOut.log` file and the com.ibm.websphere.java2secman.norethrow property.

**The WebSphere SystemOut.log File**

The AccessControl exception logged in the `SystemOut.log` file contains the permission violation that causes the exception, the exception call stack, and the permissions granted to each stack frame. This information is usually enough to determine the missing permission and the code requiring the permission.

**The com.ibm.websphere.java2secman.norethrow Property**

When Java 2 security is enabled in WebSphere Application Server, the security manager component throws a java.security.AccessControl exception when a permission violation occurs. This exception, if not handled, often causes a run-time failure. This exception is also logged in the `SystemOut.log` file.

However, when the JVM com.ibm.websphere.java2secman.norethrow property is set and has a value of **true**, the security manager does not throw the AccessControl exception. This information is logged.

To set the `com.ibm.websphere.java2secman.norethrow` property for the server, go to the WebSphere Application Server administrative console and click **Servers > Application Servers**. Under Additional Properties, click **Process Definition > Java Virtual Machine > Custom Properties > New**. In the Name field, type **com.ibm.websphere.java2secman.norethrow**. In the Value field, type **true**.

To set the `com.ibm.websphere.java2secman.norethrow` property for the node agent, go to the WebSphere Application Server administrative console and click **System Administration > Node Agents**. Under

Additional Properties, click **Process Definition > Java Virtual Machine > Custom Properties > New**. In the Name field, type **com.ibm.websphere.java2secman.norethrow**. In the Value field, type **true**.

**Note:** This property is intended for a sandbox or debug environment because it instructs the security manager not to throw the AccessControl exception. Java 2 security is not enforced. This property should not be used in a production environment where a relaxed Java 2 security environment weakens the integrity that Java 2 security is intended to produce.

This property is valuable in a sandbox or test environment where the application can be thoroughly tested and the where the `SystemOut.log` file can be inspected for AccessControl exceptions. Since this property does not throw the AccessControl exception , it does not propagate the call stack and does not cause a failure. Without this property, you have to find and fix AccessControl exceptions one at a time.

**Handling applications that are not Java 2 security ready**

If the increased system integrity that Java 2 security provides is important, then contact the application provider to have the application support Java 2 security or at least communicate the required additional permissions beyond the default WebSphere policy that must be granted.

The easiest way to deal with such applications is to disable Java 2 security in WebSphere Application Server. The downside is that this solution applies to the entire system and the integrity of the system is not as strong as it might be. Disabling Java 2 security might not be acceptable depending on the organization security policies or risk tolerances.

Another approach is to leave Java 2 security enabled, but to grant either just enough additional permissions or grant all permissions to just the problematic application. Granting permissions however, might not be a trivial thing to do. If the application provider has not communicated the required permissions in some way, there is no easy way to determine what the required permissions are and granting all permissions might be the only choice. You minimize this risk by locating this application on a different node, which might help isolate it from certain resources. Grant the java.security.AllPermission permission in the `was.policy` file embedded in the application's .ear file, for example:

```
grant codeBase "file:${application}" {
            permission java.security.AllPermission;
        };
```

*install_root***/properties/server.policy**

This policy defines the policy for the WebSphere classes. At present, all the server processes on the same installation share the same `server.policy` file. However, you can configure this file so that each server process can have a separate `server.policy` file. Define the desired policy file as the value of the Java system properties `java.security.policy`. For details of how to define Java system properties, Refer to the Process definition section of the Manage application servers file.

The `server.policy` file is not a configuration file managed by the repository and the file replication service. Changes to this file are local and do not get replicated to other machines. Use the `server.policy` file to define Java 2 security policy for server resources. Use the `app.policy` file (per node) or `was.policy` file (per enterprise application) to define Java 2 security policy for enterprise application resources.

*WAS_HOME***/java/jre/lib/security/java.policy**

The file represents the default permissions granted to all classes. The policy of this file applies to all the processes launched by the WebSphere Application Server JVM.

**Troubleshooting**

**Symptom:**

`Error message CWSCJ0314E`: Current Java 2 security policy reported a potential violation of Java 2 security permission. Refer to Problem Determination Guide for further information.{0}Permission\:{1}Code\:{2}{3}Stack Trace\:{4}Code Base Location\:{5} Current Java 2 security policy reported a potential violation of Java 2 Security Permission. Refer to Problem Determination Guide for further information.{0}Permission\:{1}Code\:{2}{3}Stack Trace\:{4}Code Base Location\:{5}

**Problem:**

The Java security manager checkPermission() reported a SecurityException on the subject permission with debugging information. The reported information can be different with respect to the system configuration. This report is enabled by either configuring RAS trace into debug mode or specifying a Java property.

See Enabling trace for information on how to configure RAS trace in debug mode.

Specify the following property in the JVM Settings panel from the administrative console: **java.security.debug**. Valid values include:

**access**
Print all debug information including: required permission, code, stack, and code base location.

**stack** Print debug information including: required permission, code, and stack.

**failure** Print debug information including: required permission and code.

**Recommended response:**

The reported exception might be critical to the secure system. Turn on security trace to determine the potential code that might have violated the security policy. Once the violating code is determined, verify if the attempted operation is permitted with respect to Java 2 security, by examining all applicable Java 2 security policy files and the application code.

**Note:** If the application is running with Java Mail, this message might be benign. User can update the `was.policy` file to grant the following permissions to the application.

```
permission java.io.FilePermission "${user.home}${/}.mailcap", "read";
permission java.io.FilePermission "${user.home}${/}.mime.types", "read";
permission java.io.FilePermission "${java.home}${/}lib${/}mailcap", "read";
permission java.io.FilePermission "${java.home}${/}lib${/}mime.types", "read";
```

**Messages**

| Message: | `CWSCJ0313E`: Java 2 security manager debug message flags are initialized\: TrDebug: {0}, Access: {1}, Stack: {2}, Failure: {3} |
|---|---|
| Problem: | Configured values of the valid debug message flags for security manager. |
| Recommended response: | None. |

| Message: | `CWSCJ0307E`: Unexpected exception is caught when trying to determine the code base location. Exception: {0} |
|---|---|
| Problem: | An unexpected exception is caught when the code base location is determined. |
| Recommended response: | Contact an IBM representative. |

## Access control exception

The Java 2 security behavior is specified by its *security policy*. The security policy is an access-control matrix that specifies which system resources certain code bases can access and who must sign them. The Java 2 security policy is declarative and it is enforced by the java.security.AccessController.checkPermission method.

The following example depicts the algorithm for the java.security.AccessController.checkPermission method. For the complete algorithm, refer to the Java 2 security check permission algorithm in Resources for learning.

```
i = m;
while (i > 0) {
 if (caller i's domain does not have the permission)
  throw AccessControlException;
 else if (caller i is marked as privileged)
  return;
 i = i - 1;
};
```

The algorithm requires that all the classes or callers on the call stack have the permissions when a java.security.AccessController.checkPermission method is performed or the request is denied and a java.security.AccessControlException exception is created. However, if the caller is marked as *privileged* and the class (caller) is granted these permissions, the algorithm returns and does not traverse the entire call stack. Subsequent classes (callers) do not need the required permission granted.

A java.security.AccessControlException exception is created when certain classes on the call stack are missing the required permissions during a java.security.AccessController.checkPermission method. Two possible resolutions to the java.security.AccessControlException exception are as follows:

- If the application is calling a Java 2 security-protected API, grant the required permission to the application Java 2 security policy. If the application is not calling a Java 2 security-protected API directly, the required permission results from the side-effect of the third-party APIs accessing Java 2 security-protected resources.
- If the application is granted the required permission, it gains more access than it needs. In this case, it is likely that the third party code that accesses the Java 2 security-protected resource is not properly marked as privileged.

### Example call stack

This example of a call stack indicates where application code is using a third-party API utility library to update the password. The following example is presented to illustrate the point. The decision of where to mark the code as privileged is application-specific and is unique in every situation. This decision requires great depth of domain knowledge and security expertise to make the correct judgement. A number of well written publications and books are available on this topic. Referencing these materials for more detailed information is recommended.

System domain

AccessController.checkPermission()
SecurityManager..checkPermission()
SecurityManager..checkWrite()
java.io.FileOutputStream()

Utility library domain

PasswordUtil.updatePasswordFile()
PasswordUtil.getPassword()

Application domain

Client Code ...

You can use the PasswordUtil utility to change the password of a user. The utility types in the old password and the new password twice to ensure that the correct password is entered. If the old password matches the one stored in the password file, the new password is stored and the password file updates. Assume that none of the stack frame is marked as privileged. According to the java.security.AccessController.checkPermission algorithm, the application fails unless all the classes on the call stack are granted write permission to the password file. The client application does not have permission to write to the password file directly and to update the password file at will.

However, if the PasswordUtil.updatePasswordFile method marks the code that accesses the password file as privileged, then the check permission algorithm does not check for the required permission from classes that call thePasswordUtil.updatePasswordFile method for the required permission as long as the PasswordUtil class is granted the permission. The client application can successfully update a password without granting the permission to write to the password file.

The ability to mark code privileged is very flexible and powerful. If this ability is used incorrectly, the overall security of the system can be compromised and security holes can be exposed. Use the ability to mark code privileged carefully.

**Resolution to the java.security.AccessControlException exception**

As described previously, you have two approaches to resolve a java.security.AccessControlException exception. Judge these exceptions individually to decide which of the following resolutions is best:
1. Grant the missing permission to the application.
2. Mark some code as privileged, after considering the issues and risks.

## Configuring Java 2 security

Java 2 security is a programming model that is very pervasive and has a huge impact on application development. It is disabled by default, but is enabled automatically when global security is enabled. However, Java 2 security is orthogonal to J2EE role-based security; you can disable or enable it independently of Global Security.

However, it does provide an extra level of access control protection on top of the J2EE role-based authorization. It particularly addresses the protection of system resources and APIs. Administrators should need to consider the benefits against the risks of disabling Java 2 Security.

The following recommendations are provided to help enable Java 2 security in a test or production environment:

1. Make sure the application is developed with the Java 2 security programming model in mind. Developers have to know whether or not the APIs used in the applications are protected by Java 2 security. It is very important that the required permissions for the APIs used are declared in the policy file (*was.policy*), or the application fails to run when Java 2 security is enabled. Developers can reference the Web site for Development Kit APIs that are protected by Java 2 security. See the Programming model and decisions section of the “Security: Resources for learning” on page 21 article to visit this Web site.
2. Make sure that migrated applications from previous releases are given the required permissions. Since Java 2 security is not supported or partially supported in previous WebSphere Application Server releases, applications developed prior to Version 5 most likely are not using the Java 2 security programming model. There is no easy way to find out all the required permissions for the application. Following are activities you can perform to determine the extra permissions required by an application:
   - Code review and code inspection
   - Application documentation review
   - Sandbox testing of migrated enterprise applications with Java 2 security enabled in a pre-production environment. Enable tracing in WebSphere Java 2 security manager to help determine the missing permissions in the application policy file. The trace specification is `com.ibm.ws.security.core.SecurityManager=all=enabled`.
   - Use the `com.ibm.websphere.java2secman.norethrow` system property to aid debuggging. This property should not be used in a production environment. Refer to “Java 2 security” on page 463.

**Note:** The default permission set for applications is the recommended permission set defined in the J2EE 1.3 Specification. The default is declared in the `profiles/`*`profile_name`*`/config/cells/`*`cell_name`*`/nodes/`*`node_name`*`/app.policy` policy file with permissions defined in the Development Kit (*`JAVA_HOME`*`/jre/lib/security/java.policy`) policy file that grant permissions to everyone. However, applications are denied permissions declared in the `profiles/`*`profile_name`*`/config/cells/`*`cell_name`*`/filter.policy` filter policy file. Permissions declared in the *filter.policy* file are filtered for applications during the permission check.

**Note:** Define the required permissions for an application in a `was.policy` file and embed the `was.policy` file in the application enterprise archive (EAR) file as *YOURAPP.ear/META-INF/was.policy* (see “Configuring Java 2 security policy files” on page 476 for details).

The following steps describe how to enforce Java 2 security on the cell level for WebSphere Application Server Network Deployment and the server level for WebSphere Application Server and WebSphere Application Server Express:

1. Click **Security > Global security**. The Global security panel is displayed.
2. Select the **Enforce Java 2 security** option.
3. Click **OK** or **Apply**.
4. Click **Save** to save the changes.
5. Restart the server for the changes to take effect.

Java 2 security is enabled and enforced for the servers. Java 2 security permission is selected when a Java 2 security protected API is called.

**When to use Java 2 security**

1. To enable protection on system resources. For example, when opening or listening to a socket connection, reading or writing to operating system file systems, reading or writing Java Virtual Machine system properties, and so on.
2. To prevent application code calling destructive APIs. For example, calling the *System.exit()* method brings down the application server.
3. To prevent application code from obtaining privileged information (passwords) or gaining extra privileges (obtaining server credentials).

The WebSphere Java 2 security manager is enhanced to dump the Java 2 security permissions granted to all classes on the call stack when an application is denied access to a resource (the `java.security.AccessControlException` exception is thrown). However, this tracing capability is disabled by default. You can enable it by specifying the server trace service with the `com.ibm.ws.security.core.SecurityManager=all=enabled` trace specification. When the exception is thrown, the trace dump provides hints to determine whether the application is missing permissions or the product run time code or third party libraries used are not properly marked as *privileged* when accessing Java 2 protected resources. See the Security Problem Determination Guide for details.

## Using PolicyTool to edit policy files

Java 2 security uses several policy files to determine the granted permission for each Java program. See Dynamic policy for the list of available policy files. The Java Development Kit provides *policytool* to edit these policy files. This tool is recommended for editing any policy file to verify the syntax of its contents. Syntax errors in the policy file cause an *AccessControlException* during application execution, including the server start. Identifying the cause of this exception is not easy because the user might not be familiar with the resource that has an access violation. Be careful when you edit these policy files.

1. Start policytool. Enter `%{was.install.root}/java/jre/bin/policytool` from a command prompt.

   The policytool window opens. The policytool looks for the `.java.policy` file in your home directory. If it does not exist, an Error message displays. Click **OK**.
2. Click **File** > **Open**.
3. Navigate the directory tree in the **Open** window to pick up the policy file that you need to update. After selecting the policy file, click **Open**. The code base entries are listed in the window.
4. Create or modify the code base entry.
   a. Modify the existing code base entry by double-clicking the code base, or click the code base and click **Edit Policy Entry**. The Policy Entry window opens with the permission list defined for the selected code base.
   b. Create a new code base entry by clicking **Add Policy Entry**. The Policy Entry window opens. At the code base column, enter the code base information as a URL format, for example, `/WebSphere/AppServer/InstalledApps/testcase.ear`.
5. Modify or add the permission specification
   a. Modify the permission specification by double-clicking the entry you want to modify, or by selecting the permission and clicking **Edit Permission**. The Permissions window opens with the selected permission information.
   b. Add a new permission by clicking **Add Permission**. The Permissions window opens. In the Permissions, window there are four rows for **Permission**, **Target Name**, **Actions**, and **Signed By**.
6. Select the permission from the Permission list. The selected permission displays. After a permission is selected, the **Target Name**, **Actions**, and **Signed By** fields automatically show the valid choices or they enable text input in the right text input area.
   a. Select **Target Name** from the list, or enter the target name in the right text input area.
   b. Select **Actions** from the list.
   c. Input **Signed By** if it is needed.

   **Important:** The Signed By keyword is not supported in the following policy files: `app.policy`, `spi.policy`, `library.policy`, `was.policy`, and `filter.policy` files. However, the Signed By keyword is supported in the following policy files:`java.policy`, `server.policy`, and `client.policy` files. The Java Authentication and Authorization Service (JAAS) is not supported in the `app.policy`, `spi.policy`, `library.policy`, `was.policy`, and `filter.policy` files. However, the JAAS principal keyword is supported in a JAAS policy file when it is specified by the Java Virtual Machine (JVM) system property, `java.security.auth.policy`.

7. Click **OK** to close the Permissions window. Modified permission entries of the specified code base display.
8. Click **Done** to close the window. Modified code base entries are listed. Repeat steps 4 through 8 until you complete editing.
9. Click **File** > **Save** after you finish editing the file.

A policy file is updated. If any policy files need editing, use the policytool. Do not edit the policy file manually. Syntax errors in the policy files can potentially cause application servers or enterprise applications to not start or function incorrectly. For the changes in the updated policy file to take effect, restart the Java processes.

## Java 2 security policy files

The J2EE 1.3 specification has a well-defined programming model of responsibilities between the container providers and the application code. Using Java 2 security manager to help enforce this programming model is recommended. Certain operations are not supported in the application code because such operations interfere with the behavior and operation of the containers. The Java 2 security manager is used in the product to enforce responsibilities of the container and the application code.

This product provides support for policy file management. A number of policy files in the product are either static or dynamic. *Dynamic policy* is a template of permissions for a particular type of resource. No relative code base is defined in the dynamic policy template. The code base is dynamically calculated from the deployment and run-time data.

### Static policy files

| Policy file | Location |
| --- | --- |
| java.policy | *install_root*/java/jre/lib/security/java.policy. Default permissions granted to all classes. The policy of this file applies to all the processes launched by WebSphere Application Server. |
| server.policy | *install_root*/profiles/*profile_name*/properties/server.policy. Default permissions granted to all the product servers. |
| client.policy | *install_root*/profiles/*profile_name*/properties/client.policy. Default permissions for all of the product client containers and applets on a node. |

The static policy files are not managed by configuration and file replication services. Changes made in these files are local and are not replicated to other nodes in the Network Deployment cell.

### Dynamic policy files

| Policy file | Location |
| --- | --- |
| spi.policy | *install_root*/profiles/*profile_name*/config/cells/*cell_name*/nodes/*node_name*/spi.policy<br><br>This template is for the Service Provider Interface (SPI) or the third-party resources that are embedded in the product. Examples of SPI are the Java Message Service (JMS) in MQ Series and JDBC drivers. The code base for the embedded resources are dynamically determined from the configuration (resources.xml file) and run-time data, and permissions that are defined in the spi.policy files are automatically applied to these resources and JAR files specified in the class path of a ResourceAdapter. The default permission of the spi.policy file is java.security.AllPermissions. |
| library.policy | *install_root*/profiles/*profile_name*/config/cells/*cell_name*/nodes/*node_name*/library.policy<br><br>This template is for the library (Java library classes). You can define a shared library to use in multiple product applications. The default permission of the library.policy is empty. |

| Policy file | Location |
| --- | --- |
| app.policy | *install_root*/profiles/*profile_name*/config/cells/*cell_name* /nodes/*node_name*/app.policy<br><br>The app.policy file defines the default permissions that are granted to all the enterprise applications running on *node_name* in *cell_name*. |
| was.policy | *install_root*/config/cells/*cell_name* /applications/*ear_file_name*/deployments/ *application_name*/META-INF/was.policy<br><br>This template is for application-specific permissions. The was.policy file is embedded in the enterprise archive (EAR) file. |
| ra.xml | *rar_file_name*/META-INF/was.policy.RAR.<br><br>This file can have a permission specification that is defined in the ra.xml file. The ra.xml file is embedded in the RAR file. |

**Note:** Grant entries that are specified in the app.policy and was.policy files must have a code base defined. If grant entries are specified without a code base, the policy files are not loaded properly and the application can fail. If the intent is to grant the permissions to all applications, use *file:${application}* as a code base in the grant entry.

### Syntax of the policy file

A policy file contains several policy entries. The following example depicts each policy entry format:

```
grant [codebase <Codebase>] {
permission <Permission>;
 permission <Permission>;
permission <Permission>;
};

<CodeBase>:  A URL.
   For example,  "file:${java.home}/lib/tools.jar"
      When [codebase <Codebase>] is not specified, listed
               permissions are applied to everything.
      If URL ends with a JAR file name,  only the classes in the
               JAR file belong to the codebase.
               If URL ends with "/", only the class files in the specified
               directory belong to the codebase.
               If URL ends with "*", all JAR and class files in the specified
               directory belong to the codebase.
               If URL ends with "-", all JAR and class files in the specified
               directory and its subdirectories belong to the codebase.
<Permissions>: Consists from
       Permission Type   : class name of the permission
          Target Name       : name specifying the target
          Actions           : actions allowed on target
   For example,
         java.io.FilePermission "/tmp/xxx", "read,write"
```

Refer to developer kit specifications for the details of each permission.

### Syntax of dynamic policy

You can define permissions for specific types of resources in dynamic policy files for an enterprise application. This action is achieved by using *product-reserved symbols*. The reserved symbol scope depends on where it is defined. If you define the permissions in the `app.policy` file, the symbol applies to all the resources on all of the enterprise applications that run on *node_name*. If you define the permissions in the `META-INF/was.policy` file, the symbol applies only to the specific enterprise application. Valid symbols for the code base are listed in the following table:

| Symbol | Meaning |
| --- | --- |
| file:${application} | Permissions apply to all resources within the application |
| file:${jars} | Permissions apply to all utility Java archive (JAR) files within the application |
| file:${ejbComponent} | Permissions apply to Enterprise JavaBeans (EJB) resources within the application |
| file:${webComponent} | Permissions apply to Web resources within the application |
| file:${connectorComponent} | Permissions apply to connector resources within the application |

Other than these entries specified by the code base symbols, you can specify the module name for a granular setting. For example:

```
grant codeBase "file:DefaultWebApplication.war" {
   permission java.security.SecurityPermission "printIdentity";
 };

grant codeBase "file:IncCMP11.jar" {
permission java.io.FilePermission
"${user.install.root}${/}bin${/}DefaultDB${/}-",
"read,write,delete";
};
```

The sixth and seventh lines in the previous code sample are one continuous line.

You can use a relative code base only in the `META-INF/was.policy` file.

Several product-reserved symbols are defined to associate the permission lists to a specific type of resources.

| Symbol | Meaning |
| --- | --- |
| file:${application} | Permissions apply to all resources within the application |
| file:${jars} | Permissions apply to all utility JAR files within the application |
| file:${ejbComponent} | Permissions apply to enterprise beans resources within the application |
| file:${webComponent} | Permissions apply to Web resources within the application |
| file:${connectorComponent} | Permissions apply to connector resources both within the application and in stand-alone connector resources. |

Five embedded symbols are provided to specify the path and the name for the java.io.FilePermission permission. These symbols enable flexible permission specification. The absolute file path is fixed after the installation of the application.

| Symbol | Meaning |
|---|---|
| ${app.installed.path} | Path where the application is installed |
| ${was.module.path} | Path where the module is installed |
| ${current.cell.name} | Current cell name |
| ${current.node.name} | Current node name |
| ${current.server.name} | Current server name |

**Note:** Do not use the `${was.module.path}` in the `${application}` entry.

Carefully determine where to add a new permission. An incorrectly specified permission causes an AccessControlException exception. Because dynamic policy resolves the code base at run time, determining which policy file has a problem is difficult. Add a permission only to the necessary resources. For example, `use ${ejbcomponent}`, and `etc` instead of `${application}`, and update the `was.policy` file instead of the `app.policy` file, if possible.

### Static policy filtering

Limited static policy filtering support exists. If the `app.policy` file and the `was.policy` file have permissions that are defined in the `filter.policy` file with the keyword, `filterMask`, the run time removes the permissions from the applications and an audit message is logged. However, if the permissions that are defined in the `app.policy` and the `was.policy` files are compound permissions, for example, `java.security.AllPermission`, the permission is not removed, but a warning message is written to the log file. The policy filtering only supports Developer Kit permissions, (the permissions package name begins with `java` or `javax`).

Run-time policy filtering support is provided to force stricter filtering. If the `app.policy` file and the `was.policy` file have permissions that are defined in the `filter.policy` file with the keyword, `runtimeFilterMask`, the run time removes the permissions from the applications no matter what permissions are granted to the application. For example, even if a was.policy file has the java.security.AllPermission permission granted to one of its modules, specified permissions such as `runtimeFilterMask` are removed from the granted permission during run time.

If the Issue Permission Warning flag in the Global Security panel is enabled and if the `app.policy` file and the `was.policy` file contain custom permissions (non-Developer Kit permissions, where the permissions package name begins with `java` or `javax`), a warning message logs. The permission is not removed. If the AllPermission permission is listed in the `app.policy` file and the `was.policy` file, a warning message logs.

### Policy file editing

Using the policy tool that is provided by the Developer Kit (*install_root*`/java/jre/bin/policytool`), to edit the previous policy files is recommended. For Network Deployment, extract the policy files from the repository before editing. After the policy file is extracted, use the policy tool to edit the file. Check the modified policy files into the repository and synchronize them with other nodes.

If syntax errors exist in the policy files, the enterprise application or the server process might fail to start. Be cautious when editing these policy files. For example, if a policy has a trailing space in the policy permission target name, the policy fails to parse the permission properly in WebSphere Application Server, Version 5.1 IBM Developer Kit, Java Technology Edition Version 1.4.x. In the following example, note the space before the last quote: * \″*\″ ″

```
grant {
    permission javax.security.auth.PrivateCredentialPermission
         "javax.resource.spi.security.PasswordCredential * \"*\" ","read";
};
```

If the permission is in a policy file loaded by the IBM Developer Kit, Java Technology Edition Version 1.4.x policy tool, the following message might display:

```
Errors have occurred while opening the policy configuration.
View the warning log for more information.
```

or the following message might display in warning log:

```
Warning: Invalid argument(s) for constructor:
javax.security.auth.PrivateCredentialPermission.
```

To fix this problem, edit the permission and remove the trailing space. When the trailing space is removed, the permission loads properly. The following code sample shows the corrected permission:

```
grant {
    permission javax.security.auth.PrivateCredentialPermission
          "javax.resource.spi.security.PasswordCredential * \"*\"","read";
}
```

**Troubleshooting**

To debug the dynamic policy, choose one of three ways to generate the detail report of the AccessControlException exception.

- **Trace** (Configured by RAS trace). Enables traces with the trace specification:

  **Attention:** The following command is one continuous line

  ```
  com.ibm.ws.security.policy.*=all=enabled:
  com.ibm.ws.security.core.SecurityManager=all=enabled
  ```
- **Trace** (Configured by property). Specifies a Java java.security.debug property . Valid values for the java.security.debug property are as follows:
  - Access. Print all debug information including required permission, code, stack, and code base location.
  - Stack. Print debug information including, required permission, code, and stack.
  - Failure. Print debug information including required permission and code.
- **ffdc**. Enable `ffdc`, modify the `ffdcRun.properties` file by changing `Level=4`and `LAE=true`. Look for an `Access Violation` keyword in the log file.

## Configuring Java 2 security policy files

Java 2 security uses several policy files to determine the granted permissions for each Java programs. See the "Java 2 security policy files" on page 472 article for the list of available policy files supported by WebSphere Application Server Version.

There are two types of policy files supported by WebSphere Application Server: dynamic policy files and static policy files. Static policy files provide the default permissions. Dynamic policy files provide application permissions. There are six dynamic policy files:

| Policy file name | Description |
| --- | --- |
| app.policy | Contains default permissions for all of the enterprise applications in the cell. |

| Policy file name | Description |
| --- | --- |
| was.policy | Contains application-specific permissions for an WebSphere Application Server enterprise application. This file is packaged in an enterprise archive (EAR) file. |
| ra.xml | Contains connector application specific permissions for a WebSphere Application Server enterprise application. This file is packaged in a resource adapter archive (RAR) file. |
| spi.policy | Contains permissions for Service Provider Interface (SPI) or third-party resources embedded in WebSphere Application Server. The default contents grant everything. Update this file carefully when the cell requires more protection against SPI in the cell. This file is applied to all of the SPIs defined in the resources.xml file. |
| library.policy | Contains permissions for the shared library of enterprise applications. |
| filter.policy | Contains the list of permissions that require filtering from the was.policy file and the app.policy file in the cell. This filtering mechanism only applies to the was.policy and app.policy files. |

In WebSphere Application Server, applications must have the appropriate thread permissions specified in the was.policy or app.policy file. Without the thread permissions specified, the application cannot manipulate threads and WebSphere Application Server throws a java.security.AccessControlException. The app.policy file applies to a specified node. If you change the permissions in one app.policy file, you must incorporate the new thread policy in the same file on the remaining nodes. Also, if you add the thread permissions to the app.policy file, you must restart WebSphere Application Server to enforce the new permissions. However, if you add the permissions to the was.policy file for a specific application, you do not need to restart WebSphere Application Server. An administrator must add the following code to a was.policy or app.policy file for an application to manipulate threads:

```
grant codeBase "file:${application}" {
permission java.lang.RuntimePermission "stopThread";
permission java.lang.RuntimePermission "modifyThread";
permission java.lang.RuntimePermission "modifyThreadGroup";
};
```

**Important:** The Signed By keyword is not supported in the following policy files: app.policy, spi.policy, library.policy, was.policy, and filter.policy files. However, the Signed By keyword is supported in the following policy files:java.policy, server.policy, and client.policy files. The Java Authentication and Authorization Service (JAAS) is not supported in the app.policy, spi.policy, library.policy, was.policy, and filter.policy files. However, the JAAS principal keyword is supported in a JAAS policy file when it is specified by the Java Virtual Machine (JVM) system property, java.security.auth.policy. You can statically set the authorization policy files in java.security.auth.policy with auth.policy.url.n=*URL* where *URL* is the location of the authorization policy.

1. Identify the policy file to update.
   - If the permission is required by an application, update the static policy file. Refer to Configuring static policy files.
   - If the permission is required by all of the WebSphere Application Server enterprise applications in the node, refer to Configuring spi.policy files.
   - If the permission is required only by specific WebSphere Application Server enterprise applications and the permission is required only by connector, update the ra.xml file. Refer to Assembling resource adapter (connector) modules. Otherwise, update the was.policy file. Refer to Configuring was.policy files and Adding the was.policy file to applications.
   - If the permission is required by shared libraries, refer to Configuring library.policy files.

- If the permission is required by SPI libraries, refer to Configuring spi.policy files.

**Note:** It is recommended to pick up the policy file with the smallest scope. You can avoid giving an extra permission to the Java programs and protect the resources. You can update the `ra.xml` file or the `was.policy` file rather than the `app.policy` file. Use specific component symbols ($(ejbcomponent), ${webComponent},${connectorComponent} and ${jars}) than ${application} symbols. Update dynamic policy files than static policy files.

Add any permission that should never be granted to the WebSphere Application Server enterprise application in the cell to the `filter.policy` file. Refer to Configuring filter.policy files.

2. Restart the WebSphere Application Server enterprise application.

The required permission is granted for the specified WebSphere Application Server enterprise application.

If an WebSphere Application Server enterprise application in a cell requires permissions, some of the dynamic policy files need updating. The symptom of the missing permission is the exception, java.security.AccessControlException. The missing permission is listed in the exception data, for example,

```
java.security.AccessControlException: access denied (java.io.FilePermission
C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar read)
```

The previous two lines were split onto two lines because of the width of the page. However, the permission should be on one line.

When a Java program receives this exception and adding this permission is justified, add a permission to an adequate dynamic policy file, for example,

```
grant codeBase "file:<user client installed location>" {
permission java.io.FilePermission
"C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar", "read";
};
```

The previous two lines were split onto two lines because of the width of the page. However, the permission should be on one line.

To decide whether to add a permission, refer to the article AccessControlException.

***Configuring app.policy files:***

Java 2 security uses several policy files to determine the granted permissions for each Java program. See the Dynamic policy article for the list of available policy files supported by WebSphere Application Server. The `app.policy` file is a default policy file shared by all of the WebSphere Application Server enterprise applications. The union of the permissions contained in the following files is applied to the WebSphere Application Server enterprise application:

- Any policy file that is specified in the policy.url.* properties in the `java.security` file.
- The `app.policy` files, which are managed by configuration and file replication services.
- The `server.policy` file.
- The `java.policy` file.
- The application `was.policy` file.
- The permission specification of the `ra.xml` file.
- The shared library, which is the `library.policy` file.

In WebSphere Application Server, applications that manipulate threads must have the appropriate thread permissions specified in the `was.policy` or `app.policy` file. Without the thread permissions specified, the application cannot manipulate threads and WebSphere Application Server throws a

java.security.AccessControlException. If an administrator adds thread permissions to the app.policy file, the permission change requires a restart of the WebSphere Application Server. An administrator must add the following code to a `was.policy` or `app.policy` file for an application to manipulate threads:

```
grant codeBase "file:${application}" {
permission java.lang.RuntimePermission "stopThread";
permission java.lang.RuntimePermission "modifyThread";
 permission java.lang.RuntimePermission "modifyThreadGroup";
};
```

**Important:** The Signed By and the Java Authentication and Authorization Service (JAAS) principal keywords are not supported in the `app.policy` file. However, the Signed By keyword is supported in the following files: `java.policy`, `server.policy`, and the `client.policy` files. The JAAS principal keyword is supported in a JAAS policy file when it is specified by the Java Virtual Machine (JVM) system property, `java.security.auth.policy`. You can statically set the authorization policy files in `java.security.auth.policy` with `auth.policy.url.n=`*URL* `where` *URL* is the location of the authorization policy.

If the default permissions for enterprise applications (the union of the permissions defined in the `java.policy` file, the `server.policy` file and the `app.policy` file) are enough, no action is required. The default `app.policy` file is used automatically. If a specific change is required to all of the enterprise applications in the cell, update the `app.policy` file. Syntax errors in the policy files cause start failures in the application servers. Edit these policy files carefully.

Modify the `app.policy` file with the Policy Tool. Changes to the `app.policy` file are local for the node.

The default Java 2 security policies have been changed for the enterprise application.

Several product-reserved symbols are defined to associate the permission lists to a specific type of resource.

| Symbol | Meaning |
| --- | --- |
| file:${application} | Permissions apply to all resources within the application |
| file:${jars} | Permissions apply to all utility Java archive (JAR) files within the application |
| file:${ejbComponent} | Permissions apply to enterprise bean resources within the application |
| file:${webComponent} | Permissions apply to Web resources within the application |
| file:${connectorComponent} | Permissions apply to connector resources both within the application and within stand-alone connector resources. |

There are five embedded symbols provided to specify the path and name for java.io.FilePermission. These symbols enable flexible permission specifications. The absolute file path is fixed after the installation of the application.

| Symbol | Meaning |
| --- | --- |
| ${app.installed.path} | Path where the application is installed |
| ${was.module.path} | Path where the module is installed |
| ${current.cell.name} | Current cell name |
| ${current.node.name} | Current node name |
| ${current.server.name} | Current server name |

**Note:** You cannot use the ${was.module.path} in the ${application} entry.

The app.policy file supplied by WebSphere Application Server resides at *install_root*/profiles/*profile_name*/config/cells/*cell_name*/nodes/*node_name*/app.policy, which contains the following default permissions:

**Attention:** In the following code sample, the first two lines related to permission java.io.FilePermission were split into two lines each due to the width of the printed page.

```
grant codeBase "file:${application}" {
  // The following are required by Java mail
  permission java.io.FilePermission "${was.install.root}${/}java${/}
  jre${/}lib${/}ext${/}mail.jar", "read";
  permission java.io.FilePermission "${was.install.root}${/}java${/}
  jre${/}lib${/}ext${/}activation.jar", "read";
};

grant codeBase "file:${jars}" {
  permission java.net.SocketPermission "*", "connect";
  permission java.util.PropertyPermission "*", "read";
};

grant codeBase "file:${connectorComponent}" {
  permission java.net.SocketPermission "*", "connect";
  permission java.util.PropertyPermission "*", "read";
};
grant codeBase "file:${webComponent}" {
  permission java.io.FilePermission "${was.module.path}${/}-", "read, write";
  permission java.lang.RuntimePermission "loadLibrary.*";
  permission java.lang.RuntimePermission "queuePrintJob";
  permission java.net.SocketPermission "*", "connect";
  permission java.util.PropertyPermission "*", "read";
};

grant codeBase "file:${ejbComponent}" {
 permission java.lang.RuntimePermission "queuePrintJob";
 permission java.net.SocketPermission "*", "connect";
 permission java.util.PropertyPermission "*", "read";
};
```

If all of the WebSphere Application Server enterprise applications in a cell require permissions that are not defined as defaults in the java.policy file, the server.policy file and the app.policy file, then update the app.policy file. The symptom of a missing permission is the exception, java.security.AccessControlException. The missing permission is listed in the exception data, for example, java.security.AccessControlException: access denied (java.io.FilePermission C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar read).

When a Java program receives this exception and adding this permission is justified, add a permission to the server.policy file, for example:

```
grant codeBase "file:<user client installed location>" {
permission java.io.FilePermission
"C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar", "read"; };
```

To decide whether to add a permission, refer to the article AccessControlException.

Restart all WebSphere Application Server enterprise applications to ensure that the updated `app.policy` file takes effect.

***Configuring filter.policy files:***

Java 2 security uses several policy files to determine the granted permission for each Java program. Java 2 security policy filtering is only in effect when Java 2 security is enabled. Refer to Configuring Java 2 security. The filtering policy defined in the `filter.policy` file is cell wide. Refer to the article, Dynamic policy, for the list of available policy files supported by WebSphere Application Server. The `filter.policy` file is the only policy file used when restricting the permission instead of granting permission. The permissions listed in the filter policy file are filtered out from the `app.policy` file and the `was.policy` file. Permissions defined in the other policy files are not affected by the `filter.policy` file.

When a permission is filtered out, an audit message is logged. However, if the permissions defined in the `app.policy` file and the `was.policy` file are compound permissions like java.security.AllPermission, for example, the permission is not removed. A warning message is logged. If the Issue Permission Warning flag is enabled (default) and if the `app.policy` file and the `was.policy` file contain custom permissions (non-Java API permission, the permission package name begins with characters other than `java` or `javax`), then a warning message is logged and the permission is not removed. You can change the value of the **Issue permission warning** option on the Global Security panel. It is not recommended that you use `AllPermission` for the enterprise application.

There are some default permissions defined in the `filter.policy` file. These permissions are the minimal ones recommended by the product. If more permissions are added to the `filter.policy` file, certain operations can fail for enterprise applications. Add permissions to the `filter.policy` file carefully.

An updated `filter.policy` file is applied to all of the WebSphere Application Server enterprise application after the servers are restarted.

The `filter.policy` file is managed by configuration and file replication services. Changes made in the file are replicated to other nodes in the Network Deployment cell.

The `filter.policy` file supplied by WebSphere Application Server resides at:
*install_root*/profiles/*profile_name*/config/cells/*cell_name*/filter.policy.

It contains these permissions as defaults:

```
filterMask {
permission java.lang.RuntimePermission "exitVM";
permission java.lang.RuntimePermission "setSecurityManager";
permission java.security.SecurityPermission "setPolicy";
permission javax.security.auth.AuthPermission "setLoginConfiguration"; };
runtimeFilterMask {
permission java.lang.RuntimePermission "exitVM";
permission java.lang.RuntimePermission "setSecurityManager";
permission java.security.SecurityPermission "setPolicy";
permission javax.security.auth.AuthPermission "setLoginConfiguration"; };
```

The permissions defined in `filterMask` are for static policy filtering. The security run time tries to remove the permissions from applications during application startup. Compound permissions are not removed but are issued with a warning, and application deployment is stopped if applications contain permissions defined in `filterMask`, and if scripting was used (wsadmin tool). The `runtimeFilterMask` defines permissions used by the security run time to deny access to those permissions to application thread. Do not add more permissions to the `runtimeFilterMask`. Application start failure or incorrect functioning might result. Be careful when adding more permissions to the `runtimeFilterMask`. Usually, you only need to add permissions to the `filterMask` stanza.

WebSphere Application Server relies on the filter policy file to restrict or disallow certain permissions that could compromise the integrity of the system. For instance, WebSphere Application Server considers the `exitVM` and `setSecurityManager` permissions as those permissions that most applications should never have. If these permissions are granted, then the following scenarios are possible:

- **exitVM** -- A servlet, JSP file, enterprise bean, or other library used by the aforementioned could call the System.exit() API and cause the entire WebSphere Application Server process to terminate.
- **setSecurityManager** -- An application could install its own SecurityManager that could either grant more permissions or bypass the default policy the WebSphere Application Server SecurityManager enforces.

For the updated `filter.policy` file to take effect, restart related Java processes.

***Configuring the was.policy file:***

Java 2 security uses several policy files to determine the granted permission for each Java program. See “Java 2 security policy files” on page 472 for the list of available policy files supported by WebSphere Application Server Version 5. The `was.policy` file is an application-specific policy file for WebSphere Application Server enterprise applications. It is embedded in the enterprise archive (EAR) file (`META-INF/was.policy`). The `was.policy` file is located in:

```
install_root/profiles/profile_name/config/cells/cell_name/applications/
ear_file_name/deployments/application_name/META-INF/was.policy
```

The union of the permissions contained in the following files is applied to the WebSphere Application Server enterprise application:

- Any policy file that is specified in the policy.url.* properties in the `java.security` file.
- The `app.policy` files, which are managed by configuration and file replication services.
- The `server.policy` file.
- The `java.policy` file.
- The application `was.policy` file.
- The permission specification of the `ra.xml` file.
- The shared library, which is the `library.policy` file.

Changes made in these files are replicated to other nodes in the Network Deployment cell.

Several product-reserved symbols are defined to associate the permission lists to a specific type of resources.

| Symbol | Definition |
| --- | --- |
| file:${application} | file:${application} |
| file:${jars} | Permissions apply to all utility Java archive (JAR) files within the application |
| file:${ejbComponent} | Permissions apply to enterprise bean resources within the application |
| file:${webComponent} | Permissions apply to Web resources within the application |
| file:${connectorComponent} | Permissions apply to connector resources within the application |

In WebSphere Application Server, applications that manipulate threads must have the appropriate thread permissions specified in the `was.policy` or `app.policy` file. Without the thread permissions specified, the application cannot manipulate threads and WebSphere Application Server throws a

java.security.AccessControlException. If you add the permissions to the was.policy file for a specific application, you do not need to restart WebSphere Application Server. An administrator must add the following code to a was.policy or app.policy file for an application to manipulate threads:

```
grant codeBase "file:${application}" {
permission java.lang.RuntimePermission "stopThread";
permission java.lang.RuntimePermission "modifyThread";
permission java.lang.RuntimePermission "modifyThreadGroup";
};
```

An administrator can add the thread permissions to the app.policy file, but the permission change requires a restart of the WebSphere Application Server.

**Important:** The Signed By and the Java Authentication and Authorization Service (JAAS) principal keywords are not supported in the was.policy file. The **Signed By** keyword is supported in the following policy files: java.policy, server.policy, and client.policy. The JAAS principal keyword is supported in a JAAS policy file when it is specified by the Java Virtual Machine (JVM) system property, java.security.auth.policy. You can statically set the authorization policy files in java.security.auth.policy with auth.policy.url.n=*URL* where *URL* is the location of the authorization policy.

Other than these blocks, you can specify the module name for granular settings. For example,

```
"file:DefaultWebApplication.war" {
      permission java.security.SecurityPermission "printIdentity";
 };

grant codeBase "file:IncCMP11.jar" {
      permission java.io.FilePermission
      "${user.install.root}${/}bin${/}DefaultDB${/}-",
      "read,write,delete";
};
```

There are five embedded symbols provided to specify the path and name for the java.io.FilePermission. These symbols enable flexible permission specification. The absolute file path is fixed after the application is installed.

| Symbol | Definition |
| --- | --- |
| ${app.installed.path} | Path where the application is installed |
| ${was.module.path} | Path where the module is installed |
| ${current.cell.name} | Current cell name |
| ${current.node.name} | Current node name |
| ${current.server.name} | Current server name |

If the default permissions for the enterprise application (union of the permissions defined in the java.policy file, the server.policy file and the app.policy file) are enough, no action is required. If an application has specific resources to access, update the was.policy file. The first two steps assume that you are creating a new policy file.

**Note:** Syntax errors in the policy files cause the application server to fail. Use care when editing these policy files.

1. Create or edit a new was.policy file using the Policy Tool. For more information, see “Using PolicyTool to edit policy files” on page 471.

2. Package the `was.policy` file into the enterprise archive (EAR) file.

   For more information, see "Adding the was.policy file to applications" on page 486.The following instructions describe how to import a `was.policy` file.

   a. Import the EAR file into an assembly tool. For more information, see Importing enterprise applications.
   b. Open the Project Navigator view.
   c. Expand the EAR file and click **META-INF**. You might find a `was.policy` file in the META-INF directory. If you want to delete the file, right-click the file name and select **Delete**.
   d. At the bottom of the Project Navigator view, click **J2EE Hierarchy**.
   e. Import the `was.policy` file by right-clicking the **Modules** directory within the deployment descriptor and clicking **Import > Import > File system**.
   f. Click **Next**.
   g. Enter the path name to the `was.policy` file in the **From directory** field or click **Browse** to locate the file.
   h. Verify that the path directory listed in the **Into directory** field lists the correct `META-INF` directory.
   i. Click **Finish**.
   j. To validate the EAR file, right-click the EAR file, which contains the Modules directory, and click **Run Validation**.
   k. To save the new EAR file, right-click the EAR file, and click **Export > Export EAR file**. If you do not save the revised EAR file, the EAR file will contain the new `was.policy` file. However, if the workspace becomes corrupted, you might lose the revised EAR file.
   l. To generate deployment code, right-click the EAR file and click **Generate Deployment Code**.
3. Update an existing installed application, if one already exists.
   a. Modify the was.policy file with the Policy Tool. For more information, see "Using PolicyTool to edit policy files" on page 471.

The updated `was.policy` file is applied to the application after the application restarts.

If an application must access a specific resource that is not defined as a default in the `java.policy` file, the `server.policy` file and the `app.policy`, then delete the `was.policy` file for that application. The symptom of the missing permission is that the exception, `java.security.AccessControlException`. The missing permission is listed in the exception data, `java.security.AccessControlException: access denied (java.io.FilePermission C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar read)`.

When a Java program receives this exception and adding this permission is justified, add a permission to the `was.policy` file: `grant codeBase "file:<user client installed location>" { permission java.io.FilePermission "C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar", "read"; };`.

To determine whether to add a permission, refer to the article, "Access control exception" on page 468.

Restart all applications for the updated `app.policy` file to take effect.

***Configuring spi.policy files:***

Java 2 security uses several policy files to determine the granted permission for each Java program. See "Java 2 security policy files" on page 472 for the list of available policy files supported by WebSphere Application Server Version 6.

Since the default permissions for Service Provider Interface (SPI) is AllPermission, the only reason to update the `spi.policy` file is a restricted SPI permission. When a change in the `spi.policy` is required, complete the following steps.

Syntax errors in the policy files cause the application server to fail. Edit these policy files carefully.

**Important:** Do not place the codebase keyword or any other keyword after the `filterMask` and `runtimeFilterMask` keywords. The Signed By and the Java Authentication and Authorization Service (JAAS) Principal keywords are not supported in the `spi.policy` file. The Signed By keyword is supported in the following policy files: `java.policy`, `server.policy`, and `client.policy`. The JAAS Principal keyword is supported in a JAAS policy file that is specified by the Java Virtual Machine (JVM) system property, `java.security.auth.policy`. You can statically set the authorization policy files in `java.security.auth.policy` with `auth.policy.url.n=`*`URL`* `where` *`URL`* is the location of the authorization policy.

Modify the `spi.policy` file with the Policy Tool.

The updated `spi.policy` is applied to the SPI libraries after the Java process is restarted.

The `spi.policy` file is the template for SPIs (Service Provider Interface) or third-party resources embedded in the product. Example of SPIs are Java Message Services (JMS) (MQSeries) and Java database connectivity (JDBC) drivers. They are specified in the `resources.xml` file. The dynamic policy grants the permissions defined in the `spi.policy` file to the class paths defined in the `resources.xml` file. The union of the permission contained in the `java.policy` file and the `spi.policy` file are applied to the SPI libraries. The `spi.policy` files are managed by configuration and file replication services. Changes made in these files are replicated to other nodes in the Network Deployment cell.

The `spi.policy` file supplied by WebSphere Application Server resides at *install_root*/profiles/*profile_name*/config/cells/*cell_name*/nodes/*node_name*/spi.policy. It contains the following default permission:

```
grant {
  permission  java.security.AllPermission;
};
```

Restart the related Java processes for the changes in the `spi.policy` file to become effective.

***Configuring library.policy files:***

Java 2 security uses several policy files to determine the granted permission for each Java program. See “Java 2 security policy files” on page 472 for the list of available policy files supported by WebSphere Application Server Version 5. The `library.policy` file is the template for shared libraries (Java library classes). Multiple enterprise applications can define and use shared libraries. Refer to Managing shared libraries for information on how to define and manage the shared libraries.

If the default permissions for a shared library (union of the permissions defined in the `java.policy` file, the `app.policy` file and the `library.policy` file) are enough, no action is required. The default library policy is picked up automatically. If a specific change is required to share a library in the cell, update the `library.policy` file.

Syntax errors in the policy files cause the application server to fail. Edit these policy files carefully.

**Important:** Do not place the codebase keyword or any other keyword after the grant keyword. The Signed By keyword and the Java Authentication and Authorization Service (JAAS) Principal keyword are not supported in the `library.policy` file. The Signed By keyword is supported in the following policy files: `java.policy`, `server.policy`, and `client.policy`. The JAAS Principal keyword is supported in a JAAS policy file when it is specified by the Java Virtual Machine (JVM) system property, `java.security.auth.policy`. You can statically set the authorization policy files in `java.security.auth.policy` with `auth.policy.url.n=`*`URL`* `where` *`URL`* is the location of the authorization policy.

An updated `library.policy` is applied to shared libraries after the servers restart.

The union of the permission contained in the `java.policy` file, the `app.policy` file, and the `library.policy` file are applied to the shared libraries. The `library.policy` file is managed by configuration and file replication services. Changes made in the file are replicated to other nodes in the Network Deployment cell.

The `library.policy` file supplied by WebSphere Application Server resides at: *install_root*/config/cells/*cell_name*/nodes/*node_name*/library.policy, contains an empty permission entry as a default. For example,

```
grant {
 };
```

If the shared library in a cell requires permissions that are not defined as defaults in the `java.policy` file, `app.policy` file and the `library.policy` file, update the `library.policy` file. The missing permission causes the exception, java.security.AccessControlException. The missing permission is listed in the exception data, for example:

```
 java.security.AccessControlException: access denied (java.io.FilePermission
C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar read)
```

The previous lines are one continuous line.

When a Java program receives this exception and adding this permission is justified, add a permission to the `library.policy` file, for example: `grant codeBase ″file:<user client installed location>″ { permission java.io.FilePermission ″C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar″, ″read″; };`

to decide whether to add a permission, refer to “Access control exception” on page 468.

Restart the related Java processes for the changes in the `library.policy` file to become effective.

***Adding the was.policy file to applications:***

When Java 2 security is enabled for a WebSphere Application Server, all the applications that run on that WebSphere Application Server undergo a security check before accessing system resources. An application might need a `was.policy` file if it accesses resources that require more permissions than those granted in the default `app.policy` file. By default, the product security reads an `app.policy` file that is located in each node and grants the permissions in the `app.policy` file to all the applications. Include any additional required permissions in the `was.policy` file. The `was.policy` file is only required if an application requires additional permissions.

The default policy file for all applications is specified in the `app.policy` file. This file is provided by the product security, is common to all applications, and should not be changed. Add any new permissions required for an application in the `was.policy` file.

The `app.policy` file is located in the *install_root*/config/cells/*cell_name*/nodes/*node_name* directory. The contents of the `app.policy` file follow:

**Attention:** In the following code sample, the two permissions that are required by JavaMail were split into two lines each due to the width of the printed page.

```
// The following permissions apply to all the components under the application.
grant codeBase ″file:${application}″ {
  // The following are required by JavaMail
  permission java.io.FilePermission ″
```

```
        ${was.install.root}${/}java${/}jre${/}lib${/}ext${/}mail.jar", "read";
  permission java.io.FilePermission "
        ${was.install.root}${/}java${/}jre${/}lib${/}ext${/}activation.jar", "read";
};

  // The following permissions apply to all utility .jar files  (other
  //  than enterprise beans JAR files) in the application.
grant codeBase "file:${jars}" {
  permission java.net.SocketPermission "*", "connect";
  permission java.util.PropertyPermission "*", "read";
};

// The following permissions apply to connector resources within the application
grant codeBase "file:${connectorComponent}" {
  permission java.net.SocketPermission "*", "connect";
  permission java.util.PropertyPermission "*", "read";
};

// The following permissions apply to all the Web modules (.war files)
// within the application.
grant codeBase "file:${webComponent}" {
  permission java.io.FilePermission "${was.module.path}${/}-", "read, write";
        //  where "was.module.path" is the path where the Web module is
        //  installed. Refer to Dynamic policy concepts for other symbols.
  permission java.lang.RuntimePermission "loadLibrary.*";
  permission java.lang.RuntimePermission "queuePrintJob";
  permission java.net.SocketPermission "*", "connect";
  permission java.util.PropertyPermission "*", "read";
};


// The following permissions apply to all the EJB modules within the application.
grant codeBase "file:${ejbComponent}" {
 permission java.lang.RuntimePermission "queuePrintJob";
 permission java.net.SocketPermission "*", "connect";
 permission java.util.PropertyPermission "*", "read";
};
```

If additional permissions are required for an application or for one or more modules of an application, use the `was.policy` file for that application. For example, use codeBase of ${application} and add required permissions to grant additional permissions to the entire application. Similarly, use codeBase of ${webComponent} and ${ejbComponent} to grant additional permissions to all the Web modules and all the enterprise bean (EJB) modules in the application. You can assign additional permissions to each module (`.war` file or `.jar` file) as shown in the following example.

An example of adding extra permissions for an application in the `was.policy` file:

**Attention:** In the following code sample, the permission for the EJB module was split into two lines due to the width of the printed page.

```
// grant additional permissions to a Web module
grant codeBase " file:aWebModule.war" {
 permission java.security.SecurityPermission "printIdentity";
};

// grant additional permission to an EJB module
```

```
grant codeBase "file:aEJBModule.jar"  {
     permission java.io.FilePermission "
         ${user.install.root}${/}bin${/}DefaultDB${/}-" ."read.write,delete";
     // where, ${user.install.root} is the system property whose value is
     // located in the <install_root> directory.
};
```

1. Create a `was.policy` file using the policy tool. For more information on using the policy tool, see "Using PolicyTool to edit policy files" on page 471
2. Add the required permissions in the `was.policy` file using the policy tool.
3. Place the `was.policy` file in the application enterprise archive (EAR) file under the `META-INF` directory. Update the application EAR file with the newly created `was.policy` file by using the **jar** command.
4. Verify that the `was.policy` file is inserted, and start an assembly tool. For more information, see Starting an assembly tool
   a. Verify that the `was.policy` file in the application is syntactically correct. In an assembly tool, right-click the enterprise application module and click **Run Validation**.

An application EAR file is now ready to run when Java 2 security is enabled.

This step is required for applications to run properly when Java 2 security is enabled. If the `was.policy` file is not created and it does not contain required permissions, the application might not access system resources.

The symptom of the missing permissions is the exception, `java.security.AccessControlException`. The missing permission is listed in the exception data, for example:

```
java.security.AccessControlException: access denied (java.io.FilePermission
C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar read)
```

The previous two lines are one continuous line.

When an application program receives this exception and adding this permission is justified, include the permission in the `was.policy` file, for example,

```
grant codeBase "file:${application}" { permission java.io.FilePermission
"C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar", "read"; };
```

The previous two lines are one continuous line.

Install the application.

## Configuring static policy files

Java 2 security uses several policy files to determine the granted permission for each Java program. See the "Java 2 security policy files" on page 472 article for the list of available policy files supported by WebSphere Application Server Version 5.

There are two types of policy files supported by WebSphere Application Server Version 5, dynamic policy files and static policy files. Static policy files provide the default permissions. Dynamic policy files provide application's permissions.

| Policy file name | Description |
| --- | --- |
| `java.policy` | Contains default permissions for all of the Java programs on the node. This file seldom changes. |

| Policy file name | Description |
|---|---|
| server.policy | Contains default permissions for all of the WebSphere Application Server programs on the node. This files is rarely updated. |
| client.policy | Contains default permissions for all of the applets and client containers on the node. |

The static policy file is not a configuration file managed by the repository and the file replication service. Changes to this file are local and do not get replicated to the other machine.

1. Identify the policy file to update.
   - If the permission is required only by an application, update the dynamic policy file. Refer to “Configuring Java 2 security policy files” on page 476.
   - If the permission is required only by applets and client containers, update the `client.policy` file. Refer to “Configuring client.policy files” on page 492.
   - If the permission is required only by WebSphere Application Server (servers, agents, managers and application servers), update the `server.policy` file. Refer to “Configuring server.policy files” on page 491.
   - If the permission is required by all of the Java programs running on the Java virtual machine (JVM), update the `java.policy` file. Refer to “Configuring java.policy files.”
2. Stop and restart the WebSphere Application Server.

The required permission is granted for all of the Java programs running with the restarted JVM.

If Java programs on a node require permissions, the policy file needs updating. If the Java program that required the permission is not part of an enterprise application, update the static policy file. The missing permission causes the exception, java.security.AccessControlException. The missing permission is listed in the exception data, for example:

```
java.security.AccessControlException: access denied (java.io.FilePermission
C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar read)
```

When a Java program receives this exception and adding this permission is justified, add a permission to an adequate policy file, for example:

```
grant codeBase "file:<user client installed location>" {
  permission java.io.FilePermission
  "C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar",
  "read";
};
```

To decide whether to add a permission, refer to “Access control exception” on page 468.

***Configuring java.policy files:***

Java 2 security uses several policy files to determine the granted permission for each Java program. See “Java 2 security policy files” on page 472 for the list of available policy files supported by WebSphere Application Server Version 5.x or later. The `java.policy` file is a global default policy file shared by all of the Java programs running in the Java Virtual Machine (JVM) on the node. Modifying this file is not recommended.

If a specific change is required to some of the Java programs on a node and the `java.policy` file requires updating, modify the java.policy file with policy tool. For more information, see “Using PolicyTool to edit

policy files" on page 471. A change to the `java.policy` file is local for the node. The default Java policy is picked up automatically. Syntax errors in the policy files cause the application server to fail. Edit these policy files carefully.

An updated `java.policy` file is applied to all the Java programs running in all the JVMs on the local node. Restart the programs for the updates to take effect

The `java.policy` file is not a configuration file managed by the repository and the file replication service. Changes to this file are local and do not get replicated to the other machine. The `java.policy` file supplied by WebSphere Application Server is located at *install_root*`/java/jre/lib/security/java.policy`. It contains these default permissions.

```
// Standard extensions get all permissions by default
grant codeBase "file:${java.home}/lib/ext/*" {
        permission java.security.AllPermission;
};
// default permissions granted to all domains
grant {
        // Allows any thread to stop itself using the java.lang.Thread.stop()
        // method that takes no argument.
        // Note that this permission is granted by default only to remain
        // backwards compatible.
        // It is strongly recommended that you either remove this permission
        // from this policy file or further restrict it to code sources
        // that you specify, because Thread.stop() is potentially unsafe.
        // See "http://java.sun.com/notes" for more information.
        // permission java.lang.RuntimePermission "stopThread";

        // allows anyone to listen on un-privileged ports
        permission java.net.SocketPermission "localhost:1024-", "listen";

        // "standard" properties that can be read by anyone

        permission java.util.PropertyPermission "java.version", "read";
        permission java.util.PropertyPermission "java.vendor", "read";
        permission java.util.PropertyPermission "java.vendor.url", "read";
        permission java.util.PropertyPermission "java.class.version", "read";
        permission java.util.PropertyPermission "os.name", "read";
        permission java.util.PropertyPermission "os.version", "read";
        permission java.util.PropertyPermission "os.arch", "read";
        permission java.util.PropertyPermission "file.separator", "read";
        permission java.util.PropertyPermission "path.separator", "read";
        permission java.util.PropertyPermission "line.separator", "read";

        permission java.util.PropertyPermission "java.specification.version", "read";
        permission java.util.PropertyPermission "java.specification.vendor", "read";
        permission java.util.PropertyPermission "java.specification.name", "read";

        permission java.util.PropertyPermission "java.vm.specification.version","read";
        permission java.util.PropertyPermission "java.vm.specification.vendor","read";
        permission java.util.PropertyPermission "java.vm.specification.name", "read";
        permission java.util.PropertyPermission "java.vm.version", "read";
        permission java.util.PropertyPermission "java.vm.vendor", "read";
        permission java.util.PropertyPermission "java.vm.name", "read";
       };
```

If some Java programs on a node require permissions that are not defined as defaults in the `java.policy` file, then consider updating the `java.policy` file. Most of the time, other policy files are updated instead of the `java.policy` file. The missing permission causes the exception, java.security.AccessControlException. The missing permission is listed in the exception data, for example:

```
java.security.AccessControlException: access denied (java.io.FilePermission
C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar read)
```

The previous two lines are one continuous line.

When a Java program receives this exception and adding this permission is justified, add a permission to the `java.policy`file, for example:

```
grant codeBase "file:<user client installed location>" {
permission java.io.FilePermission
"C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar", "read"; };
```

To decide whether to add a permission, refer to “Access control exception” on page 468.

Restart all of the Java processes for the updated `java.policy` file to take effect.

***Configuring server.policy files:***

Java 2 security uses several policy files to determine the granted permission for each Java program. See “Java 2 security policy files” on page 472 for the list of available policy files supported by WebSphere Application Server Version 5. The `server.policy` file is a default policy file shared by all of the WebSphere servers on a node. The `server.policy` file is not a configuration file managed by the repository and the file replication service. Changes to this file are local and do not replicate to the other machine.

If the default permissions for a server (the union of the permissions defined in the `java.policy` file and the `server.policy` file) are enough, no action is required. The default server policy is picked up automatically. If a specific change is required to some of the server programs on a node, update the `server.policy` file with the Policy Tool. Refer to the “Using PolicyTool to edit policy files” on page 471 article to edit policy files. Changes to the `server.policy` file are local for the node. Syntax errors in the policy files cause the application server to fail. Edit these policy files carefully.

An updated `server.policy` file is applied to all the server programs on the local node. Restart the servers for the updates to take effect.

If you want to add permissions to an application, use the `app.policy` file and the `was.policy` file.

When you do need to modify the `server.policy` file, locate this file at: *install_root*`/properties/server.policy`. This file contains these default permissions:

```
// Allow to use sun tools
grant codeBase "file:${java.home}/../lib/tools.jar" {
  permission java.security.AllPermission;
};

// WebSphere system classes
grant codeBase "file:${was.install.root}/lib/-" {
  permission java.security.AllPermission;
};
grant codeBase "file:${was.install.root}/classes/-" {
  permission java.security.AllPermission;
};
```

```
// Allow the WebSphere deploy tool all permissions
grant codeBase "file:${was.install.root}/deploytool/-" {
  permission java.security.AllPermission;
};
```

If some server programs on a node require permissions that are not defined as defaults in the `server.policy` file and the `server.policy` file, update the `server.policy` file. The missing permission causes the exception, java.security.AccessControlException. The missing permission is listed in the exception data, for example:

```
java.security.AccessControlException: access denied (java.io.FilePermission
C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar read)
```

The previous two lines are one continuous line.

When a Java program receives this exception and adding this permission is justified, add a permission to the `server.policy` file, for example:

```
grant codeBase "file:<user client installed location>" {
permission java.io.FilePermission
"C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar", "read"; };
```

To decide whether to add a permission, refer to "Access control exception" on page 468.

Restart all of the Java processes for the updated `server.policy` file to take effect.

***Configuring client.policy files:***

Java 2 security uses several policy files to determine the granted permission for each Java program. See "Java 2 security policy files" on page 472 for the list of available policy files supported by WebSphere Application Server Version 5. The `client.policy` file is a default policy file shared by all of the WebSphere Application Server client containers and applets on a node. The union of the permissions contained in the `java.policy` file and the `client.policy` file are given to all of the WebSphere client containers and applets running on the node. The `client.policy` file is not a configuration file managed by the repository and the file replication service. Changes to this file are local and do not replicate to the other machine. The `client.policy` file supplied by WebSphere Application Server is located at *install_root*/profiles/*profile_name*/properties/client.policy. It contains these default permissions:

```
grant codeBase "file:${java.home}/lib/ext/*" {
  permission java.security.AllPermission;
};
// IBM Developer Kit, Java Technology Edition classes
grant codeBase "file:${java.home}/lib/ext/-" {
  permission java.security.AllPermission;
};
grant codeBase "file:${java.home}/../lib/tools.jar" {
  permission java.security.AllPermission;
};
// WebSphere system classes
grant codeBase "file:${was.install.root}/lib/-" {
  permission java.security.AllPermission;
};
grant codeBase "file:${was.install.root}/classes/-" {
  permission java.security.AllPermission;
};
```

```
grant codeBase "file:${was.install.root}/installedConnectors/-" {
  permission java.security.AllPermission;
};
// J2EE 1.3 permissions for client container WAS applications
// in $WAS_HOME/installedApps
grant codeBase "file:${was.install.root}/installedApps/-" {
  //Application client permissions
  permission java.awt.AWTPermission "accessClipboard";
  permission java.awt.AWTPermission "accessEventQueue";
  permission java.awt.AWTPermission "showWindowWithoutWarningBanner";
  permission java.lang.RuntimePermission "exitVM";
  permission java.lang.RuntimePermission "loadLibrary";
  permission java.lang.RuntimePermission "queuePrintJob";
  permission java.net.SocketPermission "*", "connect";
  permission java.net.SocketPermission "localhost:1024-", "accept,listen";
  permission java.io.FilePermission "*", "read,write";
  permission java.util.PropertyPermission "*", "read";
};
// J2EE 1.3 permissions for client container - expanded ear file code base
grant codeBase "file:${com.ibm.websphere.client.applicationclient.archivedir}/-"
 {
  permission java.awt.AWTPermission "accessClipboard";
  permission java.awt.AWTPermission "accessEventQueue";
  permission java.awt.AWTPermission "showWindowWithoutWarningBanner";
  permission java.lang.RuntimePermission "exitVM";
  permission java.lang.RuntimePermission "loadLibrary";
  permission java.lang.RuntimePermission "queuePrintJob";
  permission java.net.SocketPermission "*", "connect";
  permission java.net.SocketPermission "localhost:1024-", "accept,listen";
  permission java.io.FilePermission "*", "read,write";
  permission java.util.PropertyPermission "*", "read";
};
// For MQ Series
grant codeBase "file:${mq.install.root}/java/*" {
  permission java.security.AllPermission;
};
```

1. If the default permissions for a client (union of the permissions defined in the `java.policy` file and the `client.policy` file) are enough, no action is required. The default client policy is picked up automatically.
2. If a specific change is required to some of the client containers and applets on a node, modify the `client.policy` file with the policy tool. Refer to "Using PolicyTool to edit policy files" on page 471, to edit policy files. Changes to the `client.policy` file are local for the node.

All of the client containers and applets on the local node are granted the updated permissions at the time of execution.

If some client containers or applets on a node require permissions that are not defined as defaults in the `java.policy` file and the default `client.policy` file, update the `client.policy` file. The missing permission causes the exception, java.security.AccessControlException. The missing permission is listed in the exception data, for example,

```
java.security.AccessControlException: access denied (java.io.FilePermission
C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar read)
```

The previous two lines of sample code are one continuous line, but extended beyond the width of the page.

When a client program receives this exception and adding this permission is justified, add a permission to the `client.policy` file, for example, grant codebase ″file:*user_client_installed_location*″ { permission java.io.FilePermission ″C:\WebSphere\AppServer\java\jre\lib\ext\mail.ja″, ″read″; };.

To decide whether to add a permission, refer to “Access control exception” on page 468.

Close and restart the browser. You also must restart the client application if you have one.

## Migrating Java 2 security policy

### Previous WebSphere Application Server releases

Starting from Version 3.x, WebSphere Application Server installed a Java 2 security manager in the server run time to prevent enterprise applications from calling the `System.exit()` and the `System.setSecurityManager()` methods. These two Java APIs have undesirable consequences if called by enterprise applications. The `System.exit()` API, for example, causes the Java virtual machine (application server process) to exit prematurely, which is an undesirable operation for an application server.

However, Java 2 security was not a fully supported feature prior to Version 5. To support Java 2 security properly, all the server run time must be marked as `privileged` (with `doPrivileged()` API calls inserted in the correct places), and identify the default permission sets or policy. Application code is not privileged and subject to the permissions defined in the policy files. The `doPrivileged` instrumentation is important and necessary to support Java 2 security. Without it, the application code must be granted the permissions required by the server run time. This is due to the design and algorithm used by Java 2 security to enforce permission checks. Please refer to the Java 2 security check permission algorithm.

The following two permissions are enforced by the WebSphere Java 2 security manager (hard coded):
- `java.lang.RuntimePermission(exitVM)`
- `java.lang.RuntimePermission(setSecurityManager)`

Application code is denied access to these permissions regardless of what is in the Java 2 security policy. However, the server run time is granted these permissions. All the other permission checks are not enforced.

Partial support was introduced since the version 4.02 product release. Prior to version 4.0.2, Java 2 security was not supported. From version 4.02 and later, only two permissions are supported:
- `java.net.SocketPermission`
- `java.net.NetPermission`

However, not all the product server run time is properly marked as `privileged`. You must grant the application code all the other permissions besides the two listed previously or the enterprise application can potentially fail to run. This Java 2 security policy for enterprise applications is liberal.

### What changed

Java 2 Security is fully supported in version 5.x and later, which means all permissions are enforced. The default Java 2 security policy for enterprise application is the recommended permission set defined by the J2EE 1.4 specification. Refer to the *install_root*/profiles/*profile_name*/config/cells/*cell_name*/nodes/*node_name*/app.policy file for the default Java 2 security policy granted to enterprise applications. This is a much more stringent policy compared to previous releases.

All policy is declarative. The product security manager honors all policy declared in the policy files. There is an exception to this rule: enterprise applications are denied access to permissions declared in the *install_root*profiles/*profile_name*/config/cells/*cell_name*/filter.policy file.

**Note:** Enterprise applications that run on Version 4.0.x with Java 2 security enabled are not guaranteed to run successfully when migrating to Version 5 (when Java 2 security is enabled), even if the Java 2 security policy is migrated properly. The default Java 2 security policy for enterprise applications is much more stringent and all permissions are enforced in Version 5. It might fail because the application code does not have the necessary permissions granted where system resources (such as file I/O for example) can be programmatically accessed and are now subject to the permission checking.

**Migrating system properties**

The following system properties are used in previous releases in relation to Java 2 security:

- **java.security.policy**. The absolute path of the policy file (action required). It contains both system permissions (permissions granted to the Java Virtual Machine (JVM) and the product server run time) and enterprise application permissions. Migrate the Java 2 security policy of the enterprise application to Version 5. For Java 2 security policy migration, see the steps for migrating Java 2 security policy.
- **enableJava2Security**. Used to enable Java 2 security enforcement (no action required). This is deprecated; a flag in the WebSphere configuration application programming interface (API) is used to control whether to enabled Java 2 security. Enable this option through the administrative console.
- **was.home**. Expanded to the installation directory of the WebSphere Application Server (action might be required). This is deprecated; superseded by ${user.install.root} and ${was.install.root} properties. If the directory contains instance specific data then ${user.install.root} is used; otherwise ${was.install.root} is used. Use these properties interchangeably for the WebSphere Application Server or the Network Deployment environments. See the steps for migrating Java 2 security policy.

**Migrating the Java 2 Security Policy**

There is no easy way of migrating the Java policy file from Version 4.0.x automatically because there is a mixture of system permissions and application permissions in the same policy file. Manually copy the Java 2 security policy for enterprise applications to a was.policy or app.policy file. However, migrating the Java 2 security policy to a was.policy file is preferable because symbols or relative codebase is used instead of absolute codebase. There are many advantages to this process. The permissions defined in the was.policy file should only be granted to the specific enterprise application, while permissions in the app.policy file apply to all the enterprise applications running on the node where the app.policy file belongs. Refer to the “Java 2 security policy files” on page 472 article for more details on policy management.

The following example illustrates the migration of a Java 2 security policy from a previous release. The contents include the Java 2 security policy file (the default is *install_root*profiles *profile_name*/properties/java.policy) for the app1.ear enterprise application and the system permissions (permissions granted to the JVM and product server run time). Default permissions are omitted for clarity:

```
// For product Samples
   grant codeBase "file:${install_root}/installedApps/app1.ear/-" {
     permission java.security.SecurityPermission "printIdentity";
     permission java.io.FilePermission "${install_root}${/}temp${/}somefile.txt",
       "read";
   };
```

For clarity of illustration, all the permissions are migrated as the application level permissions in this example. However, you can grant permissions at a more granular level at the component level (Web, enterprise beans, connector or utility Java archive (JAR) component level) or you can grant permissions to a particular component.

1. Ensure that Java 2 security is disabled on the application server.
2. Create a new `was.policy` file (if one is not present) or update the `was.policy` for migrated applications in the configuration repository in (`profiles/`*profile_name*`config/cells/`*cell_name*`/applications/app.ear/deployments/app/META-INF/was.policy`) with the following contents:

   ```
   grant codeBase "file:${application}" {
        permission java.security.SecurityPermission "printIdentity";
        permission java.io.FilePermission "
                ${user.install.root}${/}temp${/}somefile.txt", "read";
      };
   ```

   The third and fourth lines in the previous code sample are one continuous line, but extended beyond the width of the page.
3. Use an assembly tool to attach the `was.policy` file to the enterprise archive (EAR) file. You also can use an assembly tool to validate the contents of the `was.policy` file. For more information, see "Configuring the was.policy file" on page 482.
4. Validate that the enterprise application does not require additional permissions to the migrated Java 2 Security permissions and the default permissions set declared in the `${was.install.root}profiles/`*profile_name*`/config/cells/`*cell_name*`/nodes/`*node_name*`/app.policy` file. This requires code review, code inspection, application documentation review, and sandbox testing of migrated enterprise applications with Java 2 security enabled in a pre-production environment. Refer to developer kit APIs protected by Java 2 security for information about which APIs are protected by Java 2 security. If you use third party libraries, consult the vendor documentation for APIs that are protected by Java 2 security. Verify that the application is granted all the required permissions, or it might fail to run when Java 2 security is enabled.
5. Perform pre-production testing of the migrated enterprise application with Java 2 security enabled. **Hint**: Enable trace for the WebSphere Application Server Java 2 security manager in the pre-production testing environment (with trace string: `com.ibm.ws.security.core.SecurityManager=all=enabled`). This can be helpful in debugging the *AccessControlException* exception thrown when an application is not granted the required permission or some system code is not properly marked as *privileged*. The trace dumps the stack trace and permissions granted to the classes on the call stack when the exception is thrown. For more information, see "Access control exception" on page 468.

   **Note:** Because the Java 2 security policy is much more stringent compared with previous releases, it is strongly advised that the administrator or deployer review their enterprise applications to see if extra permissions are required before enabling Java 2 security. If the enterprise applications are not granted the required permissions, they fail to run.

# Chapter 13. Configuring security with scripting

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

If you enable security for a WebSphere Application Server cell, supply authentication information to communicate with servers.

The `sas.client.props` and the `soap.client.props` files are located in the properties directory for each WebSphere Application Server profile, *profilePath*/`properties`.

- The nature of the properties file updates required for running in secure mode depend on whether you connect with a Remote Method Invocation (RMI) connector, or a Simple Object Access Protocol (SOAP) connector:
  - If you use a Remote Method Invocation (RMI) connector, set the following properties in the `sas.client.props` file with the appropriate values:

    ```
    com.ibm.CORBA.loginUserid=
    com.ibm.CORBA.loginPassword=
    ```

    Also, set the following property:

    ```
    com.ibm.CORBA.loginSource=properties
    ```

    The default value for this property is `prompt` in the `sas.client.props` file. If you leave the default value, a dialog box appears with a password prompt. If the script is running unattended, it appears to hang.
  - If you use a Simple Object Access Protocol (SOAP) connector, set the following properties in the `soap.client.props` file with the appropriate values:

    ```
    com.ibm.SOAP.securityEnabled=true
    com.ibm.SOAP.loginUserid=
    com.ibm.SOAP.loginPassword=
    ```
- To specify user and password information, choose one of the following methods:
  - Specify user name and password on a command line, using the **-user** and **-password** commands. For example:

    ```
    wsadmin.sh -conntype RMI -port 2809 -user u1 -password secret1
    ```
  - Specify user name and password in the `sas.client.props` file for a RMI connector or the `soap.client.props` file for a SOAP connector.

  If you specify user and password information on a command line and in the `sas.client.props` file or the `soap.client.props` file, the command line information overrides the information in the `props` file.

  Warning: On UNIX system, the use of -password option may result in security exposure as the password information becomes visible to the system status program such as ps command which can be invoked by other user to display all the running processes. Do not use this option if security exposure is a concern. Instead, specify user and password information in the soap.client.props file for SOAP connector or sas.client.props file for RMI connector. The soap.client.props and sas.client.props files are located in the properties directory of your WebSphere profile.

## Enabling and disabling global security using scripting

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

The default profile sets up procedures so that you can enable and disable global security based on LocalOS registry.

- You can use the **help** command to find out the arguments that you need to provide with this call, for example:

– Using Jacl:

```
securityon help
```

Example output:

```
Syntax: securityon user password
```

– Using Jython:

```
securityon()
```

Example output:

```
Syntax: securityon(user, password)
```

- To enable global security based on the LocalOS registry, use the following procedure call and arguments:
  – Using Jacl:

```
securityon user1 password1
```

  – Using Jython:

```
securityon('user1', 'password1')
```

- To disable global security based on the LocalOS registry, use the following procedure call:
  – Using Jacl:

```
securityoff
```

  – Using Jython:

```
securityoff()
```

## Enabling and disabling Java 2 security using scripting

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

Perform the following steps to enable or disable Java 2 security:

1. Identify the security configuration object and assign it to the security variable:
   - Using Jacl:

```
set security [$AdminConfig list Security]
```

   - Using Jython:

```
security = AdminConfig.list('Security')
print security
```

   Example output:

```
(cells/mycell|security.xml#Security_1)
```

2. Modify the enforceJava2Security attribute to enable or disable Java 2 security. For example:
   - To enable Java 2 security:
     – Using Jacl:

```
$AdminConfig modify $security {{enforceJava2Security true}}
```

     – Using Jython:

```
AdminConfig.modify(security, [['enforceJava2Security', 'true']])
```

   - To disable Java 2 security:
     – Using Jacl:

```
$AdminConfig modify $security {{enforceJava2Security false}}
```

     – Using Jython:

```
AdminConfig.modify(security, [['enforceJava2Security', 'false']])
```

3. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.

4. In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

# Chapter 14. Learn about WebSphere applications

Use this section as a starting point to investigate the technologies used in and by applications that you deploy on the application server.

See Learn about WebSphere applications: Overview and new features for an introduction to each technology.

| | | | | |
|---|---|---|---|---|
| Web applications | How do I?... | Overview | | Samples |
| EJB applications | How do I?... | Overview | Tutorials | Samples |
| Client applications | How do I?... | Overview | | Samples |
| Web services | How do I?... | Overview | Tutorials | Samples |
| Data access resources | How do I?... | Overview | Tutorials | Samples |
| Messaging resources | How do I?... | Overview | Tutorials | Samples |
| Mail, URLs, and other J2EE resources | How do I?... | Overview | | |
| Security | How do I?... | Overview | Tutorials | Samples |
| Naming and directory | How do I?... | Overview | | |
| Object Request Broker | How do I?... | Overview | | |
| Transactions | How do I?... | Overview | | Samples |
| ActivitySessions | How do I?... | Overview | | Samples |
| Application profiling | How do I?... | Overview | | Samples |
| Asynchronous beans | How do I?... | Overview | | Samples |
| Dynamic caching | How do I?... | Overview | | |
| Dynamic query | How do I?... | Overview | | Samples |
| Internationalization | How do I?... | Overview | | Samples |
| Object pools | How do I?... | Overview | | |
| Scheduler | How do I?... | Overview | | Samples |
| Startup beans | How do I?... | Overview | | |
| Work areas | How do I?... | Overview | | |

## Web applications

## Security constraints

Security constraints determine how Web content is to be protected.

These properties associate security constraints with one or more Web resource collections. A constraint consists of a Web resource collection, an authorization constraint and a user data constraint.

- A Web resource collection is a set of resources (URL patterns) and HTTP methods on those resources. All requests that contain a request path that matches the URL pattern described in the Web resource collection are subject to the constraint. If no HTTP methods are specified, then the security constraint applies to all HTTP methods.

© Copyright IBM Corp. 2004

- An authorization constraint is a set of roles that users must be granted in order to access the resources described by the Web resource collection. If a user who requests access to a specified URI is not granted at least one of the roles specified in the authorization constraint, the user is denied access to that resource.
- A user data constraint indicates that the transport layer of the client or server communications process must satisfy the requirement of either guaranteeing content integrity (preventing tampering in transit) or guaranteeing confidentiality (preventing reading while in transit).

# EJB applications

## Configuring security for message-driven beans that use listener ports

Use this task to configure resource security and security permissions for EJB 2.0 message-driven beans deployed to use listener ports.

Messages arriving at a listener port have no client credentials associated with them. The messages are anonymous.

To call secure enterprise beans from a message-driven bean, the message-driven bean needs to be configured with a RunAs Identity deployment descriptor. Security depends on the role specified by the RunAs Identity for the message-driven bean as an EJB component.

For more information about EJB security, see EJB component security. For more information about configuring security for your application, see Assembling secured applications.

JMS connections used by message-driven beans can benefit from the added security of using J2C container-managed authentication. To enable the use of J2C container authentication aliases and mapping, define a J2C container-managed alias on the JMS connection factory definition that the message-driven bean is using to listen upon (defined by the **Connection factory JNDI name** property of the listener port). If defined, the listener uses the container-managed authentication alias for its JMSConnection security credentials instead of any application-managed alias. To set the container-managed alias, use the administrative console to complete the following steps:

1. To display the listener port settings, click **Servers** → **Application Servers** → *application_server* → **[Communications] Messaging** → **Message Listener Service** → **Listener Ports** → *listener_port*
2. To get the name of the JMS connection factory, look at the **Connection factory JNDI name** property.
3. Display the JMS connection factory properties. For example, to display the properties of a WebSphere queue connection factory provided by the default messaging provider, click **Resources** → **JMS Providers** → **Default Messaging Provider** → → **[Content pane] WebSphere Queue Connection Factories** → *connection_factory*
4. Set the **Authentication alias** property.
5. Click **OK**

## Configuring security for EJB 2.1 message-driven beans

Use this task to configure resource security and security permissions for EJB 2.1 message-driven beans.

Messages handled by message-driven beans have no client credentials associated with them. The messages are anonymous.

To call secure enterprise beans from a message-driven bean, the message-driven bean needs to be configured with a RunAs Identity deployment descriptor. Security depends on the role specified by the RunAs Identity for the message-driven bean as an EJB component.

For more information about EJB security, see EJB component security. For more information about configuring security for your application, see Assembling secured applications.

Connections used by message-driven beans can benefit from the added security of using J2C container-managed authentication. To enable the use of J2C container authentication aliases and mapping, define an authentication alias on the J2C activation specification that the message-driven bean is configured with. If defined, the message-driven bean uses the authentication alias for its JMSConnection security credentials instead of any application-managed alias.

To set the authentication alias, you can use the administrative console to complete one the following steps. This task description assumes that you have already created an activation specification. If you want to create a new activation specification, see the related tasks.

- For a message-driven bean listening on a JMS destination of the default messaging provider, set the authentication alias on a JMS activation specification.
  1. To display the JMS activation specification settings, click **Resources** → **JMS Providers** → **Default messaging** → **[Activation Specifications] JMS activation specification**
  2. If you have already created a JMS activation specification, click its name in the list displayed. Otherwise, click **New** to create a new JMS activation specification.
  3. Set the **Authentication alias** property.
  4. Click **OK**
  5. Save your changes to the master configuration.
- For a message-driven bean listening on a destination (or endpoint) of another JCA provider, set the authentication alias on a J2C activation specification.
  1. To display the J2C activation specification settings, click **Resources** → **Resource Adapters** → **adapter_name** → **J2C Activation specifications** → **activation specification_name**
  2. Set the **Authentication alias** property.
  3. Click **OK**
  4. Save your changes to the master configuration.

# Client applications

## Accessing secure resources using SSL and applet clients

By default, the applet client is configured to have security enabled. If you have global security turned on at the server from which you are accessing resources, then you can use secure sockets layer (SSL) when needed. If you decide that the security requirements for the applet differ from other application client types, then create a new version of the `sas.client.props` file.

1. Make a copy of the following file so that you can use it for an applet:

   `<product_install_directory>/properties/sas.client.props`
2. Edit the copy of `sas.client.props` file that you made with your changes.
3. Click **Start** > **Control panel** > select the product Java plug-in to open the Java control panel.
   - To use the file you created in step 1, modify the following value:

```
-Dcom.ibm.CORBA.ConfigURL=file:<product_install_directory>\properties\sas.client.props
```

   For more information on the `sas.client.props` file and WebSphere Application Server security, see the Security section of the information center.

### Applet client security requirements

When code is loaded, it is assigned permissions based on the security policy in effect. This policy specifies the permissions that are available for code from various locations. You can initialize this policy from an external policy file. By default, the client uses the `<product_installation_dir>/properties/client.policy` file. You must update this file with the following permission:

SocketPermission grants permission to open a port and make a connection to a host machine, which is your WebSphere Application Server. In the following example, `yourserver.yourcompany.com` is the complete host name of your WebSphere Application Server:

```
permission java.util.PropertyPermission "*", "read";
permission java.net.SocketPermission "yourserver.yourcompany.com ,"connect";
```

# Web services

## Transport level security

*Transport level security* is based on Secure Sockets Layer (SSL) or Transport Layer Security (TLS) that runs beneath HTTP.

Transport level security can be used to secure Web services messages. It is orthogonal to the security support provided by WS-Security or HTTP Basic Authentication.

SSL and TLS provide security features including authentication, data protection, and cryptographic token support for secure HTTP connections. To run with HTTPS, the service port address must be in the form `https://`.

The integrity and confidentiality of transport data, including SOAP messages and HTTP basic authentication, is confirmed when you use SSL and TLS. See Secure Sockets Layer for more information.

Web services applications can also use Federal Information Processing Standard (FIPS) approved ciphers for more secure TLS connections. For information on FIPS, see “Global security settings” on page 145.

WebSphere Application Server uses the Java Secure Sockets Extension (JSSE) package to support SSL and TLS.

## Configuring HTTP outbound transport level security with the administrative console

This topic explains how to configure HTTP outbound transport level security with the administrative console.

This task is one of three ways that you can configure the HTTP outbound transport level security for a Web service acting as a client to another Web service client. You can also configure the HTTP outbound transport level security with an assembly tool or by using the Java properties.

Configuring the HTTP outbound transport-level security for a Web service is based on the Secured Sockets Layer (SSL) configuration repertoires of the WebSphere Application Server. Review Configuring Secure Sockets Layer for more information.

If you choose to configure the HTTP outbound transport level security with the administrative console or an assembly tool, the Web services security binding information is modified. You can use the administrative console to configure the Web services client security bindings if you have deployed or installed the Web services application into WebSphere Application Server. If you have not installed the Web services application, you can configure the HTTP SSL configuration with an assembly tool. This task assumes that you have deployed the Web services application into the WebSphere product.

If you configure the HTTP outbound transport level security using Java properties, the properties are configured as system properties. However, the configuration that is specified in the binding takes precedence over the Java properties.

Configure the HTTP outbound transport level security with the following steps provided in this task section.

Open the administrative console.

1. Click **Applications** > **Enterprise Applications** > *application_instance* > **Web Modules** or **EJB Modules** > *module_instance* > **Web Services Client Security Bindings**.
2. Click **HTTP SSL Configuration** to access the HTTP SSL configuration panel. Select **HTTP SSL enabled**. Select the SSL configuration from the list in the HTTP Basic Authentication panel.

You have configured the HTTP outbound transport level security for a Web service acting as a client to another Web service with the administrative console.

### HTTP SSL Configuration collection

Use this page to configure transport-level Secure Sockets Layer (SSL) security. You can use this configuration when a Web service is a client to another Web service.

You can use transport-level security to enable HTTP SSL (or HTTPS). Transport-level security can be enabled or disabled independently from message-level security. Because transport-level security provides minimal security, use message-level security when security is essential to the Web service application.

To view this administrative console page, complete the following steps:

1. Click **Applications > Enterprise Applications >***application_name*.
2. Under Related Items, click **Web module >** *URI_file_name* **> Web Services: Client Security Bindings**.
3. Under HTTP SSL Configuration, click **Edit**.

***HTTP SSL enabled:***

Specifies secure socket communications for the HTTP transport for this port. When enabled, WebSphere Application Server uses the HTTP SSL Configuration setting.

***HTTP SSL configuration:***

Specifies which alias of the SSL configuration to use with the HTTP transport for this port.

This option is used if you select **HTTP SSL Enabled**. SSL aliases are defined in the Secure Sockets Layer configuration repertoire, which you can configure by clicking **Security > SSL**.

## Configuring HTTP outbound transport level security with an assembly tool

This topic explains how to configure the HTTP outbound transport level security with an assembly tool.

You can configure HTTP outbound transport level security with assembly tools provided with WebSphere Application Server.

You must configure the assembly tool before you can use it.

This task is one of three ways that you can configure the HTTP outbound transport level security for a Web Service acting as a client to another Web service. You can also configure the HTTP outbound transport level security with the administrative console or by using the Java properties.

The configuration of HTTP outbound transport-level security for a Web service is based on the Secured Sockets Layer (SSL) configuration repertoires of the WebSphere Application Server. Review Configuring Secure Sockets Layer for more information.

If you choose to configure the HTTP outbound transport level security with assembly tool or with the administrative console, the Web services security binding information is modified. If you have not yet

installed the Web services application into WebSphere Application Server, you can configure the HTTP SSL configuration with an assembly tool. This task assumes that you have not deployed the Web services application into the WebSphere product.

If you configure the HTTP outbound transport level security using the Java properties, the properties are configured as system properties. However, the configuration specified in the binding takes precedence over the Java properties.

Configure the HTTP outbound transport level security with the following steps provided in this task section.

1. Start an assembly tool. The assembly tools, Application Server Toolkit (AST) and Rational Web Developer, provide a graphical interface for developing code artifacts, assembling the code artifacts into various archives (modules) and configuring related Java 2 Platform, Enterprise Edition (J2EE) Version 1.2, 1.3 or 1.4 compliant deployment descriptors.
2. Configure the HTTP outbound transport level security in the Web Services Client Port Binding page for a Web service client. The Web Services Client Port Binding page is available after double-clicking the client deployment descriptor file.

You have configured the HTTP outbound transport level security for a Web Service acting as a client to another Web service with an assembly tool.

## Configuring HTTP outbound transport-level security using Java properties

This topic explains how to configure the HTTP outbound transport level security for a Web service using Java properties

This task is one of three ways that you can configure HTTP outbound transport-level security for a Web service that is acting as a client to another Web service. You can also configure the HTTP outbound transport level security with the administrative console or an assembly tool. However, you can also use this task to configure the HTTP outbound transport-level security for a Web service client.

If you choose to configure the HTTP outbound transport-level security with the administrative console or an assembly tool, the Web services security binding information is modified.

If you configure the HTTP outbound transport-level security using Java properties, the properties are configured as system properties. However, the configuration specified in the binding takes precedence over the Java properties.

You can configure the HTTP outbound transport-level security using WebSphere SSL properties or JSSE SSL properties. However, the WebSphere SSL properties take precedence over the JSSE SSL properties.

Configure the HTTP outbound transport-level security with the following steps provided in this task section.

1. Create a property file that includes the following properties:
   ```
   com.ibm.ssl.protocol
   com.ibm.ssl.keyStoreType
   com.ibm.ssl.keyStore
   com.ibm.ssl.keyStorePassword
   com.ibm.ssl.trustStoreType
   com.ibm.ssl.trustStore
   com.ibm.ssl.trustStorePassword
   ```
2. Set the `com.ibm.webservices.sslConfigURL` Java system property to the absolute path of the created property file. If no WebSphere SSL properties are defined, the JSSE SSL properties are used. Set the JSSE SSL properties as JVM custom properties. See Using Java Secure Socket Extension and Java Cryptography Extension with servlets and enterprise bean files for more information about setting the JSSE SSL properties.

You have configured the HTTP outbound transport-level security for a Web service acting as a client to another Web service.

## HTTP basic authentication

*HTTP basic authentication* uses a user name and password to authenticate a service client to a secure endpoint.

WebSphere Application Server can have several resources, including Web services, protected by a Java 2 Platform, Enterprise Edition (J2EE) security model.

HTTP basic authentication is orthogonal to the security support provided by WS-Security or HTTP Secure Sockets Layer (SSL) configuration.

A simple way to provide authentication data for the service client is to authenticate to the protected service endpoint using HTTP basic authentication. The basic authentication is encoded in the HTTP request that carries the SOAP message. When the application server receives the HTTP request, the user name and password are retrieved and verified using the authentication mechanism specific to the server.

Although the basic authentication data is base64-encoded, sending data over HTTPS is recommended. The integrity and confidentiality of the data can be protected by the SSL protocol.

In some cases, a firewall is present using a pass-thru HTTP proxy server. The HTTP proxy server forwards the basic authentication data into the J2EE application server. The proxy server can also be protected. Applications can specify the proxy data by setting properties in a stub object.

## Configuring HTTP basic authentication with the administrative console

This topic explains how to configure HTTP basic authentication with the administrative console.

This task is one of three ways that you can configure HTTP basic authentication. You can also configure HTTP basic authentication with an assembly tool or by modifying the HTTP properties programmatically.

If you choose to configure HTTP basic authentication with the administrative console or an assembly tool, the Web services security binding information is modified. You can use the administrative console to configure HTTP basic authentication if you have deployed or installed the Web services application into WebSphere Application Server. If you have not installed the Web services application, then you can configure the security bindings with an assembly tool. This task assumes that you have deployed the Web services application into the WebSphere product.

If you configure HTTP basic authentication programmatically, the properties are configured in the Stub or Call instance. The values set programmatically take precedence over the values defined in the binding. However, you only can programmatically configure HTTP proxy authentication.

The HTTP basic authentication that is discussed in this topic is orthogonal to WS-Security and is distinct from basic authentication that WS-Security supports. WS-Security supports basic authentication token, not HTTP basic authentication.

Configure HTTP basic authentication with the following steps provided in this task section.

1. Open the administrative console.
   a. Click **Applications** > **Enterprise Applications** > *application_instance* > **Web Modules** or **EJB Modules** > *module_instance* > **Web Services Client Bindings**.
   b. Click **HTTP Basic Authentication** to access the HTTP basic authentication panel. Enter the values in the HTTP Basic Authentication panel.
2. Click **Applications** > **Enterprise Applications** > *application_instance*. Under Additional Properties, click **Publish WSDL files** which brings you to the **Publish WSDL zip files** panel.

### HTTP basic authentication collection

Use this page to specify a user name and password for transport-level basic authentication security for this port. You can use this configuration when a Web service is a client to another Web service.

You can use transport-level security to enable basic authentication. Transport-level security can be enabled or disabled independently from message-level security. Because transport-level security provides minimal security, use message-level security when security is essential to the Web service application.

To view this administrative console page, complete the following steps:

1. Click **Applications > Enterprise Applications >***application_name*.
2. Under Related Items, click **Web module >***URI_file_name* **> Web Services: Client Security Bindings**.
3. Under HTTP Basic Authentication, click **Edit**.

***Basic authentication ID:***

Specifies the user name for the HTTP basic authentication for this port.

***Basic authentication password:***

Specifies the password for the HTTP basic authentication for this port.

## Configuring HTTP basic authentication with an assembly tool

This topic explains how to configure HTTP basic authentication with an assembly tool.

You can configure HTTP basic authentication with assembly tools provided with WebSphere Application Server.

You must configure the assembly tool before you can use it.

This task is one of three ways that you can configure HTTP basic authentication. You can also configure HTTP basic authentication with the administrative console or by modifying the HTTP properties programmatically.

If you choose to configure the HTTP basic authentication with an assembly tool or with the administrative console , the Web services security binding information is modified. You can use an assembly tool to configure HTTP basic authentication before you deploy or install the Web services application into WebSphere Application Server. This task assumes that you have not deployed the Web services application into the WebSphere product.

If you configure HTTP basic authentication programmatically, the properties are configured in the Stub or Call instance. The values set programmatically take precedence over the values defined in the binding. However, you only can programmatically configure HTTP proxy authentication.

The HTTP basic authentication that is discussed in this topic is orthogonal to WS-Security and is distinct from basic authentication that WS-Security supports. WS-Security supports basic authentication token, not HTTP basic authentication.

To configure HTTP basic authentication, use the WebSphere Application Server tools to modify the binding information.

1. Start an assembly tool. The assembly tools, Application Server Toolkit (AST) and Rational Web Developer, provide a graphical interface for developing code artifacts, assembling the code artifacts into various archives (modules) and configuring related Java 2 Platform, Enterprise Edition (J2EE) Version 1.2, 1.3 or 1.4 compliant deployment descriptors.

2. Configure the HTTP basic authentication in the Web Services Client Port Binding page for a Web service or a Web service client. The Web Services Client Port Binding page is available after double-clicking the client deployment descriptor file.

### HTTP basic authentication collection

Use this page to specify a user name and password for transport-level basic authentication security for this port. You can use this configuration when a Web service is a client to another Web service.

You can use transport-level security to enable basic authentication. Transport-level security can be enabled or disabled independently from message-level security. Because transport-level security provides minimal security, use message-level security when security is essential to the Web service application.

To view this administrative console page, complete the following steps:

1. Click **Applications** > **Enterprise Applications** >*application_name*.
2. Under Related Items, click **Web module** >*URI_file_name* > **Web Services: Client Security Bindings**.
3. Under HTTP Basic Authentication, click **Edit**.

***Basic authentication ID:***

Specifies the user name for the HTTP basic authentication for this port.

***Basic authentication password:***

Specifies the password for the HTTP basic authentication for this port.

## Configuring HTTP basic authentication programmatically

This topic explains how to configure HTTP basic authentication by programmatically modifying HTTP properties.

This task is one of three ways that you can configure HTTP basic authentication. You can also configure HTTP basic authentication with an assembly tool or with the administrative console.

If you programmatically configure HTTP basic authentication, the properties are configured in the Stub or Call instance. If you choose to configure HTTP basic authentication with the administrative console or an assembly tool, the Web services security binding information is modified. The values that are set programmatically take precedence over the values defined in the binding. However, you can only configure HTTP proxy authentication programmatically.

The HTTP basic authentication that is discussed in this topic is orthogonal to WS-Security and is distinct from basic authentication that WS-Security supports. WS-Security supports basic authentication token, not HTTP basic authentication.

Configure HTTP basic authentication programmatically with the following steps provided in this task section.

1. Set the properties in the Stub or Call instance for a Web service or a Web service client You can set the following properties:

   ```
   javax.xml.rpc.Call.USERNAME_PROPERTY
   javax.xml.rpc.Call.PASSWORD_PROPERTY
   javax.xml.rpc.Stub.USERNAME_PROPERTY
   javax.xml.rpc.Stub.PASSWORD_PROPERTY
   ```

2. Set the properties in the Stub or Call instance to configure the HTTP proxy authentication.
   a. You can set the following properties for HTTP:

```
com.ibm.wsspi.webservices.HTTP_PROXYHOST_PROPERTY
com.ibm.wsspi.webservices.HTTP_PROXYPORT_PROPERTY
com.ibm.wsspi.webservices.HTTP_PROXYUSER_PROPERTY
com.ibm.wsspi.webservices.HTTP_PROXYPASSWORD_PROPERTY
```

3. You can set the following properties for HTTPS:

```
com.ibm.wsspi.webservices.HTTPS_PROXYHOST_PROPERTY
com.ibm.wsspi.webservices.HTTPS_PROXYPORT_PROPERTY
com.ibm.wsspi.webservices.HTTPS_PROXYUSER_PROPERTY
com.ibm.wsspi.webservices.HTTPS_PROXYPASSWORD_PROPERTY
```

### HTTP basic authentication collection

Use this page to specify a user name and password for transport-level basic authentication security for this port. You can use this configuration when a Web service is a client to another Web service.

You can use transport-level security to enable basic authentication. Transport-level security can be enabled or disabled independently from message-level security. Because transport-level security provides minimal security, use message-level security when security is essential to the Web service application.

To view this administrative console page, complete the following steps:

1. Click **Applications** > **Enterprise Applications** >*application_name*.
2. Under Related Items, click **Web module** >*URI_file_name* **> Web Services: Client Security Bindings**.
3. Under HTTP Basic Authentication, click **Edit**.

***Basic authentication ID:***

Specifies the user name for the HTTP basic authentication for this port.

***Basic authentication password:***

Specifies the password for the HTTP basic authentication for this port.

## Configuring additional HTTP transport properties using the administrative console

This topic explains how to configure additional HTTP transport properties with the JVM custom properties panel in the administrative console.

This task is one of three ways that you can configure additional HTTP transport properties for a Web Service acting as a client to another Web service. You can also configure the additional HTTP transport properties in the following ways:

- Configure the properties with an assembly tool
- Configure the properties using the **wsadmin** command-line tool

If you want to programmatically configure the properties using the Java API XML-based Remote Procedure Call (JAX-RPC) programming model, review the JAX-RPC specification that is available through Web services: Resources for learning.

See Additional HTTP transport properties for Web services applications for more information about the following properties that you can configure:

- com.ibm.websphere.webservices.http.requestContentEncoding
- com.ibm.websphere.webservices.http.responseContentEncoding
- com.ibm.websphere.webservices.http.connectionKeepAlive
- com.ibm.websphere.webservices.http.requestResendEnabled
- http.proxyHost

- http.proxyPort
- https.proxyHost
- https.proxyPort

These additional properties are configured for Web services applications that use the HTTP protocol. The properties affect the content encoding of the message in the HTTP request, the HTTP response, the HTTP connection persistence and the behavior of an HTTP request that is resent after a `java.net.ConnectException` error occurs when there is a read time-out.

Configure the additional HTTP properties with the administrative console with the following steps provided in this task section:

1. Open the administrative console.
   a. Click **Servers** > **Application Servers** > *server* > **Process Definition** > *Control* > **Java Virtual Machine** > **Custom Properties** to define the property in the Control, or click **Application Server** > *server* > **Process Definition** > *Servant* > **Java Virtual Machine** > **Custom Properties** to define the property in the Servant.
2. (Optional) If the property is not listed, create a new property name.
3. Enter the name and value.
4. (Optional) Accept the redirection of the HTTP request to a different URI in HTTPS.

   A redirection of the HTTP request to a different URI in HTTPS can occur if the transport guarantee of CONFIDENTIAL or INTEGRAL is configured in the application. To accept the redirection, you can do either of the following tasks:
   - Set the `com.ibm.ws.webservices.HttpRedirectEnabled` Java system property to `true`.
   - Programmatically set the `com.ibm.wsspi.webservices.Constants.HTTP_REDIRECT_ENABLED` property to `true` in the stub or call object before invoking the service.

You have configured HTTP transport properties for a Web services application.

## Configuring additional HTTP transport properties with an assembly tool

This topic explains how to configure additional HTTP transport properties with an assembly tool. The assembly tool is used to configure the `ibm-webservicesclient-bnd.xmi` deployment descriptor binding file.

You can configure additional HTTP transport properties with assembly tools provided with WebSphere Application Server.

You must configure the assembly tool before you can use it.

This task is one of three ways that you can configure additional HTTP transport properties for a Web Service acting as a client to another Web service. You can also configure the additional HTTP transport properties in the following ways:

- Configure the properties the JVM custom property panel in the administrative console.
- Configure the properties using the **wsadmin** command-line tool.

If you want to programmatically configure the properties using the Java API XML-based Remote Procedure Call (JAX-RPC) programming model, review the JAX-RPC specification that is available through Web services: Resources for learning.

See Additional HTTP transport properties for Web services applications for more information about the following properties that you can configure:

- com.ibm.websphere.webservices.http.requestContentEncoding
- com.ibm.websphere.webservices.http.responseContentEncoding

- com.ibm.websphere.webservices.http.connectionKeepAlive
- com.ibm.websphere.webservices.http.requestResendEnabled
- http.proxyHost
- http.proxyPort
- https.proxyHost
- https.proxyPort

These additional properties are configured for Web services applications that use the HTTP protocol. The properties affect the content encoding of the message in the HTTP request, the HTTP response, the HTTP connection persistence and the behavior of an HTTP request that is resent after a `java.net.ConnectException` error occurs when there is a read time-out.

Configure the additional HTTP properties with an assembly tool with the following steps provided in this task section:

1. Start an assembly tool. The assembly tools, Application Server Toolkit (AST) and Rational Web Developer, provide a graphical interface for developing code artifacts, assembling the code artifacts into various archives (modules) and configuring related Java 2 Platform, Enterprise Edition (J2EE) Version 1.2, 1.3 or 1.4 compliant deployment descriptors.
2. Create and specify the name/value pair in the **Web Services Client Port Binding** page for a Web service client. The Web Services Client Port Binding page is available after double-clicking the client deployment descriptor file.

You have configured additional HTTP transport properties for a Web services application.

## Configuring additional HTTP transport properties using wsadmin

This topic explains how to configure additional HTTP transport properties with the **wsadmin** command-line tool.

The WebSphere Application Server wsadmin tool provides the ability to run scripts. You can use the wsadmin tool to manage a WebSphere Application Server installation, as well as configuration, application deployment, and server run-time operations. The WebSphere Application Server only supports the Jacl and Jython scripting languages. For more information about the wsadmin tool options, review Options for the AdminApp object install, installInteractive, edit, editInteractive, update, and updateInteractive commands

This task is one of three ways that you can configure additional HTTP transport properties for a Web Service acting as a client to another Web service. You can also configure the additional HTTP transport properties in the following ways:

- Configure the properties with an assembly tool
- Configure the properties using the administrative console

If you want to programmatically configure the properties using the Java API XML-based Remote Procedure Call (JAX-RPC) programming model, review the JAX-RPC specification that is available through Web services: Resources for learning.

See Additional HTTP transport properties for Web services applications for more information about the following properties that you can configure:

- com.ibm.websphere.webservices.http.requestContentEncoding
- com.ibm.websphere.webservices.http.responseContentEncoding
- com.ibm.websphere.webservices.http.connectionKeepAlive
- com.ibm.websphere.webservices.http.requestResendEnabled
- http.proxyHost

- http.proxyPort
- https.proxyHost
- https.proxyPort

These additional properties are configured for Web services applications that use the HTTP protocol. The properties affect the content encoding of the message in the HTTP request, the HTTP response, the HTTP connection persistence and the behavior of an HTTP request that is resent after a `java.net.ConnectException` error occurs when there is a read time-out.

Configure the additional HTTP properties with the wsadmin tool by following steps provided in this task section:

1. Launch a scripting command.
2. At the **wsadmin** command prompt, enter the command syntax. You can use install, installInteractive, edit, editInteractive, update, and updateInteractive commands.
3. If you are configuring the `com.ibm.websphere.webservices.http.responseContentEncoding` property, use the **WebServicesServerCustomProperty** command option.
4. Configure all other properties using the **WebServicesClientCustomProperty** command option.
5. Save the configuration changes with the **$AdminConfig save** command.

You have configured HTTP transport properties for a Web services application.

The following illustrates an example of the Jython script syntax:

```
AdminApp.edit ( 'PlantsByWebSphere', '[ -WebServicesClientCustomProperty [[PlantsByWebSphere.war ""
service/FrontGate_SEIService FrontGate http.proxyHost+http.proxyPort myhost+80]]]')
AdminConfig.save()

AdminApp.edit ( 'WebServicesSamples', '[ -WebServicesServerCustomProperty [[AddressBookW2JE.jarAddressBookService Address
AdminConfig.save()
```

The following illustrates an example of the Jacly script syntax:

```
$AdminApp edit PlantsByWebSphere {  -WebServicesClientCustomProperty {{PlantsByWebSphere.war {} service/FrontGate_SEIServ
$AdminConfig save

$AdminApp edit WebServicesSamples {-WebServicesServerCustomProperty {{AddressBookW2JE.jar
AddressBookService AddressBook http.proxyHost+http.proxyPort myhost+80}}}
$AdminConfig save
```

To convert these examples from **edit** to **install**, add `.ear` to form a file name, and add any extra keywords for deployment, like `-usedefaultbindings` and `-deployejb`.

# Provide HTTP endpoint URL information

Use this page to specify endpoint URL prefix information for Web services accessed by HTTP. Prefixes are used to form complete endpoint addresses included in published Web Services Description Language (WSDL) files.

To view this administrative console page, click **Applications** >**Enterprise Applications** > *application_instance* > **Provide HTTP endpoint URL information**.

You can specify a portion of the endpoint URL to be used in each Web service module. In a published WSDL file, the URL defining the target endpoint address is found in the location attribute of the port's `soap:address` element.

## Specify endpoint URL prefixes for Web services

Specifies the *protocol* (either http or https), *host_name*, and *port_number* to be used in the endpoint URL.

You can select a prefix from a predefined list using the **HTTP URL prefix** or **Custom HTTP URL prefix** field.

The URL prefix format is *protocol://host_name:port_number*, for example, http://*myHost*:9045. The actual endpoint URL that appears in a published WSDL file consists of the prefix followed by the module's context-root and the Web service url-pattern, for example, http://*myHost*:9045/services/*myService*.

### Select default HTTP URL prefix

Specifies the drop down list associated with a default list of URL prefixes. This list is the intersection of the set of ports for the module's virtual host and the set of ports for the module's application server. Use items from this list if the Web services application server is accessed directly.

To set an HTTP endpoint URL prefix, select **Select default HTTP URL prefix** and select a value from the drop down list. Select the check box of the modules that are to use the prefix and click **Apply**. When you click **Apply**, the entry in the **Select default HTTP URL prefix** or **Select custom HTTP URL prefix** fields, depending on which is selected, is copied into the **HTTP URL prefix** field of any module whose check box is selected.

### Select custom HTTP URL prefix

Specifies the *protocol*, *host*, and *port_number* of the intermediate service if the Web services in a module are accessed through an intermediate node, for example the Web services gateway or an IHS server.

To set a custom HTTP endpoint URL prefix, select **Select custom HTTP URL prefix** and enter a value. Select the check box of the modules that are to use the prefix and click **Apply**. When you click **Apply**, the entry in the **Select default HTTP URL prefix** or **Select custom HTTP URL prefix** fields, depending on which is selected, is copied into the **HTTP endpoint URL prefix** field of any module whose check box is selected.

## Publish WSDL zip files settings

Use this page to publish Web Services Description Language (WSDL) files.

To view this administrative console page, click **Applications** >**Enterprise Applications** > *application_instance* > **Publish WSDL zip files**.

When you click **OK**, a panel showing one or several zip file names displays. Each zip file contains a WSDL file that represents the Web services-enabled modules in the application. When you select a zip file to publish, a dialogue displays from which you can choose where to create the zip file. Within the published zip files, the directory structure is *application_name/module_name/[META-INF|WEB-INF]/wsdl/wsdl_file_name*.

In a published WSDL file, the location attribute of a port's `soap:address` element contains the endpoint URL through which the Web service is accessed. Using the **Provide HTTP endpoint URL information** and the **Provide JMS and EJB endpoint URL information** panels, configure the endpoint URLs to be used for the Web services in each module.

### *application_name*_WSDLFiles.zip

Specifies the `application_name_WSDLFiles.zip` file containing the WSDL that describes Web services that are accessible by standard SOAP-based ports.

### *application_name*_ExtendedWSDLFiles.zip

Specifies the `application_name_ExtendedWSDLFiles.zip` file containing the WSDL file that describes the Web services available, including SOAP-based and non-SOAP based (for example, EJB) ports.

If there are no Web services configured for direct EJB access, this zip file name does not appear. Do not use this zip file if you want to produce a WSDL file compliant to standards.

# Securing Web services for version 6 applications based on WS-Security

Web services security for WebSphere Application Server is based on standards included in the Organization for the Advancement of Structured Information Standards (OASIS) Web services security (WSS) Version 1.0 specification, the Username token Version 1.0 profile, and the X.509 token Version 1.0 profile. These standards and profiles address how to provide protection for messages exchanged in a Web service environment. The specification defines the core facilities for protecting the integrity and confidentiality of a message and provides mechanisms for associating security-related claims with the message. Web services security is a message-level standard based on securing Simple Object Access Protocol (SOAP) messages through XML digital signature, confidentiality through XML encryption, and credential propagation through security tokens.

To secure Web services, you must consider a broad set of security requirements, including authentication, authorization, privacy, trust, integrity, confidentiality, secure communications channels, federation, delegation, and auditing across a spectrum of application and business topologies. One of the key requirements for the security model in today's business environment is the ability to interoperate between formerly incompatible security technologies (such as public key infrastructure, Kerberos and so on) in heterogeneous environments (such as Microsoft .NET and Java 2 Platform, Enterprise Edition (J2EE)). The complete Web services security protocol stack and technology roadmap is described in Security in a Web Services World: A Proposed Architecture and Roadmap.

The Web Services Security: SOAP Message Security 1.0 specification outlines a standard set of SOAP extensions that you can use to build secure Web services. These standards confirm integrity and confidentiality, which are generally provided with digital signature and encryption technologies. In addition, Web services security provides a general purpose mechanism for associating security tokens with messages. A typical example of the security token is a username token, in which a user name and password are included as text. Web services security defines how to encode binary security tokens using methods such as X.509 certificates and Kerberos tickets. However, the required security tokens are not defined in the Web service security Version 1.0 specification. Instead, the tokens are defined in separate profiles such as the Username token profile, the X.509 token profile, the SAML profile, the Kerberos profile, the XrML profile and so on.

Web service security is supported in the managed Web service container. To establish a managed environment and to enforce constraints for Web services security, you must perform a Java Naming and Directory Interface (JNDI) lookup on the client to resolve the service reference. For more information on the recommended client programming model, see ″Service lookup″ in the Java Specification Request (JSR) 109 specification available at: ftp://www-126.ibm.com/pub/jsr109/spec/1.0/websvcs-1_0-fr.pdf.

**WebSphere Application Server Version 6 and Version 5.x compatibility**

In WebSphere Application Server Version 6, you can run a version 5.x Web services-secured application on a version 6 application server. However, when you use a Web services-secured application, the client and the server must use the same version of the application server. For example, a Web services-secured application does not work properly when the client uses WebSphere Application Server Version 6 and the server uses version 5.x. Conversely, a Web services-secured application does not work properly when the client uses WebSphere Application Server Version 5.x and the server uses version 6. This issue occurs because the SOAP message format is different between a version 5.x application and a version 6 application.

**Configurations**

To secure Web services with WebSphere Application Server, you must specify several different configurations. Although there is not a specific sequence in which you must specify these different configurations, some configurations reference other configurations. The following table shows an example

of the relationship between each of the configurations. However, the requirements for the bindings depend upon the deployment descriptor. Some binding information depends upon other information in the binding or server and cell-level configuration. For instance, the signing information references the key information.

*Table 3. The relationship between the configurations.*

| Configuration level | Configuration name | Configurations it references |
| --- | --- | --- |
| Application-level request generator | Token generator | • Collection certificate store<br>• Nonce<br>• Timestamp<br>• Callback handler |
| Application-level request generator | Key information | • Key locator<br>• Key name<br>• Token |
| Application-level request generator | Signing information | • Key information |
| Application-level request generator | Encryption information | • Key information |
| Application-level request consumer | Token consumer | • Trust anchor<br>• Collection certificate store<br>• Trusted ID evaluators<br>• Java Authentication and Authorization Service (JAAS) configuration |
| Application-level request consumer | Key information | • Key locator<br>• Token |
| Application-level request consumer | Signing information | • Key information |
| Application-level request consumer | Encryption information | • Key information |
| Application-level response generator | Token generator | • Collection certificate store<br>• Callback handler |
| Application-level response generator | Key information | • Key locator<br>• Token |
| Application-level response generator | Signing information | • Key information |
| Application-level response generator | Encryption information | • Key information |
| Application-level response consumer | Token consumer | • Trust anchor<br>• Collection certificate store<br>• JAAS configuration |
| Application-level response consumer | Key information | • Key locator<br>• Key name<br>• Token |
| Application-level response consumer | Signing information | • Key information |
| Application-level response consumer | Encryption information | • Key information |
| Server-level default generator bindings | Token generator | • Collection certificate store<br>• Callback handler |

*Table 3. The relationship between the configurations. (continued)*

| Configuration level | Configuration name | Configurations it references |
| --- | --- | --- |
| Server-level default generator bindings | Key information | • Key locator<br>• Token |
| Server-level default generator bindings | Signing information | • Key information |
| Server-level default generator bindings | Encryption information | • Key information |
| Server-level default consumer bindings | Token consumer | • Trust anchor<br>• Collection certificate store<br>• Trusted ID evaluator<br>• JAAS configuration |
| Server-level default consumer bindings | Key information | • Key locator<br>• Token |
| Server-level default consumer bindings | Signing information | • Key information |
| Server-level default consumer bindings | Encryption information | • Key information |
| Cell-level default generator bindings | Token generator | • Collection certificate store<br>• Callback handler |
| Cell-level default generator bindings | Key information | • Key locator<br>• Token |

If multiple applications will use the same binding information, consider configuring the binding information on the server level. For example, you might have a global key locator configuration that is used by multiple applications.

Because of the relationship between the different Web services security configurations, it is recommended that you specify the configurations in following order:

- Assemble your Web services security-enabled application using an assembly tool. Prior to modifying an Web services security-enabled application in the WebSphere Application Server administrative console, you must assemble your application using an assembly tool. Although you can modify some of the application settings using the administrative console, you must configure the generator and the consumer security constraints using an assembly tool such as the Application Server Toolkit or the Rational Application Developer. For information on the assembly tools, see ″Assembly tools″ in the ″Developing and deploying applications″ PDF. For information on how to add Web services security to an application using an assembly tool, see “Configuring an application for Web services security with an assembly tool” on page 567. Return to this article after you have assembled your application and imported it into the administrative console.
- **Optional:** Modify the application-level configurations in the administrative console.
  1. Configure the trust anchors for the generator binding. For more information, see “Configuring trust anchors for the generator binding on the application level” on page 659.
  2. Configure the collection certificate store for the generator binding. For more information, see “Configuring the collection certificate store for the generator binding on the application level” on page 663.
  3. Configure the token for the generator binding. For more information, see “Configuring the token generator on the application level” on page 675.

4. Configure the key locators for the generator binding. For more information, see “Configuring the key locator for the generator binding on the application level” on page 692.
5. Configure the key information for the generator binding. For more information, see “Configuring the key information for the generator binding on the application level” on page 699.
6. Configure the signing information for the generator binding. For more information, see “Configuring the signing information for the generator binding on the application level” on page 712.
7. Configure the encryption information for the generator binding. For more information, see “Configuring the encryption information for the generator binding on the application level” on page 727.
8. Configure the trust anchors for the consumer binding. For more information, see “Configuring trust anchors for the consumer binding on the application level” on page 736.
9. Configure the collection certificate store for the consumer binding. For more information, see “Configuring the collection certificate store for the consumer binding on the application level” on page 738.
10. Configure the token for the consumer binding. For more information, see “Configuring token consumer on the application level” on page 740
11. Configure the key locators for the consumer binding. For more information, see “Configuring the key locator for the consumer binding on the application level” on page 749.
12. Configure the key information for the consumer binding. For more information, see “Configuring the key information for the consumer binding on the application level” on page 750.
13. Configure the signing information for the consumer binding. For more information, see “Configuring the signing information for the consumer binding on the application level” on page 753.
14. Configure the encryption information for the consumer binding. For more information, see “Configuring the encryption information for the consumer binding on the application level” on page 757.

- Specify the server-level configurations.
  1. Configure the trust anchors for the server level. For more information, see “Configuring trust anchors on the server or cell level” on page 761
  2. Configure the collection certificate store for the server level. For more information, see “Configuring the collection certificate store on the server or cell-level bindings” on page 762
  3. Configure a token generator. For more information, see “Configuring token generators on the server or cell level” on page 765.
  4. Configure a nonce for the server level. For more information, see “Configuring a nonce on the server or cell level” on page 764.
  5. Configure the key locators for the generator binding. For more information, see “Configuring the key locator on the server or cell level” on page 777.
  6. Configure the key information for the generator binding. For more information, see “Configuring the key locator on the server or cell level” on page 777.
  7. Configure the signing information for the generator binding. For more information, see “Configuring the signing information for the generator binding on the server or cell level” on page 780.
  8. Configure the encryption information for the generator binding. For more information, see “Configuring the encryption information for the generator binding on the server or cell level” on page 783.
  9. Configure the trusted ID evaluators for the server level. For more information, see “Configuring trusted ID evaluators on the server or cell level” on page 784
  10. Configure a token consumer. For more information, see “Configuring token consumers on the server or cell level” on page 787.
  11. Configure the key information for the consumer binding. For more information, see “Configuring the key information for the consumer binding on the server or cell level” on page 795.

12. Configure the signing information for the consumer binding. For more information, see “Configuring the signing information for the consumer binding on the server or cell level” on page 796.
13. Configure the encryption information for the consumer binding. For more information, see “Configuring the encryption information for the consumer binding on the server or cell level” on page 799.

After completing these steps on the appropriate level of WebSphere Application Server, you have secured Web services.

**Note:** Configuration information for the application-level precedes similar configuration information on the server-level.

**Related tasks**

“Configuring an application for Web services security with an assembly tool” on page 567

Importing enterprise applications
Importing an enterprise archive (EAR) file migrates the EAR file to the assembly tool and defines a new enterprise application project using the tool.

## What is new for securing Web services

In WebSphere Application Server Version 6, there are many security enhancements for Web services. The enhancements include supporting sections of the Web services security specifications and providing architectural support for plugging in and extending the capabilities of security tokens.

### Enhancements from the supported Web services security specifications

Since September 2002, the Organization for the Advancement of Structured Information Standards (OASIS) has been developing the Web Services Security (WSS) for SOAP message standard. In April 2004, OASIS released the Web Services security Version 1.0 specification, which is a major milestone for securing Web services. This specification is the foundation for other Web services security specifications and is also the basis for the Basic Security Profile (WS-I BSP) Version 1.0 work. Web services security Version 1.0 is a strategic move towards Web services security interoperability and it is the first step in the Web services security roadmap. For more information on the Web services security roadmap, see Security in a Web Services World: A Proposed Architecture and Roadmap.

WebSphere Application Server Version 6 supports the following specifications and profiles:

- OASIS: Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)
- OASIS: Web Services Security: UsernameToken Profile 1.0
- OASIS: Web Services Security X.509 Certificate Token Profile 1.0

For details on what parts of the previous specifications are supported in WebSphere Application Server version 6, see “Supported functionality from OASIS specifications” on page 527.

### High level features overview in WebSphere Application Server Version 6

The Web Services Security for SOAP message Version 1.0 specification is designed to be flexible and accommodate the requirements of Web services. For example, the specification does not have a mandatory security token definition in the Web services security Version 1.0 specification. Rather the specification defines a generic mechanism to associate the security token with a Simple Object Access Protocol (SOAP) message. The use of security tokens is defined in the various security token profiles such as:

- The username token profile
- The X.509 token profile
- The WS-Security Kerberos token profile
- The Security Assertion Markup Language (SAML) token profile

- The Rights Express Language (REL) token profile

For more information on security token profile development at OASIS, see Organization for the Advancement of Structured Information Standards.

**Important:** The wire format in the Web services security Version 1.0 specification changed and is not compatible with the previous drafts of the Web services security specification. It is not possible to make an implementation of the wire format using a previous draft of the Web services security specification to interoperate with the Web Services Security Version 1.0 specification.

Support for pluggable security tokens has been available since WebSphere Application Server Version 5.0.2. However, in WebSphere Application Server Version 6, the pluggable architecture is enhanced to support the Web services security Version 1.0 specification, other profiles, and other Web services security specifications. WebSphere Application Server Version 6 includes the following key enhancements:

- Support for the client (sender or generator) to send multiple security tokens in a SOAP message.
- Ability to derive keys from a security token for digital signature (verification) and encryption (decryption).
- Support to sign or encrypt any element in a SOAP message. However, some limitations exist. For example, encrypting some parts of a message might break the SOAP message format. If you encrypt the SOAP body element, the SOAP message format breaks.
- Support for signing the SOAP Envelope, the SOAP Header, and the Web services security header.
- Ability to configure the order of the digital signature and encryption.
- Support for various mechanisms to reference the security tokens such as direct references, key identifiers, key names, and embedded references.
- Support for the PKCS#7 format certificate revocation list (CRL) encoding for an X.509 security token.
- Support for CRL verification.
- Ability to insert nonce and time stamps into elements within the Web services security header, into signed elements, or into encrypted elements.
- Support for identity assertion using the Run As (invocation) identity in the current security context for WebSphere Application Server.
- Support for a default binding, which is a set of default Web services security bindings for applications.
- Ability to use pluggable digital signature (verification) and encryption (decryption) algorithms.

For more information on some of these enhancements, see “Web services security enhancements” on page 531.

**Configuration**

WebSphere Application Server Version 6 uses the deployment model for implementing the Web services security Version 1.0 specification, the Username token Version 1.0 profile, and the X.509 token Version 1.0 profile. The deployment model is an extension of the Web services deployment model for Java 2 Platform, Enterprise Edition (J2EE). The Web services security constraints are defined in the IBM extension deployment descriptor and the binding file based on the Web service port.

The format of the deployment descriptor and the binding file is IBM proprietary material and is not available. However, WebSphere Application Server provides the following tools that you can use to edit the deployment descriptor and the binding file:

**Rational Application Developer Version 6**

You can use Rational Application Developer Version 6 to develop Web services and configure the deployment descriptor and the binding file for Web services security. The Rational Application Developer enables you to assemble both Web and EJB modules.

**Rational Web Developer Version 6**

You can use Rational Web Developer Version 6 to develop Web services and configure the

deployment descriptor and the binding file for Web services security. However, you cannot assemble EJB modules using this tool. Instead, use the Application Server Toolkit or the Rational Application Developer.

**Application Server Toolkit**
You can use the Application Server Toolkit (AST), which is an assembly tool designer for WebSphere Application Server Version 6, to specify the deployment descriptor and the binding file for Web services security.

**WebSphere Application Server Administrative Console**
You can use the administrative console to configure the Web services security binding of a deployed application with Web services security constraints defined in the deployment descriptor.

**Important:** The format of the deployment descriptor and the binding file for Web services security in WebSphere Application Server Version 6 is different from WebSphere Application Server Versions 5.0.2, 5.1, and 5.1.1. Web services security support in WebSphere Application Server Versions 5.0.2, 5.1, and 5.1.1 is based on the Web services security draft 13 specification and the username token draft 2 profile. Thus, this support is deprecated. However, applications that you configured using the Web service security Versions 5.0.2, 5.1, and 5.1.1 deployment descriptor and binding file can work with WebSphere Application Server 6. These applications use a deployment descriptor and binding file that emit SOAP message security using the draft 13 specification format. The Web services security deployment descriptor and binding file for WebSphere Application Server Version 6 is available for a J2EE Version 1.4 application only. Therefore, the Web services security Version 1.0 specification is supported for a J2EE Version 1.4 application only.

To take advantage of implementations associated with the Web services security Version 1.0 specification, you must:

- Migrate existing applications to J2EE Version 1.4
- Reconfigure the Web services security constraints in the new deployment descriptor and binding format

**Important:** An automatic process does not exist for migrating the deployment descriptor and the binding file for Web services security from the version 5.0.2, 5.1, and 5.1.1 format to the new version 6 format using the Rational Web Developer and Application Server Toolkit. You must migrate the configuration manually.

### What is not supported

Web service security is still fairly new and some of the standards are still being defined or standardized. The following functionality is not supported in WebSphere Application Server Version 6:

- Application programming interfaces (API) do not exist for Web services security in WebSphere Application Server version 6. The following standards exist for the Java application programming interface for XML security and Web services security:
  - JSR-105 (Java API for XML-Signature XPath Filter Version 2.0

    W3C Recommendation, November 2002
  - JSR-106 (Java API for XML Encryption Syntax and Processing)

    W3C Recommendation, December 2002
  - JSR-183 (Java API for Web Services Security: SOAP Message Security 1.0 specification)
- SAML token profile is not supported out of the box.
- WS-SecuredConversation is not supported out of the box.
- WS-Trust is not supported out of the box.
- WS-SecurityKerberos token profile is not supported out of the box.
- REL token profile is not supported.

- Web services security SOAP messages with an attachments profile (SwA) is not supported.
- WS-I Basic Security Profile 1.0 is not supported.
- Non-Web services container managed client is not supported out of the box.

For information on what is supported for Web services security in WebSphere Application Server Version 6, see “Supported functionality from OASIS specifications” on page 527.

**Related concepts**

“Web services security enhancements” on page 531

***Web services security specification for version 6 - a chronology:*** This article describes the development of the Web services security specification. The article provides information on the Organization for the Advancement of Structured Information Standards (OASIS) Web services security Version 1.0 specification, which is the specification that serves as a basis for securing Web services in WebSphere Application Server Version 6.

### Non-OASIS activities

Web services is gaining rapid acceptance as a viable technology for interoperability and integration. However, securing Web services is one of the paramount quality of services that makes the adoption of Web services a viable industry and commercial solution for businesses. IBM and Microsoft jointly published a security white paper on Web services entitled Security in a Web Services World: A Proposed Architecture and Roadmap. The white paper discusses the following initial and subsequent specifications in the proposed Web services security roadmap:

**Web service security**
: This specification defines how to attach a digital signature, use encryption, and use security tokens in Simple Object Access Protocol (SOAP) messages.

**WS-Policy**
: This specification defines the language that is used to describe security constraints and the policy of intermediaries or endpoints.

**WS-Trust**
: This specification defines a framework for trust models to establish trust between Web services.

**WS-Privacy**
: This specification defines a model of how to express a privacy policy for a Web service and a requester.

**WS-SecureConversation**
: This specification defines how to exchange and establish a secured context, which derives session keys between Web services.

**WS-Federation**
: This specification defines a model for trust relationships in a heterogeneous, federated environment, including federated identities management.

**WS-Authorization**
: This specification defines the authorization policy for a Web service.

This following figure shows the relationship between these specifications:

WS-SecureConversation | WS-Federation | WS-Authorization
WS-Policy | WS-Trust | WS-Privacy
Web service security: SOAP message security
SOAP messaging

In April 2002, IBM, Microsoft, and VeriSign proposed the Web Services Security (WS-Security) specification on their Web sites as depicted by the green box in the previous figure. This specification included the basic ideas of a security token, XML digital signature, and XML encryption. The specification also defined the format for user name tokens and encoded binary security tokens. After some discussion and an interoperability test based on the specification, the following issues were noted:

- The specification requires that the Web services security processors understand the schema correctly so that the processor distinguishes between the ID attribute for XML digital signature and XML encryption.
- The freshness of the message, which indicates whether the message complies with predefined time constraints, cannot be determined.
- Digested password strings do not strengthen security.

In August 2002, IBM, Microsoft, and VeriSign published the *Web Services Security Addendum*, which attempted to address the previously listed issues. The following solutions were addressed in the addendum:

- Require a global ID attribute for XML signature and XML encryption.
- Use time stamp header elements that indicate the time of the creation, receipt, or expiration of the message.
- Use password strings that are digested with a time stamp and nonce, which is a randomly generated token.

The specifications for the blue boxes in the previous figure have been proposed by various industry vendors and various interoperability events have been organized by the vendors to verify and refine the proposed specifications.

### OASIS activities

In June 2002, OASIS received a proposed Web services security specification from IBM, Microsoft, and Verisign. The Web Services Security Technical Committee (WSS TC) was organized at OASIS soon after the submission. The technical committee included many companies including IBM, Microsoft, VeriSign, Sun Microsystems, and BEA Systems.

In September 2002, WSS TC published its first specification, Web Services Security Core Specification, Working Draft 01. This specification included the contents of both the original Web services security specification and its addendum.

The coverage of the technical committee became larger as the discussion proceeded. Because the Web Services Security Core Specification allows arbitrary types of security tokens, proposals were published as profiles. The profiles described the method for embedding tokens, including Security Assertion Markup

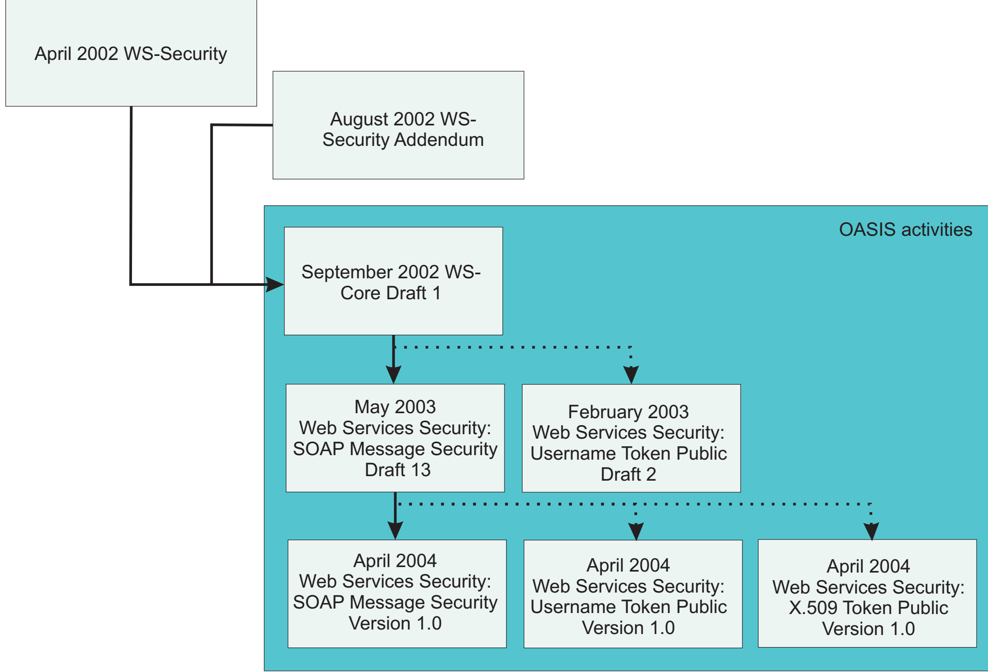
Language (SAML) tokens and Kerberos tokens embedded into the Web services security messages. Subsequently, the definitions of the usage for user name tokens and X.509 binary security tokens, which were defined in the original Web Services Security Specification, were divided into the profiles.

WebSphere Application Server Versions 5.0.2, 5.1, and 5.1.1 support the following specifications:
- Web Services Security: SOAP Message Security Draft 13 (formerly Web Services Security Core Specification)
- Web Services Security: Username Token Profile Draft 2

In April 2004, the Web service security specification (officially called Web Services Security: SOAP Message Security Version 1.0) became the Version 1.0 OASIS standard. Also, the Username token and X.509 token profiles are Version 1.0 specifications.

WebSphere Application Server 6 supports the following Web services security specifications from OASIS:
- Web Services Security: SOAP Message Security 1.0 specification
- Web Services Security: Username Token 1.0 Profile
- Web Services Security: X.509 Token 1.0 Profile

The following figure shows the various Web services security-related specifications.




WebSphere Application Server Version 6 also extends and provides plug-in capability to enable security providers to extend the run-time capability and implement some of the higher level specifications in the Web service security stack. The plug-in points are exposed as Service Provider Programming Interfaces (SPI). For more information on these SPIs, see "Default implementations of the Web services security service provider programming interfaces" on page 558.

**Web services security specification development**

The OASIS Web services security Version 1.0 specification defines the enhancements that are used to provide message integrity and confidentiality. It also provides a general framework for associating the security tokens with a Simple Object Access Protocol (SOAP) message. The specification is designed to be extensible to support multiple security token formats. The particular security token usage is addressed

with the security token profile. The OASIS Web services security specification is based upon the following World Wide Web Consortium (W3C) specifications. Most of the W3C specifications are in the standard body recommended status.

- XML-Signature Syntax and Processing

  W3C recommendation, February 2002 (Also, IETF RFC 3275, March 2002)
- Canonical XML Version 1.0

  W3C recommendation, March 2001
- Exclusive XML Canonicalization Version 1.0

  W3C recommendation, July 2002
- XML-Signature XPath Filter Version 2.0

  W3C Recommendation, November 2002
- XML Encryption Syntax and Processing

  W3C Recommendation, December 2002
- Decryption Transform for XML Signature

  W3C Recommendation, December 2002

These specifications are supported in WebSphere Application Server 6 in the context of Web services security. For example, you can sign a SOAP message by specifying the integrity option in the deployment descriptors. However, there is no application programming interface (API) that an application can use for XML signature on an XML element in a SOAP message.

The OASIS Web services security Version 1.0 specification defines the enhancements that are used to provide message integrity and confidentiality. It also provides a general framework for associating the security tokens with a Simple Object Access Protocol (SOAP) message. The specification is designed to be extensible to support multiple security token formats. The particular security token usage is addressed with the security token profile.

**Specification and profile support in WebSphere Application Server Version 6**

OASIS is working on various profiles. For more information, see Organization for the Advancement of Structured Information Standards Committees. WebSphere Application Server Version 6 does not support these profiles. The following list is some of the published draft profiles and OASIS Web services security technical committee work in progress:

- Web Services Security: SAML token profile
- Web Services Security: REL token profile
- Web Services Security: Kerberos token profile
- Web Services Security: SOAP Messages with Attachments (SwA) profile

Because WebSphere Application Server Version 6 supports the following specifications, support for Web services security draft 13 and Username token profile draft 2 in WebSphere Application 5.0.2, 5.1.0 and 5.1.1 is deprecated:

- OASIS Web Services Security Version 1.0 specification
- Web Services Security Username token profile
- X.509 token profile

The wire format of the SOAP message with Web services security in Web services security Version 1.0 has changed and is not compatible with previous drafts of the OASIS Web services security specification. Interoperability between OASIS Web services security Version 1.0 and previous Web services security drafts is not supported. However, it is possible to run an application that is based on Web services security

draft 13 on WebSphere Application Server Version 6. The application can interoperate with an application that is based on Web services security draft 13 on WebSphere Application Server Version 5.0.2, 5.1 or 5.1.1.

WebSphere Application Server Version 6 supports both the OASIS Web services security draft 13 and the OASIS Web services security 1.0 specification. But in WebSphere Application Server Version 6, the support of OASIS Web services security draft 13 is deprecated. However, applications that were developed using OASIS Web services security draft 13 on WebSphere Application Server 5.0.2, 5.1.0 and 5.1.1 can run on WebSphere Application Server Version 6. OASIS Web services security Version 1.0 support is available only for Java 2 Platform, Enterprise Edition (J2EE) Version 1.4 applications. The configuration format for the deployment descriptor and the binding is different from previous versions of WebSphere Application Server. You must migrate the existing applications to J2EE 1.4 and migrate the Web services security configuration to the WebSphere Application Server Version 6 format. For migration information, see “Migrating Version 5.x applications with Web services security to Version 6 applications” on page 544.

**Web Services Interoperability Organization (WS-I) activities**

Web Services Interoperability Organization (WS-I) is an open industry effort to promote Web services interoperability across vendors, platforms, programming languages and applications. The organization is a consortium of companies across many industries including IBM, Microsoft, Oracle, Sun, Novell, VeriSign, and Daimler Chrysler. WS-I began working on the basic security profile (BSP) in the spring of 2003. BSP consists of a set of non-proprietary Web services specifications that clarifies and amplifies those specifications to promote Web services security interoperability across different vendor implementations. As of June 2004, BSP is a public draft. For more information, see the Web Services Interoperability Organization. Since the spring of 2004, WS-I has been working on sample application work. The BSP sample application is the testing tool for the BSP. WebSphere Application Server Version 6 does not support the WS-I BSP and BSP sample application.

**Related concepts**

“Default implementations of the Web services security service provider programming interfaces” on page 558

**Related tasks**

“Migrating Version 5.x applications with Web services security to Version 6 applications” on page 544

***XML token:***

XML tokens are offered in two well-known formats called Security Assertion Markup Language (SAML) and eXtensible rights Markup Language (XrML).

In WebSphere Application Server Version 6, you can plugin your own implementation. Using extensibility of the <wsse:Security> header in XML-based security tokens, you can directly insert these security tokens into the header. SAML assertions are attached to Web services security messages using Web services by placing assertion elements inside the <wsse:Security> header. The following example illustrates a Web services security message with a SAML assertion token.

```
<S:Envelope xmlns:S="...">
<S:Header>
      <wsse:Security xmlns:wsse="...">
          <saml:Assertion
                    MajorVersion="1"
                    MinorVersion="0"
                    AssertionID="SecurityToken-ef375268"
                       Issuer="elliotw1"
                       IssueInstant="2002-07-23T11:32:05.6228146-07:00"
                     xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion">
                     ...
           </saml:Assertion>
      </wsse:Security>
```

```
 </S:Header>
 <S:Body>
 ...
 </S:Body>
</S:Envelope>
```

For more information on SAML and XrML, see Web services: Resources for learning.

**Related reference**

Web services: Resources for learning

***Supported functionality from OASIS specifications:*** WebSphere Application Server Version 6 supports the following Web services security specifications and profiles.

- OASIS: Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)
- OASIS: Web Services Security: UsernameToken Profile 1.0
- OASIS: Web Services Security X.509 Certificate Token Profile 1.0

### OASIS: Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)

The following list shows the aspects of the OASIS: Web Services Security: SOAP Message Security 1.0 (WS-Security 2004) specification that is supported in WebSphere Application Server Version 6.

| Supported topic | Specific aspect that is supported |
| --- | --- |
| Security header | • @S11 :actor (for an intermediary)<br>• @S11:mustUnderstand |
| Security tokens | • Username token (user name and password)<br>• Binary security token (X.509 and Lightweight Third Party Authentication (LTPA))<br>• Custom token<br>– Other binary security token<br>– XML token<br>**Note:** WebSphere Application Server does not provide an implementation, but you can use an XML token with plug-in point. |
| Token references | • Direct reference<br>• Key identifier<br>• Key name<br>• Embedded reference |

| Supported topic | Specific aspect that is supported |
|---|---|
| Signature algorithms | • **Digest**<br>**SHA1** http://www.w3.org/2000/09/xmldsig#sha1<br>• **MAC**<br>**HMAC-SHA1**<br>http://www.w3.org/2000/09/xmldsig#hmac-sha1<br>• **Signature**<br>**DSA with SHA1**<br>http://www.w3.org/2000/09/xmldsig#dsa-sha1<br>**RSA with SHA1**<br>http://www.w3.org/2000/09/xmldsig#rsa-sha1<br>• **Canonicalization**<br>**Canonical XML (with comments)**<br>http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments<br>**Canonical XML (without comments)**<br>http://www.w3.org/TR/2001/REC-xml-c14n-20010315<br>**Exclusive XML canonicalization (with comments)**<br>http://www.w3.org/2001/10/xml-exc-c14n#WithComments<br>**Exclusive XML canonicalization (without comments)**<br>http://www.w3.org/2001/10/xml-exc-c14n#<br>• **Transform**<br>**STR transform**<br>http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform<br>http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soapmessage-security-1.0#STR-Transform<br>**XPath** http://www.w3.org/TR/1999/REC-xpath-19991116<br>**Enveloped signature**<br>http://www.w3.org/2000/09/xmldsig#enveloped-signature<br>**XPath Filter2**<br>http://www.w3.org/2002/06/xmldsig-filter2<br>**Decryption transform**<br>http://www.w3.org/2002/07/decrypt#XML |

<table>
<tr><th>Supported topic</th><th>Specific aspect that is supported</th></tr>
<tr><td>Signature signed parts</td><td>• WebSphere Application Server key words:<br>– body, which signs the Simple Object Access Protocol (SOAP) message body<br>– timestamp, which signs all of the time stamps<br>– securitytoken, which signs all of the security tokens<br>– dsigkey, which signs the signing key<br>– enckey, which signs the encryption key<br>– messageid, which signs the wsa :MessageID element in WS-Addressing.<br>– to, which signs the wsa:To element in WS-Addressing<br>– action, which signs the wsa:Action element in WS-Addressing<br>– relatesto, which signs the wsa:RelatesTo element in WS-Addressing<br>wsa is the namespace prefix of http://schemas.xmlsoap.org/ws/2004/08/addressing<br>• XPath expression to select an XML element in a Simple Object Access protocol (SOAP) message. For more information, see http://www.w3.org/TR/1999/REC-xpath-19991116.</td></tr>
<tr><td>Encryption algorithms</td><td>• Block encryption<br>– Triple DES in CBC: http://www.w3.org/2001/04/xmlenc#tripledes-cbc<br>– AES128 in CBC: http://www.w3.org/2001/04/xmlenc#aes128-cbc<br>– AES192 in CBC: http://www.w3.org/2001/04/xmlenc#aes192-cbc<br>This algorithm requires the unrestricted JCE policy file. For more information, see the Key encryption algorithm description in the “Encryption information configuration settings” on page 729.<br>– AES256 in CBC: http://www.w3.org/2001/04/xmlenc#aes256-cbc<br>This algorithm requires the unrestricted JCE policy file. For more information, see the Key encryption algorithm description in the “Encryption information configuration settings” on page 729.<br>• Key transport<br>– RSA Version 1.5: http://www.w3.org/2001/04/xmlenc#rsa-1_5<br>• Symmetric key wrap<br>– Triple DES key wrap: http://www.w3.org/2001/04/xmlenc#kw-tripledes<br>– AES key wrap (aes128): http://www.w3.org/2001/04/xmlenc#kw-aes128<br>– AES key wrap (aes192): http://www.w3.org/2001/04/xmlenc#kw-aes192<br>This algorithm requires the unrestricted JCE policy file. For more information, see the Key encryption algorithm description in the “Encryption information configuration settings” on page 729.<br>– AES key wrap (aes256): http://www.w3.org/2001/04/xmlenc#kw-aes256<br>This algorithm requires the unrestricted JCE policy file. For more information, see the Key encryption algorithm description in the “Encryption information configuration settings” on page 729.<br>• Manifests-xenc is the namespace prefix of http://www.w3.org/TR/xmlenc-core<br>– xenc:ReferenceList<br>– xenc:EncryptedKey<br><br>Advanced Encryption Standard (AES) is designed to provide stronger and better performance for symmetric key encryption over Triple-DES. Therefore, it is recommended that you use AES, if possible, for symmetric key encryption.</td></tr>
</table>

| Supported topic | Specific aspect that is supported |
| --- | --- |
| Encryption message parts | • WebSphere Application Server keywords<br>– bodycontent, which is used to encrypt the SOAP body content<br>– usernametoken, which is used to encrypt the username token<br>– digestvalue, which is used to encrypt the digest value of the digital signature<br>• XPath expression to select the XML element in the SOAP message<br>– XML elements<br>– XML element contents |
| Time stamp | • Within Web services security header<br>• WebSphere Application Server is extended to allow you to insert time stamps into other elements so that the age of those elements can be determined. |
| Error handling | SOAP faults |

### OASIS: Web Services Security: UsernameToken Profile 1.0

The following list shows the aspects of the OASIS: Web Services Security: UsernameToken Profile 1.0 specification that is supported in WebSphere Application Server Version 6.

| Supported topic | Specific aspect that is supported |
| --- | --- |
| Password types | Text |
| Token references | Direct reference |

### OASIS: Web Services Security X.509 Certificate Token Profile

The following list shows the aspects of the OASIS: Web Services Security X.509 Certificate Token Profile specification that is supported in WebSphere Application Server Version 6.

| Supported topic | Specific aspect that is supported |
| --- | --- |
| Token types | • X.509 Version 3: Single certificate<br>http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509- token-profile-1.0#X509v3<br>• X.509 Version 3: X509PKIPathv1 without certificate revocation lists (CRL)<br>http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509- token-profile-1.0#X509PKIPathv1<br>• X.509 Version 3: PKCS7 with or without CRLs. The IBM software development kit (SDK) supports both. The Sun Java Development Kit (JDK) supports PKCS7 without CRL only. |
| Token references | • Key identifier – subject key identifier<br>• Direct reference<br>• Custom reference – issuer name and serial number |

### Functionality that is not supported

The following list shows the functionality that is supported in the OASIS specifications, OASIS drafts, and other recommendations, but is not supported by WebSphere Application Server Version 6:

- Non-managed client with Web services security. For example, a Java 2 Platform, Standard Edition (J2SE) client or a Dynamic Invocation Interface (DII) client
- The Web services security binding is not collected during the application installation process. It can be configured after the application is deployed.

- Web services security for SOAP attachment
- SAML token profile, WS-SecurityKerberos token profile, and XrML token profile
- Web Services Interoperability Organization (WS-I) basic security profile
- XML enveloping digital signature
- XML enveloping digital encryption
- Security header
  - @S12:role

    S12 is the namespace prefix of http://www.w3.org/2003/05/soap-envelope
- The following transport algorithms for digital signatures are not supported:
  - XSLT: http://www.w3.org/TR/1999/REC-xslt-19991116
  - SOAP Message Normalization

    For more information, see SOAP Version 1.2 Message Normalization.
- The following key transport algorithm for encryption is not supported:
  - RSA-OAEP: http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p.
- The following key agreement algorithm for encryption is not supported:
  - Diffie-Hellman: http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/Overview.html#sec-DHKeyValue
- The following canonicalization algorithm for encryption, which is optional in the XML encryption specification, is not supported:
  - Canonical XML with or without comments
  - Exclusive XML canonicalization with or without comments
- In the Username Token Version 1.0 Profile specification, the digest password type is not supported.

**Related reference**

"Encryption information configuration settings" on page 729
Use this page to configure the encryption and decryption parameters. You can use these parameters to encrypt and decrypt various parts of the message, including the body and user name token.

## Web services security enhancements

WebSphere Application Server Version 6 includes a number of enhancements for securing Web services. These enhancements are explained in detail within this article.

**Building your applications**

To assemble your applications and to specify the security constraints for Web services security in the deployment descriptor and bindings, it is recommended that you use Rational Web Developer and the Application Server Toolkit. For more information on these tools, see "Assembly tools" in the "Developing and deploying applications" PDF. You can also use the WebSphere Application Server administrative console to edit the application binding file.

**Using identity assertion**

In a secured environment such as an intranet, a secure sockets layer (SSL) connection or through a Virtual Private Network (VPN), it is useful to send the requester identity only without credentials, such as password, with other trusted credentials, such as the server identity. WebSphere Application Server Version 6 supports the following types of identity assertions:

- A username token without a password
- An X.509 Token for a X.509 certificate

For the X.509 certificate, WebSphere Application Server uses the distinguished name in the certificate as a requester identity. There are two trust modes for validating the trust of the upstream server:

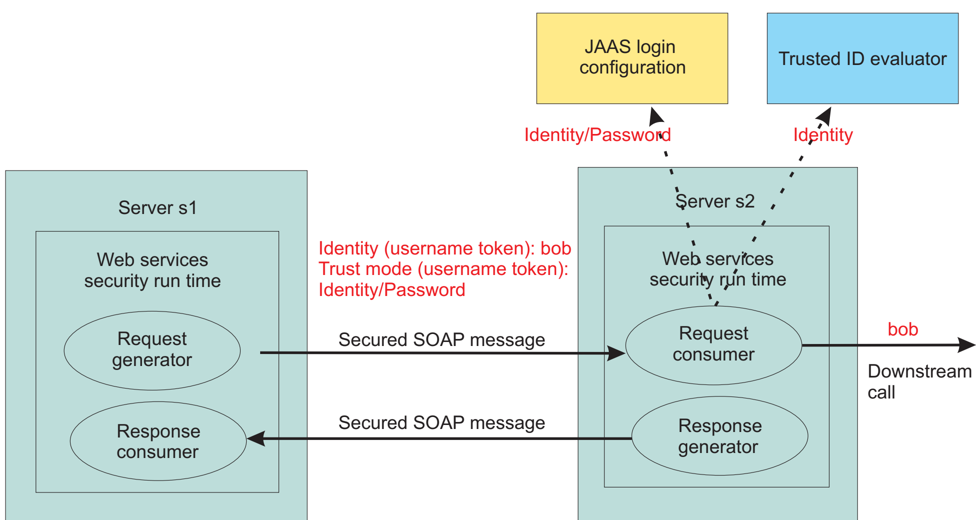
**Basic authentication (username token)**

The upstream server sends a username token with a user name and password to a downstream server. The consumer or receiver of the message authenticates the username token and validates the trust based upon the TrustedIDEvaluator implementation. The TrustedIDEvaluator implementation must implement the com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator Java interface.

**Signature**

The upstream server signs the message, which can be any message part such as the Simple Object Access Protocol (SOAP) body. The upstream server sends the X.509 token to a downstream server. The consumer or receiver of the message verifies the signature and validates the X.509 token. The identity or the distinguished name from the X.509 token that is used in the digital signature is validated based on the TrustedIDEvaluator implementation. The TrustedIDEvaluator implementation must implement the com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator Java interface.

The following figure demonstrates the identity assertion trust process.

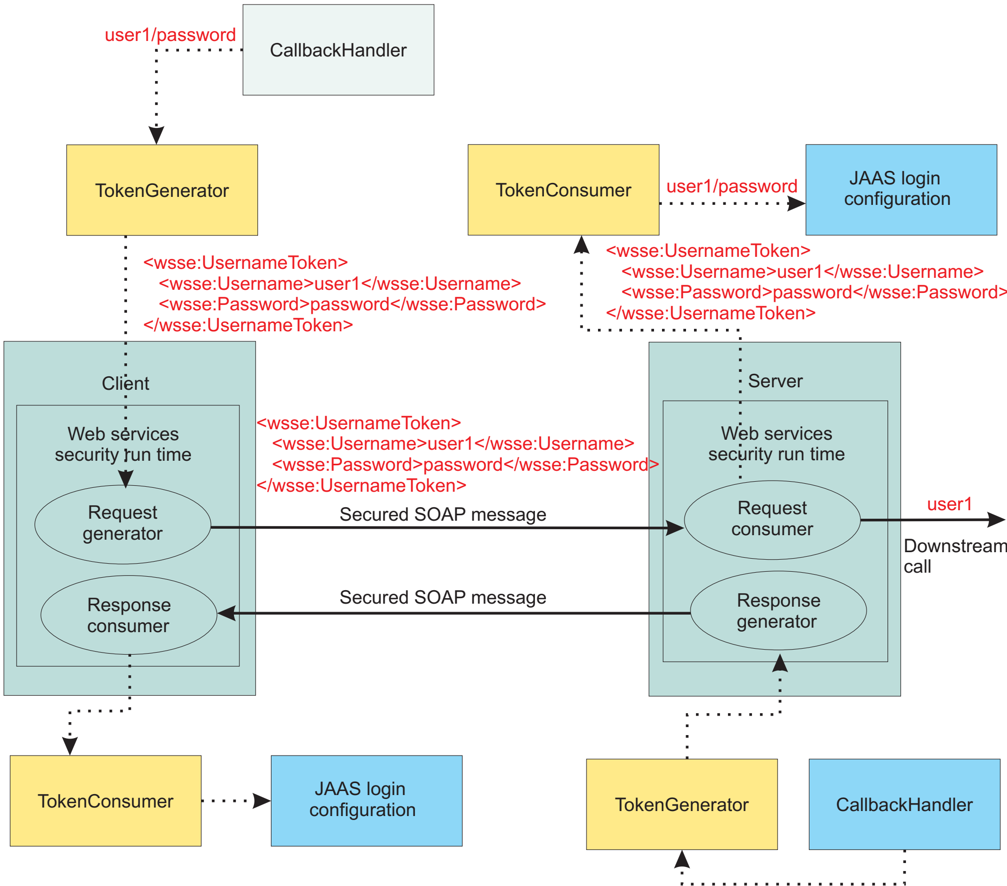



In this figure, server s1 is the upstream server and identity assertion is set up between server s1 and server s2. The s1 server authenticates the identity called *bob*. Server s1 wants to send *bob* to the s2 server with a password. The trust mode is an s1 credential that contains the identity and a password. Server s2 receives the request, authenticates the user using a Java Authentication and Authorization Service (JAAS) login module, and uses the trusted ID evaluator to determine whether to trust the identity. If the identity is trusted, *bob* is used as the caller that invokes the service. If authorization is required, *bob* is the identity that is used for authorization verification.

In WebSphere Application Server Version 6, the identity can be asserted as the RunAs (invocation) identity of the current security context. For example, the Web services gateway authenticates a requester using a secure method such as password authentication and then sends the requester identity only to a back-end server. You might also use identity assertion for interoperability with another Web services security implementation.

**Using the pluggable token framework**

The Organization for the Advancement of Structured Information Standards (OASIS) Web Services Security Version 1.0 specification defines a generic mechanism to associate security tokens with a SOAP message. In WebSphere Application Server Version 6, the pluggable token framework is enhanced to handle this flexible mechanism. The following figure shows this pluggable framework.




The following terms are used in the previous figure:

**TokenGenerator**

The token generator, or the com.ibm.wsspi.wssecurity.token.TokenGeneratorComponent Java interface, is responsible for the following actions:

- Marshalling the token into the correct XML representation for the SOAP message. In this case, marshalling is the process of converting a token to a standardized format before transmitting it over the network.
- Setting the token to the local JAAS Subject.
- Generating the correct token identifier based on the key information type.

The token generator invokes the CallbackHandler or the javax.security.auth.callback.CallbackHandler Java interface for token acquisition. The javax.security.auth.callback.Callback Java interface is used to pass information from the callback handler to the token generator.

**CallbackHandler**

The callback handler, or the javax.security.auth.callback.CallbackHandler Java interface, is responsible for acquiring the token using a method such as GUI prompt, a standard-in prompt, talking to external token service, and so on.

**TokenConsumer**

The token consumer, or the com.ibm.wsspi.wssecurity.token.TokenConsumerComponent Java interface, is responsible for the following actions:

- Un-marshalling the token from the XML format within the SOAP message. In this case, un-marshalling is the process of converting the token from the standard network format to the local or native format.
- Calling the JAAS login configuration to validate the token

- Setting the correct WSSToken, or com.ibm.wsspi.wssecurity.auth.token.WSSToken Java abstract class, to the local JAAS Subject.

At the final stage of Web services security processing, the local JAAS Subject content is used to create the WebSphere credentials and principals. The Caller Subject is created based on the content of the local JAAS Subject.

**JAAS login configuration**
The JAAS login configuration is responsible for validating the token. The validation process might involve making a call to the WebSphere Application Server authentication module or calling a third-party token service.

### Signing or encrypting data with a custom token

The key locator, or the com.ibm.wsspi.wssecurity.keyinfo.KeyLocator Java interface, is enhanced to support the flexibility of the specification. The key locator is responsible for locating the key. The local JAAS Subject is passed into the KeyLocator.getKey() method in the context. The key locator implementation can derive the key from the token, which is created by the token generator or the token consumer, to sign a message, to verify the signature within a message, to encrypt a message, or to decrypt a message.

**Important:** The com.ibm.wsspi.wssecurity.keyinfo.KeyLocator Java interface is different from the version in WebSphere Application Server Version 5.x. The com.ibm.wsspi.wssecurity.config.KeyLocator interface from version 5.x is deprecated. There is no automatic migration for the key locator from version 5.x to version 6. You must migrate the source code for the version 5.x key locator implementation to the key locator programming model for version 6.

### Signing or encrypting any XML element

The deployment descriptor supports the XPath expression for selecting which XML element to sign or encrypt. However, an envelope signature is used when you sign the SOAP envelope, SOAP header, or Web services security header.

### Supporting LTPA

Lightweight Third Party Authentication (LTPA) is supported as a binary security token in Web services security. The token type is http://www.ibm.com/websphere/appserver/tokentype/5.0.2/LTPA.

### Extending the support for time stamps

You can insert a time stamp in other elements during the signing process besides the Web services security header. This time stamp provides a mechanism for adding a time limit to an element. However, this support is an extension for WebSphere Application Server Version 6. Other vendor implementations might not have the ability to consume a message that is generated with an additional time stamp that is inserted in the message.

### Extending the support for nonce

You can insert a nonce, which is a randomly generated value, in other elements beside the username token. The nonce is used to reduce the chance of a replay attack. However, this support is an extension for WebSphere Application Server Version 6. Other vendor implementations might not have the ability to consume messages with a nonce that is inserted into elements other than a username token.

### Supporting distributed nonce caching

Distributed nonce caching is a new feature for Web services in WebSphere Application Server Version 6 that enables you to replicate nonce data between servers in a cluster. For example, you might have

application server A and application server B in cluster C. If application server A accepts a nonce with a value of X, then application server B throws a SoapSecurityException if it receives the nonce with the same value within a specified period of time. For more information, see the information that explains nonce cache timeout, nonce maximum age and nonce clock skew in “Web services: Default bindings for the Web services security collection” on page 811. However, if application server B receives another nonce with a value of Y, then it does not throw an exception, but caches the nonce and copies it into the other application servers within the same cluster.

**Important:** The distributed nonce caching feature uses the WebSphere Application Server Distributed Replication Service (DRS). The data in the local cache is pushed to the cache in other servers in the same replication domain. The replication is an out-of-process call and, in some cases, is a remote call. Therefore, there is a possible delay in replication while the content of the cache in each application server within the cluster is updated. The delay might be due to network traffic, network workload, machine workload, and so on.

### Caching the X.509 certificate

WebSphere Application Server Version 6 caches the X.509 certificates it receives, by default, to avoid certificate path validation and improve its performance. However, this change might lead to security exposure. You can disable X.509 certificate caching using the following steps:

On the cell level:

- Click **Security > Web services**.
- Under Additional properties, click **Properties > New**.
- In the Property name field, type `com.ibm.ws.wssecurity.config.token.certificate.useCache`.
- In the Property value field, type `false`.

On the server level:

- Click **Servers > Application servers >** *server_name*.
- Under Security, click **Web services: Default bindings for Web services security**.
- Under Additional properties, click **Properties > New**.
- In the Property name field, type `com.ibm.ws.wssecurity.config.token.certificate.useCache`.
- In the Property value field, type `false`.

### Providing support for a certificate revocation list

The certificate revocation list (CRL) in WebSphere Application Server Version 6 is used to enhance certificate path validation. You can specify a CRL in the collection certificate store for validation. You can also encode a CRL in an X.509 token using PKCS#7 encoding. However, WebSphere Application Server Version 6 does not support X509PKIPathv1 CRL encoding in a X.509 token.

**Important:** The PKCS#7 encoding was tested with the IBM certificate path (IBM CertPath) provider only. The encoding is not supported for other certificate path providers.

**Related concepts**

“Nonce, a randomly generated token” on page 567
*Nonce* is a randomly generated, cryptographic token used to prevent replay attacks. Although Nonce can be inserted anywhere in the SOAP message, it is typically inserted in the <UsernameToken> element.

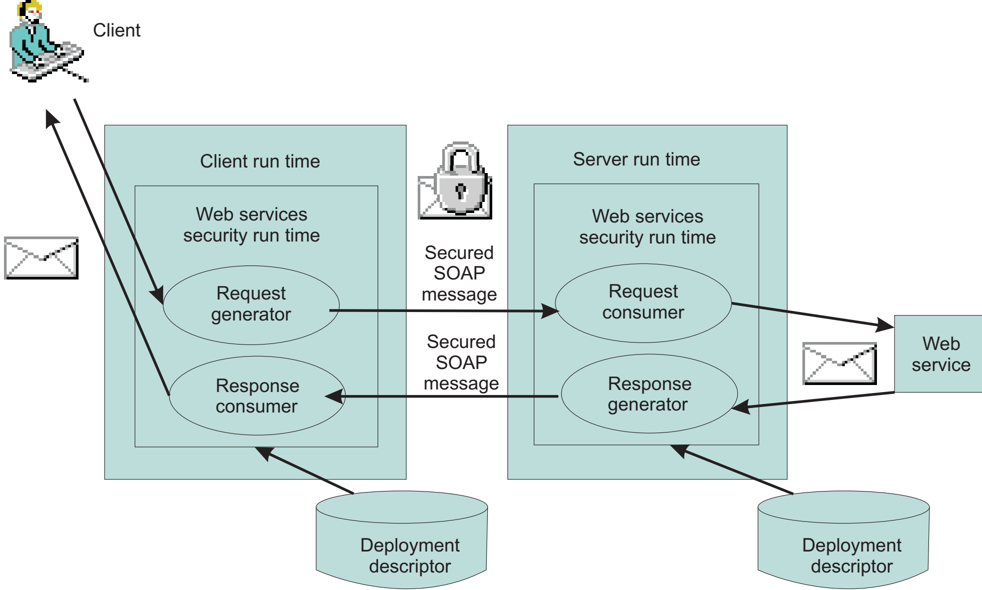
“Collection certificate store” on page 585
A *collection certificate store* is a collection of non-root, certificate authority (CA) certificates and certificate revocation lists (CRLs). This collection of CA certificates and CRLs is used to check the signature of a digitally signed Simple Object Access Protocol (SOAP) message.

“Certificate revocation list” on page 585
A *certificate revocation list* is a time-stamped list of certificates that have been revoked by a certificate authority (CA).

## High-level architecture for Web services security

WebSphere Application Server Version 6 uses the Java 2 Platform, Enterprise Edition (J2EE) Version 1.4 Web services deployment model to implement Web services security. The Web services security constraints are specified in the IBM extension of the Web services deployments descriptors and bindings. The Web services security run time enforces the security constraints specified in the deployment descriptors. One of the advantages of deployment model is that you can define the Web services security requirements outside of the application business logic. With the separation of roles, the application developer can focus on the business logic and the security expert can specify the security requirement.

The following figure shows the high-level architecture model that is used to secure Web services in WebSphere Application Server Version 6.




The deployment descriptor and binding for Web services security is based on Web service ports. Each Web service port can have its own unique Web services security constraints defined. For example, you might configure Web service port A to sign the Simple Object Access Protocol (SOAP) body and the username token. You might configure Web service port B to encrypt the SOAP body content and so on.

As shown in the previous figure, there are 2 sets of configurations on both the client side and the server side:

**Request generator**
: This client-side configuration defines the Web services security requirements for the outgoing SOAP message request. These requirements might involve generating a SOAP message request that uses a digital signature, incorporates encryption, and attaches security tokens. In WebSphere Application Server Versions 5.0.2, 5.1, and 5.1.1, the request generator was known as the request sender.

**Request consumer**
: This server-side configuration defines the Web services security requirements for the incoming SOAP message request. These requirements might involve verifying that the required integrity parts are digitally signed; verifying the digital signature; verifying that the required confidential parts were encrypted by the request generator; decrypting the required confidential parts; validating the

security tokens, and verifying that the security context is set up with the appropriate identity. In WebSphere Application Server Versions 5.0.2, 5.1, and 5.1.1, the request consumer was known as the request receiver.

**Response generator**
This server-side configuration defines the Web services security requirements for the outgoing SOAP message response. These requirements might involve generating the SOAP message response with Web services security; including digital signature; and encrypting and attaching the security tokens, if necessary. In WebSphere Application Server Versions 5.0.2, 5.1, and 5.1.1, the response generator was known as the response sender.

**Response consumer**
This client-side configuration defines the Web services security requirements for the incoming SOAP response. The requirements might involve verifying that the integrity parts are signed and the signature is verified; verifying that the required confidential parts are encrypted and that the parts are decrypted; and validating the security tokens. In WebSphere Application Server Versions 5.0.2, 5.1, and 5.1.1, the response consumer was known as the response receiver.

WebSphere Application Server Version 6 does not include security policy negotiation or exchange between the client and server. This security policy negotiation is defined by the WS-Policy, WS-PolicyAssertion, and WS-SecurityPolicy specifications and are not supported in WebSphere Application Server Version 6.

**Note:** The Web services security requirements that are defined in the request generator must match the request consumer. The requirements that are defined in the response generator must match the response consumer. Otherwise, the request or response is rejected because the Web services security constraints can not be met by the request consumer and response consumer.

The format of the Web services security deployment descriptors and bindings are IBM proprietary. However, the following tools are available to edit the deployment descriptors and bindings:

**Rational Application Developer Version 6**
Use this tool to edit the Web services security deployment descriptor and binding. You can use this tool to assemble both Web and EJB modules.

**Rational Web Developer Version 6**
Use this tool to edit the Web services security deployment descriptor and binding. You can use this tool to assemble Web modules only.

**Application Server Toolkit**
Use this tool to edit the Web services security deployment descriptor and binding.

**WebSphere Application Server Version 6 administrative console**
Use this tool to edit the Web services security binding of a deployed application.

**Related reference**

“Request generator (sender) binding configuration settings” on page 680
Use this page to specify the binding configuration for the request generator.

“Request consumer (receiver) binding configuration settings” on page 745
Use this page to specify the binding configuration for the request consumer.

“Response generator (sender) binding configuration settings” on page 682
Use this page to specify the binding configuration for the response generator or response sender.

“Response consumer (receiver) binding configuration settings” on page 747
Use this page to specify the binding configuration for the response consumer.

## Configuration overview

The Web services security constraints are defined in an IBM extension of the Web services deployment descriptor for Java 2 Platform, Enterprise Edition (J2EE). The IBM extension deployment descriptor and binding for Web services security are IBM proprietary. Due to the complexity of these files, it is not recommended that you edit the deployment descriptor and binding files manually with a text editor

because they might cause errors. It is recommended, however, that you use the tools provided by IBM to configure the Web services security constraints for an application. These tools are the Rational Application Developer, Rational Web Developer, the Application Server Toolkit, and the WebSphere Application Server administrative console.

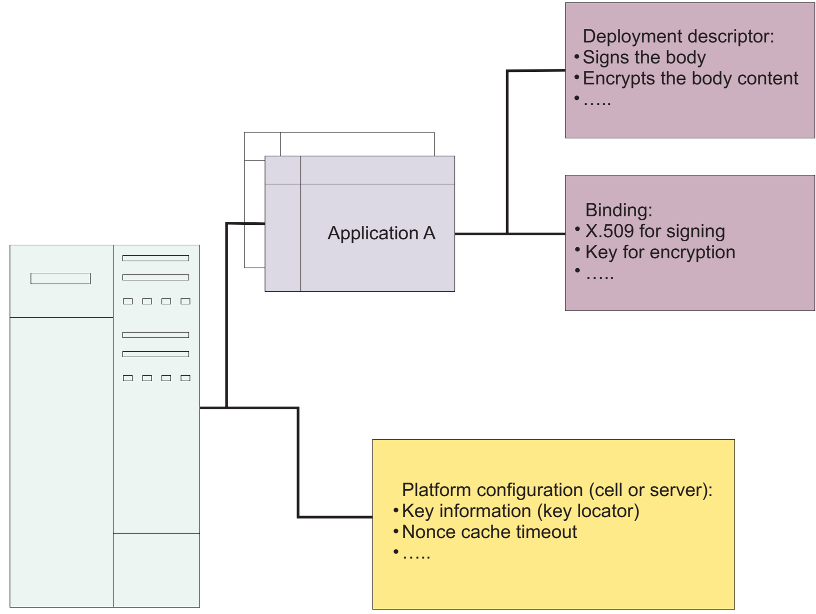
The following table provides the names of the deployment descriptor and binding files for the client and the server.

| File type | Client side | Server side |
| --- | --- | --- |
| Deployment descriptor | ibm-webservicesclient-ext.xmi | ibm-webservices-ext.xmi |
| Binding file | ibm-webservicesclient-bnd.xmi | ibm-webservices-bnd.xmi |

The ″what″ is specified in the deployment descriptor such as what message part to sign and which token to encrypt. The ″how″ is specified in the binding file such as how the message is signed, how to generate and consume the security token.

In addition to the application deployment descriptor and binding files, WebSphere Application Server Version 6 has a server level WSS configuration. These configurations are global for all applications. Because WebSphere Application Server Version 6 supports 5.x applications, some of the configurations are valid for version 5.x applications only and some are valid for version 6 applications only.

The following figure represents the relationship of the application deployment descriptor and binding files to the cell or server level configuration.




## Platform configuration overview

The following options are available in the administrative console:

**Nonce cache timeout**
This option, which is found on the cell level (Network Deployment only) and server level, specifies the cache timeout value for a nonce in seconds.

**Nonce maximum age**
: This option, which is found on the cell level (Network Deployment only) and server level, specifies the default life span for the nonce in seconds.

**Nonce clock skew**
: This option, which is found on the cell level (Network Deployment only) and server level, specifies the default clock skew to account for network delay, processing delay, and so on. It is used to calculate when the nonce expires. Its unit of measurement is seconds.

**Distribute nonce caching**
: This feature enables you to distribute the cache for the nonce to different servers in a cluster. It is a new feature for WebSphere Application Server Version 6.

The following features can be referenced in the application binding:

**Key locator**
: This feature specifies how the keys are retrieved for signing, encryption, and decryption. The implementation classes for the key locator are different in WebSphere Application Server Version 6 and Version 5.x.

**Collection certificate store**
: This feature specifies the certificate store for certificate path validation. It is typically used for validating X.509 tokens during signature verification or constructing the X.509 token with a certificate revocation list that is encoded in the PKCS#7 format. The certificate revocation list is supported for WebSphere Application Server Version 6 applications only.

**Trust anchors**
: This feature specifies the trust level for the signer certificate and is typically used in the X.509 token validation during signature verification.

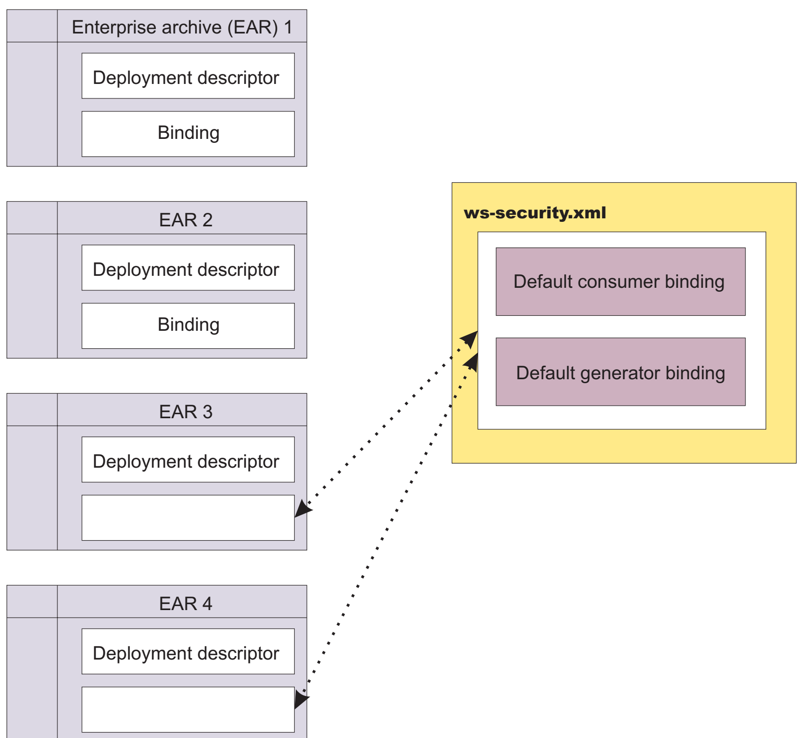
**Trusted ID evaluators**
: This feature specifies how to verify the trust level for the identity. The feature is used with identity assertion.

**Login mappings**
: This feature specifies the login configuration binding to the authentication methods. This feature is used by WebSphere Application Server Version 5.x applications only and it is deprecated.

### Default bindings

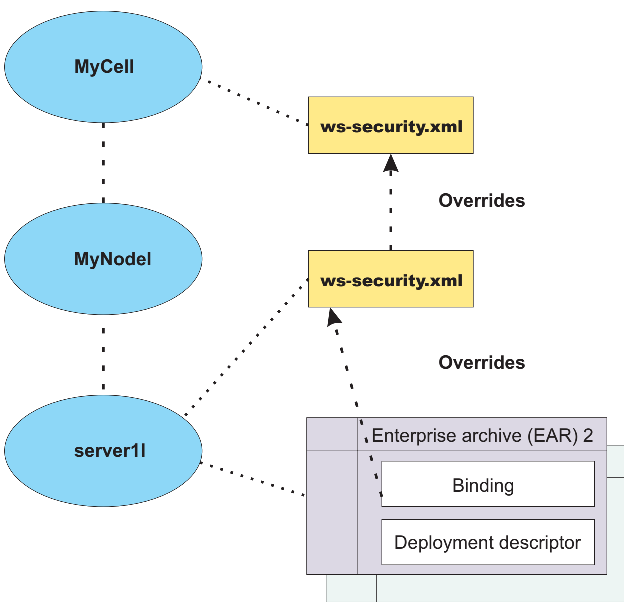
The default bindings specify the default binding so that applications do not have to define the binding in the application binding files for Web services security. There is only one set of default bindings and they can be shared by multiple applications. This feature is available for WebSphere Application Server Version 6 applications only.

The following figure shows the relationship between the application enterprise archive (EAR) file and the `ws-security.xml` file.




Application EAR 1 and EAR 2 have specific bindings in the application binding file. However, application EAR 3 and EAR 4 do not have a binding in the application binding file, but rather use the default binding defined in the `ws-security.xml` file. The configuration is resolved by nearest configuration in the hierarchy. For example, there might be three key locators named "mykeylocator" defined in the application binding file, the server level, and the cell level. If `mykeylocator` is referenced in the application binding, then the key locator defined in the application binding is used. The visibility scope of the data depends upon where the data is defined. If the data is defined in the application binding, then its visibility is scoped to that particular application. If the data is defined on the server level, then the visibility scope is all of the applications deployed on that server. If the data is defined on the cell level, then the visibility scope is all of the applications deployed on servers in the cell. In general, if data is not meant to be shared by other applications, define the configuration in the application binding level.

The following figure shows the relationship of the bindings on the application, server, and cell levels.




**Related concepts**

"Nonce, a randomly generated token" on page 567
*Nonce* is a randomly generated, cryptographic token used to prevent replay attacks. Although Nonce can be inserted anywhere in the SOAP message, it is typically inserted in the <UsernameToken> element.

"Distributed nonce caching" on page 763
The *distributed nonce caching* feature enables you to distribute the cache for a nonce to different servers in a cluster.

"Key locator" on page 592
A key locator or the com.ibm.wsspi.wssecurity.keyinfo.KeyLocator class, is an abstraction of the mechanism that retrieves the key for digital signature and encryption.

"Collection certificate store" on page 585
A *collection certificate store* is a collection of non-root, certificate authority (CA) certificates and certificate revocation lists (CRLs). This collection of CA certificates and CRLs is used to check the signature of a digitally signed Simple Object Access Protocol (SOAP) message.

"Trust anchor" on page 587
A *trust anchor* specifies the key stores that contain trusted root certificates. These certificates are used to validate the X.509 certificate that is embedded in the Simple Object Access Protocol (SOAP) message.

"Trusted ID evaluator" on page 613
A *trusted ID evaluator* (com.ibm.wsspi.wssecurity.id.TrustedIDEvaluatorImpl) is an abstraction of the mechanism that evaluates whether the given ID name is trusted.

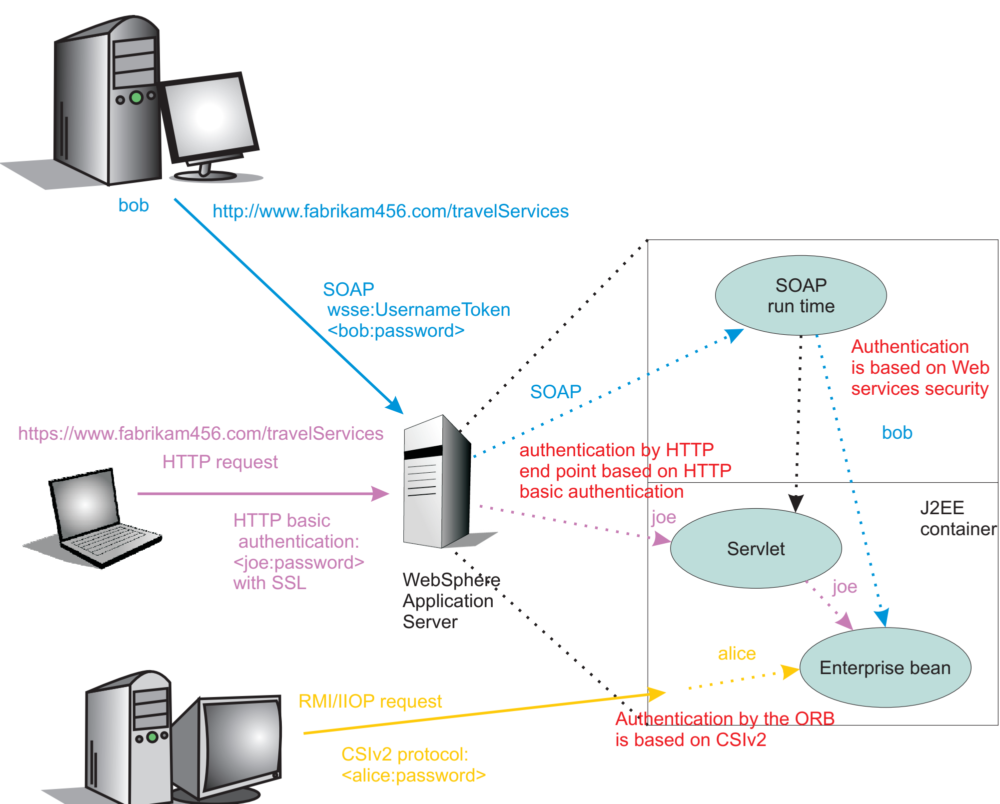
## Security model mixture

There can be multiple protocols and channels in the WebSphere Application Server Version 6 programming environment. For example, you might access a Web-based application through the HTTP transport. For example, a servlet, JavaServer pages (JSP) file, HTML and so on. You might access an enterprise application through the Remote Method Invocation over the Internet Inter-ORB (RMI/IIOP) protocol and a Web service application through the Simple Object Access Protocol (SOAP) over HTTP, SOAP over the Java Message Service (JMS) or SOAP over RMI/IIOP protocol. Each of these applications serve different business needs. More importantly, Web services are often implemented as servlets with a JavaBean or EJB file. Therefore, you can mix and match the Web services security model with the Java 2 Platform, Enterprise Edition (J2EE) security model for Web and EJB components. It is intended that Web service security compliment the J2EE role-based security and the security run time for WebSphere Application Server Version 6.

Web services security also can take advantage of the security features in J2EE and the security run time for WebSphere Application Server Version 6. For example, Web services security can use the following security features to provide an end-to-end security deployment:

- Use the local OS, Lightweight Directory Access Protocol (LDAP), and custom user registries for authenticating the username token
- Propagate the Lightweight Third Party Authentication (LTPA) security token in the SOAP message
- Use identity assertion
- Use a trust association interceptor (TAI)
- Enable security attribute propagation
- Use J2EE role-based authorization
- Use a Java Authorization Contract for Containers (JACC) authorization provider such as Tivoli Access Manager

The following figure shows that different security protocols are used to send authentication information to the application server. For a Web service, you might use either HTTP basic authentication with Secure Sockets Layer (SSL) or a Web services security username token with encryption. In the following figure, when identity *bob* from Web services security is authenticated and set as the caller identity of the SOAP message request, the J2EE Enterprise JavaBean (EJB) container performs authorization using *bob* before the call is dispatched to the service implementation, which, in this case, is the enterprise bean.




You can secure a Web service using the transport layer security. For example, when you are using SOAP over HTTP, HTTPs can be used to secure the Web service. However, transport layer security provides point-to-point security only. This layer of security might be adequate for certain scenarios. However, when the SOAP message must travel through intermediary servers (multi-hop) before it is consumed by the target endpoint, you might use SOAP over the Java Message Service (JMS). The usage scenarios and security requirements dictate how to secure Web services. The requirements depend upon the operating

environment and the business needs. However, one key advantage of using Web services security is that it is transport layer independent; the same Web services security constraints can be used for SOAP over HTTP, SOAP over JMS, or SOAP over RMI/IIOP.

## Security considerations for Web services

In WebSphere Application Server Version 6, when you enable integrity, confidentiality, and the associated tokens within a Simple Object Access Protocol (SOAP) message, security is not guaranteed. When you configure Web services security, you must make every effort to verify that the result is not vulnerable to a wide range of attack mechanisms. This article provides some information about the possible security concerns that arise when you are securing Web services. This list of security concerns is not complete. You must conduct your own security analysis for your environment.

- Ensuring the message freshness

  Message freshness involves protecting resources from a replay attack in which a message is captured and resent. Digital signatures, by themselves, cannot prevent a replay attack because a signed message can be captured and resent. It is recommended that you allow message recipients to detect message replay attacks when messages are exchanged through an open network. You can use the following elements, which are described in the Web services security specifications, for this purpose:

  **Timestamp**
  : You can use the timestamp element to keep track of messages and to detect replays of previous messages. The WS-Security 2004 specification recommends that you cache time stamps for a given period of time. As a guideline, you can use five minutes as a minimum period of time to detect replays. Messages that contain an expired timestamp are rejected.

  **Nonce**
  : A nonce is a child element of UsernameToken in the UsernameToken profile. Because each Nonce element has a unique value, recipients can detect replay attacks with relative ease.

  **Important:** Both the time stamp and nonce element must be signed. Otherwise, these elements can be altered easily and therefore cannot prevent replay attacks.

- Using XML digital signature and XML encryption properly to avoid a potential security hole

  The WS-Security 2004 specification defines how to use XML digital signature and XML encryption in Simple Object Access Protocol (SOAP) headers. Therefore, users must understand XML digital signature and XML encryption in the context of other security mechanisms and their possible threats to an entity. For XML digital signature, you must be aware of all of the security implications resulting from the use of digital signatures in general and XML digital signature in particular. When you build trust into an application based on a digital signature, you must incorporate other technologies such as certification trust validation based upon the Public Key Infrastructure (PKI). For XML encryption, the combination of digital signing and encryption over a common data item might introduce some cryptographic vulnerabilities. For example, when you encrypt digitally signed data, you might leave the digital signature in plain text and leave your message vulnerable to plain text guessing attacks. As a general practice, when data is encrypted, encrypt any digest or signature over the data. For more information, see http://www.w3.org/TR/xmlenc-core/#sec-Sign-with-Encrypt.

- Protecting the integrity of security tokens

  The possibility of a token substitution attack exists. In this scenario, a digital signature is verified with a key that is often derived from a security token and is included in a message. If the token is substituted, a recipient might accept the message based on the substituted key, which might not be what you expect. One possible solution to this problem is to sign the security token (or the unique identifying data from which the signing key is derived) together with the signed data. In some situations, the token that is issued by a trusted authority is signed. In this case, there might not be an integrity issue. However, because application semantics and the environment might change over time, the best practice is to prevent this attack. You must assess the risk assessment based upon the deployed environment.

- Verifying the certificate to leverage the certificate path verification and the certificate revocation list

  It is recommended that you verify that the authenticity or validity of the token identity that is used for digital signature is properly trusted. Especially for an X.509 token, this issue involves verifying the

certificate path and using a certificate revocation list (CRL). In the Web services security implementation in WebSphere Application Server Version 6, the certificate is verified by the TokenConsumer element. WebSphere Application Server provides a default implementation for the X.509 certificate that uses the Java CertPath library to verify and validate the certificate. In the implementation, there is no explicit concept of a CRL. Rather, proper root certificates and intermediate certificates are prepared in files only. For a sophisticated solution, you might develop your own TokenConsumer implementation that performs certificate and CRL verification using the online CRL database or the Online Certificate Status Protocol (OCSP).

- Protecting the username token with a password

  It is recommended that you do not send a password in a UsernameToken to a downstream server without protection. You can use transport-level security such as SSL (for example, HTTPS) or use XML encryption within Web services security to protect the password. The preferred method of protection depends upon your environment. However, you might be able to send a password to a downstream server as plain text in some special environments where you are positive that you are not vulnerable to an attack.

Securing Web services involves more work than just enabling XML digital signature and XML encryption. To properly secure a Web service, you must have knowledge about the Public Key Infrastructure (PKI). The amount of security that you need depends upon the deployed environment and the usage patterns. However, there are some basic rules and best practices for securing Web services. It is recommended that you read some books on PKI and read information on the Web Services Interoperability Organization (WS-I) Basic Security Profile (BSP).

## Migrating Version 5.x applications with Web services security to Version 6 applications

You can install Java 2 Platform, Enterprise Edition (J2EE) Version 1.3 applications that use Web services security on a WebSphere Application Server Version 6 server. However, if you want J2EE Version 1.3 applications to use the Web services security (WSS) Version 1.0 specification and the other new features added in Version 6, you must migrate the J2EE Version 1.3 applications to J2EE Version 1.4. This article provides general information about migrating a J2EE Version 1.3 application that uses Web services security to a J2EE Version 1.4 application.

Complete the following steps to migrate a Version 5.x application, along with the Web services security configuration information, to a Version 6 application:

1. Save the original J2EE Version 1.3 application. You need the Web services security configuration files of the J2EE Version 1.3 application to recreate the configuration in the new format for the J2EE Version 1.4 application.
2. Use the Java 2 Platform, Enterprise Edition (J2EE) Migration Wizard in an assembly tool to migrate the J2EE Version 1.3 application to J2EE Version 1.4.

   **Important:** After you migrate to J2EE Version 1.4 using the J2EE Migration Wizard, you cannot view the J2EE Version 1.3 extension and binding information within an assembly tool. You can view the J2EE Version 1.3 Web services security extension and binding information using a text editor. However, do not edit the extension and binding information using a text editor. The J2EE Migration Wizard does not migrate the Web services security configuration files to the new format in the J2EE Version 1.4 application. Rather the wizard is used to migrate your files from J2EE Version 1.3 to Version 1.4.

   To access the J2EE Migration Wizard, complete the following steps:

   a. Right-click the name of your application.
   b. Click **Migrate > J2EE Migration Wizard**.
3. Manually delete all of the Web services security configuration information from the binding and extension files of the application that is migrated to J2EE Version 1.4.

a. Delete the `<securityRequestReceiverServiceConfig>` and `<securityResponseSenderServiceConfig>` sections from the server-side `ibm-webservices-ext.xmi` extension file.
b. Delete the `<securityRequestReceiverBindingConfig>` and `<securityResponseSenderBindingConfig>` sections from the server-side `ibm-webservices-bnd.xmi` binding file.
c. Delete the `<securityRequestSenderServiceConfig>` and `<securityResponseReceiverServiceConfig>` sections from the client-side `ibm-webservicesclient-ext.xmi` extension file.
d. Delete the `<securityRequestSenderBindingConfig>` and `<securityResponseReceiverBindingConfig>` sections from client-side `ibm-webservicesclient-bnd.xmi` binding file.

4. Recreate the Web services security configuration information in the new J2EE Version 1.4 format. At this stage, because the application is already migrated to the J2EE Version 1.4, you can use the Application Server Toolkit to configure the original Web services security information in the new Version 6 format.

This task provides general information about how to migrate J2EE Version 1.3 applications to J2EE Version 1.4.

The following articles contain some general scenarios that map some of the basic Web services security information specified in a J2EE Version 1.3 application to a J2EE Version 1.4 application and specify this information using the Application Server Toolkit. The Web services security configuration information is contained in four configuration files: two server-side configuration files and two client-side configuration files. The migration of all of the configuration information is divided into four sections; one for each configuration file. When you recreate the Web services security information in the new J2EE Version 1.4 format, it is recommended that you configure the extensions and binding files in the following order:

1. Configure the `ibm-webservices-ext.xmi` server-side extensions file. For more information, see “Migrating the server-side extensions configuration.”
2. Configure the `ibm-webservicesclient-ext.xmi` client-side extensions file. For more information, see “Migrating the client-side extensions configuration” on page 547.
3. Configure the `ibm-webservices-bnd.xmi` server-side bindings file. For more information, see “Migrating the server-side bindings file” on page 548.
4. Configure the `ibm-webservicesclient-bnd.xmi` client-side bindings file. For more information, see “Migrating the server-side bindings file” on page 548.

***Migrating the server-side extensions configuration:***

This article provides general information about migrating the Web services security server-side extensions configuration for a Java 2 Platform, Enterprise Edition (J2EE) Version 1.3 application to a J2EE Version 1.4 application. The steps are based on typical scenarios, but the steps are not all-inclusive.

The following table lists the mappings for the top-level sections under the server-side **Security Extensions** tab within an assembly tool from a J2EE Version 1.3 application to a J2EE Version 1.4 application.

*Table 4. The mapping of the configuration sections*

| J2EE Version 1.3 extensions configuration | J2EE Version 1.4 extensions configuration |
| --- | --- |
| Request Receiver Service Configuration Details | Request Consumer Service Configuration Details |
| Response Sender Service Configuration Details | Response Generator Service Configuration Details |

For information about the assembly tools that are available for WebSphere Application Server Version 6, see ″Assembly tools″ in the ″Developing and deploying applications″ PDF.

Consider the following steps to migrate the server-side extensions from a J2EE Version 1.3 application to a J2EE Version 1.4 application. These steps are dependent upon your specific configuration.

- Import the J2EE Version 1.3 application into an assembly tool and identify all the message parts that are required to be signed and encrypted. The message parts are listed in the Required Integrity and Required Confidentiality sections under the Request Receiver Service Configuration Details section. In a J2EE Version 1.4 application, these message parts map to the Message parts field of the **Required integrity** and **Required confidentiality** dialogs windows within the assembly tool. To specify these message parts within an assembly tool, complete the following steps in the Web Services editor:
  1. Click the **Extensions** tab.
  2. Navigate to the Required integrity subsection within the Request Consumer Service Configuration Details section.
  3. Specify each message part to be signed in the Message Parts field.

  For example, if the message part in the J2EE Version 1.3 application is `body`, you need to specify **body** in the Message parts keyword field. Similarly, on the **Extensions** tab, configure the message parts to be encrypted using the Required Confidentiality dialog. Also, for all the message parts that are migrated from a J2EE Version 1.3 application, you must select **http://www.ibm.com/websphere/webservices/wssecurity/dialect-was** in the Message parts dialect field and **Required** in the Usage type field.
- **Optional:** Configure the Required Security Token and Caller Part sections on the **Extensions** tab if the authentication method of BasicAuth is configured under the Login Config section of the J2EE Version 1.3 application. When you configure the Required Security Token section, select **Username** in the name field and **Required** in the Usage type field within the Required Security Token Dialog window. The following table shows how the authentication method values for a J2EE Version 1.3 application map to the token type values within the J2EE Version 1.4 application.

*Table 5. Authentication method to token type mappings*

| Login Config Authentication method values in the J2EE Version 1.3 extensions configuration | Token type values in the J2EE Version 1.4 extensions configuration |
| --- | --- |
| BasicAuth | UsernameToken |
| Signature | X509 certificate |
| LTPA | LTPAToken |

  If the authentication method value is IDAssertion within the Login Config section, the token type that you must specify in the J2EE Version 1.4 application depends upon the IDType value within the IDAssertion section. The following table shows how the IDType values for J2EE Version 1.3 application map to the token type values in the J2EE Version 1.4 application.

*Table 6. IDType values to token type mappings*

| IDType values in the J2EE Version 1.3 application extensions configuration | Token type values in the J2EE Version 1.4 application extensions configuration |
| --- | --- |
| X509Certificate | X509 certificate |
| Username | Username |

- Select the appropriate token type in the Name field of the Call Part Dialog window based on the previous two tables. Select the **Username** token type when you are configuring the caller part for the basic authentication method. Configuring the other token types in the Caller part dialog is similar to configuring token types in the Required Security Token dialog. If you need to map the IDAssertion authentication method from a J2EE Version 1.3 application to a J2EE Version 1.4 application, select the **Use IDAssertion** option and configure the ID assertion section of the Caller Part Dialog window. The Trust Mode field under the IDAssertion section maps to the Trust method name field of the Trust method property section in the Caller Part Dialog window. If Signature is selected for the Trust method, specify the Required Integrity part that specifies the signature of the trusted intermediary certificate.

- Configure a nonce in the Version 6 Binding Configurations section if nonce is specified in the Add Authentication Method dialog under Login Config within the J2EE Version 1.3 application extensions configuration.

  **Important:** Nonce is configured in the bindings for a J2EE Version 1.4 application and not in the extensions.

  To configure a nonce on the Binding Configurations tab, set the com.ibm.wsspi.wssecurity.token.Username.verifyNonce property in the Token Consumer configuration for the Username token.
- Configure the Add Timestamp section to migrate the time stamp information if the `<addReceivedTimestamp>` element is configured in the J2EE Version 1.3 extensions. To migrate the Response Sender Service Configuration Details section in the J2EE Version 1.3 extensions, identify all of the message parts listed within the Integrity and Confidentiality sections. Configure these message parts using the Integrity and Confidentiality dialogs under the Response Generator Service Configuration details section. This configuration is similar to the configuration for Required Integrity and Required Confidentiality, with the exception of the Order field in the Integrity Dialog. The value of this Order field specifies the order in which the message parts specified in the Message Parts field are digitally signed or encrypted in the Simple Object Access Protocol (SOAP) message. For example, the extensions contain the following information:
  - One integrity entry called `int_part1` with a value of `1` in the Order field
  - One confidentiality entry called `conf_part1` with a value of `2` in the Order field

  In this example, the message parts that are specified by the `int_part1` integrity entry are signed before the message parts specified by the `conf_part1` confidentiality entry are encrypted. The same rule for the order attribute applies for multiple integrity or confidentiality elements.

This set of steps describe the types of information that you need to migrate the Web services security server-side extensions for a J2EE Version 1.3 application to a J2EE Version 1.4 application.

Migrate the client-side extensions for a J2EE Version 1.3 application to a J2EE Version 1.4 application. For more information, see “Migrating the client-side extensions configuration.”

**Related tasks**

“Migrating the client-side extensions configuration”
“Migrating the server-side bindings file” on page 548
“Migrating the server-side bindings file” on page 548

***Migrating the client-side extensions configuration:***

This article provides general information about migrating the Web services security client-side extensions configuration for a Java 2 Platform, Enterprise Edition (J2EE) Version 1.3 application to a J2EE Version 1.4 application. The steps are based on typical scenarios, but the steps are not all-inclusive.

The following table lists the mappings of the top-level sections under the client-side **Security Extensions** tab for Web services security from a J2EE Version 1.3 application to a J2EE Version 1.4 application.

*Table 7. The mapping of the configuration sections*

| J2EE Version 1.3 security extensions for Web services security | J2EE Version 1.4 extensions for Web services security |
| --- | --- |
| Request Sender Configuration | Request Generator Configuration |
| Response Receiver Configuration | Response Consumer Configuration |

Consider the following steps to migrate the client-side extensions configuration from a J2EE Version 1.3 application to a J2EE Version 1.4 application. These steps are dependent upon your specific configuration.

- Migrate the message parts that you need to sign or encrypt from the Integrity and Confidentiality sections in the J2EE Version 1.3 application to the Integrity and Confidentiality sections on the **WS Extensions** tab in an assembly tool for a J2EE Version 1.4 application.
- Configure the Security Token section under the Request Generator Configuration on the **WS Extensions** tab if Login Config section is configured in the J2EE Version 1.3 extensions configuration. When you configure the security token, select the token type in the Token type field that matches the authentication method value of the Login Config in the J2EE Version 1.3 application. For example, if the authentication method in the J2EE Version 1.3 extensions configuration is BasicAuth, then select **Username** in the Token type field within the assembly tool. For more information on how the authentication methods for Web services security map from a J2EE Version 1.3 application to a J2EE Version 1.4 application, see Table 5 on page 546. If the authentication method is IDAssertion, there is no action required because in a J2EE Version 1.4 application the identity assertion configuration is not required in the client-side extensions configuration. In a J2EE Version 1.4 application, the identity assertion configuration is specified in the server-side extensions configuration and in the client-side bindings configuration.
- Migrate the Required Integrity and Required Confidentiality sections by configuring the Required Integrity and Required Confidentiality sections in an assembly tool. Migrating the Response Receiver Configuration section is similar to migrating the Request Receiver Service Configuration Details section of the server-side extensions configuration. For more information, see "Migrating the server-side extensions configuration" on page 545.
- Migrate the nonce configuration in the Login Config section in a J2EE Version 1.3 extensions configuration for Web services security to a J2EE Version 1.4 application.

  **Important:** Nonce is not configured in a J2EE Version 1.4 extension file for Web services security. Rather, it is configured in the binding file for Web services security.

  To configure a nonce in the binding file, define the com.ibm.wsspi.wssecurity.token.username.addNonce property in the token generator of the username token.
- Configure the Add Timestamp section under the Request Generator Configuration in the assembly tool if the **Add Created Time Stamp** option is configured in the J2EE Version 1.3 extensions.

This set of steps describe the types of information that you need to migrate the client-side extensions configuration for Web services security for a J2EE Version 1.3 application to a J2EE Version 1.4 application.

Migrate the server-side bindings configuration for a J2EE Version 1.3 application to a J2EE Version 1.4 application. For more information, see "Migrating the server-side bindings file."

**Related tasks**

"Migrating the server-side extensions configuration" on page 545

"Migrating the server-side bindings file"

"Migrating the server-side bindings file"

***Migrating the server-side bindings file:***

This article provides general information about migrating the server-side bindings configuration for a Java 2 Platform, Enterprise Edition (J2EE) Version 1.3 application to a J2EE Version 1.4 application. The steps are based on typical scenarios, but the steps are not all-inclusive.

The following table lists the mappings of the top-level sections under the server-side **Binding Configurations** tab from a J2EE Version 1.3 application to a J2EE Version 1.4 application.

*Table 8. The mapping of the configuration sections*

| J2EE Version 1.3 Binding Configurations | J2EE Version 1.4 Binding Configurations |
| --- | --- |
| Request Receiver Binding Configuration Details | Request Consumer Service Binding Configuration Details |

*Table 8. The mapping of the configuration sections (continued)*

| J2EE Version 1.3 Binding Configurations | J2EE Version 1.4 Binding Configurations |
| --- | --- |
| Response Sender Binding Configuration Details | Response Generator Binding Configuration Details |

Consider the following steps to migrate the server-side bindings from J2EE Version 1.3 to J2EE Version 1.4. These steps are dependent upon your specific configuration.

- Migrate the configuration information under the Request Receiver Binding Configuration Details section of a J2EE Version 1.3 application.
  1. Migrate any trust anchor information that is specified in the J2EE Version 1.3 application to J2EE Version 1.4 using the Trust Anchor dialog.
  2. Migrate the information under the certificate store list that is specified in the J2EE Version 1.3 application to J2EE Version 1.4 by configuring the Certificate Store List section in the J2EE Version 1.4 application.
  3. Configure the key locator and token consumer information that is referenced from the Key Information dialog window. The configuration of the key locator and the token consumer depends upon the key information type. For example, if an X.509 certificate that is embedded in the `<wsse:Security>` security header is used for digital signature, complete the following steps:
     a. For configuring the key locator, specify the com.ibm.wsspi.wssecurity.keyinfo.X509TokenKeyLocator class as the key locator class and do not specify a key store.
     b. For configuring the token consumer, select the com.ibm.wsspi.wssecurity.token.509TokenConsumer class, specify `X509 certificate token` for the value type Uniform Resource Identifier (URI), and specify `system.wssecurity.X509BST` in the jaas.config.name field. Also, you must specify the certificate path settings (the trust anchor reference and the certificate store reference) as part of the token consumer configuration.
  4. Explicitly specify the key information type in the Key Information Dialog window. In a J2EE Version 1.3 application, the key information type, such as the security token reference and the key identifier, is not explicitly specified. The key information type is implied by the configuration. In a J2EE Version 1.4 application, you must specify the key information type explicitly using the Key Information Dialog when you have digital signature or encryption information in the binding file. Before you configure the key information, make sure that you have configured the key locator and token consumer information that is referenced from the Key Information dialog.

     When you configure the key information for either digital signature or encryption, you need to specify the correct key information type. The value of the key information type depends upon the type of mechanism that is used to reference the security token that is used for digitally signing or encrypting. The following information describes the Security token reference (or Direct reference) and the Key identifier, which are the most common, recommended key information types that are used for digitally signing and encrypting:

     **Security token reference (or Direct reference)**
     The security token is directly referenced using the Uniform Resource Identifiers (URIs). The following `<KeyInfo>` element is generated in the Simple Object Access Protocol (SOAP) message for this key information type:

     ```
     <ds:KeyInfo>
        <wsse:SecurityTokenReference>
           <wsse:Reference URI="#mytoken" />
        </wsse:SecurityTokenReference>
     </ds:KeyInfo>
     ```

     **Key identifier**
     The security token is referenced using an opaque value that uniquely identifies the token. The algorithm that is used for generating the `KeyIdentifier` value depends upon the token

type. For example, a hash of the important elements of the security token is used for generating the `KeyIdentifier` value. The following `<KeyInfo>` element is generated in the SOAP message for this key information type:

```
<ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
   <wsse:SecurityTokenReference>
      <wsse:KeyIdentifier ValueType="wsse:X509v3">/62wXO...</wsse:KeyIdentifier>
   </wsse:SecurityTokenReference>
</ds:KeyInfo>
```

In the Key Information Dialog window, specify the names of the key locator and the token consumer that you configured previously. The Key name field is optional for the consumer side.

5. Migrate the information in the Signing Information section by configuring the Signing Information, Part References, and Transforms sections.
   - Specify the Signature method and Canonicalization method algorithms in the Signing Information Dialog window.
   - Specify the Digest method algorithm in the Part Reference Dialog window.
6. Migrate the information under the Encryption Information section. In the Encryption Information Dialog window, select the name of the Key Information element that is configured for encryption, and specify the `RequiredConfidentiality` part. Verify that the value for the selected `RequiredConfidentiality` part is the same name as the Required Confidentiality part that is configured in the extension file.

   The Login Mapping section in the J2EE Version 1.3 application maps to the Token Consumer configuration for the type of token that is specified by the authentication method. For example, to migrate a Login Mappings configuration that uses the `BasicAuth` authentication method, configure a token consumer for the username token. To configure a token consumer for a username token, complete the following steps:
   a. Select the com.ibm.wsspi.wssecurity.UsernameTokenConsumer token consumer class.
   b. Specify the name of the Required Security Token configuration from the Extensions within in the Security Token field.
   c. Select **Username Token** for value type.
   d. Specify the system.wssecurity.UsernameToken value in the jaas.config.name field.

- Migrate the configuration information in the Response Sender Binding Configuration Details section of the J2EE Version 1.3 bindings file to the Response Generator Binding Configuration Details section of the J2EE Version 1.4 application. Configuring the Response Generator section is very similar to configuring the Request Consumer section.
  1. Migrate the information from the Key Locators section by using the Key Locator Dialog window in an assembly tool.
  2. Configure a token generator, which is referenced in the Key Information Dialog window. You must configure a token generator for every security token that is generated in the SOAP message. If the token generator is for an X.509 certificate that is used for digital signature or encryption, complete the following steps:
     a. For configuring the key locator, specify the com.ibm.wsspi.wssecurity.keyinfo.X509TokenKeyLocator class as the key locator class and do not specify a key store.
     b. For configuring the token generator, select the com.ibm.wsspi.wssecurity.X509TokenGenerator class and specify `X509 certificate token` for the value type Uniform Resource Identifier (URI). The key store information that is specified for the token generator is the same information that is used for configuring the key locator. Therefore, the keystore information from the Key Locators configuration in a J2EE Version 1.3 application is used to configure the key locator and the token generator in a J2EE Version 1.4 application.
     c. In the Token Generator Dialog window, specify the key store information that is required by the callback handler to obtain the key information that is required for generating the token.

d. For the callback handler, select the com.ibm.wsspi.wssecurity.auth.callback.X509CallbackHandler class.

3. Specify the names of the key locator and the token generator in the Key Information Dialog window that you configured previously. The Key name is required for the generator side. The key that is specified in the Key Information Dialog window must exist in the list of keys that is specified in the key locator configuration. Also, migrating the Signing Information and the Encryption Information configurations is similar to migrating the Signing Information and the Encryption Information configurations for the Request Receiver Binding Configuration section. Configuring the key information for the response generator section is similar to configuring the key information for the request consumer section.

This set of steps describse the types of information that you need to migrate the server-side bindings configuration for a J2EE Version 1.3 application to a J2EE Version 1.4 application.

Migrate the client-side binding configuration for a J2EE Version 1.3 application to a J2EE Version 1.4 application. For more information, see “Migrating the server-side bindings file” on page 548.

**Related tasks**

“Migrating the server-side extensions configuration” on page 545

“Migrating the client-side extensions configuration” on page 547

“Migrating the server-side bindings file” on page 548

***Migrating the client-side bindings file:***

This article provides general information about migrating the Web services security client-side binding configuration for a Java 2 Platform, Enterprise Edition Version 1.3 application to a J2EE Version 1.4 application. The steps are based on typical scenarios, but the steps are not all-inclusive.

The following table lists the mapping of the top-level sections under the client-side **Port Bindings** tab within a J2EE Version 1.3 application to a J2EE Version 1.4 application.

*Table 9. The mapping of the configuration sections*

| J2EE Version 1.3 binding configuration for Web services security | J2EE Version 1.4 binding configuration for Web services security |
|---|---|
| Security Request Sender Binding Configuration | Security Request Generator Binding Configuration |
| Security Response Receiver Binding Configuration | Security Response Consumer Binding Configuration |

Consider the following steps to migrate the client-side binding configuration from a J2EE Version 1.3 application to a J2EE Version 1.4 application. These steps are dependent upon your specific configuration.

- Migrate the information in the Security Request Sender Binding Configuration section in a J2EE Version 1.3 application to a J2EE Version 1.4 application. The migrations process for the Security Request Sender Binding Configuration section is similar to the process for the Response Sender Binding Configuration Details section in the server-side binding configuration. For more information, see “Migrating the server-side bindings file” on page 548.
- Migrate the information in the Key Locators, Signing Information, and the Encryption Information sections of the J2EE Version 1.3 application to a J2EE Version 1.4 application. The migration process for these elements on the client side is similar to migration process on the server side. For more information, see “Migrating the server-side bindings file” on page 548.
- Migrate the information in the Login Bindings section in a J2EE Version 1.3 application to a J2EE Version 1.4 application. The migration of the Login Bindings section depends upon the value of the authentication method. If the authentication method is `BasicAuth` or `IDAssertion`, configure a token generator for the username token. If the authentication method is LTPA, select the com.ibm.wsspi.wssecurity.token.LTPATokenGenerator class as the token generator class. If the client-side bindings for the Web service uses IDAssertion, complete the following steps:

1. Configure a token generator for the authentication token of the original client.
2. Define the com.ibm.wsspi.wssecurity.token.IDAssertion.isUsed property and set its value to `true` in the Token Generator Dialog window within an assembly tool. If the original client is using a username token for authentication and if the target Web service is using `BasicAuth` for authentication, configure the following token generators in the client-side binding file:
   - The username token of the original client. You must set the `com.ibm.wsspi.wssecurity.token.IDAssertion.isUsed` property in the token generator of the original client.
   - The username token of the intermediary Web service.

- Migrate the Security Response Receiver Binding Configuration section from a J2EE Version 1.3 application to a J2EE Version 1.4 application. Migrating the Security Response Receiver Binding Configuration section is similar to migrating the Request Receiver Binding Configuration Details section of the server-side bindings configuration. Migrate this information under the Security Response Consumer Binding Configuration section. For more information, see "Migrating the server-side bindings file" on page 548.

  To configure a nonce in the binding file, define the com.ibm.wsspi.wssecurity.token.username.addNonce property in the token generator of the username token.

This set of steps describe the types of information that you need to migrate the Web services security client-side bindings configuration for a J2EE Version 1.3 application to a J2EE Version 1.4 application.

Verify that you have migrated both the server-side and the client-side extension and binding configurations for a J2EE Version 1.3 application to a J2EE Version 1.3 application. For more information, see "Migrating Version 5.x applications with Web services security to Version 6 applications" on page 544.

**Related tasks**

"Migrating the server-side extensions configuration" on page 545
"Migrating the client-side extensions configuration" on page 547
"Migrating the server-side bindings file" on page 548

***View Web services client deployment descriptor:***

Use this page to view your client deployment descriptor.

Before you begin this task, the Web services application must be installed.

By completing this task, you can gather information that enables your to maintain or configure binding information. After the Web services application is installed, you can view the Web services deployment descriptors.

To view this administrative console page, complete the following steps:

1. Click **Applications > Enterprise applications >***application_name*.
2. Under Related Items, click **EJB modules** or **Web modules** > *URI_file_name*.
3. Under Additional properties, click **View Web services client deployment descriptor extension**.

Application-level and server-level bindings are the two levels of bindings that WebSphere Application Server offers. The information in the following implementation descriptions indicates how to configure your application-level bindings. If the Web server is acting as a client, the default bindings are used. To configure the server-level bindings, which are the defaults, complete the following steps:

1. Click **Servers > Application servers >** *server_name*.
2. Under Security, click **Web Services: Default bindings for Web services security**.

If you are using any of the following configurations, verify that the deployment descriptor is configured properly:

- Request signing
- Request encryption
- BasicAuth authentication
- Identity (ID) assertion authentication
- Identity (ID) assertion authentication with the signature TrustMode
- Response digital signature verification
- Response decryption

**Request signing**

If the integrity constraints (digital signature) are specified, verify that you configured the signing information in the binding files.

To configure the signing parameters, complete the following steps:

1. Click **Applications > Enterprise applications >***application_name*.
2. Under Related Items, click **Web modules** > *URI_file_name*
3. Under Additional properties, click **Web Services: Client security bindings**.
4. In the Response receiver binding column, click **Edit > Signing information > New**.

To configure the key locators, complete the following steps:

1. Click **Servers > Application servers >***server_name*.
2. Under Additional properties, click **Web Services: Default bindings for Web services security > Key locators**.

**Request encryption**

If the confidentiality constraints (encryption) are specified, verify that you configured the encryption information in the binding files.

To configure the encryption parameters, complete the following steps:

1. Click **Applications > Enterprise applications >***application_name*.
2. Under Related Items, click **EJB modules** or **Web modules** >*URI_file_name* **> Web services: Client security bindings** .
3. In the Response receiver binding column, click **Edit > Encryption Information > New**.

To configure the key locators, complete the following steps:

1. Click **Servers > Application servers >***server_name*.
2. Under Additional properties, click **Web Services: Default bindings for Web services security > Key locators**.

**BasicAuth authentication**

If BasicAuth authentication is configured as the required security token, specify the callback handler in the binding file to collect the basic authentication data. The following list contains the CallBack support implementations:

**com.ibm.wsspi.wssecurity.auth.callback.GuiPromptCallbackHandler**
: This implementation prompts for basic authentication information, the user name and password, in an interface.

**com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler**
: This implementation reads the basic authentication information from the binding file.

**com.ibm.wsspi.wssecurity.auth.callback.StdPromptCallbackHandler**
: This implementation prompts for a user name and password using the standard in (stdin) prompt.

To configure the login binding information, complete the following steps:

1. Click **Applications > Enterprise applications >***application_name*.
2. Under Related items, click **EJB modules** or **Web modules >***URI_file_name* **> Web services: Client security bindings**.
3. Under Request sender bindings, click **Edit > Login binding**.

**Identity (ID) Assertion authentication with BasicAuth TrustMode**

Configure a login binding in the bindings file with a com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler implementation. Specify a BasicAuth user name and password that a trusted ID evaluator on a downstream server trusts.

To configure the login binding information, complete the following steps:

1. Click **Applications > Enterprise applications >***application_name*.
2. Under Related items, click **EJB modules** or **Web modules >***URI_file_name* **> Web services: Client security bindings**.
3. Under Request sender bindings, click **Edit > Login binding**.

**Identity (ID) Assertion authentication with the Signature TrustMode**

Configure the signing information in the bindings file with a signing key pointing to a key locator. The key locator contains the X.509 certificate that is trusted by the downstream server.

To configure ID assertion, complete the following steps:

1. Click **Servers > Application servers >***server_name*.
2. Under Additional properties, click **Web services: Default bindings for Web services security > Login mappings > IDAssertion**.

To configure the login binding information, complete the following steps:

1. Click **Applications > Enterprise applications >***application_name*.
2. Under Related items, click **EJB modules** or **Web modules >***URI_file_name* > **Web services: Client security bindings**.
3. Under Request sender bindings, click **Edit > Login binding**.

**Response digital signature verification**

If the integrity constraints, which require a signature, are defined, verify that you configured the signing information in the binding files.

To configure the signing parameters, complete the following steps:

1. Click **Applications > Enterprise applications >***application_name*.
2. Under Related Items, click **EJB modules** or **Web modules >***URI_file_name* **> Web services: Client security bindings**.
3. In the Response receiver binding column, click **Edit > Signing information > New**.

To configure the trust anchors, complete the following steps:

1. Click **Servers > Application servers >***server_name*.
2. Under Additional properties, click **Web Services: Default bindings for Web services security > Trust anchors > New**.

To configure the collection certificate store, complete the following steps:

1. Click **Servers > Application servers >***server_name*.

2. Under Additional properties, click **Web Services: Default bindings for Web services security > Collection certificate store > New**.

**Response decryption**

If the confidentiality constraints (encryption) are specified, verify that you defined the encryption information.

To configure the encryption information, complete the following steps:

1. Click **Applications > Enterprise applications >***application_name*.
2. Under Related Items, click **EJB modules** or **Web modules >**<*URI_file_name* **> Web services: Client security bindings**.
3. In the Response receiver binding column, click **Edit > Encryption information > New**.

To configure the key locators, complete the following steps:

1. Click **Servers > Application servers >***server_name*.
2. Under Additional properties, click **Web Services: Default bindings for Web services security > Key locators**.

   **Related reference**

   "View Web services server deployment descriptor"
   Use this page to view your server deployment descriptor settings.

***View Web services server deployment descriptor:***

Use this page to view your server deployment descriptor settings.

Before you begin this task, the Web services application must be installed.

By completing this task, you can gather information that enables you to maintain or configure binding information. After the Web services application is installed, you can view the Web services deployment descriptors.

To view this administrative console page, complete the following steps:

1. Click **Applications > Enterprise applications >***application_name*.
2. Under Related items, click **EJB modules** or **Web modules** > *URI_file_name* > **View Web services server deployment descriptor**.

WebSphere Application Server has two levels of bindings: application-level and server-level. The information in the following implementation descriptions indicate how to configure your application-level bindings. To configure the server-level bindings, which are the defaults, complete the following steps:

1. Click **Servers > Application servers >** *server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.

- Request digital signature verification
- Request decryption
- Basic authentication
- Identity (ID) assertion authentication
- Identity (ID) assertion authentication with the signature TrustMode
- Response signing
- Response encryption

**Request digital signature verification**

If the integrity constraints, which require a signature, are defined, verify that you configured the signing information in the binding files.

To configure the signing parameters, complete the following steps:
1. Click **Applications > Enterprise applications >***application_name*.
2. Under Related items, click **EJB modules** or **Web modules >***URI_file_name* > **Web services: Server security bindings**.
3. Under Request consumer (receiver) binding, click **Edit custom > Signing information**.

To configure the trust anchor, complete the following steps:
1. Click **Servers > Application servers >***server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Trust anchors**.

To configure the collection certificate store, complete the following steps:
1. Click **Servers > Application servers >***server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Collection certificate store**.

To configure the key locators, complete the following steps:
1. Click **Servers > Application servers >***server_name*.
2. Under Security, click **Web Services: Default bindings for Web services security**.
3. Under Additional properties, click **Key locators**.

**Request decryption**

If the confidentiality constraints (encryption) are specified, verify that the encryption information is defined.

To configure the encryption information parameters, complete the following steps:
1. Click **Enterprise applications >** *application_name*.
2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.
3. Under Additional properties, click **Web services: Server security bindings**.
4. Under Request consumer (receiver) binding, click **Edit custom > Encryption information**.

To configure the key locators, complete the following steps:
1. Click **Servers > Application servers >***server_name*.
2. Under Security, click **Web Services: Default bindings for Web services security**.
3. Under Additional properties, click **Key locators**.

**Basic authentication**

If BasicAuth authentication is configured as the required security token, specify the callback handler in the binding file to collect the basic authentication data. The following list contains callback support implementations:

**com.ibm.wsspi.wssecurity.auth.callback.GuiPromptCallbackHandler**
: The implementation prompts for BasicAuth information (user name and password) in an interface panel.

**com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler**
: This implementation reads the BasicAuth information from the binding file.

**com.ibm.wsspi.wssecurity.auth.callback.StdPromptCallbackHandler**
This implementation prompts for a user name and password using the standard in (stdin) prompt.

To configure the login mapping information, complete the following steps:
1. Click **Server > Application Servers >***server_name*.
2. Under Security, click **Web Services: Default bindings for Web services security**.
3. Under Additional properties, click **Login mappings**.

**Identity (ID) assertion authentication with the BasicAuth TrustMode**

Configure a login binding in the bindings file with a com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler implementation. Specify a user name and password for basic authentication that a TrustedIDEvaluator on a downstream server trusts.

To configure the login mapping information, complete the following steps:
1. Click **Server > Application servers >***server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Login mappings**.

**Identity (ID) assertion authentication with the signature TrustMode**

Configure the signing information in the bindings file with a signing key that points to a key locator. The key locator contains the X.509 certificate that is trusted by the downstream server.

To configure the login mapping information, complete the following steps:
1. Click **Server > Application servers >***server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Login mappings**.

The Java Authentication and Authorization Service (JAAS) uses `WSLogin` as the name of the login configuration. To configure JAAS, complete the following steps:
1. Click **Security > Global security**.
2. Under Authentication, click **JAAS configuration >Application logins**.

The value of the <TrustedIDEvaluatorRef> tag in the binding must match the value of the <TrustedIDEvaluator> name.

To configure the trusted ID evaluators, complete the following steps:
1. Click **Servers > Application servers >***server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Trusted ID evaluators**.

**Response signing**

If the integrity constraints (digital signature) are defined, verify that you have the signing information configured in the binding files.

To specify the signing information, complete the following steps:
1. Click **Applications > Enterprise applications >***application_name*.
2. Under Related items, click **EJB modules** or **Web modules >***URI_file_name* **> Web services: Server security bindings**.
3. In the Request receiver binding column, click **Edit > Signing information**.

To configure the key locators, complete the following steps:

1. Click **Servers > Application servers >***server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Key locators**.

**Response encryption**

If the confidentiality constraints (encryption) are specified, verify that the encryption information is defined.

To specify the encryption information, complete the following steps:

1. Click **Enterprise applications >** *application_name*.
2. Under Related items, click **EJB modules** or **Web modules**.
3. Under Additional properties, click **Web services: Server security bindings**.
4. Under Request consumer (receiver) binding, click **Edit custom > Encryption information**.

To configure the key locators, complete the following steps:

1. Click **Servers > Application servers >***server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Key locators**.

**Related reference**

"View Web services client deployment descriptor" on page 552
Use this page to view your client deployment descriptor.

## Default implementations of the Web services security service provider programming interfaces

The following information describes the default implementations of the service provider interfaces (SPI) for Web services security within WebSphere Application Server Version 6. The default implementations of the service provider interfaces for WebSphere Application Server Version 5.x are not described in this document. Instead, see "Securing Web services for version 5.x applications based on WS-Security" on page 801 for the Version 5.x implementations that are deprecated in version 6.

**com.ibm.wsspi.wssecurity.token.X509TokenGenerator**
: This class implements the com.ibm.wsspi.wssecurity.token.TokenGeneratorComponent interface. It is responsible for creating the X.509 token object from the X.509 certificate, which is returned by the com.ibm.wsspi.wssecurity.auth.callback.{X509,PKCS7,PkiPath}CallbackHandler interface. Encode the token using the base 64 format and insert its XML representation into the Simple Object Access Protocol (SOAP) message, if necessary.

**com.ibm.wsspi.wssecurity.auth.callback.X509CallbackHandler**
: This class implements the javax.security.auth.callback.CallbackHandler interface and it retrieves the X.509 certificate from the keystore file.

**com.ibm.wsspi.wssecurity.token.UsernameTokenGenerator**
: This class implements the com.ibm.wsspi.wssecurity.token.TokenGeneratorComponent interface. It is responsible for creating the username token object from user name and password that is returned by a javax.security.auth.callback.CallbackHandler implementation such as com.ibm.wsspi.wssecurity.auth.callback.{GUIPrompt,NonPrompt,StdinPrompt}CallbackHandler. It also inserts the XML representation of the token into the SOAP message, if necessary.

**com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator**
: This class implements the com.ibm.wsspi.wssecurity.keyinfo.KeyLocator interface and it retrieves the keys from the keystore files for digital signature and encryption.

**com.ibm.wsspi.wssecurity.token.X509TokenConsumer**
: This class implements the com.ibm.wsspi.wssecurity.token.TokenConsumerComponent interface and processes the X.509 token from the binary security token. This class decodes the Base64

encryption within the X.509 token and then invokes the system.wssecurity.X509BST Java Authentication and Authorization Service (JAAS) Login Configuration with the com.ibm.wsspi.wssecurity.auth.module.X509LoginModule login module to validate the X.509 token. An object of the com.ibm.wsspi.wssecurity.auth.token.X509Token is created for the validated X.509 token and stored in JAAS Subject.

**com.ibm.wsspi.wssecurity.token.IDAssertionUsernameTokenConsumer**
This class implements com.ibm.wsspi.wssecurity.token.TokenConsumerComponent interface and processes the username token for identity assertion (IDAssertion), which does not have a password element. This interface invokes the system.wssecurity.IDAssertionUsernameToken JAAS login configuration with the com.ibm.wsspi.wssecurity.auth.module.IDAssertionUsernameLoginModule login module to validate the IDAssertion username token. An object of the com.ibm.wsspi.wssecurity.auth.token.UsernameToken class is created for the validated username token and stored in the JAAS Subject.

**com.ibm.wsspi.wssecurity.auth.module.IDAssertionUsernameLoginModule**
This class implements the javax.security.auth.spi.LoginModule interface and checks whether the username value is not empty. The login module assumes that the UsernameToken is valid if the username value is not empty.

**com.ibm.wsspi.wssecurity.token.LTPATokenGenerator**
This class implements the com.ibm.wsspi.wssecurity.token.TokenGeneratorComponent interface and is responsible for Base 64 encoding the LTPA token object obtained from the com.ibm.wsspi.wssecurity.auth.callback.LTPATokenCallbackHandler callback handler. The object is inserted into the Web services security header within the SOAP message, if necessary.

**com.ibm.wsspi.wssecurity.token.LTPATokenConsumer**
This class implements the com.ibm.wsspi.wssecurity.token.TokenConsumerComponent interface, processes the LTPA token from the binary security token, and decodes the Base64 encoding within the LTPA token. An object of the com.ibm.wsspi.wssecurity.auth.token.LTPAToken class is created for the validated LTPA token and stored in the JAAS Subject.

**com.ibm.wsspi.wssecurity.auth.module.X509LoginModule**
This class implements the javax.security.auth.spi.LoginModule interface and validates the X.509 Certificate based on the trust anchor and the collection certification store configuration.

**com.ibm.wsspi.wssecurity.token.UsernameTokenConsumer**
This class implements the com.ibm.wsspi.wssecurity.token.TokenConsumerComponent interface, processes the username token, extracts the user name and password, and then invokes the system.wssecurity.UsernameToken JAAS login configuration using the com.ibm.wsspi.wssecurity.auth.module.UsernameLoginModule login module to validate the user name and password. An object of the com.ibm.wsspi.wssecurity.auth.token.UsernameToken class is created for the validated username token and stored in the JAAS Subject.

**com.ibm.wsspi.wssecurity.keyinfo.X509TokenKeyLocator**
This class implements the com.ibm.wsspi.wssecurity.keyinfo.KeyLocator interface and it is used to retrieve a public key from a X.509 certificate. The X.509 certificate is stored in the X.509 token (com.ibm.wsspi.wssecurity.auth.token.X509Token) in the JAAS Subject. The X.509 token is created by the X.509 Token Consumer (com.ibm.wsspi.wssecurity.tokenX509TokenConsumer).

**com.ibm.wsspi.wssecurity.keyinfo.SignerCertKeyLocator**
This class implements the com.ibm.wsspi.wssecurity.keyinfo.KeyLocator interface, which is used to retrieve a public key from the X.509 certificate of the request signer and encrypt the response. You can use this key locator in the response generator binding configuration only.

**Important:** This implementation assumes that only one signer certificate is used in the request.

**com.ibm.wsspi.wssecurity.auth.token.UsernameToken**
This implementation extends the com.ibm.wsspi.wssecurity.auth.token.WSSToken abstract class to represent the username token.

**com.ibm.wsspi.wssecurity.auth.token.X509Token**
This implementation extends the com.ibm.wsspi.wssecurity.auth.token.WSSToken abstract class to represent the X.509 binary security token (X.509 certificate).

**com.ibm.wsspi.wssecurity.auth.token.LTPAToken**
This implementation extends the com.ibm.wsspi.wssecurity.auth.token.WSSToken abstract class as a wrapper to the LTPA token that is extracted from the binary security token.

**com.ibm.wsspi.wssecurity.auth.callback.PKCS7CallbackHandler**
This class implements the javax.security.auth.callback.CallbackHandler interface and is responsible for creating a certificate and binary data with or without a certificate revocation list (CRL) using the PKCS#7 encoding. The certificate and the binary data is passed back to the com.ibm.wsspi.wssecurity.token.X509TokenGenerator implementation through the com.ibm.wsspi.wssecurity.auth.callback.X509BSCallback callback handler.

**com.ibm.wsspi.wssecurity.auth.callback.PkiPathCallbackHandler**
This class implements the javax.security.auth.callback.CallbackHandler interface and it is responsible for creating a certificate and binary data without a CRL using the PkiPath encoding. The certificate and binary data is passed back to the com.ibm.wsspi.wssecurity.token.X509TokenGenerator implementation through the com.ibm.wsspi.wssecurity.auth.callback.X509BSCallback callback handler.

**com.ibm.wsspi.wssecurity.auth.callback.X509CallbackHandler**
This class implements the javax.security.auth.callback.CallbackHandler interface and it is responsible for creating a certificate from the key store file. The X.509 token certificate is passed back to the com.ibm.wsspi.wssecurity.token.X509TokenGenerator implementation through the com.ibm.wsspi.wssecurity.auth.callback.X509BSCallback callback handler.

**com.ibm.wsspi.wssecurity.auth.callback.LTPATokenCallbackHandler**
This implementation generates a Lightweight Third Party Authentication (LTPA) token in the Web services security header as a binary security token. If basic authentication data is defined in the application binding file, it is used to perform a login, to extract the LTPA token from the WebSphere Application Server credentials, and to insert the token in the Web services security header. Otherwise, it extracts the LTPA security token from the invocation credentials (run as identity) and inserts the token in the Web services security header.

**com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler**
This implementation reads the basic authentication data from the application binding file. You might use this implementation on the server side to generate a username token.

**com.ibm.wsspi.wssecurity.auth.callback.GUIPromptCallbackHandler**
This implementation presents you with a login prompt to gather the basic authentication data. Use this implementation on the client side only.

**com.ibm.wsspi.wssecurity.auth.callback.StdinPromptCallbackHandler**
This implementation collects the basic authentication data using a standard in (stdin) prompt. Use this implementation on the client side only.

**com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator**
This interface is used to evaluate the level of trust for identity assertion. The default implementation is com.ibm.wsspi.wssecurity.id.TrustedIDEvaluatorImpl, which enables you to define a list of trusted identities.

**com.ibm.wsspi.wssecurity.id.TrustedIDEvaluatorImpl**
This default implementation enables you to define a list of trusted identities for identity assertion.

**com.ibm.wsspi.wssecurity.id.TrustedIDEvaluatorException**

This exception class is used by an implementation of the com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator to communicate the exception and errors to the Web services security run time.

**Related tasks**

"Securing Web services for version 5.x applications based on WS-Security" on page 801

## Default configuration

WebSphere Application Server Version 6 provides a variety of sample configurations that you can configure through the administrative console. The configurations that you specify are reflected on the cell or server level. Do not use these configurations in a production environment as they are for sample and testing purposes only. To make modifications to these sample configurations, it is recommended that you use the administrative console provided by WebSphere Application Server.

For a Web services security-enabled application, you must correctly configure a deployment descriptor and a binding. In WebSphere Application Server Version 6, one set of default bindings is shared by the applications to make application deployment easier. The default binding information for server level can be overridden by the binding information on the application level. The Application Server searches for binding information for an application on the application level before searching the server level.

This article contains information on the sample default bindings, keystores, key locators, collection certificate store, trust anchors, and trusted ID evaluators.

### Default generator binding

WebSphere Application Server Version 6 provides a sample set of default generator binding. The default generator binding contain both signing information and encryption information.

The sample signing information configuration is called `gen_signinfo` and contains the following configurations:

- Uses the following algorithms for the `gen_signinfo` configuration:
  - Signature method: `http://www.w3.org/2000/09/xmldsig#rsa-sha1`
  - Canonicalization method: `http://www.w3.org/2001/10/xml-exc-c14n#`
- References the `gen_signkeyinfo` signing key information. The following information pertains to the `gen_signkeyinfo` configuration:
  - Contains a part reference configuration that is called `gen_signpart`. The part reference is not used in default binding. The signing information applies to all of the Integrity or Required Integrity elements within the deployment descriptors and the information is used for naming purposes only. The following information pertains to the `gen_signpart` configuration:
    - Uses the transform configuration called `transform1`. The following transforms are configured for the default signing information:
      - Uses the `http://www.w3.org/2001/10/xml-exc-c14n#` algorithm
      - Uses the `http://www.w3.org/2000/09/xmldsig#sha1` digest method
  - Uses the security token reference, which is the configured default key information.
  - Uses the `SampleGeneratorSignatureKeyStoreKeyLocator` key locator. For more information on this key locator, see "Sample key locators" on page 564.
  - Uses the gen_signtgen token generator, which contains the following configuration:
    - Contains the X.509 token generator, which generates the X.509 token of the signer.
    - Contains the `gen_signtgen_vtype` value type URI.
    - Contains the `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509` value type local name value.

- Uses X.509 Callback Handler. The callback handler calls the ${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-sender.ks keystore.
  - The keystore password is client.
  - The alias name of the trusted certificate is soapca.
  - The alias name of the personal certificate is soaprequester.
  - The key password client issued by intermediary certificate authority Int CA2, which is, in turn, issued by soapca.

The sample encryption information configuration is called gen_encinfo and contains the following configurations:

- Uses the following algorithms for the gen_encinfo configuration:
  - Data encryption method: http://www.w3.org/2001/04/xmlenc#tripledes-cbc
  - Key encryption method: http://www.w3.org/2001/04/xmlenc#rsa-1_5
- References the gen_enckeyinfo encryption key information. The following information pertains to the gen_enckeyinfo configuration:
  - Uses the key identifier as the default key information.
  - Contains a reference to the SampleGeneratorEncryptionKeyStoreKeyLocator key locator. For more information on this key locator, see "Sample key locators" on page 564.
  - Uses the gen_signtgen token generator, which has the following configuration:
    - Contains the X.509 token generator, which generates the X.509 token of the signer.
    - Contains the gen_enctgen_vtype value type URI.
    - Contains the http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509 value type local name value.
  - Uses X.509 Callback Handler. The callback handler calls the ${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-sender.jceks keystore.
    - The keystore password is storepass.
    - The secret key CN=Group1 has an alias name of Group1 and a key password of keypass.
    - The public key CN=Bob, O=IBM, C=US has an alias name of bob and a key password of keypass.
    - The private key CN=Alice, O=IBM, C=US has an alias name of alice and a key password of keypass.

**Default consumer binding**

WebSphere Application Server Version 6 provides a sample set of default consumer binding. The default consumer binding contain both signing information and encryption information.

The sample signing information configuration is called con_signinfo and contains the following configurations:

- Uses the following algorithms for the con_signinfo configuration:
  - Signature method: http://www.w3.org/2000/09/xmldsig#rsa-sha1
  - Canonicalization method: http://www.w3.org/2001/10/xml-exc-c14n#
- Uses the con_signkeyinfo signing key information reference. The following information pertains to the con_signkeyinfo configuration:
  - Contains a part reference configuration that is called con_signpart. The part reference is not used in default binding. The signing information applies to all of the Integrity or RequiredIntegrity elements within the deployment descriptors and the information is used for naming purposes only. The following information pertains to the con_signpart configuration:
    - Uses the transform configuration called reqint_body_transform1. The following transforms are configured for the default signing information:

- Uses the http://www.w3.org/2001/10/xml-exc-c14n# algorithm.
    - Uses the http://www.w3.org/2000/09/xmldsig#sha1 digest method.
  - Uses the security token reference, which is the configured default key information.
  - Uses the SampleX509TokenKeyLocator key locator. For more information on this key locator, see "Sample key locators" on page 564.
  - References the con_signtcon token consumer configuration. The following information pertains to the con_signtcon configuration:
    - Uses the X.509 Token Consumer, which is configured as the consumer for the default signing information.
    - Contains the signtconsumer_vtype value type URI.
    - Contains the http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509 value type local name value.
  - Contains a JAAS configuration called system.wssecurity.X509BST that references the following information:
    - Trust anchor: SampleClientTrustAnchor
    - Collection certificate store: SampleCollectionCertStore

The encryption information configuration is called con_encinfo and contains the following configurations:
- Uses the following algorithms for the con_encinfo configuration:
  - Data encryption method: http://www.w3.org/2001/04/xmlenc#tripledes-cbc
  - Key encryption method: http://www.w3.org/2001/04/xmlenc#rsa-1_5
- References the con_enckeyinfo encryption key information. This key actually decrypts the message. The following information pertains to the con_enckeyinfo configuration:
  - Uses the key identifier, which is configured as the key information for the default encryption information.
  - Contains a reference to the SampleConsumerEncryptionKeyStoreKeyLocator key locator. For more information on this key locator, see "Sample key locators" on page 564.
  - References the con_enctcon token consumer configuration. The following information pertains to the con_enctcon configuration:
    - Uses the X.509 token consumer, which is configured for the default encryption information.
    - Contains the enctconsumer_vtype value type URI.
    - Contains the http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509 value type local name value.
  - Contains a JAAS configuration called system.wssecurity.X509BST.

### Sample keystore configurations

WebSphere Application Server provides the following keystores. You can work with these keystores outside of the Application Server by using the iKeyman utility or the key tool.

The iKeyman utility is located in the following directories:
- Windows *install_dir*/bin/ikeyman
- UNIX *install_dir*\bin\ikeyman.sh

The key tool is located in the following directories:
- Windows *install_dir*/java/jre/bin/keytool
- UNIX *install_dir*\java\jre\bin\keytool.sh

The following sample keystores are for testing purposes only; do not use these keystores in a production environment:

- ${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-sender.ks
  - The keystore format is JKS.
  - The keystore password is client.
  - The trusted certificate has a soapca alias name.
  - The personal certificate has a soaprequester alias name and a client key password that is issued by the Int CA2 intermediary certificate authority, which is, in turn, issued by soapca.
- ${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-receiver.ks
  - The keystore format is JKS.
  - The keystore password is server.
  - The trusted certificate has a soapca alias name.
  - The personal certificate has a soapprovider alias name and a server key password that is issued by the Int CA2 intermediary certificate authority, which is, in turn, issued by soapca.
- ${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-sender.jceks
  - The keystore format is JCEKS.
  - The keystore password is storepass.
  - The CN=Group1 DES secret key has a Group1 alias name and a keypass key password.
  - The CN=Bob, O=IBM, C=US public key has a bob alias name and a keypass key password.
  - The CN=Alice, O=IBM, C=US private key has a alice alias name and a keypass key password.
- ${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-receiver.jceks
  - The keystore format is JCEKS.
  - The keystore password is storepass.
  - The CN=Group1 DES secret key has a Group1 alias name and a keypass key password.
  - The CN=Bob, O=IBM, C=US private key has a bob alias name and a keypass key password.
  - The CN=Alice, O=IBM, C=US public key has a alice alias name and a keypass key password.
- ${USER_INSTALL_ROOT}/etc/ws-security/samples/intca2.cer
  - The intermediary certificate is signed by soapca and it signs both the soaprequester and the soapprovider.

### Sample key locators

### Key locators

Key locators are used to locate the key for digital signature, encryption, and decryption. For information on how to modify these sample key locator configurations, see the following articles:


- "Configuring the key locator for the generator binding on the application level" on page 692
- "Configuring the key locator for the consumer binding on the application level" on page 749
- "Configuring the key locator on the server or cell level" on page 777


**SampleClientSignerKey**
: This key locator is used by the request sender for a Version 5.x application to sign the Simple Object Access Protocol (SOAP) message. The signing key name is clientsignerkey, which is referenced in the signing information as the signing key name.

**SampleServerSignerKey**
: This key locator is used by the response sender for a Version 5.x application to sign the SOAP message. The signing key name is serversignerkey, which can be referenced in the signing information as the signing key name.

**SampleSenderEncryptionKeyLocator**
: This key locator is used by the sender for a Version 5.x application to encrypt the SOAP message. It is configured to use the ${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-sender.jceks

keystore and the com.ibm.wsspi.wssecurity.config.KeyStoreKeyLocator keystore key locator. The implementation is configured for the DES secret key. To use asymmetric encryption (RSA), you must add the appropriate RSA keys.

**SampleReceiverEncryptionKeyLocator**
: This key locator is used by the receiver for a Version 5.x application to decrypt the encrypted SOAP message. The implementation is configured to use the `${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-receiver.jceks` keystore and the com.ibm.wsspi.wssecurity.config.KeyStoreKeyLocator keystore key locator. The implementation is configured for symmetric encryption (DES or TRIPLEDES). To use RSA, you must add the private key `CN=Bob`, `O=IBM`, `C=US`, alias name `bob`, and key password `keypass`.

**SampleResponseSenderEncryptionKeyLocator**
: This key locator is used by the response sender for a Version 5.x application to encrypt the SOAP response message. It is configured to use the `${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-receiver.jceks` keystore and the com.ibm.wsspi.wssecurity.config.WSIdKeyStoreMapKeyLocator keystore key locator. This key locator maps an authenticated identity (of the current thread) to a public key for encryption. By default, WebSphere Application Server is configured to map to public key `alice`, and you must change WebSphere Application Server to the appropriate user. The SampleResponseSenderEncryptionKeyLocator key locator also can set a default key for encryption. By default, this key locator is configured to use public key `alice`.

**SampleGeneratorSignatureKeyStoreKeyLocator**
: This key locator is used by generator to sign the SOAP message. The signing key name is `SOAPRequester`, which is referenced in the signing information as the signing key name. It is configured to use the `${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-sender.ks` keystore and the com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator keystore key locator.

**SampleConsumerSignatureKeyStoreKeyLocator**
: This key locator is used by the consumer to verify the digital signature in the SOAP message. The signing key is `SOAPProvider`, which is referenced in the signing information as the signing key name. It is configured to use the `${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-receiver.ks` keystore and the com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator keystore key locator.

**SampleGeneratorEncryptionKeyStoreKeyLocator**
: This key locator is used by the generator to encrypt the SOAP message. It is configured to use the `${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-sender.jceks` keystore and the com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator keystore key locator.

**SampleConsumerEncryptionKeyStoreKeyLocator**
: This key locator is used by the consumer to decrypt an encrypted SOAP message. It is configured to use the `${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-receiver.jceks` keystore and the com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator keystore key locator.

**SampleX509TokenKeyLocator**
: This key locator is used by the consumer to verify a digital certificate in an X.509 certificate. It is configured to use the `${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-receiver.jceks` keystore and the com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator keystore key locator.

### Sample collection certificate store

Collection certificate stores are used to validate the certificate path. For information on how to modify this sample collection certificate store, see the following articles:

- “Configuring the collection certificate store for the generator binding on the application level” on page 663
- “Configuring the collection certificate store for the consumer binding on the application level” on page 738

- “Configuring the collection certificate store on the server or cell-level bindings” on page 762

**SampleCollectionCertStore**
: This collection certificate store is used by the response consumer and the request generator to validate the signer certificate path.

### Sample trust anchors

Trust anchors are used to validate the trust of the signer certificate. For information on how to modify the sample trust anchor configurations, see the following articles:

- “Configuring trust anchors for the generator binding on the application level” on page 659
- “Configuring trust anchors for the consumer binding on the application level” on page 736
- “Configuring trust anchors on the server or cell level” on page 761

**SampleClientTrustAnchor**
: This trust anchor is used by the response consumer to validate the signer certificate. This trust anchor is configure to access the `{USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-sender.ks` keystore.

**SampleServerTrustAnchor**
: This trust anchor is used by the request consumer to validate the signer certificate. This trust anchor is configure to access the `{USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-sender.ks` keystore.

### Sample trusted ID evaluators

Trusted ID evaluators are used to establish trust before asserting the identity in identity assertion. For information on how to modify the sample trusted ID evaluator configuration, see “Configuring trusted ID evaluators on the server or cell level” on page 784.

**SampleTrustedIDEvaluator**
: This trusted ID evaluator uses the com.ibm.wsspi.wssecurity.id.TrustedIDEvaluatorImpl implementation. The default implementation of com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator contains a list of trusted identities. This list is defined as properties with trustedId_* as the key and the value as the trusted identity.

  Complete the following steps to define this information for the server level in the administrative console:

  1. Click **Servers > Application servers >***server_name*.
  2. Under Security, click **Web services: Default bindings for Web services security**.
  3. Under Additional properties, click **Trusted ID evaluators** > `SampleTrustedIDEvaluator`.

**Related tasks**

“Configuring the key locator for the generator binding on the application level” on page 692

“Configuring the key locator for the consumer binding on the application level” on page 749

“Configuring the key locator on the server or cell level” on page 777

“Configuring the collection certificate store for the generator binding on the application level” on page 663

“Configuring the collection certificate store for the consumer binding on the application level” on page 738

“Configuring the collection certificate store on the server or cell-level bindings” on page 762

“Configuring trust anchors for the generator binding on the application level” on page 659

“Configuring trust anchors for the consumer binding on the application level” on page 736

“Configuring trust anchors on the server or cell level” on page 761

“Configuring trusted ID evaluators on the server or cell level” on page 784

## Nonce, a randomly generated token

*Nonce* is a randomly generated, cryptographic token used to prevent replay attacks. Although Nonce can be inserted anywhere in the SOAP message, it is typically inserted in the <UsernameToken> element.

Without nonce, when a user name token is passed from one machine to another machine using a non-secure transport, such as HTTP, the token might be intercepted and used in a replay attack. The same password might be reused when the username token is transmitted between the client and the server, which leaves it vulnerable to attack. The user name token can be stolen even if you use XML digital signature and XML encryption.

To help eliminate these replay attacks, the `<wsse:Nonce>` and `<wsu:Created>` elements are generated within the `<wsse:UsernameToken>` element and used to validate the message. The server checks the freshness of the message by verifying that the difference between the nonce creation time, which is specified by the <wsu:Created> element, and the current time falls within a specified time period. Also, the server checks a cache of used nonces to verify that the username token in the received Simple Object Access Protocol (SOAP) message has not been processed within the specified time period. These two features are used to lessen the chance that a user name token is used for a replay attack.

To add nonce for the username token, you can specify it in the token generator for the username token. When the token generator for the username token is specified, you can select the **Add nonce** option if you want to include nonce in the username token.

**Related concepts**

"Distributed nonce caching" on page 763
The *distributed nonce caching* feature enables you to distribute the cache for a nonce to different servers in a cluster.

## Configuring an application for Web services security with an assembly tool

Prior to completing this task, you must import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF

There are eight parts of Web services security that you must configure to secure your Simple Object Access Protocol (SOAP) messages using either digital signature or encryption. Four of these parts involve the deployment descriptor extensions and four parts involve the bindings that correspond to the deployment descriptors. The following table illustrates these eight parts that involve both the client and the server or a server acting as a client. It is recommended that you configure each of these parts in order from left to right in the table. For example, configure the request generator extensions and then the request consumer extensions because the configurations must match. After you configure the request generator and request consumer extensions, configure the request generator and the request consumer bindings, and so on.

*Table 10. Client and server extensions and bindings relationship*

| Client | Server |
|---|---|
| 1. Request generator extensions | 2. Request consumer extensions |
| 3. Request generator bindings | 4. Request consumer bindings |
| 5. Response consumer extensions | 6. Request generator extensions |
| 7. Response consumer bindings | 8. Response generator bindings |

In Web services security for WebSphere Application Server Version 6, integrity refers to digital signature and confidentiality refers to encryption. *Integrity* decreases the risk of data modification when data is transmitted across a network. *Confidentiality* reduces the risk of someone intercepting the message as it

moves across a network. With confidentiality, however, the message is encrypted before it is sent and decrypted when it is received by its target server. The article provides the steps needed to secure your Web services using either integrity or confidentiality.

In the generator bindings, you can specify which message parts to sign (integrity) or encrypt (confidentiality) and what method is used. In the consumer bindings, you specify when the message parts are signed or encrypted. After you verify the digital signature or encryption in the consumer, the consumer verifies that the specified message parts are actually signed or encrypted. If the digital signature or encryption is required and the message is not signed or encrypted, the message is rejected by the consumer.

There are two different methods to specify what needs to be signed (integrity) or encrypted (confidentiality). You can use either keywords or an XPath expression to configure message parts, a nonce, or a time stamp. When you use keywords, you can specify only certain elements within a message. With an XPath expression, you can specify any part of the message.

In addition to securing Web services for integrity and confidentiality, the assembly tools enable you to complete the following tasks:

- Configure a stand-alone time stamp for the generator and the consumer extensions. For more information, see “Adding a stand-alone time stamp to generator security constraints” on page 650 and “Adding a stand-alone time stamp in consumer security constraints” on page 651.
- Configure the security token in the generator and consumer constraints. For more information, see “Configuring the security token in generator security constraints” on page 653 and “Configuring the security token requirement in consumer security constraints” on page 653.
- Configure a caller part for the consumer security constraints. For more information, see “Configuring the caller in consumer security constraints” on page 654.
- Configure identity assertion. For more information, see “Configuring identity assertion” on page 656.
- Configure the client and the server for integrity. To properly configure Web services security for integrity, complete the following steps for the request generator and the request consumer and then repeat the steps for the response generator and the response consumer.
  1. Specify which message elements to sign in the generator security constraints using either keywords on an XPath expression. For more information, see either “Signing message elements in generator security constraints with keywords” on page 571 or “Signing message elements in generator security constraints with an XPath expression” on page 579. When you sign the message elements, you can also add a nonce or a time stamp configuration. For more information on these configurations, see:
     - “Adding time stamps for integrity to generator security constraints with keywords” on page 574
     - “Adding time stamps for integrity to generator security constraints with an XPath expression” on page 581
     - “Adding a nonce for integrity in generator security constraints with keywords” on page 577
     - “Adding a nonce for integrity to generator security constraints with an XPath expression” on page 583
  2. Configure a collection certificate store for the generator security constraints. For more information, see “Configuring the collection certificate store for the generator binding with an assembly tool” on page 586.
  3. Configure the token generator. For more information, see “Configuring token generators with an assembly tool” on page 587.
  4. Configure the key locators in the generator binding. For more information, see “Configuring key locators for the generator binding with an assembly tool” on page 594.
  5. Configure the key information in the generator binding. For more information, see “Configuring key information for the generator binding with an assembly tool” on page 596.

6. Configure the signing information in the generator binding. For more information, see “Configuring signing information for the generator binding with an assembly tool” on page 598.
7. Specify which message elements to sign in the consumer security constraints using either keywords on an XPath expression. For more information, see either “Signing message elements in consumer security constraints with keywords” on page 600 or “Signing message elements in consumer security constraints with an XPath expression” on page 606. When you sign the message elements, you can also add a nonce or a time stamp configuration. For more information on these configurations, see:
   - “Adding time stamps for integrity in consumer security constraints with keywords” on page 602
   - “Adding a nonce for integrity in consumer security constraints with keywords” on page 604
   - “Adding time stamps for integrity in consumer security constraints with an XPath expression” on page 608
   - “Adding a nonce for integrity in consumer security constraints with an XPath expression” on page 610
8. Configure a collection certificate store for the consumer security constraints. For more information, see “Configuring the collection certificate store for the consumer binding with an assembly tool” on page 611.
9. Configure a token consumer. For more information, see “Configuring token consumers with an assembly tool” on page 613.
10. Configure the key locators in the consumer binding. For more information, see “Configuring the key locator for the consumer binding with an assembly tool” on page 616.
11. Configure the key information in the consumer bindings. For more information, see “Configuring key information for the consumer binding with an assembly tool” on page 617.
12. Configure the signing information in the consumer binding. For more information, see “Configuring signing information for the consumer binding with an assembly tool” on page 620.

- Configure the client and the server for confidentiality. To properly configure Web services security for confidentiality, complete the following steps for the request generator and the request consumer, and then repeat the steps for the response generator and the response consumer.
  1. Specify which message elements to encrypt in the generator security constraints using either keywords on an XPath expression. For more information, see either “Encrypting the message elements in generator security constraints with keywords” on page 623 or “Encrypting the message elements in generator security constraints with an XPath expression” on page 629. When you encrypt the message elements, you can also add a nonce or a time stamp configuration. For more information on these configurations, see:
     - “Adding time stamps for confidentiality to generator security constraints with keywords” on page 624
     - “Adding the nonce for confidentiality to generator security constraints with keywords” on page 627
     - “Adding time stamps for confidentiality to generator security constraints with an XPath expression” on page 630
     - “Adding the nonce for confidentiality to generator security constraints with an XPath expression” on page 632
  2. Configure the token generator. For more information, see “Configuring token generators with an assembly tool” on page 587.
  3. Configure the key locators in the generator binding. For more information, see “Configuring key locators for the generator binding with an assembly tool” on page 594.
  4. Configure the key information in the generator binding. For more information, see “Configuring key information for the generator binding with an assembly tool” on page 596.
  5. Configure the encryption information in the generator binding. For more information, see “Configuring encryption information for the consumer binding with an assembly tool” on page 637.

6. Specify which message elements to encrypt in the consumer security constraints using either keywords on an XPath expression. For more information, see either “Encrypting message elements in consumer security constraints with keywords” on page 639 or “Encrypting message elements in consumer security constraints with an XPath expression” on page 643. When you encrypt the message elements, you can also add a nonce or a time stamp configuration. For more information on these configurations, see:
   - “Adding time stamps for confidentiality in consumer security constraints with keywords” on page 640
   - “Adding a nonce for confidentiality in consumer security constraints with keywords” on page 642
   - “Adding time stamps for confidentiality in consumer security constraints with an XPath expression” on page 645
   - “Adding the nonce for confidentiality in consumer security constraints with an XPath expression” on page 646
7. Configure a token consumer. For more information, see “Configuring token consumers with an assembly tool” on page 613.

   Also, the token consumer article provides the steps that are needed to optionally configure a trust anchor.
8. Configure the key locators in the consumer binding. For more information, see “Configuring the key locator for the consumer binding with an assembly tool” on page 616.
9. Configure the key information in the consumer bindings. For more information, see “Configuring key information for the consumer binding with an assembly tool” on page 617.
10. Configure the encryption information in the consumer binding. For more information, see “Configuring encryption information for the generator binding with an assembly tool” on page 648.

By completing the previous steps, you have configured your application for either digital signature (integrity) or encryption (confidentiality).

**Related concepts**

“Nonce, a randomly generated token” on page 567
*Nonce* is a randomly generated, cryptographic token used to prevent replay attacks. Although Nonce can be inserted anywhere in the SOAP message, it is typically inserted in the <UsernameToken> element.

***XML digital signature:***

XML-Signature Syntax and Processing (XML digital signature) is a specification that defines XML syntax and processing rules to sign and verify digital signatures for digital content. The specification was developed jointly by the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF).

XML digital signature does not introduce new cryptographic algorithms. WebSphere Application Server uses XML digital signature with existing algorithms such as RSA, HMAC, and SHA1. XML signature defines many methods for describing key information and enables the definition of a new method.

XML canonicalization (c14n) is often needed when you use XML signature. Information can be represented in various ways within serialized XML documents. For example, although their octet representations are different, the following examples are identical:

- `<person first="John" last="Smith"/>`
- `<person last="Smith" first="John"></person>`

C14n is a process that is used to canonicalize XML information. Select an appropriate c14n algorithm because the information that is canonicalized is dependent upon this algorithm. One of the major c14n

algorithms, Exclusive XML Canonicalization, canonicalizes the character encoding scheme, attribute order, namespace declarations, and so on. The algorithm does not canonicalize white space outside tags, namespace prefixes, or data type representation.

**XML signature in the Web Services Security-Core specification**

The Web Services Security-Core (WSS-Core) specification defines a standard way for Simple Object Access Protocol (SOAP) messages to incorporate an XML signature. You can use almost all of the XML signature features in WSS-Core except enveloped signature and enveloping signature. However, WSS-Core has some recommendations such as exclusive canonicalization for the c14n algorithm and some additional features such as SecurityTokenReference and KeyIdentifier. The KeyIdentifier is the value of the SubjectKeyIdentifier field within the X.509 certificate. For more information on the KeyIdentifier, see ″Reference to a Subject Key Identifier″ within the OASIS Web Services Security X.509 Certificate Token Profile documentation.

By including XML signature in SOAP messages, the following issues are realized:

**Message integrity**
: A message receiver can confirm that attackers or accidents have not altered parts of the message after these parts are signed by a key.

**Authentication**
: You can assume that a valid signature is *proof of possession*. A message with a digital certificate that is issued by a certificate authority and a signature in the message that is validated successfully by a public key in the certificate, is proof that the signer has the corresponding private key. The receiver can authenticate the signer by checking the trustworthiness of the certificate.

**Related information**

Exclusive XML Canonicalization

***Signing message elements in generator security constraints with keywords:***

Prior to completing this task, you must import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF.

Complete the following steps to specify which message parts to digitally sign when you configure the generator security constraints for either the request generator or the response generator. The request generator is configured for the client and the response generator is configured for the server. In the following steps, you must configure either the client-side extensions in step 2 or the server-side extensions in step 3.

1. Start the assembly tool and click **Window > Open Perspective > J2EE**.
2. **Optional:** Locate the client-side extensions using the Project Explorer window. The Client Deployment Descriptor window is displayed. This Web service contains the extensions that you need to configure.
   a. Expand the Web Services > Client section and double-click the name of the Web service.
   b. Click the **WS Extension** tab and expand the Request Generator Configuration section.
3. **Optional:** Locate the server-side extensions using the Project Explorer window. The Web Services Editor window is displayed. This Web service contains the extensions that you need to configure.
   a. Expand the Web Services > Services section and double-click the name of the Web service.
   b. Click the **Extensions** tab and expand the Response Generator Service Configuration Details section.
4. Expand the Integrity section. Integrity refers to digital signature while confidentiality refers to encryption. Integrity decreases the risk of data modification when you transmit data across a network. For more information on digitally signing Simple Object Access Protocol (SOAP) messages, see “XML digital signature” on page 570.

5. Click **Add** to indicate which parts of the message to sign. The Integrity Dialog window is displayed.
   a. Specify a name for the integrity element in the Integrity Name field. For example, you might specify `int_webskey`.
   b. Specify an order in the Order field. The value, which must be a positive integer value, specifies the order in which the digital signature is processed. An order value of 1 specifies that the signing is done first.
6. Click **Add** under the Message Parts section and select the Message parts dialect. The http://www.ibm.com/websphere/webservices/wssecurity/dialect-was dialect specifies the message part that is signed using keywords. If you select this dialect, you can select one of the following keywords under the Message parts keyword heading:

   **body** Specifies the user data portion of the message. If you select this keyword, the body is signed.

   **timestamp**
   Specifies that the stand-alone timestamp element within the message is signed. The timestamp element determines whether the message is valid based upon the time that the message is sent and then received. If the timestamp option is selected, make sure that there is a stand-alone timestamp element in the message. If the element does not exist, see "Adding a stand-alone time stamp to generator security constraints" on page 650.

   **securitytoken**
   Specifies that the UsernameToken in the SOAP message is signed.

   **dsigkey**
   Specifies that the key information element, which is used for digital signature, is signed.

   **enckey**
   Specifies that the key information element, which is used for encryption, is signed.

   **messageid**
   Specifies that the `<wsa:MessageID>` element within the message is signed.

   **to** Specifies that the `<wsa:To>` element within the message is signed.

   **action** Specifies that the `<wsa:Action>` element is signed.

   **relatesto**
   Specifies that the `<wsa:RelatesTo>` element within the message is signed.
7. Click **OK** to save the configuration changes.

   **Note:** These configurations for the generator and the consumer must match.

   In addition to the message parts, you also can specify that WebSphere Application Server sign the nonce and timestamp elements. For more information, see the following articles:
   
   - "Adding time stamps for integrity to generator security constraints with keywords" on page 574
   - "Adding time stamps for integrity to generator security constraints with an XPath expression" on page 581
   - "Adding a nonce for integrity in generator security constraints with keywords" on page 577
   - "Adding a nonce for integrity to generator security constraints with an XPath expression" on page 583
   

The following example is a SOAP message whose body is signed using the body keyword and the http://www.ibm.com/websphere/webservices/wssecurity/dialect-was dialect:

```
<soapenv:Envelope xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Header soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <wsse:Security soapenv:mustUnderstand="1" xmlns:wsse="http://docs.oasis-open.org/wss/
       2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
```

```
        <wsse:BinarySecurityToken EncodingType="http://docs.oasis-open.org/wss/2004/01/
         oasis-200401-wss-soap-message-security-1.0#Base64Binary" ValueType=
         "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3"
         wsu:Id="x509bst_956396521418196" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
         oasis-200401-wss-wssecurity-utility-1.0.xsd"> MA0GCSqGSIb3DQEBBQUAA4GBAHkthdGDgCvdIL9/
         vXUo74xpfOQd/rr1owBmMdb1TWdOyzwbOHC7lkUlnKrkI7SofwSLSDUP571iiMXUx3tRdmAVCoDMMFuDXh9V7212lu
         Xccx0s1S5KN0D3xW97LLNegQC0/b+aFD8XKw2U5ZtwbnFTRgs097dmz09RosDKkLlM
        </wsse:BinarySecurityToken>
        <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
           <ds:SignedInfo>
              <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
                 <ec:InclusiveNamespaces PrefixList="wsse ds xsi soapenc xsd soapenv "
                  xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#"/>
              </ds:CanonicalizationMethod>
              <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
              <ds:Reference URI="#wssecurity_signature_id_5945817517184298591">
                 <ds:Transforms>
                    <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
                       <ec:InclusiveNamespaces PrefixList="p896 xsi soapenc xsd wsu soapenv "
                        xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                    </ds:Transform>
                 </ds:Transforms>
                 <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                 <ds:DigestValue>vyu0JwXXSAvRCUCi6TPkeH8yUTU=</ds:DigestValue>
              </ds:Reference>
           </ds:SignedInfo>
           <ds:SignatureValue>dtbYA609wwAUAA8BmDmJlVHrWShy6OLJB3n4A6ToU0lI9tJrNhBksGqks17cykf+uHTJ
            cgOY18XrRDN4wHTW4zm/tmD5WqQd8K1WpYaGpbwlFoiwKVFNyfQn2K/WbZ2JccmZvJGFaOtqStg6TqSUGLQSA5
            MCSpZUhck545IY2F4=
           </ds:SignatureValue>
           <ds:KeyInfo>
              <wsse:SecurityTokenReference>
                 <wsse:Reference URI="#x509bst_956396521418196" ValueType="http://docs.oasis-open.org/
                  wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3"/>
              </wsse:SecurityTokenReference>
           </ds:KeyInfo>
        </ds:Signature>
     </wsse:Security>
  </soapenv:Header>
  <soapenv:Body soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
   wsu:Id="wssecurity_signature_id_5945817517184298591" xmlns:wsu="http://docs.oasis-open.org/
   wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
     <p896:getVersion xmlns:p896="http://msgsec.wssecfvt.ws.ibm.com"/>
  </soapenv:Body>
</soapenv:Envelope>
```

After you specify which message parts to digitally sign, you must specify which method is used to digitally sign the message. For more information, see “Configuring signing information for the generator binding with an assembly tool” on page 598.

**Related concepts**

“XML digital signature” on page 570
XML-Signature Syntax and Processing (XML digital signature) is a specification that defines XML syntax and processing rules to sign and verify digital signatures for digital content. The specification was developed jointly by the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF).

**Related tasks**

“Adding a stand-alone time stamp to generator security constraints” on page 650

“Adding time stamps for integrity to generator security constraints with an XPath expression” on page 581

“Adding a nonce for integrity to generator security constraints with an XPath expression” on page 583

“Configuring signing information for the generator binding with an assembly tool” on page 598

*Adding time stamps for integrity to generator security constraints with keywords:*

Prior to completing this task, you must import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF.

This task is used to specify that a time stamp is embedded in a particular element and that the element is signed. Complete the following steps to specify the time stamp for integrity using keywords when you configure the generator security constraints for either the request generator or the response generator. The request generator is configured for the client and the response generator is configured for the server. In the following steps, you must configure either the client-side extensions in step 2 or the server-side extensions in step 3.

1. Start the assembly tool and click **Window > Open Perspective > J2EE**.
2. **Optional:** Locate the client-side extensions using the Project Explorer window. The Client Deployment Descriptor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the client-side extensions:
   a. Expand the Web Services > Client section and double-click the name of the Web service.
   b. Click the **WS Extension** tab and expand the Request Generator Configuration section.
3. **Optional:** Locate the server-side extensions using the Project Explorer window. The Web Services Editor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the server-side extensions:
   a. Expand the Web Services > Services section and double-click the name of the Web service.
   b. Click the **Extensions** tab and expand the Response Generator Service Configuration Details section.
4. Expand the Integrity section. Integrity refers to digital signature while confidentiality refers to encryption. Integrity decreases the risk of data modification when you transmit data across a network. For more information on digitally signing Simple Object Access Protocol (SOAP) messages, see “XML digital signature” on page 570.
5. Click **Add** to specify a time stamp for integrity. The Integrity Dialog window is displayed. Complete the following steps to specify a configuration:
   a. Specify a name for the integrity element in the Integrity Name field.
   b. Specify an order in the Order field. The value, which must be a positive integer value, specifies the order in which the digital signature is processed. An order value of 1 specifies that the signing is done first.
6. In the Timestamp section, click **Add** and select the Timestamp dialect. The http://www.ibm.com/websphere/webservices/wssecurity/dialect-was dialect specifies the message element to which the time stamp is added prior to signing the element using the keywords. If you select this dialect, you can select one of the following keywords under the Timestamp keyword heading:

   **body** Specifies the user data portion of the message. If you select the body option, a time stamp is embedded in Simple Object Access Protocol (SOAP) body and the body is signed.

   **timestamp**
   Specifies that the time stamp is embedded in the stand-alone timestamp element within the message and that it is signed. If you select the timestamp option, make sure that there is a stand-alone timestamp element in the message. If the element does not exist, see “Adding a stand-alone time stamp to generator security constraints” on page 650.

   **securitytoken**
   Specifies that the security token authenticates the client. If you select this option, the timestamp element is embedded in the securitytoken element and the security token is signed.

**dsigkey**
: Specifies that the time stamp is inserted into the key information element, which is used for digital signature, and the key information element is signed.

**enckey**
: Specifies that the time stamp is inserted into the key information element, which is used for encryption, and the key information element is signed.

**messageid**
: Specifies that the time stamp is inserted into the `<wsa:MessageID>` element and the `<wsa:MessageID>` element is signed.

**to**
: Specifies that the time stamp is inserted into the `<wsa:To>` element within the message and that the `<wsa:To>` element is signed.

**action**
: Specifies that the `<wsa:Action>` element is signed.

**relatesto**
: Specifies that the time stamp is inserted into the `<wsa:RelatesTo>` element within the message and the `<wsa:RelatesTo>` element is signed.

7. Specify an expiration time for the time stamp in the Timestamp expires field. The time stamp helps defend against replay attacks. The lexical representation for the duration is the [ISO 8601] extended format `PnYnMnDTnHnMnS`, where:

   **P** Precedes the date and time values.

   **nY** Represents the number of years in which the time stamp is in effect. Select a value from 0 to 99 years.

   **nM** Represents the number of months in which the time stamp is in effect. Select a value from 0 to 11 months.

   **nD** Represents the number of days in which the time stamp is in effect. Select a value from 0 to 30 days.

   **T** Separates the date and time values.

   **nH** Represents the number of hours in which the time stamp is in effect. Select a value from 0 to 23 hours.

   **nM** Represents the number of minutes in which the time stamp is in effect. Select a value from 0 to 59 minutes.

   **nS** Represents the number of seconds in which the time stamp is in effect. The number of seconds can include decimal digits to aribrary precision. You can select a value from 0 to 59 for the seconds and from 0 to 9 for tenths of a second.

   For example, to indicate 1 year, 2 months, 3 days, 10 hours, and 30 minutes, the format is `P1Y2M3DT10H30M`. Typically, you might configure a message time stamp for between 10 and 30 minutes. For example, 10 minutes is represented as `P0Y0M0DT0H10M0S` or `PT10M`.
8. In the Message Parts section, click **Add** and select http://www.ibm.com/websphere/webservices/wssecurity/dialect-was in the Message parts dialect field.
9. In the Message Parts section, select the message parts keyword.

   **Important:** You must define at least one message part in the Message Parts section in order to specify a time stamp for integrity.
10. Click **OK** to save the configuration changes.

    **Note:** These configurations for the generator and the consumer must match.

    In addition to the time stamp, you can specify that the nonce is signed. For more information, see the following articles:

    - “Adding a nonce for integrity in generator security constraints with keywords” on page 577

- “Adding a nonce for integrity to generator security constraints with an XPath expression” on page 583

The following example shows a time stamp that is inserted in the SOAP message body and signed:

```
<soapenv:Envelope xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Header soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <wsse:Security soapenv:mustUnderstand="1" xmlns:wsse="http://docs.oasis-open.org/wss/
       2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
         <wsse:BinarySecurityToken EncodingType="http://docs.oasis-open.org/wss/2004/01/
          oasis-200401-wss-soap-message-security-1.0#Base64Binary" ValueType=
          "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3"
          wsu:Id="x509bst_6212871821454005389" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
          oasis-200401-wss-wssecurity-utility-1.0.xsd"> AgBgkqhkiG9w0BCQEWE21hcnV5YW1hQGpwLmlibS5jb22
          CAgEBMA0GCSqGSIb3DQEBBQUAA4GBAHkthdGDgCvdIL9/vXUo74xpfOQd/rr1owBmMdb1TWdOyzwbOHC7lkUlnKrkI7
          SofwSLSDUP571iiMXUx3tRdmAVCoDMMFuDXh9V7212luXccx0s1S5KN0D3xW97LLNegQC0/b+aFD8XKw2U5ZtwbnFTRgs
          097dmz09RosDKkLlM</wsse:BinarySecurityToken>
         <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
            <ds:SignedInfo>
               <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
                  <ec:InclusiveNamespaces PrefixList="wsse ds xsi soapenc xsd soapenv "
                   xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#"/>
               </ds:CanonicalizationMethod>
               <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
               <ds:Reference URI="#wssecurity_signature_id_493518228178200731">
                  <ds:Transforms>
                     <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
                        <ec:InclusiveNamespaces PrefixList="xsi soapenc xsd wsu soapenv "
                         xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                     </ds:Transform>
                  </ds:Transforms>
                  <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                  <ds:DigestValue>kKrcmc8saJ91JCNiE33UECoNYz8=</ds:DigestValue>
               </ds:Reference>
            </ds:SignedInfo>
            <ds:SignatureValue>XBpPju5+qH4bBFodO1kbBO54kEdBD0Pr5ohnXa3TPrDwXqmr67zDP3ZTk7iBSADnH+dlfKup
             Fhx+NZu2h5/j1/KYWaR2HTTv/KYE6IdqXVz3EFglUIBLzQnJ2Zbn62eBx5Th285Cn2Vrxtdb5BvUa1dt6M6k61CvRlz3
            /nMhQxk=</ds:SignatureValue>
            <ds:KeyInfo>
               <wsse:SecurityTokenReference>
                  <wsse:Reference URI="#x509bst_6212871821454005389" ValueType="http://docs.oasis-open.org/
                   wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3"/>
               </wsse:SecurityTokenReference>
            </ds:KeyInfo>
         </ds:Signature>
      </wsse:Security>
   </soapenv:Header>
   <soapenv:Body soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    wsu:Id="wssecurity_signature_id_493518228178200731" xmlns:wsu="http://docs.oasis-open.org/
    wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
      <getVersion/>
      <wsu:Timestamp wasextention="wedsig">
         <wsu:Created>2004-10-12T15:58:19.201Z</wsu:Created>
      </wsu:Timestamp>
   </soapenv:Body>
</soapenv:Envelope>
```

After you specify which message parts to digitally sign, you must specify which method is used to digitally sign the message. For more information, see “Configuring signing information for the generator binding with an assembly tool” on page 598.

**Related tasks**

“Adding a stand-alone time stamp to generator security constraints” on page 650

"Adding a nonce for integrity in generator security constraints with keywords"
"Adding a nonce for integrity to generator security constraints with an XPath expression" on page 583
"Configuring signing information for the generator binding with an assembly tool" on page 598

*Adding a nonce for integrity in generator security constraints with keywords:*

Prior to completing this task, you must import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF.

Nonce for integrity is used to specify that the nonce is embedded in a particular element and the element is signed. Nonce is a randomly generated, cryptographic token. When nonce is added to the specific parts of a message, it might prevent theft and replay attacks because a generated nonce is unique. For example, without nonce, when a user name token is passed from one machine to another machine using a non-secure transport, such as HTTP, the token might be intercepted and used in a replay attack. The user name token can be stolen even if you use XML digital signature and XML encryption. However, it might be prevented by adding a nonce.

Complete the following steps to specify a nonce for integrity using keywords when you configure the generator security constraints for either the request generator or the response generator. The request generator is configured for the client and the response generator is configured for the server. In the following steps, you must configure either the client-side extensions in step 2 or the server-side extensions in step 3.

1. Start the assembly tool and click **Window > Open Perspective > J2EE**.
2. **Optional:** Locate the client-side extensions using the Project Explorer window. The Client Deployment Descriptor window is displayed. This Web service contains the extensions that you need to configure.
   a. Expand the Web Services > Client section and double-click the name of the Web service.
   b. Click the **WS Extension** tab and expand the Request Generator Configuration section.
3. **Optional:** Locate the server-side extensions using the Project Explorer window. The Web Services Editor window is displayed. This Web service contains the extensions that you need to configure.
   a. Expand the Web Services > Services section and double-click the name of the Web service.
   b. Click the **Extensions** tab and expand the Response Generator Service Configuration Details section.
4. Expand the Integrity section. Integrity refers to digital signature while confidentiality refers to encryption. Integrity decreases the risk of data modification when you transmit data across a network. For more information on digitally signing Simple Object Access Protocol (SOAP) messages, see "XML digital signature" on page 570.
5. Click **Add** to specify a nonce for integrity. The Integrity Dialog window is displayed.
   a. Specify a name for the integrity element in the Required Integrity Name field. For example, you might specify `int_nonce`.
   b. Specify an order in the Order field. The value, which must be a positive integer value, specifies the order in which the digital signature is processed. An order value of 1 specifies that the signing is done first.
6. Under Nonce, click **Add** and select the nonce dialect. The http://www.ibm.com/websphere/webservices/wssecurity/dialect-was dialect specifies the message part to which a nonce is added and signed. If you select this dialect, you can select one of the following keywords under Nonce keyword:

   **body** Specifies the user data portion of the message. If this option is selected, a nonce is embedded in the Simple Object Access Protocol (SOAP) body element and the body element is signed.

   **timestamp**
   Specifies that the nonce is embedded in the stand-alone timestamp element within the

message and the element is signed. If timestamp is selected, make sure that there is a stand-alone timestamp element in the message. If not, see “Adding a stand-alone time stamp to generator security constraints” on page 650.

**securitytoken**
: Specifies that the securitytoken element is signed. The security token authenticates the client. If this option is selected, the nonce element is embedded in the securitytoken element.

**dsigkey**
: Specifies that the nonce is inserted into the key information element, which is used for digital signature, and the key information element is signed.

**enckey**
: Specifies that the nonce is inserted into the key information element, which is used for encryption, and the key information element is signed.

**messageid**
: Specifies that the nonce is inserted into the `<wsa:MessageID>` element and that the `<wsa:MessageID>` element is signed.

**to**
: Specifies that the nonce is inserted into the `<wsa:To>` element within the message and that the `<wsa:To>` element is signed.

**action**
: Specifies that the `<wsa:Action>` element is signed.

**relatesto**
: Specifies that the nonce is inserted into the `<wsa:RelatesTo>` element within the message and that the `<wsa:RelatesTo>` element is signed.

7. In the Message Parts section, click **Add** and select http://www.ibm.com/websphere/webservices/wssecurity/dialect-was in the Message parts dialect field.
8. In the Message Parts section, select the message parts keyword.

   **Important:** You must define at least one message part in the Message Parts section in order to specify a nonce for integrity. This message part is signed as well as the parent element of the nonce.
9. Click **OK** to save the configuration changes.

   **Note:** These configurations on the consumer side and the generator side must match.

   In addition to the nonce, you can specify that the timestamp element is signed. For more information, see the following articles:
   - “Adding time stamps for integrity in consumer security constraints with keywords” on page 602
   - “Adding time stamps for integrity in consumer security constraints with an XPath expression” on page 608

The following example is a SOAP message that has a nonce that is inserted in the SOAP message body and signed:

```
<soapenv:Envelope xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Header soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <wsse:Security soapenv:mustUnderstand="1" xmlns:wsse="http://docs.oasis-open.org/
       wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
         <wsse:BinarySecurityToken EncodingType="http://docs.oasis-open.org/wss/2004/01/
          oasis-200401-wss-soap-message-security-1.0#Base64Binary" ValueType=
          "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3"
          wsu:Id="x509bst_1179110083179840266" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
          oasis-200401-wss-wssecurity-utility-1.0.xsd"> E21hcnV5YW1hQGpwLmlibS5jb22CAgEBMA0GCSqGSIb3
          DQEBBQUAA4GBAHkthdGDgCvdIL9/vXUo74xpfOQd/rr1owBmMdb1TWdOyzwbOHC7lkUlnKrkI7SofwSLSDUP571iiMX
          Ux3tRdmAVCoDMMFuDXh9V7212luXccx0s1S5KN0D3xW97LLNegQC0/b+aFD8XKw2U5ZtwbnFTRgs097dmz09RosDKkLlM
```

```
        </wsse:BinarySecurityToken>
        <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
           <ds:SignedInfo>
              <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
                 <ec:InclusiveNamespaces PrefixList="wsse ds xsi soapenc xsd soapenv "
                  xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#"/>
              </ds:CanonicalizationMethod>
              <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
              <ds:Reference URI="#wssecurity_signature_id_8451968259110349556">
                 <ds:Transforms>
                    <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
                       <ec:InclusiveNamespaces PrefixList="xsi soapenc xsd wsu soapenv "
                        xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                    </ds:Transform>
                 </ds:Transforms>
                 <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                 <ds:DigestValue>HgfL7FiG/TGECE/L0zg5mJldfgc=</ds:DigestValue>
              </ds:Reference>
           </ds:SignedInfo>
           <ds:SignatureValue>iE2G53VMwCFBI6BwOWiaOLYvemZUJTXJocXpy8loyW1LiR8bBQcFioDOuDXXzVj3K+ZD2p
            Yhc0krVYqkYY0IZoRx7xpWt+9qn7aSbxKjuHlMFNCdB1Uxp608zCZcSwvuoCffjlO0ltUQ8JTEBnmMB0cfaoiG5bF
            kUOEpkFo2P9c=</ds:SignatureValue>
           <ds:KeyInfo>
              <wsse:SecurityTokenReference>
                 <wsse:Reference URI="#x509bst_1179110083179840266" ValueType="http://docs.oasis-open.org/
                  wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3"/>
              </wsse:SecurityTokenReference>
           </ds:KeyInfo>
        </ds:Signature>
     </wsse:Security>
  </soapenv:Header>
  <soapenv:Body soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
   wsu:Id="wssecurity_signature_id_8451968259110349556" xmlns:wsu="http://docs.oasis-open.org/
   wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
     <getVersion/>
     <wsse:Nonce wasextention="wedsig" xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/
      oasis-200401-wss-wssecurity-secext-1.0.xsd">1u0otbnjkPiCWDhh25yEyBHD/r3VPSbQ1oZTs0Ks1GE/iDL4YbKDT
      wdL+e2Hb7nNZn397nRJQ9ePGgf7PRdEuqATFbfq0/T+6j6Fk/MbSHmZnHHoBscFX8W/dYssyCmWDp99447kRhnJbNg5JxarkFmM
      LqpxKfm1iP3hKP5DpJY=</wsse:Nonce>
  </soapenv:Body>
</soapenv:Envelope>
```

After you specify which message parts to digitally sign, you must specify which method is used to digitally sign the message. For more information, see “Configuring signing information for the consumer binding with an assembly tool” on page 620.

**Related concepts**

“XML digital signature” on page 570
XML-Signature Syntax and Processing (XML digital signature) is a specification that defines XML syntax and processing rules to sign and verify digital signatures for digital content. The specification was developed jointly by the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF).

**Related tasks**

“Adding a stand-alone time stamp in consumer security constraints” on page 651

“Adding time stamps for integrity in consumer security constraints with keywords” on page 602

“Adding time stamps for integrity in consumer security constraints with an XPath expression” on page 608

“Adding a nonce for integrity in consumer security constraints with an XPath expression” on page 610

“Configuring signing information for the consumer binding with an assembly tool” on page 620

***Signing message elements in generator security constraints with an XPath expression:***

Prior to completing this task, you must import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF.

The following information explains the difference between using an XPath expression and using keywords to specify which part of the message to sign:

**XPath expression**
: Specify any part of the message using an XPath expression. XPath is a language that is used to address parts of an XML document. You can find information on XPath syntax at the following Web site: http://www.w3.org/TR/1999/REC-xpath-19991116.

**Keywords**
: Specify only elements within the message using predefined keywords.

Complete the following steps to specify which message parts to digitally sign when you configure the generator security constraints for either the request generator or the response generator. The request generator is configured for the client and the response generator is configured for the server. In the following steps, you must configure either the client-side extensions in step 2 or the server-side extensions in step 3.

1. Start the assembly tool and click **Window > Open Perspective > J2EE**.
2. **Optional:** Locate the client-side extensions using the Project Explorer window. The Client Deployment Descriptor window is displayed. This Web service contains the extensions that you need to configure.
   a. Expand the Web Services > Client section and double-click the name of the Web service.
   b. Click the **WS Extension** tab and expand the Request Generator Configuration section.
3. **Optional:** Locate the server-side extensions using the Project Explorer window. The Web Services Editor window is displayed. This Web service contains the extensions that you need to configure.
   a. Expand the Web Services > Services section and double-click the name of the Web service.
   b. Click the **Extensions** tab and expand the Response Generator Service Configuration Details section.
4. Expand the Integrity section. Integrity refers to digital signature while confidentiality refers to encryption. Integrity decreases the risk of data modification when you transmit data across a network. For more information on digitally signing Simple Object Access Protocol (SOAP) messages, see “XML digital signature” on page 570.
5. Click **Add** to indicate which parts of the message to sign. The Integrity Dialog window is displayed.
   a. Specify a name for the integrity element in the Integrity Name field. For example, you might specify `int_xpath`.
   b. Specify an order in the Order field. The value, which must be a positive integer value, specifies the order in which the digital signature is processed. An order value of 1 specifies that the signing is done first.
6. Click **Add** under the Message Parts section of the Integrity Dialog window.
   a. Select the Message parts dialect from the Message Parts section of the Integrity Dialog window. If you select the http://www.w3.org/TR/1999/REC-xpath-19991116 dialect, the message part that will be signed is specified by an XPath expression.
   b. Specify the message part to be signed using an XPath expression in the Message parts keyword field. For example, to specify that the body is signed, you might add the following expression in the Message parts keyword field as one continuous line:
      ```
      /*[namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/' and local-name()='Envelope']/*
      [namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/' and local-name()='Body']
      ```
7. Click **OK** to save the configuration changes.

   **Note:** These configurations for the generator and the consumer must match.

In addition to the message parts, you also can specify that WebSphere Application Server sign the nonce and timestamp elements. For more information, see the following articles:

- "Adding time stamps for integrity to generator security constraints with keywords" on page 574
- "Adding time stamps for integrity to generator security constraints with an XPath expression"
- "Adding a nonce for integrity in generator security constraints with keywords" on page 577
- "Adding a nonce for integrity to generator security constraints with an XPath expression" on page 583

After you specify which message parts to digitally sign, you must specify which method is used to digitally sign the message. For more information, see "Configuring signing information for the generator binding with an assembly tool" on page 598.

**Related concepts**

"XML digital signature" on page 570
XML-Signature Syntax and Processing (XML digital signature) is a specification that defines XML syntax and processing rules to sign and verify digital signatures for digital content. The specification was developed jointly by the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF).

**Related tasks**

"Adding a stand-alone time stamp to generator security constraints" on page 650

"Adding time stamps for integrity to generator security constraints with keywords" on page 574

"Adding a nonce for integrity in generator security constraints with keywords" on page 577

"Configuring signing information for the generator binding with an assembly tool" on page 598

*Adding time stamps for integrity to generator security constraints with an XPath expression:*

Prior to completing this task, you must import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF.

The following information explains the difference between using an XPath expression and using keywords to specify which part of the message to sign:

**XPath expression**
: Specify any part of the message using an XPath expression. XPath is a language that is used to address parts of an XML document. You can find information on XPath syntax at the following Web site: http://www.w3.org/TR/1999/REC-xpath-19991116.

**Keywords**
: Specify only elements within the message using predefined keywords.

This task is used to specify that a time stamp is embedded in a particular element and that the element is signed. Complete the following steps to specify the time stamp for integrity using keywords when you configure the generator security constraints for either the request generator or the response generator. The request generator is configured for the client and the response generator is configured for the server. In the following steps, you must configure either the client-side extensions in step 2 or the server-side extensions in step 3.

1. Start the assembly tool and click **Window > Open Perspective > J2EE**.
2. **Optional:** Locate the client-side extensions using the Project Explorer window. The Client Deployment Descriptor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the client-side extensions.
   a. Expand the Web Services > Client section and double-click the name of the Web service.
   b. Click the **WS Extension** tab and expand the Request Generator Configuration section.

3. **Optional:** Locate the server-side extensions using the Project Explorer window. The Web Services Editor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the server-side extensions.
   a. Expand the Web Services > Services section and double-click the name of the Web service.
   b. Click the **Extensions** tab and expand the Response Generator Service Configuration Details section.
4. Expand the Integrity section. Integrity refers to digital signature while confidentiality refers to encryption. Integrity decreases the risk of data modification when you transmit data across a network. For more information on digitally signing Simple Object Access Protocol (SOAP) messages, see "XML digital signature" on page 570.
5. Click **Add** to specify a time stamp for integrity. The Integrity Dialog window is displayed. Complete the following steps to specify a configuration:
   a. Specify a name for the integrity element in the Integrity Name field. For example, you might specify `int_tmstmp`.
   b. Specify an order in the Order field. The value, which must be a positive integer value, specifies the order in which the digital signature is processed. An order value of 1 specifies that the signing is done first.
6. Click **Add** in the Timestamp section of the Integrity Dialog window. Complete the following steps to specify a time stamp configuration:
   a. Select the Timestamp dialect from the Timestamp section. The http://www.w3.org/TR/1999/REC-xpath-19991116 dialect specifies the message part to which the time stamp is added and signed using the XPath expression.
   b. Select the message part in the Timestamp keyword field to which the time stamp is added and signed using an XPath expression. For example, to specify that the time stamp is added to the body and is signed, you might specify the following expression for the Timestamp keyword:

```
/*[namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/' and local-name()='Envelope']/*
[namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/' and local-name()='Body']
```

   c. Specify an expiration time for the time stamp in the Timestamp expires field. The time stamp helps defend against replay attacks. The lexical representation for the duration is the [ISO 8601] extended format `PnYnMnDTnHnMnS`, where:

**P**
Precedes the date and time values.

**nY**
Represents the number of years in which the time stamp is in effect. Select a value from 0 to 99 years.

**nM**
Represents the number of months in which the time stamp is in effect. Select a value from 0 to 11 months.

**nD**
Represents the number of days in which the time stamp is in effect. Select a value from 0 to 30 days.

**T**
Separates the date and time values.

**nH**
Represents the number of hours in which the time stamp is in effect. Select a value from 0 to 23 hours.

**nM**
Represents the number of minutes in which the time stamp is in effect. Select a value from 0 to 59 minutes.

**nS**
Represents the number of seconds in which the time stamp is in effect. The number of seconds can include decimal digits to arbitrary precision. You can select a value from 0 to 59 for the seconds and from 0 to 9 for tenths of a second.

For example, to indicate 1 year, 2 months, 3 days, 10 hours, and 30 minutes, the format is `P1Y2M3DT10H30M`. Typically, you might configure a message time stamp for between 10 and 30 minutes. For example, 10 minutes is represented as `P0Y0M0DT0H10M0S` or `PT10M`.

7. In the Message Parts section, click **Add** and select http://www.ibm.com/websphere/webservices/wssecurity/dialect-was in the Message parts dialect field.
8. In the Message Parts section, select the message parts keyword.

   **Important:** You must define at least one message part in the Message Parts section in order to specify a time stamp for integrity. This message part is signed as well as the parent element of the time stamp.
9. Click **OK** to save the configuration changes.

   **Note:** These configurations for the generator and the consumer must match.

   In addition to the time stamp, you can specify that the nonce is signed. For more information, see the following articles:
   - “Adding a nonce for integrity in generator security constraints with keywords” on page 577
   - “Adding a nonce for integrity to generator security constraints with an XPath expression”

After you have specified which message parts to digitally sign, you must specify which method is used to digitally sign the message. For more information, see “Configuring signing information for the generator binding with an assembly tool” on page 598.

**Related tasks**

“Adding a stand-alone time stamp to generator security constraints” on page 650

“Adding a nonce for integrity in generator security constraints with keywords” on page 577

“Adding a nonce for integrity to generator security constraints with an XPath expression”

“Configuring signing information for the generator binding with an assembly tool” on page 598

*Adding a nonce for integrity to generator security constraints with an XPath expression:*

Prior to completing this task, you must import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF.

The following information explains the difference between using an XPath expression and using keywords to specify which part of the message to sign:

**XPath expression**
: Specify any part of the message using an XPath expression. XPath is a language that is used to address parts of an XML document. You can find information on XPath syntax at the following Web site: http://www.w3.org/TR/1999/REC-xpath-19991116.

**Keywords**
: Specify only elements within the message using predefined keywords.

Nonce for integrity is used to specify that the nonce is embedded in a particular element and the element is signed. Nonce is a randomly generated, cryptographic token. When nonce is added to the specific parts of a message, it might prevent theft and replay attacks because a generated nonce is unique. For example, without nonce, when a user name token is passed from one machine to another machine using a non-secure transport, such as HTTP, the token might be intercepted and used in a replay attack. The user name token can be stolen even if you use XML digital signature and XML encryption. However, it might be prevented by adding a nonce.

Complete the following steps to specify a nonce for integrity using an XPath expression when you configure the generator security constraints for either the request generator or the response generator. The request generator is configured for the client and the response generator is configured for the server. In the following steps, you must configure either the client-side extensions in step 2 or the server-side extensions in step 3.

1. Start the assembly tool and click **Window > Open Perspective > J2EE**.
2. **Optional:** Locate the client-side extensions using the Project Explorer window. The Client Deployment Descriptor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the client-side extensions:
   a. Expand the Web Services > Client section and double-click the name of the Web service.
   b. Click the **WS Extension** tab and expand the Request Generator Configuration section.
3. **Optional:** Locate the server-side extensions using the Project Explorer window. The Web Services Editor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the server-side extensions:
   a. Expand the Web Services > Services section and double-click the name of the Web service.
   b. Click the **Extensions** tab and expand the Response Generator Service Configuration Details section.
4. Expand the Integrity section. Integrity refers to digital signature while confidentiality refers to encryption. Integrity decreases the risk of data modification when you transmit data across a network. For more information on digitally signing Simple Object Access Protocol (SOAP) messages, see “XML digital signature” on page 570.
5. Click **Add** to specify a nonce for integrity. The Integrity Dialog window is displayed. Complete the following steps to specify a configuration:
   a. Specify a name for the integrity element in the Integrity Name field.
   b. Specify an order in the Order field. The value, which must be a positive integer value, specifies the order in which the digital signature is processed. An order value of 1 specifies that the signing is done first.
6. Click **Add** in the Nonce section of the Integrity Dialog window. Complete the following steps to specify a nonce dialect and message part:
   a. Select the Nonce dialect from the Nonce section. The http://www.w3.org/TR/1999/REC-xpath-19991116 dialect specifies the message part to which a nonce is added and signed using an XPath expression.
   b. Select the message part in the Nonce keyword field to which a nonce is added and signed using an XPath expression. For example, to specify that a nonce is added to the body and that it is signed, you might specify the following expression for the Nonce keyword:
   ```
   /*[namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/' and local-name()='Envelope']/*
   [namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/' and local-name()='Body']
   ```
7. In the Message Parts section, click **Add** and select http://www.w3.org/TR/1999/REC-xpath-19991116 in the Message parts dialect field.
8. In the Message Parts section, select the message parts keyword.

   **Important:** You must define at least one message part in the Message Parts section in order to specify a nonce for integrity. This message part is signed as well as the parent element of the nonce.
9. Click **OK** to save the configuration changes.

   **Note:** These configurations for the consumer and the generator must match.

   In addition to the nonce, you can specify that the timestamp element is signed. For more information, see the following articles:
   - “Adding time stamps for integrity to generator security constraints with keywords” on page 574
   - “Adding time stamps for integrity to generator security constraints with an XPath expression” on page 581

After you specify that a nonce is added to the message parts and signed, you must specify which method is used to digitally sign the message. For more information, see “Configuring signing information for the generator binding with an assembly tool” on page 598.

**Related concepts**

"XML digital signature" on page 570
XML-Signature Syntax and Processing (XML digital signature) is a specification that defines XML syntax and processing rules to sign and verify digital signatures for digital content. The specification was developed jointly by the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF).

**Related tasks**

"Adding time stamps for integrity to generator security constraints with keywords" on page 574

"Adding time stamps for integrity to generator security constraints with an XPath expression" on page 581

"Configuring signing information for the generator binding with an assembly tool" on page 598

***Collection certificate store:***

A *collection certificate store* is a collection of non-root, certificate authority (CA) certificates and certificate revocation lists (CRLs). This collection of CA certificates and CRLs is used to check the signature of a digitally signed Simple Object Access Protocol (SOAP) message.

A collection certificate store is used when WebSphere Application Server is processing a received SOAP message. This collection is configured in the Request Consumer Service Configuration Details section of the binding file for servers and in the Response Consumer Configuration section of the binding file for clients. You can configure these two sections using one of the assembly tools provided by WebSphere Application Server. For more information on the assembly tools, see ″Assembly tools″ in the ″Developing and deploying applications″ PDF.

A collection certificate store is one kind of certificate store. A certificate store is defined as javax.security.cert.CertStore in the Java CertPath application programming interface (API). The Java CertPath API defines the following types of certificate stores:

**Collection certificate store**
A collection certificate store accepts the certificates and CRLs as Java collection objects.

**Lightweight Directory Access Protocol certificate store**
The Lightweight Directory Access Protocol (LDAP) certificate store accepts certificates and CRLs as LDAP entries.

The CertPath API uses the certificate store and the trust anchor to validate the incoming X.509 certificate that is embedded in the SOAP message. The Web services security implementation in the WebSphere Application Server supports the collection certificate store. Each certificate and CRL is passed as an encoded file.

**Related concepts**

"Trust anchor" on page 587
A *trust anchor* specifies the key stores that contain trusted root certificates. These certificates are used to validate the X.509 certificate that is embedded in the Simple Object Access Protocol (SOAP) message.

*Certificate revocation list:*

A *certificate revocation list* is a time-stamped list of certificates that have been revoked by a certificate authority (CA).

A certificate that is found in a certificate revocation list (CRL) might not be expired, but is no longer trusted by the certificate authority that issued the certificate. The certificate authority creates the CRL that contains the serial number and issuing CA distinguished name of the certificate that has been revoked. The CA

might add the certificate to the certificate revocation list if it believes that the client certificate is compromised. The certificate revocation list is maintained and issued by the certificate authority.

***Configuring the collection certificate store for the generator binding with an assembly tool:***

Prior to completing this task, you must import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF.

This task describes the steps to specify the collection certificate store for the generator bindings at the application level using an assembly tool. A collection certificate store is a collection of non-root certificate authority (CA) certificates and certificate revocation lists (CRLs) that is used for validating an X.509 certificate embedded within the received SOAP message. The request generator is configured for the client and the response generator is configured for the server. On the generator side, a configuration for the collection certificate store is required only when you configure CRLs that are embedded in the PKCS#7 format. Complete the following steps to configure a collection certificate store for the generator. Specifying either the client-side bindings in step 2 or the server-side bindings in step 3.

1. Start the assembly tool and click **Window > Open Perspective > J2EE**.
2. **Optional:** Locate the client-side bindings using the Project Explorer window. The Client Deployment Descriptor window is displayed. This Web service contains the bindings that you need to configure.
   a. Expand the Web Services > Client section and double-click the name of the Web service.
   b. Click the **WS Binding** tab and expand the Security Request Generator Binding Configuration section.
3. **Optional:** Locate the server-side bindings using the Project Explorer window. The Web Services Editor window is displayed. This Web service contains the bindings that you need to configure.
   a. Expand the Web Services > Services section and double-click the name of the Web service.
   b. Click the **Binding Configurations** tab and expand the Response Generator Binding Configuration Details section.
4. Expand the Certificate Store List > Collection Certificate Store section and click **Add**.
5. Specify a unique certificate store name in the Name field. For example, specify `cert1`. The name of the collection certificate store must be unique on the level in which it is defined. For example, the name must be unique at the application level. The name specified in the certificate store name field is used by other configurations to refer to a predefined collection certificate store. WebSphere Application Server looks up the collection certificate store based on proximity. For example, if an application binding refers to certificate store cert1, WebSphere Application Server looks first for cert1 at the application level. If it is not found, it looks at the server level, and finally at the cell level.
6. Specify a certificate store provider in the Provider field. The `IBMCertPath` certificate path provider is supported. To use another certificate path provider, you must define the provider implementation in the provider list within the `java.security` file in the Software Development Kit (SDK).
7. Click **Add** under X509 Certificate to specify a fully qualified path to an X.509 certificate, click the name of an existing certificate path entry to edit it, or click **Remove** to delete it. This collection certificate store is used to validate the certificate path of the incoming X.509-formatted security tokens.

   You can use the `USER_INSTALL_ROOT` variable as part of the path name. For example, you might specify `${USER_INSTALL_ROOT}/etc/ws-security/samples/intca2.cer`. However, do not use this X.509 certificate path for production use. Obtain your own X.509 certificate from a certificate authority before putting your WebSphere Application Server environment into production.

   In the WebSphere Application Server administrative console, you can click **Environment > WebSphere Variables** to configure the **USER_INSTALL_ROOT** variable.
8. Click **Add** under CRL to specify the fully qualified path to a certificate revocation list (CRL), click an existing CRL entry to edit it or click **Remove** to delete it.

   For portability reasons, it is recommended that you use the WebSphere Application Server variables to specify a relative path to the certificate revocation list. For example, you might use the

`USER_INSTALL_ROOT` variable to define a path such as `${USER_INSTALL_ROOT}/mycertstore/mycrl`. For a list of the supported variables in the WebSphere Application Server administrative console, click **Environment > WebSphere Variables**.

The following list provides recommendations for using CRLs:

- If CRLs are added to the collection certificate store, add the CRLs for the root certificate authority and each intermediate certificate, if applicable. When the CRL is in the certificate collection store, the certificate revocation status for every certificate in the chain is checked against the CRL of the issuer.
- When the CRL file is updated, the new CRL does not take effect until you restart the Web service application.
- Before a CRL expires, you must load a new CRL into the certificate collection store to replace the old CRL. An expired CRL in the collection certificate store results in a certificate path (CertPath) build failure.

9. Click **OK** to save your configuration.

**Related concepts**

"Collection certificate store" on page 585
A *collection certificate store* is a collection of non-root, certificate authority (CA) certificates and certificate revocation lists (CRLs). This collection of CA certificates and CRLs is used to check the signature of a digitally signed Simple Object Access Protocol (SOAP) message.

**Related tasks**

"Configuring token generators with an assembly tool"

***Trust anchor:***

A *trust anchor* specifies the key stores that contain trusted root certificates. These certificates are used to validate the X.509 certificate that is embedded in the Simple Object Access Protocol (SOAP) message.

These key stores are used by the following message points to validate the X.509 certificate that is used for digital signature or XML encryption:

- Request consumer, as defined in the `ibm-webservices-bnd.xmi` file
- Response consumer, as defined in the `ibm-webservicesclient-bnd.xmi` file when a Web service is acting as a client to another Web service

The key stores are critical to the integrity of the digital signature validation. If the key stores are tampered with, the result of the digital signature verification is doubtful and compromised. Therefore, it is recommended that you secure these key stores. The binding configuration specified for the request consumer in the `ibm-webservices-bnd.xmi` file must match the binding configuration for the request generator in the `ibm-webservicesclient-bnd.xmi` file.

The trust anchor is defined as java.security.cert.TrustAnchor in the Java CertPath application programming interface (API). The Java CertPath API uses the trust anchor and the certificate store to validate the incoming X.509 certificate that is embedded in the Simple Object Access Protocol (SOAP) message. The Web services security implementation in WebSphere Application Server supports this trust anchor. In WebSphere Application Server, the trust anchor is represented as a Java key store object. The type, path, and password of the key store are passed to the implementation through the administrative console or by scripting.

**Related concepts**

"Collection certificate store" on page 585
A *collection certificate store* is a collection of non-root, certificate authority (CA) certificates and certificate revocation lists (CRLs). This collection of CA certificates and CRLs is used to check the signature of a digitally signed Simple Object Access Protocol (SOAP) message.

***Configuring token generators with an assembly tool:***

Prior to completing this task, you must complete the following steps:

- Import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF.
- Configure the security token in the extensions file. For example, if you are configuring a token generator for a Lightweight Third Party Authentication (LTPA) token, you must first configure the LTPA token under the Security Token section on the **Extensions** tab. For more information, see "Configuring the security token in generator security constraints" on page 653.
- Configure a collection certificate store if the token generator uses the PKCS#7 token type and you want to package the certificate revocation lists (CRL) in the security token. For more information, see "Configuring the collection certificate store for the generator binding with an assembly tool" on page 586.

A security token represents a set of claims that are made by a client. This set of claims might include a name, password, identity, key, certificate, group, privilege, and so on. A security token is embedded in the Simple Object Access Protocol (SOAP) message within the SOAP header. The security token within the SOAP header is propagated from the message sender to the intended message receiver.

Complete the following steps to configure either the client-side bindings for the token generator in step 2 or the server-side bindings for the token generator in step 3:

1. Start the assembly tool and click **Window > Open Perspective > J2EE**.
2. **Optional:** Locate the client-side bindings using the Project Explorer window. The Client Deployment Descriptor window is displayed. This Web service contains the bindings that you need to configure.
   a. Expand the Web Services > Clients section and double-click the name of the Web service.
   b. Click the **WS Binding** tab and expand the Security Request Generator Binding Configuration section.
3. **Optional:** Locate the server-side bindings using the Project Explorer window. The Web Services Editor window is displayed. This Web service contains the bindings that you need to configure.
   a. Expand the Web Services > Services section and double-click the name of the Web service.
   b. Click the **Binding Configurations** tab and expand the Response Generator Binding Configuration Details section.
4. **Optional:** Configure a trust anchor if you are configuring this token consumer for an X.509 security token.
   a. Expand the Trust anchor section and click **Add** to add a new entry or click **Edit** to edit a selected entry. The Trust anchor dialog window is displayed.
   b. Specify a name for the trust anchor configuration in the Trust anchor name field.
   c. Specify a keystore password in the Key store storepass field. The keystore storepass is the password that is required to access the keystore file.
   d. Specify the path to the keystore file in the Key store path field. The key store path is the directory where the keystore resides. Make sure that wherever you deploy your application that can locate your keystore file.
   e. Select a key store type from the Key store type field. The key store type that you select must match the keystore file that is specified in the Key store path field.
5. Expand the Token generator section and click **Add** to add a new entry or click **Edit** to edit a selected entry. The Token Generator Dialog window is displayed.
6. Specify a unique name in the Token generator name field. For example, you might specify `gen_signtgen`. If this token generator is for an X.509 certificate and is used for signature generation or encryption, the token generator name is referenced in the Token field of the Key Information dialog window.

7. Select a token generator class in the Token generator class field. Select the token generator class that matches the type of token that you are configuring. This class must implement the com.ibm.wsspi.wssecurity.token.TokenGeneratorComponent interface. The following default token generator implementations are supported:
   - `com.ibm.wsspi.wssecurity.token.LTPATokenGenerator`
   - `com.ibm.wsspi.wssecurity.token.UsernameTokenGenerator`
   - `com.ibm.wsspi.wssecurity.token.X509TokenGenerator`
8. Select a security token reference in the Security token field. The value in this field references the security token that is configured in the extensions file.
9. Select the **Use value type** option and select the value type in the Value type field. Select the value type of the security token that matches the type of token generator that you are configuring. When you select the value type, the assembly tool automatically enters the correct values in the Local name and URI fields depending upon the type of security token that is specified by the value type. If you select **Custom Token**, you must specify the local name and the namespace URI of the value type for the generated token. The following value types are supported:
   - Username Token
   - X509 certificate token
   - X509 certificates in a PKIPath
   - A list of X509 certificates and CRLs in a PKCS#7
   - LTPA Token
   - Custom Token
10. Specify the Callback handler class name in the Call back handler field. This name is the callback handler implementation class that is used to plug-in a security token framework. The specified callback handler class must implement the javax.security.auth.callback.CallbackHandler interface. The implementation of the Java Authentication and Authorization Service (JAAS) javax.security.auth.callback.CallbackHandler interface must provide a constructor using the following syntax:

    ```
    MyCallbackHandler(String username, char[] password, java.util.Map properties)
    ```

    Where:
    - *username* specifies the user name that is passed into the configuration.
    - *password* specifies the password that is passed into the configuration.
    - *properties* specifies the other configuration properties that are passed into the configuration.

    The following default callback handler implementations are supported:

    **com.ibm.wsspi.wssecurity.auth.callback.GUIPromptCallbackHandler**
    This callback handler uses a login prompt to gather the user name and password information. However, if you specify the user name and password on this panel, a prompt is not displayed and WebSphere Application Server returns the user name and password to the token generator if it is specified on this panel. Use this implementation for a Java 2 Platform, Enterprise Edition (J2EE) application client only.

    **com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler**
    This callback handler does not issue a prompt and returns the user name and password if it is specified on this panel. You can use this callback handler when the Web service is acting as a client.

    **com.ibm.wsspi.wssecurity.auth.callback.StdinPromptCallbackHandler**
    This callback handler uses a standard-in prompt to gather the user name and password. However, if the user name and password is specified on this panel, WebSphere Application Server does not issue a prompt, but returns the user name and password to the token generator. Use this implementation for a J2EE application client only.

    **com.ibm.wsspi.wssecurity.auth.callback.LTPATokenCallbackHandler**
    This callback handler is used to obtain the Lightweight Third Party Authentication (LTPA)

security token from the Run As invocation Subject. This token is inserted in the Web services security header within the SOAP message as a binary security token. However, if the user name and password are specified on this panel, WebSphere Application Server authenticates the user name and password to obtain the LTPA security token rather than obtaining it from the Run As Subject. Use this callback handler only when the Web service is acting as a client on the application server. It is recommended that you do not use this callback handler on a J2EE application client.

**com.ibm.wsspi.wssecurity.auth.callback.X509CallbackHandler**
: This callback handler is used to create the X.509 certificate that is inserted in the Web services security header within the SOAP message as a binary security token. A keystore and a key definition is required for this callback handler.

**com.ibm.wsspi.wssecurity.auth.callback.PKCS7CallbackHandler**
: This callback handler is used to create X.509 certificates encoded with the PKCS#7 format. The certificate is inserted in the Web services security header in the SOAP message as a binary security token. A keystore is required for this callback handler. You must specify a certificate revocation list (CRL) in the collection certificate store. The CRL is encoded with the X.509 certificate in the PKCS#7 format.

**com.ibm.wsspi.wssecurity.auth.callback.PkiPathCallbackHandler**
: This callback handler is used to create X.509 certificates encoded with the PkiPath format. The certificate is inserted in the Web services security header within the SOAP message as a binary security token. A keystore is required for this callback handler. A CRL is not supported by the callback handler; therefore, the collection certificate store is not required or used.

The callback handler implementation obtains the required security token and passes it to the token generator. The token generator inserts the security token in the Web services security header within the SOAP message. Also, the token generator is a plug-in point for the pluggable security token framework. Service providers can provide their own implementation, but the implementation must use the com.ibm.wsspi.wssecurity.token.TokenConsumerComponent interface.

11. Specify the basic authentication User ID in the User ID field. This user name is passed to the constructors of the callback handler implementation. The basic authentication user name and password are used if you select one of the following default callback handler implementations, as described in the previous step:
    - com.ibm.wsspi.wssecurity.auth.callback.GUIPromptCallbackHandler
    - com.ibm.wsspi.wssecurity.auth.callback.LTPATokenCallbackHandler
    - com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler
    - com.ibm.wsspi.wssecurity.auth.callback.StdinPromptCallbackHandler
12. Specify the basic authentication password in the Password field. This password is passed to the constructors of the callback handler implementation.
13. **Optional:** Select the **Use key store** option and complete the following substeps if you previously selected one of the following callback handlers: com.ibm.wsspi.wssecurity.auth.callback.PKCS7CallbackHandler. com.ibm.wsspi.wssecurity.auth.callback.PkiPathCallbackHandler, or com.ibm.wsspi.wssecurity.auth.callback.X509CallbackHandler.
    a. Specify the password for the keystore in the Key store storepass field. This password is used to access the keystore file.
    b. Specify the location of the keystore in the Key store path field.
    c. Specify the type of keystore in the Key store type field. The following keystore types are supported:

       **JKS** Use this option if the keystore uses the Java Keystore (JKS) format.

       **JCEKS**
       : Use this option if the Java Cryptography Extension is configured in the software

development kit (SDK). The default IBM JCE is configured in WebSphere Application Server. This option provides stronger protection for stored private keys by using Triple DES encryption.

**PKCS11**
Use this format if your keystore uses the PKCS#11 file format. Keystores using this format might contain RSA keys on cryptographic hardware or might encrypt keys that use cryptographic hardware to ensure protection.

**PKCS12**
Use this option if your keystore uses the PKCS#12 file format.

d. In the Keys section, click **Add** to add a key. You can also click **Remove** to remove an existing key.
e. In the Key section, specify an alias for the key in the Alias field. For example, you might specify `bob`. The key alias is used by the key locator to locate the key within the keystore file.
f. In the Keys section, specify a password for the key in the Key pass field. This password is needed to access the key object within the keystore file.
g. In the Keys section, specify a name in the Key name field. The key name must be a fully qualified, distinguished name. For example, you might specify `CN=Bob,O=IBM,C=US`.

14. **Optional:** Select the **Use certificate path settings** option if the token generator uses the PKCS#7 token type and you want to package the certificate revocation lists (CRL) in the security token.
    a. Select the **Certificate path reference** option and a certificate store reference. This selection references a certificate store that is configured in the Certificate Store List section. For more information, see "Configuring the collection certificate store for the generator binding with an assembly tool" on page 586.
15. **Optional:** Click **Add** and specify additional properties in the Property section.

    If the token generator includes a nonce in the user name token, add the following name and value pair:

    **Name** `com.ibm.wsspi.wssecurity.token.username.addNonce`

    **Value** `true`

    Nonce is a unique cryptographic number that is embedded in a message to help stop repeat, unauthorized attacks of user name tokens. A property is valid only when the generated token type is a user name token. This option is available for the request generator binding only.

    If this token generator might include a time stamp in the user name token, add the following name and value pair:

    **Name** `com.ibm.wsspi.wssecurity.token.username.addTimestamp`

    **Value** `true`

    This option is valid only when the generated token type is a user name token and it is available for the request generator binding only.

    If you have defined identity assertion in the IBM extended deployment descriptor, add the following name and value pair:

    **Name** `com.ibm.wsspi.wssecurity.token.IDAssertion.isUsed`

    **Value** `true`

    This option indicates that only the identity of the initial sender is required and inserted into the Web services security header within the SOAP message. For example, WebSphere Application Server only sends the user name of the original caller for a Username token generator. For an X.509 token generator, the application server sends the original signer certification only.

    If you have defined identity assertion in the IBM extended deployment descriptor and you want to use the Run As identity instead of the initial caller identity for identity assertion for a downstream call, add the following name and value pair:

**Name** `com.ibm.wsspi.wssecurity.token.IDAssertion.useRunAsIdentity`

**Value** `true`

This option is valid only when the generated token type is a user name token.

16. Click **OK** to save your configuration.

Configure the key information if this token generator configuration is for an X.509 security token. For more information, see "Configuring key information for the generator binding with an assembly tool" on page 596.

**Related concepts**

"Trust anchor" on page 587
A *trust anchor* specifies the key stores that contain trusted root certificates. These certificates are used to validate the X.509 certificate that is embedded in the Simple Object Access Protocol (SOAP) message.

**Related tasks**

"Configuring the security token in generator security constraints" on page 653

"Configuring key information for the generator binding with an assembly tool" on page 596

"Configuring the collection certificate store for the generator binding with an assembly tool" on page 586

***Key locator:***

A key locator or the com.ibm.wsspi.wssecurity.keyinfo.KeyLocator class, is an abstraction of the mechanism that retrieves the key for digital signature and encryption.

You can use any of the following infrastructure from which to retrieve the keys depending upon the implementation:

- Java keystore file
- Database
- Lightweight Third Party Authentication (LTPA) server

Key locators search for the key using some type of a clue. The following types of clues are supported:

- A string label of the key, which is explicitly passed through the application programming interface (API). The relationship between each key and its name (string label) is maintained inside the key locator.
- The implementation context of the key locator; explicit information is not passed to the key locator. A key locator determines the appropriate key according to the implementation context.

WebSphere Application Server Version 6 supports a secret key-based signature called HMAC-SHA1. If you use HMAC-SHA1, the Simple Object Access Protocol (SOAP) message does not contain a binary security token. In this case, it is assumed that the key information within the message contains the key name that is used to specify the secret key within the keystore.

Because the key locators support the public key-based signature, the key for verification is embedded in the X.509 certificate as a `<BinarySecurityToken>` element in the incoming message. For example, key locators can obtain the identity of the caller from the context and can retrieve the public key of the caller for response encryption.

This section describes the usage scenarios for key locators.

**Signing**

The name of the signing key is specified in the Web services security configuration. This value is passed to the key locator and the actual key is returned. The corresponding X.509 certificate also can be returned.

**Verification**

By default, WebSphere Application Server Version 6 supports the following types of key locators:

**KeyStoreKeyLocator**
Uses the keystore to retrieve the key that is used for digital signature and verification or encryption and decryption.

**X509CertKeyLocator**
Uses an X.509 certificate within a message to retrieve the key for verification or decryption.

**SignerCertKeyLocator**
Uses the X.509 certificate within the request message to retrieve the key that is used for encryption in the response message.

**Encryption**

The name of the encryption key is specified in the Web services security configuration. This value is passed to the key locator and the actual key is returned. On the server side, you can use the SignerCertKeyLocator to retrieve the key for encryption in the response message from the X.509 certificate in the request message.

**Decryption**

The Web services security specification recommends using the key identifier instead of the key name. However, while the algorithm for computing the identifier for the public keys is defined in Internet Engineering Task Force (IETF) Request for Comment (RFC) 3280, there is no agreed-upon algorithm for the secret keys. Therefore, the current implementation of Web services security uses the identifier only when public key-based encryption is performed. Otherwise, the ordinal key name is used.

When you use public key-based encryption, the value of the key identifier is embedded in the incoming encrypted message. Then, the Web services security implementation searches for all of the keys managed by the key locator and decrypts the message using the key whose identifier value matches the one in the message.

When you use secret key-based encryption, the value of the key name is embedded in the incoming encrypted message. The Web services security implementation asks the key locator for the key with the name that matches the name in the message and decrypts the message using the key.

**Related reference**

"Key collection" on page 687
Use this page to view a list of logical names that is mapped to a key alias in the keystore file.

"Key configuration settings" on page 688
Use this page to define the mapping of a logical name to a key alias in a keystore file.

*Keys:*

Keys are used for XML digital signature and encryption.

There are two predominant kinds of keys used in the current Web services security implementation:

- Public key - such as Rivest Shamir Adleman (RSA) encryption and Digital Signature Algorithm (DSA) encryption
- Secret key - such as triple-strength DES (3DES) encryption

In public key-based signature, a message is signed using the sender private key and is verified using the sender public key. In public key-based encryption, a message is encrypted using the receiver public key and is decrypted using the receiver private key. In secret key-based signature and encryption, the same key is used by both parties.

While the current implementation of Web services security can support both kinds of keys, the format of the message differs slightly between public key-based encryption and secret key-based encryption.

***Configuring key locators for the generator binding with an assembly tool:***

Prior to completing this task, you must import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF. Before configuring a key locator, you should know which key information configuration will reference this key locator. For example, if you configure this key locator for the STRREF key information type, select the com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator key locator class.

WebSphere Application Server Version 6 provides default key locator implementations that you can choose or you can write your own implementation. Custom key locators must implement the com.ibm.wsspi.wssecurity.keyinfo.KeyLocator interface. Using this implementation, you can locate keys within any data source.

Complete the following steps to configure a key locator for the generator using an assembly tool. The purpose of the key locators is to retrieve keys from the keystore for digital signature and encryption. The request generator is configured for the client and the response generator is configured for the server. In the following steps, you must configure either the client-side bindings in step 2 or the server-side bindings in step 3.

1. Start the assembly tool and click **Window > Open Perspective > J2EE**.
2. **Optional:** Locate the client-side bindings using the Project Explorer window. The Client Deployment Descriptor window is displayed. This Web service contains the bindings that you need to configure.
   a. Expand the Web Services > Client section and double-click the name of the Web service.
   b. Click the **WS Binding** tab and expand the Security Request Generator Binding Configuration section.
3. **Optional:** Locate the server-side bindings using the Project Explorer window. The Web Services Editor window is displayed. This Web service contains the bindings that you need to configure.
   a. Expand the Web Services > Services section and double-click the name of the Web service.
   b. Click the **Binding Configurations** tab and expand the Response Generator Binding Configuration Details section.
4. Expand the Key locators section and click **Add** to add a new entry or click **Edit** to edit a selected entry.
5. Specify a name for this configuration in the Key locator name field. This configuration name is referenced in the Key locator field of the Key Information dialog.
6. Select a key locator implementation in the Key locator class field. Select the key locator class that matches the Key Information configuration that references this key locator. The following default key locator class implementations are supported for version 6 applications:

   **`com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator`**
   This implementation locates and obtains the key from the specified keystore file.

   **`com.ibm.wsspi.wssecurity.keyinfo.SignerCertKeyLocator`**
   This implementation uses the public key from the certificate of the signer. This class implementation is used by the response generator.

**com.ibm.wsspi.wssecurity.keyinfo.X509TokenKeyLocator**
This implementation uses the X.509 security token from the sender message for digital signature validation and encryption. This class implementation is used by the request consumer and the response consumer.

7. Select the **Use key store** option if you need to configure a key store for this key locator. Whether you need to configure the key store information for a key locator depends upon the key locator class and your application configuration. For example, if you select the com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator key locator class in the previous step, configure the key store information for this key locator.
   a. Specify a keystore password in the Key store storepass field. The keystore storepass is the password that is required to access the keystore file.
   b. Specify the path to the keystore file in the Key store path field. The key store path is the directory where the keystore resides. Make sure that wherever you deploy your application that can locate your keystore file. Thus it is recommended that you use `${USER_INSTALL_ROOT}` in the path name as this variable expands to the WebSphere Application Server path on your machine.
   c. Select a key store type from the Key store type field. The key store type that you select must match the keystore file that is specified in the Key store path field. The following keystore types are supported:

      **JKS** Use this option if you are not using Java Cryptography Extensions (JCE) and if your keystore file uses the Java Keystore (JKS) format.

      **JCEKS**
      Use this option if you are using Java Cryptography Extensions.

      **PKCS11**
      Use this format if your keystore uses the PKCS#11 file format. Keystores using this format might contain RSA keys on cryptographic hardware or might encrypt keys that use cryptographic hardware to ensure protection.

      **PKCS12**
      Use this option if your keystore uses the PKCS#12 file format.
8. Click **Add** under the Key field to add a key entry from the keystore file that you specified in the previous step. This key is used for signature generation or encryption. The key that you specify must match the key that is used for validation or decryption for the consumer.
   a. Specify an alias name for the key in the Alias field. The key alias is used by the key locator to find the key within the keystore file.
   b. Specify the password that is associated with the key in the Key pass field. This password is needed to access the key object within the keystore file.
   c. Specify the key name in the Key name field. For digital signatures, the key name is used in the signing information for the request generator or response generator to determine which key is used to digitally sign the message. For encryption, the key name is used to determine which key is used for encryption. You must specify a fully qualified, distinguished name for the key name. For example, you might specify `CN=Bob,O=IBM,C=US`.
9. Click **OK** to save the configuration.

After you configure the key locator and any token generator that you need to configure, you can configure the key information that references this key locator. For more information, see “Configuring key information for the generator binding with an assembly tool” on page 596.

**Related concepts**

“Key locator” on page 592
A key locator or the com.ibm.wsspi.wssecurity.keyinfo.KeyLocator class, is an abstraction of the mechanism that retrieves the key for digital signature and encryption.

**Related tasks**

“Configuring key information for the generator binding with an assembly tool” on page 596

***Configuring key information for the generator binding with an assembly tool:***

Prior to completing this task, you must complete the following steps:

1. Import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF.
2. Configure the key locator that is referenced by the key information configuration. For more information, see “Configuring key locators for the generator binding with an assembly tool” on page 594.
3. Configure the token generator that is referenced by the key information configuration. For more information, see “Configuring token generators with an assembly tool” on page 587

Complete the following steps to configure the key information for the server-side and client-side bindings using an assembly tool. This key information is used to specify the configuration that is needed to generate the key for digital signature and encryption. The signing information and encryption information configurations can share the key information. The key information on the consumer side is used for specifying the information about the key that is used for validating the digital signature in the received message or for decrypting the encrypted parts of the message. The request generator is configured for the client and the response generation is configured for the server. In the following steps, you must configure either the client-side bindings in step 2 or the server-side bindings in step 3.

1. Start the assembly tool and click **Window > Open Perspective > J2EE**.
2. **Optional:** Locate the client-side bindings using the Project Explorer window. The Client Deployment Descriptor window is displayed. This Web service contains the bindings that you need to configure.
   a. Expand the Web Services > Client section and double-click the name of the Web service.
   b. Click the **WS Binding** tab and expand the Security Request Generator Binding Configuration section.
3. **Optional:** Locate the server-side bindings using the Project Explorer window. The Web Services Editor window is displayed. This Web service contains the bindings that you need to configure.
   a. Expand the Web Services > Services section and double-click the name of the Web service.
   b. Click the **Binding Configurations** tab and expand the Response Generator Binding Configuration Details section.
4. Expand the Key Information section and click **Add** to add a new entry or click **Edit** to edit a selected entry.
5. Specify a unique name for this configuration in the Key information name field. For example, you might specify `gen_signkeyinfo`. This configuration name is referenced by the Key information element within the Signing Information and Encryption Information dialog windows. For more information, see “Configuring signing information for the generator binding with an assembly tool” on page 598 and “Configuring encryption information for the generator binding with an assembly tool” on page 648.
6. Select a key information type from the Key information type field. The key information types specify different mechanisms for referencing security tokens. The assembly tools support the following key information types:

   **STRREF**
   This type is the security token reference. The security token is directly referenced using Universal Resource Identifiers (URIs). The following `<KeyInfo>` element is generated in the SOAP message for this key information type:

   ```
   <ds:KeyInfo>
      <wsse:SecurityTokenReference>
         <wsse:Reference URI="#mytoken" />
      </wsse:SecurityTokenReference>
   </ds:KeyInfo>
   ```

   **EMB** This type is the embedded token. The security token is directly embedded within the `<SecurityTokenReference>` element. The following `<KeyInfo>` element is generated in the SOAP message for this key information type:

```
<ds:KeyInfo>
   <wsse:SecurityTokenReference>
      <wsse:Embedded wsu:Id="tok1" />
         ...
      </wsse:Embedded>
   </wsse:SecurityTokenReference>
</ds:KeyInfo>
```

**KEYID**

This type is a key identifier. The security token is referenced using an opaque value that uniquely identifies the token. The algorithm that is used for generating the key identifier value depends upon the token type. The following `<KeyInfo>` element is generated in the Simple Object Access Protocol (SOAP) message for this key information type:

```
<ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
   <wsse:SecurityTokenReference>
      <wsse:KeyIdentifier ValueType="wsse:X509v3">/62wXO...</wsse:KeyIdentifier>
   </wsse:SecurityTokenReference>
</ds:KeyInfo>
```

**KEYNAME**

This type is the key name. The security token is referenced using a name that matches an asserted identity within the token. The following `<KeyInfo>` element is generated in the SOAP message for this key information type:

```
<ds:KeyInfo>
   <ds:KeyName>CN=Group1</ds:KeyName>
</ds:KeyInfo>
```

**X509ISSUER**

This type is the X.509 certificate issuer name and serial number. The security token is referenced by an issuer name and issuer serial number of an X.509 certificate. The following `<KeyInfo>` element is generated in the SOAP message for this key information type:

```
<ds:KeyInfo>
   <wsse:SecurityTokenReference>
      <ds:X509Data>
         <ds:X509IssuerSerial>
            <ds:X509IssuerName>CN=Jones, O=IBM, C=US</ds:X509IssuerName>
            <ds:X509SerialNumber>1040152879</ds:X509SerialNumber>
         </ds:X509IssuerSerial>
      </ds:X509Data>
   </wsse:SecurityTokenReference>
</ds:KeyInfo>
```

7. Select the **Use key locator** option.
   a. Select the name of a key locator configuration from the Key locator field. The value of this field is a reference to a key locator that specifies how to find keys or certificates. For more information, see “Configuring key locators for the generator binding with an assembly tool” on page 594.
   b. Specify a key name in the Key name field. The value is the name of a key that is used for generating the digital signature and for encryption. The list of key names come from the key locator that you specified previously.
8. **Optional:** Select the **Use token** option and a token generator configuration in the Token field if a token generator is required for the key information configuration. The token that you select specifies a reference to a token generator that is used for processing the security token within the message. Before you specify a token reference, you must configure a token generator. For more information on token generator configurations, see “Configuring token generators with an assembly tool” on page 587.

After completing this task, configure the signing information or encryption information that references the key information that is specified by this task. For more information, see “Configuring signing information for the generator binding with an assembly tool” on page 598 or “Configuring encryption information for the generator binding with an assembly tool” on page 648.

**Related tasks**

"Configuring key locators for the generator binding with an assembly tool" on page 594
"Configuring token generators with an assembly tool" on page 587
"Configuring signing information for the generator binding with an assembly tool"
"Configuring encryption information for the generator binding with an assembly tool" on page 648

***Configuring signing information for the generator binding with an assembly tool:***

Prior to completing this task, you must complete the following steps:

1. Import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF.
2. Specify which message parts to digitally sign. For more information, see "Signing message elements in generator security constraints with keywords" on page 571 or "Signing message elements in generator security constraints with an XPath expression" on page 579.
3. Configure the key information that is referenced by the Key information element within the Signing information dialog window. For more information, see "Configuring key information for the generator binding with an assembly tool" on page 596.

Complete the following steps to configure the signing information for the server-side and client-side bindings using an assembly tool. The request generator is configured for the client and the response generator is configured for the server. In the following steps, you must configure either the client-side bindings in step 2 or the server-side bindings in step 3.

1. Start the assembly tool and click **Window > Open Perspective > J2EE**.
2. **Optional:** Locate the client-side bindings using the Project Explorer window. The Client Deployment Descriptor window is displayed. This Web service contains the bindings that you need to configure.
   a. Expand the Web Services > Client section and double-click the name of the Web service.
   b. Click the **WS Binding** tab and expand the Security Request Generator Binding Configuration section.
3. **Optional:** Locate the server-side bindings using the Project Explorer window. The Web Services Editor window is displayed. This Web service contains the bindings that you need to configure.
   a. Expand the Web Services > Services section and double-click the name of the Web service.
   b. Click the **Binding Configurations** tab and expand the Response Generator Binding Configuration Details section.
4. Expand the Signing Information section and click **Add** to add a new entry or select an existing entry and click **Edit**. The Signing Information dialog window is displayed.
   a. Specify a name for the signing information configuration in the Signing information name field. For example, you might specify `gen_signinfo`.
   b. Select a canonicalization method from the Canonicalization method algorithm field. The canonicalization method algorithm is used to canonicalize the signing information before it is digested as part of the signature operation. The following pre-configured algorithms are supported:
      - http://www.w3.org/2001/10/xml-exc-c14n#
      - http://www.w3.org/2001/10/xml-exc-c14n#WithComments
      - http://www.w3.org/TR/2001/REC-xml-c14n-20010315
      - http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments

      You must specify the same canonicalization algorithm for both the generator and the consumer. For more information on configuring the signing information for the consumer, see "Configuring signing information for the consumer binding with an assembly tool" on page 620.
   c. Select a signature method algorithm from the Signature method algorithm field. The following pre-configured algorithms are supported:
      - http://www.w3.org/2000/09/xmldsig#rsa-sha1
      - http://www.w3.org/2000/09/xmldsig#dsa-sha1

- http://www.w3.org/2000/09/xmldsig#hmac-sha1

You must specify the same canonicalization algorithm for both the generator and the consumer. For more information on configuring the signing information for the consumer, see "Configuring signing information for the consumer binding with an assembly tool" on page 620.

5. Click **Add** in the Signing Key Information section to add a new key information entry or click **Remove** to delete a selected entry. Complete the following substeps if you are adding a new key information entry.
   a. Specify a name in the Key information name field. For example, you might specify `gen_skeyinfo`.
   b. Select a key information reference from the list under the Key information element field. The value in this field references the key information configuration that you specified previously. If you have a key information configuration called `gen_signkeyinfo` that you want to use with this signing information configuration, specify `gen_signkeyinfo` in the Key information element field. For more information, see "Configuring key information for the generator binding with an assembly tool" on page 596.
   c. Optional: Select the **Use key information signature** option if you want to sign the key information within the Simple Object Access Protocol (SOAP) message.
   d. Optional: Select a key information signature type from the Type field if you select the **Use key information signature** option. Select the **keyinfo** value to specify that the entire KeyInfo element must be signed within the SOAP message. Select the **keyinfochildelements** value to specify that the child elements within the KeyInfo element must be signed, but the KeyInfo element itself does not need to be signed.
6. Click **OK** to save your signing information configuration.
7. Expand the Part References subsection and select the signing information configuration from the Signing Information section.
8. Click **Add** in the Part References subsection to add a new entry or select an existing entry and click **Edit**. The Part References dialog window is displayed.
   a. Specify a name for the part reference configuration in the Part reference name field.
   b. Select a integrity part configuration in the Integrity part field. For more information on how to configure the integrity part, see "Signing message elements in generator security constraints with keywords" on page 571 or "Signing message elements in generator security constraints with an XPath expression" on page 579.
   c. Select the http://www.w3.org/2000/09/xmldsig#sha1 digest method algorithm in the Digest method algorithm field. This digest method algorithm is used to create the digest for each message part that is specified by this part reference.
   d. Expand the Transforms subsection and the part reference configuration from the Part reference subsection.
   e. Click **Add** in the Transforms subsection to add a new entry or select an existing entry and click **Edit**. The Transform dialog window is displayed.
   f. Specify a transform name in the Name field. For example, you might specify `reqint_body_transform1`.
   g. Select a transform algorithm from the Algorithm field. The following transform algorithms are supported:

      **http://www.w3.org/2001/10/xml-exc-c14n#**
      This algorithm specifies the World Wide Web Consortium (W3C) Exclusive Canonicalization recommendation.

      **http://www.w3.org/TR/1999/REC-xpath-19991116**
      This algorithm specifies the W3C XML path language recommendation. If you specify this algorithm, you must specify the property name and value by clicking **Properties**, which is displayed under Additional properties. For example, you might specify the following information:

**Property**
: `com.ibm.wsspi.wssecurity.dsig.XPathExpression`

**Value**
: `not(ancestor-or-self::*[namespace-uri()='http://www.w3.org/2000/09/xmldsig#' and local-name()='Signature'])`

**http://www.w3.org/2002/06/xmldsig-filter2**

This algorithm specifies the XML-Signature XPath Filter Version 2.0 proposed recommendation.

When you use this algorithm, you must specify a set of properties in the Transform property fields. You can use multiple property sets for the XPath Filter Version 2. Thus, it is recommended that your property names end with the number of the property set, which is denoted by an asterisk in the following examples:

- To specify an XPath expression for the XPath filter2, you might use:

  ```
  name com.ibm.wsspi.wssecurity.dsig.XPath2Expression_*
  ```

- To specify a filter type for each XPath, you might use:

  ```
  name com.ibm.wsspi.wssecurity.dsig.XPath2Filter_*
  ```

  Following this expression, you can have a value, `[intersect]`, `[subtract]`, or `[union]`.

- To specify the processing order for each XPath, you might use:

  ```
  name com.ibm.wsspi.wssecurity.dsig.XPath2Order_*
  ```

  Following this expression, indicate the processing order of the XPath.

The following is a list of complete examples:

```
com.ibm.wsspi.wssecurity.dsign.XPath2Filter_1 = [intersect]
com.ibm.wsspi.wssecurity.dsign.XPath2Order_1 = [1]
com.ibm.wsspi.wssecurity.dsign.XPath2Expression_2 = [XPath expression#2]
com.ibm.wsspi.wssecurity.dsign.XPath2Filter_2 = [subtract]
com.ibm.wsspi.wssecurity.dsign.XPath2Filter_2 = [1]
```

**http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform**

**http://www.w3.org/2002/07/decrypt#XML**

This algorithm specifies the W3C decryption transform for XML Signature recommendation.

**http://www.w3.org/2000/09/xmldsig#enveloped-signature**

This algorithm specifies the W3C recommendation for XML digital signatures.

The transform algorithm that you select for the generator must match the transform algorithm for the consumer.

After you complete this task for the generator binding, you must configure the signing information for consumer binding.

**Related tasks**

"Signing message elements in generator security constraints with keywords" on page 571
"Signing message elements in generator security constraints with an XPath expression" on page 579
"Configuring key information for the generator binding with an assembly tool" on page 596
"Configuring signing information for the consumer binding with an assembly tool" on page 620

***Signing message elements in consumer security constraints with keywords:***

Prior to completing this task, you must import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF.

Complete the following steps to specify which message parts or elements must be signed when you configure the consumer security constraints for either the response consumer or the request consumer. The response consumer is configured for the client and the request consumer is configured for the server. If the required parts are not signed, the request or response is rejected and a Simple Object Access Protocol (SOAP) fault is returned to the caller. In the following steps, you must configure either the client-side extensions in step 2 or the server-side extensions in step 3.

1. Start the assembly tool and click **Window > Open Perspective > J2EE**.
2. **Optional:** Locate the client-side extensions using the Project Explorer window. The Client Deployment Descriptor window is displayed. This Web service contains the extensions that you need to configure.
   a. Expand the Web Services > Client section and double-click the name of the Web service.
   b. Click the **WS Extension** tab and expand the Response Consumer Configuration section.
3. **Optional:** Locate the server-side extensions using the Project Explorer window. The Web Services Editor window is displayed. This Web service contains the extensions that you need to configure.
   a. Expand the Web Services > Services section and double-click the name of the Web service.
   b. Click the **Extensions** tab and expand the Request Consumer Service Configuration Details section.
4. Expand the Required Integrity section. Integrity refers to digital signature while confidentiality refers to encryption. Integrity decreases the risk of data modification when you transmit data across a network. For more information on digitally signing Simple Object Access Protocol (SOAP) messages, see “XML digital signature” on page 570.
5. Click **Add** to indicate which message parts or elements the consumer expects to be signed. The Required Integrity Dialog window is displayed.
   a. Specify a name for the integrity element under Required Integrity Name field.
   b. Specify a usage type in the Usage type field. This field specifies the requirement for the integrity element. The following options are available:

      **Required**
         If you select **Required** and the required message parts or elements are not signed, then the message is rejected with SOAP fault.

      **Optional**
         If you select **Optional**, then the digital signature of the selected message parts or elements is verified if they are signed. However, the consumer does not reject the message if the selected message parts or elements are not signed.
6. Click **Add** under the Message Parts section and select the Message parts dialect. The http://www.ibm.com/websphere/webservices/wssecurity/dialect-was dialect specifies which message parts or elements are expected to be signed using keywords. If you select this dialect, you can select one of the following keywords under the Message parts keyword heading:

   **body** Specifies the user data portion of the message. If you select this keyword, the body is checked to see if it is signed.

   **timestamp**
      Specifies that the stand-alone timestamp element within the message is checked for a digital signature. The timestamp element determines whether the message is valid based upon the time that the message is sent and then received. If the timestamp option is selected, make sure that there is a stand-alone timestamp element in the message. If the element does not exist, see “Adding a stand-alone time stamp in consumer security constraints” on page 651.

   **securitytoken**
      Specifies that the security token authenticates the client. If this keyword is selected, the

security token or tokens in the SOAP message are checked to determine if they are signed. For example, if you are sending a UsernameToken element within the message, you can specify that it is signed using this keyword.

**dsigkey**
Specifies that the key information element, which is used for digital signature, is checked to determine if it is signed.

**enckey**
Specifies that the key information element, which is used for encryption, is checked to determine if it is signed.

**messageid**
Specifies that the `<wsa:MessageID>` element within the message is checked to determine if it is signed.

**to** Specifies that the `<wsa:To>` element within the message is checked to determine if it is signed.

**action** Specifies that the `<wsa:Action>` element is checked to determine if it is signed.

**relatesto**
Specifies that the `<wsa:RelatesTo>` element within the message is checked to determine if it is signed.

7. Click **OK** to save the configuration changes.

**Note:** These configurations for the consumer and the generator must match.

In addition to specifying the message parts or elements that are expected to be signed, you also can specify that nonce and timestamp elements are expected to be included in the signed elements. For more information, see the following articles:

- "Adding time stamps for integrity in consumer security constraints with keywords"
- "Adding time stamps for integrity in consumer security constraints with an XPath expression" on page 608
- "Adding a nonce for integrity in consumer security constraints with keywords" on page 604
- "Adding a nonce for integrity in consumer security constraints with an XPath expression" on page 610

After you specify which message parts to check for a digital signature, you must specify which signature algorithm is used to validate the signature. For more information, see "Configuring signing information for the consumer binding with an assembly tool" on page 620.

**Related concepts**

"XML digital signature" on page 570
XML-Signature Syntax and Processing (XML digital signature) is a specification that defines XML syntax and processing rules to sign and verify digital signatures for digital content. The specification was developed jointly by the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF).

**Related tasks**

"Adding a stand-alone time stamp in consumer security constraints" on page 651

"Adding time stamps for integrity in consumer security constraints with an XPath expression" on page 608

"Adding a nonce for integrity in consumer security constraints with an XPath expression" on page 610

"Configuring signing information for the consumer binding with an assembly tool" on page 620

*Adding time stamps for integrity in consumer security constraints with keywords:*

Prior to completing this task, you must import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF.

This task is used to specify that when a time stamp is embedded in a particular element, the parent of the time stamp is expected to be signed with the message parts. Complete the following steps to specify that the parent element of the time stamp is expected in the element. Also, the time stamp is included in the signature for the message parts. Configure the consumer security constraints for either the response consumer or the request consumer. The response consumer is configured for the client and the request consumer is configured for the server. In the following steps, you must configure either the client-side extensions in step 2 or the server-side extensions in step 3.

1. Start the assembly tool and click **Window > Open Perspective > J2EE**.
2. **Optional:** Locate the client-side extensions using the Project Explorer window. The Client Deployment Descriptor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the client-side extensions:
   a. Expand the Web Services > Client section and double-click the name of the Web service.
   b. Click the **WS Extension** tab and expand the Response Consumer Configuration section.
3. **Optional:** Locate the server-side extensions using the Project Explorer window. The Web Services Editor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the server-side extensions:
   a. Expand the Web Services > Services section and double-click the name of the Web service.
   b. Click the **Extensions** tab and expand the Request Consumer Service Configuration Details section.
4. Expand the Required Integrity section. Integrity refers to digital signature while confidentiality refers to encryption. Integrity decreases the risk of data modification when you transmit data across a network. For more information on digitally signing Simple Object Access Protocol (SOAP) messages, see “XML digital signature” on page 570.
5. Click **Add** to specify a time stamp that is expected in the parent element of the keyword. The parent element of the time stamp is also expected to be included in the signature for the message part. The Required Integrity Dialog window is displayed. Before you configure the time stamp in the Required Integrity, you must configure at least one message part or element that is expected to be signed. Complete the following steps to specify a configuration:
   a. Specify a name for the integrity element in the Required Integrity Name field.
   b. Specify a usage type in the Usage type field. This field specifies the requirement for the integrity element. The value of this attribute is either Required or Optional. The following options are available:

      **Required**
         If you select **Required** and the required message parts or elements are not signed, then the message is rejected with SOAP fault.

      **Optional**
         If you select **Optional**, then the digital signature of the selected message parts or elements is verified if they are signed. However, the consumer does not reject the message if the selected message parts or elements are not signed.
6. In the Timestamp section, click **Add** and select the Timestamp dialect. The http://www.ibm.com/websphere/webservices/wssecurity/dialect-was dialect specifies the parent element of the expected time stamp. If you select this dialect, you can select one of the following keywords under the Timestamp keyword heading:

   **body** Specifies the user data portion of the message. If you select the body option, a time stamp is embedded in Simple Object Access Protocol (SOAP) body. Also, the parent of the time stamp (SOAP body) is expected to be signed with the message parts in the Required Integrity.

**securitytoken**
: Specifies that a time stamp is expected to be embedded in the security token element. Also, the parent of the time stamp (security token) is expected to be signed with the message parts in the Required Integrity.

**dsigkey**
: Specifies that the time stamp is inserted into the key information element, which is used for digital signature, and the key information element is signed.

**enckey**
: Specifies that the time stamp is inserted into the key information element, which is used for encryption, and the key information element is signed.

**messageid**
: Specifies that the time stamp is inserted into the `<wsa:MessageID>` element and the `<wsa:MessageID>` element is signed.

**to** Specifies that the time stamp is inserted into the `<wsa:To>` element within the message and that the `<wsa:To>` element is signed.

**action** Specifies that the `<wsa:Action>` element is signed.

**relatesto**
: Specifies that the times tamp is inserted into the `<wsa:RelatesTo>` element within the message and the `<wsa:RelatesTo>` element is signed.

7. If you have not defined a message part for Required Integrity, you must define at least one message part to add a time stamp for Required Integrity. Complete the following steps to define a message part:
   a. In the Message Parts section, click **Add** and select http://www.ibm.com/websphere/webservices/wssecurity/dialect-was in the Message parts dialect field.
   b. In the Message Parts section, select the message parts keyword.
   c. Click **OK** to save the configuration changes.

   **Note:** These configurations for the consumer and the generator must match.

   In addition to the time stamp, you can specify that the nonce is signed. For more information, see the following articles:
   - "Adding a nonce for integrity in consumer security constraints with keywords"
   - "Adding a nonce for integrity in consumer security constraints with an XPath expression" on page 610

After you have specified which message parts to digitally sign, you must specify which method is used to digitally sign the message. For more information, see "Configuring signing information for the consumer binding with an assembly tool" on page 620.

**Related tasks**

"Adding a stand-alone time stamp in consumer security constraints" on page 651

"Adding a nonce for integrity in consumer security constraints with keywords"

"Adding a nonce for integrity in consumer security constraints with an XPath expression" on page 610

"Configuring signing information for the consumer binding with an assembly tool" on page 620

*Adding a nonce for integrity in consumer security constraints with keywords:*

Prior to completing this task, you must import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF.

Nonce for integrity is used to specify that a nonce is embedded in a particular element. The parent element of the nonce is signed with the message parts in the required integrity. Complete the following steps to specify a nonce for integrity using keywords when you configure the consumer security constraints for either the response consumer or the request consumer. The response consumer is configured for the client and the request consumer is configured for the server. In the following steps, you must configure either the client-side bindings in step 2 or the server-side bindings in step 3.

1. Start the assembly tool and click **Window > Open Perspective > J2EE**.
2. **Optional:** Locate the client-side bindings using the Project Explorer window. The Client Deployment Descriptor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the client-side bindings:
   a. Expand the Web Services > Client section and double-click the name of the Web service.
   b. Click the **WS Extension** tab and expand the Response Consumer Configuration section.
3. **Optional:** Locate the server-side bindings using the Project Explorer window. The Web Services Editor window is displayed. This Web service contains the bindings that you need to configure. Complete the following steps to locate the server-side bindings:
   a. Expand the Web Services > Services section and double-click the name of the Web service.
   b. Click the **Extensions** tab and expand the Request Consumer Service Configuration Details section.
4. Expand the Required Integrity section. Integrity refers to digital signature while confidentiality refers to encryption. Integrity decreases the risk of data modification when you transmit data across a network. For more information on digitally signing Simple Object Access Protocol (SOAP) messages, see “XML digital signature” on page 570.
5. Click **Add** to specify a nonce for integrity. The Required Integrity Dialog window is displayed. Complete the following steps to specify a configuration:
   a. Specify a name for the integrity element in the Required Integrity Name field.
   b. Specify a usage type in the Usage type field. This field specifies the requirement for the integrity element. The value of this attribute is either Required or Optional.

      **Required**
      If you select **Required** and the required message parts or elements are not signed, then the consumer rejects the message and issues a SOAP fault.

      **Optional**
      If you select **Optional** and the message parts or elements are signed, then the digital signature is verified. However, the consumer does not reject the message if the selected message parts or elements are not signed.
6. Under Nonce, click **Add** and select the nonce dialect. The http://www.ibm.com/websphere/webservices/wssecurity/dialect-was dialect specifies the message part to which a nonce is added as a child element and signed with the message parts in the required integrity. If you select this dialect, you can select one of the following keywords under Nonce keyword:

   **body** Specifies the user data portion of the message. If this option is selected, a nonce is embedded in the Simple Object Access Protocol (SOAP) body element. Also, the parent of the nonce (SOAP body) is expected to be signed with the message parts in the required integrity.

   **timestamp**
   Specifies that the nonce is embedded in the stand-alone timestamp element within the message. Also, the parent of the nonce (timestamp) is expected to be signed with the message parts in the required integrity. If timestamp keyword is selected, make sure that there is a standalone timestamp element in the message. If not, refer to the “Adding a stand-alone time stamp in consumer security constraints” on page 651 article.

**securitytoken**
: Specifies that the nonce element is expected to exist within the security token element. Also, the parent of the nonce (securitytoken) is expected to be signed with the message parts in the required integrity.

**dsigkey**
: Specifies that the nonce is inserted into the key information element, which is used for digital signature, and the key information element is signed.

**enckey**
: Specifies that the nonce is inserted into the key information element, which is used for encryption, and the key information element is signed.

**messageid**
: Specifies that the nonce is inserted into the `<wsa:MessageID>` element and that the `<wsa:MessageID>` element is signed.

**to**
: Specifies that the nonce is inserted into the `<wsa:To>` element within the message and that the `<wsa:To>` element is signed.

**action**
: Specifies that the `<wsa:Action>` element is signed.

**relatesto**
: Specifies that the nonce is inserted into the `<wsa:RelatesTo>` element within the message and that the `<wsa:RelatesTo>` element is signed.

7. If you have not previously specified message part in required integrity, click **Add** in the Message Parts section to add the message parts. You must define at least one message part in required integrity to specify a nonce in required integrity.
8. In the Message Parts section, select the message parts keyword.
9. Click **OK** to save the configuration changes.

   **Note:** These configurations on the consumer side and the generator side must match.

   In addition to the nonce, you can specify that the timestamp element is signed. For more information, see the following articles:

   - “Adding time stamps for integrity in consumer security constraints with keywords” on page 602
   - “Adding time stamps for integrity in consumer security constraints with an XPath expression” on page 608

After you specify which message parts to digitally sign, you must specify which method is used to digitally sign the message. For more information, see “Configuring signing information for the consumer binding with an assembly tool” on page 620.

**Related concepts**

“XML digital signature” on page 570
XML-Signature Syntax and Processing (XML digital signature) is a specification that defines XML syntax and processing rules to sign and verify digital signatures for digital content. The specification was developed jointly by the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF).

**Related tasks**

“Adding a stand-alone time stamp in consumer security constraints” on page 651

“Adding time stamps for integrity in consumer security constraints with keywords” on page 602

“Adding time stamps for integrity in consumer security constraints with an XPath expression” on page 608

“Adding a nonce for integrity in consumer security constraints with an XPath expression” on page 610

“Configuring signing information for the consumer binding with an assembly tool” on page 620

***Signing message elements in consumer security constraints with an XPath expression:***

Prior to completing this task, you must import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF.

The following information explains the difference between using an XPath expression and using keywords to specify which part of the message to sign:

**XPath expression**
: Specify any part of the message using an XPath expression. XPath is a language that is used to address parts of an XML document. You can find information on XPath syntax at the following Web site: http://www.w3.org/TR/1999/REC-xpath-19991116.

**Keywords**
: Specify only elements within the message using predefined keywords.

Complete the following steps to specify which message parts are expected to be signed using an XPath expression. In the following steps, you must configure either the client-side extensions in step 2 or the server-side extensions in step 3.

1. Start the assembly tool and click **Window > Open Perspective > J2EE**.
2. **Optional:** Locate the client-side extensions using the Project Explorer window. The Client Deployment Descriptor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the client-side extensions:
   a. Expand the Web Services > Client section and double-click the name of the Web service.
   b. Click the **WS Extension** tab and expand the Response Consumer Configuration section.
3. **Optional:** Locate the server-side extensions using the Project Explorer window. The Web Services Editor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the server-side extensions:
   a. Expand the Web Services > Services section and double-click the name of the Web service.
   b. Click the **Extensions** tab and expand the Request Consumer Service Configuration Details section.
4. Expand the Required Integrity section. Integrity refers to digital signature while confidentiality refers to encryption. Integrity decreases the risk of data modification when you transmit data across a network. For more information on digitally signing Simple Object Access Protocol (SOAP) messages, see “XML digital signature” on page 570.
5. Click **Add** to indicate which message parts to validate for digital signature. The Required Integrity Dialog window is displayed. Complete the following steps to specify a configuration:
   a. Specify a name for the integrity element under Required Integrity Name field.
   b. Specify a usage type in the Usage type field. This field specifies the requirement for the integrity element. The value of this attribute is either Required or Optional.

      **Required**
      : If you select **Required** and the required message parts or elements are not signed, then the message is rejected with SOAP fault.

      **Optional**
      : If you select **Optional**, then the digital signature of the selected message parts or elements is verified if they are signed. However, the consumer does not reject the message if the selected message parts or elements are not signed.
6. Click **Add** under the Message Parts section of the Required Integrity Dialog window. Complete the following steps to specify the message parts dialect and its message part:
   a. Select the Message parts dialect from the Message Parts section of the Required Integrity Dialog window. If you select the http://www.w3.org/TR/1999/REC-xpath-19991116 dialect, the message part that is validated for digital signature is specified by the XPath expression.

b. Specify the message part to be validated for digital signature using an XPath expression in the Message parts keyword field. For example, to specify that the message body is checked to determine if it is signed, you might add the following expression in the Message parts keyword field as one continuous line:

```
/*[namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/' and local-name()='Envelope']/*
[namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/' and local-name()='Body']
```

**Important:** Verify that your XPath syntax is correct.

7. Click **OK** to save the configuration changes.

**Note:** These configurations for the consumer and the generator must match.

In addition to the message parts, you also can specify that WebSphere Application Server check the nonce and timestamp elements for a digital signature. For more information, see the following articles:

- “Adding time stamps for integrity in consumer security constraints with keywords” on page 602
- “Adding time stamps for integrity in consumer security constraints with an XPath expression”
- “Adding a nonce for integrity in consumer security constraints with keywords” on page 604
- “Adding a nonce for integrity in consumer security constraints with an XPath expression” on page 610

After you specify which message parts to check for a digital signature, you must specify which method is used to validate the signature. For more information, see “Configuring signing information for the consumer binding with an assembly tool” on page 620.

**Related concepts**

“XML digital signature” on page 570
XML-Signature Syntax and Processing (XML digital signature) is a specification that defines XML syntax and processing rules to sign and verify digital signatures for digital content. The specification was developed jointly by the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF).

**Related tasks**

“Adding a stand-alone time stamp in consumer security constraints” on page 651

“Adding time stamps for integrity in consumer security constraints with keywords” on page 602

“Adding a nonce for integrity in consumer security constraints with keywords” on page 604

“Configuring signing information for the consumer binding with an assembly tool” on page 620

*Adding time stamps for integrity in consumer security constraints with an XPath expression:*

Prior to completing this task, you must import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF.

The following information explains the difference between using an XPath expression and using keywords to specify which part of the message to sign:

**XPath expression**
: Specify any part of the message using an XPath expression. XPath is a language that is used to address parts of an XML document. You can find information on XPath syntax at the following Web site: http://www.w3.org/TR/1999/REC-xpath-19991116.

**Keywords**
: Specify only elements within the message using predefined keywords.

This task is to specify that a time stamp is expected to be added in an element of the SOAP message that is specified by the XPath language syntax. The element is expected to be signed with the message part that is specified in the Required Integrity Dialog window. Complete the following steps to specify the time

stamp for integrity using an XPath expression when you configure the consumer security constraints for either the response consumer or the request consumer. The response consumer is configured for the client and the request consumer is configured for the server. In the following steps, you must configure either the client-side extensions in step 2 or the server-side extensions in step 3.

1. Start the assembly tool and click **Window > Open Perspective > J2EE**.
2. **Optional:** Locate the client-side extensions using the Project Explorer window. The Client Deployment Descriptor window is displayed. This Web service contains the extensions that you need to configure.
   a. Expand the Web Services > Client section and double-click the name of the Web service.
   b. Click the **WS Extension** tab and expand the Response Consumer Configuration section.
3. **Optional:** Locate the server-side extensions using the Project Explorer window. The Web Services Editor window is displayed. This Web service contains the extensions that you need to configure.
   a. Expand the Web Services > Services section and double-click the name of the Web service.
   b. Click the **Extensions** tab and expand the Request Consumer Service Configuration Details section.
4. Expand the Required Integrity section. Integrity refers to digital signature while confidentiality refers to encryption. Integrity decreases the risk of data modification when you transmit data across a network. For more information on digitally signing Simple Object Access Protocol (SOAP) messages, see “XML digital signature” on page 570.
5. Click **Add** to specify a time stamp for integrity. The Required Integrity Dialog window is displayed.
   a. Specify a name for the integrity element in the Required Integrity Name field.
   b. Specify a usage type in the Usage type field. This field specifies the requirement for the integrity element. The value of this attribute is either Required or Optional.
6. Click **Add** in the Timestamp section of the Required Integrity Dialog window.
   a. Select the Timestamp dialect from the Timestamp section. The http://www.w3.org/TR/1999/REC-xpath-19991116 dialect specifies the message part to which the time stamp is added and signed using the XPath expression.
   b. Select the message part in the Timestamp keyword field to which the time stamp is added and signed using an XPath expression. For example, to specify that the time stamp is added to the body and is signed, you might specify the following expression for the Timestamp keyword:

```
/*[namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/' and local-name()='Envelope']/*
[namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/' and local-name()='Body']
```

      **Important:** Verify that your XPath expression syntax is correct.
7. If you have not previously defined a message part in the Required Integrity Dialog window, click **Add** under the Message Parts section and define a message part.
8. In the Message Parts section, select the message parts keyword.
9. Click **OK** to save the configuration changes.

   **Note:** These configurations for the consumer and the generator must match.

   In addition to the time stamp, you can specify that the nonce is signed. For more information, see the following articles:
   - “Adding a nonce for integrity in consumer security constraints with keywords” on page 604
   - “Adding a nonce for integrity in consumer security constraints with an XPath expression” on page 610

   **Important:** You must define one message part in the required integrity if you want to use the time stamp feature for required integrity.

After you have specified which message parts to digitally sign, you must specify which method is used to digitally sign the message. For more information, see “Configuring signing information for the consumer binding with an assembly tool” on page 620.

**Related tasks**

“Adding a stand-alone time stamp in consumer security constraints” on page 651
“Adding a nonce for integrity in consumer security constraints with keywords” on page 604
“Adding a nonce for integrity in consumer security constraints with an XPath expression”
“Configuring signing information for the consumer binding with an assembly tool” on page 620

*Adding a nonce for integrity in consumer security constraints with an XPath expression:*

Prior to completing this task, you must import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF.

The following information explains the difference between using an XPath expression and using keywords to specify which part of the message to sign:

**XPath expression**
: Specify any part of the message using an XPath expression. XPath is a language that is used to address parts of an XML document. You can find information on XPath syntax at the following Web site: http://www.w3.org/TR/1999/REC-xpath-19991116.

**Keywords**
: Specify only elements within the message using predefined keywords.

Nonce for integrity is used to specify that a nonce is embedded in a particular element within the message and that the element is signed. Complete the following steps to specify a nonce for integrity using an XPath expression when you configure the consumer security constraints for either the response consumer or the request consumer. The response consumer is configured for the client and the request consumer is configured for the server. In the following steps, you must configure either the client-side bindings in step 2 or the server-side bindings in step 3.

1. Start the assembly tool and click **Window > Open Perspective > J2EE**.
2. **Optional:** Locate the client-side bindings using the Project Explorer window. The Client Deployment Descriptor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the client-side bindings:
   a. Expand the Web Services > Client section and double-click the name of the Web service.
   b. Click the **WS Extension** tab and expand the Response Consumer Configuration section.
3. **Optional:** Locate the server-side bindings using the Project Explorer window. The Web Services Editor window is displayed. This Web service contains the bindings that you need to configure. Complete the following steps to locate the server-side bindings:
   a. Expand the Web Services > Services section and double-click the name of the Web service.
   b. Click the **Extensions** tab and expand the Request Consumer Service Configuration Details section.
4. Expand the Required Integrity section. Integrity refers to digital signature while confidentiality refers to encryption. Integrity decreases the risk of data modification when you transmit data across a network. For more information on digitally signing Simple Object Access Protocol (SOAP) messages, see “XML digital signature” on page 570.
5. Click **Add** to specify a nonce for integrity. The Required Integrity Dialog window is displayed. Complete the following steps to specify a configuration:
   a. Specify a name for the integrity element in the Required Integrity Name field.
   b. Specify a usage type in the Usage type field. This field specifies the requirement for the integrity element. The value of this attribute is either Required or Optional.
6. Click **Add** in the Nonce section of the Required Integrity Dialog window. Complete the following steps to configure the nonce dialect and message part:

a. Select the Nonce dialect from the Nonce section. The http://www.w3.org/TR/1999/REC-xpath-19991116 dialect specifies the message part to which a nonce is added and signed using an XPath expression.
b. Select the message part in the Nonce keyword field to which a nonce is added and signed using an XPath expression. For example, to specify that a nonce is added to the body and that it is signed, you might specify the following expression for the Nonce keyword:

```
/*[namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/' and local-name()='Envelope']/*
[namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/' and local-name()='Body']
```

7. In the Message Parts section, click **Add** and select http://www.w3.org/TR/1999/REC-xpath-19991116 in the Message parts dialect field.
8. In the Message Parts section, select the message parts keyword.

   **Important:** You must select the same keyword in the Message parts keyword field as the keyword that you selected in the Nonce keyword field.
9. Click **OK** to save the configuration changes.

   **Note:** These configurations on the consumer side and the generator side must match.

   In addition to the nonce, you can specify that the timestamp element is signed. For more information, see the following articles:
   - "Adding time stamps for integrity in consumer security constraints with keywords" on page 602
   - "Adding time stamps for integrity in consumer security constraints with an XPath expression" on page 608

After you specify which message parts to digitally sign, you must specify which method is used to digitally sign the message. For more information, see "Configuring signing information for the consumer binding with an assembly tool" on page 620.

**Related concepts**

"XML digital signature" on page 570
XML-Signature Syntax and Processing (XML digital signature) is a specification that defines XML syntax and processing rules to sign and verify digital signatures for digital content. The specification was developed jointly by the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF).

**Related tasks**

"Adding a stand-alone time stamp in consumer security constraints" on page 651

"Adding time stamps for integrity in consumer security constraints with keywords" on page 602

"Adding time stamps for integrity in consumer security constraints with an XPath expression" on page 608

"Adding a nonce for integrity in consumer security constraints with keywords" on page 604

"Configuring signing information for the consumer binding with an assembly tool" on page 620

***Configuring the collection certificate store for the consumer binding with an assembly tool:***

Prior to completing this task, you must import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF.

This task describes the steps to specify the collection certificate store for the consumer bindings at the application level using an assembly tool. A collection certificate store is a collection of non-root certificate authority (CA) certificates and certificate revocation lists (CRLs) that is used for validating an X.509 certificate embedded within the received SOAP message. The response consumer is configured for the client and the request consumer is configured for the server. In the following steps, you must configure either the client-side bindings in step 2 or the server-side bindings in step 3.

1. Start the assembly tool and click **Window > Open Perspective > J2EE**.
2. **Optional:** Locate the client-side bindings using the Project Explorer window. The Client Deployment Descriptor window is displayed. This Web service contains the bindings that you need to configure. Complete the following steps to locate the client-side bindings:
   a. Expand the Web Services > Client section and double-click the name of the Web service.
   b. Click the **WS Binding** tab and expand the Security Response Consumer Binding Configuration section.
3. **Optional:** Locate the server-side bindings using the Project Explorer window. The Web Services Editor window is displayed. This Web service contains the bindings that you need to configure. Complete the following steps to locate the server-side bindings:
   a. Expand the Web Services > Services section and double-click the name of the Web service.
   b. Click the **Binding Configurations** tab and expand the Request Consumer Binding Configuration Details section.
4. Expand the Certificate Store List > Collection Certificate Store section and click **Add**.
5. Specify a unique certificate store name in the Name field. For example, specify `cert1`. The name of the collection certificate store must be unique on the level in which it is defined. For example, the name must be unique at the application level. The name specified in the certificate store name field is used by other configurations to refer to a predefined collection certificate store. WebSphere Application Server looks up the collection certificate store based on proximity. For example, if an application binding refers to certificate store cert1, WebSphere Application Server will look first for cert1 at the application level. If it is not found, it will look at the server level, and finally at the cell level.
6. Specify a certificate store provider in the Provider field. The `IBMCertPath` certificate path provider is supported. To use another certificate path provider, you must define the provider implementation in the provider list within the `java.security` file in the Software Development Kit (SDK).
7. Click **Add** under X509 Certificate to specify a fully qualified path to an X.509 certificate, click the name of an existing certificate path entry to edit it, or click **Remove** to delete it. This collection certificate store is used to validate the certificate path of the incoming X.509-formatted security tokens.

   You can use the `USER_INSTALL_ROOT` variable as part of the path name. For example you might specify `${USER_INSTALL_ROOT}/etc/ws-security/samples/intca2.cer`. However, do not use this X.509 certificate path for production use. Obtain your own X.509 certificate from a certificate authority before putting your WebSphere Application Server environment into production.

   In the WebSphere Application Server administrative console, you can click **Environment > WebSphere Variables** to configure the **USER_INSTALL_ROOT** variable.
8. Click **Add** under CRL to specify the fully qualified path to a certificate revocation list (CRL), click an existing CRL entry to edit it or click **Remove** to delete it.

   For portability reasons, it is recommended that you use the WebSphere Application Server variables to specify a relative path to the certificate revocation list. For example, you might use the `USER_INSTALL_ROOT` variable to define a path such as `${USER_INSTALL_ROOT}/mycertstore/mycrl`. For a list of the supported variables in the WebSphere Application Server administrative console, click **Environment > WebSphere Variables**.

   The following list provides recommendations for using CRLs:
   - If CRLs are added to the collection certificate store, add the CRLs for the root certificate authority and each intermediate certificate, if applicable. When the CRL is in the certificate collection store, the certificate revocation status for every certificate in the chain is checked against the CRL of the issuer.
   - When the CRL file is updated, the new CRL does not take effect until you restart the Web service application.
   - Before a CRL expires, you must load a new CRL into the certificate collection store to replace the old CRL. An expired CRL in the collection certificate store results in a certificate path (CertPath) build failure.
9. Click **OK** to save your configuration.

**Related tasks**

"Configuring token consumers with an assembly tool"

***Trusted ID evaluator:***

A *trusted ID evaluator* (com.ibm.wsspi.wssecurity.id.TrustedIDEvaluatorImpl) is an abstraction of the mechanism that evaluates whether the given ID name is trusted.

Depending upon the implementation, you can use various types of infrastructure to store a list of the trusted IDs, such as:

- Plain text file
- Database
- Lightweight Directory Access Protocol (LDAP) server

The trusted ID evaluator is typically used by the eventual receiver in a multi-hop environment. The Web services security implementation invokes the trusted ID evaluator and passes the identity name of the intermediary as a parameter. If the identity is evaluated and deemed trustworthy, the procedure continues. Otherwise, an exception is created and the procedure is stopped.

**Related reference**

"Trusted ID evaluator collection" on page 784
Use this page to view a list of trusted identity (ID) evaluators. The trusted ID evaluator determines whether to trust the identity-asserting authority. After the ID is trusted, WebSphere Application Server issues the proper credentials based on the identity, which are used in a downstream call for invoking resources. The trusted ID evaluator implements the com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator interface.

"Trusted ID evaluator configuration settings" on page 785
Use this information to configure trust identity (ID) evaluators.

***Configuring token consumers with an assembly tool:***

Prior to completing this task, you must complete the following steps:

- Import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF.
- Configure the security token in the extension file. For example, if you are configuring a token consumer for a Lightweight Third Party Authentication (LTPA) token, you must first configure the LTPA token under the Required Security Token section on the **Extensions** tab. For more information, see "Configuring the security token requirement in consumer security constraints" on page 653
- Configure a collection certificate store if the token consumer uses the PKCS#7 token type and you want to package the certificate revocation lists (CRL) in the security token. For more information, see "Configuring the collection certificate store for the consumer binding with an assembly tool" on page 611.

A security token represents a set of claims that are made by a client. This set of claims might include a name, password, identity, key, certificate, group, privilege, and so on. A security token is embedded in the Simple Object Access Protocol (SOAP) message within the SOAP header. The security token within the SOAP header is propagated from the message sender to the intended message receiver. On the receiving side, the security handler for WebSphere Application Server authenticates the security token and sets up the caller identity on the running thread.

Complete the following steps to configure a token consumer for either the client-side bindings in step 2 or the server-side bindings in step 3:

1. Start the assembly tool and click **Window > Open Perspective > J2EE**.

2. **Optional:** Locate the client-side bindings using the Project Explorer window. The Client Deployment Descriptor window is displayed. This Web service contains the bindings that you need to configure. Complete the following steps to locate the client-side bindings:
   a. Expand the Web Services > Client section and double-click the name of the Web service.
   b. Click the **WS Binding** tab and expand the Security Response Consumer Configuration section.
3. **Optional:** Locate the server-side bindings using the Project Explorer window. The Web Services Editor window is displayed. This Web service contains the bindings that you need to configure. Complete the following steps to locate the server-side bindings:
   a. Expand the Web Services > Services section and double-click the name of the Web service.
   b. Click the **Binding Configurations** tab and expand the Request Consumer Binding Configuration Details section.
4. **Optional:** Configure a trust anchor if you are configuring this token consumer for an X.509 security token. Complete the following steps to configure the trust anchors:
   a. Expand the Trust anchor section and click **Add** to add a new entry or click **Edit** to edit a selected entry. The Trust anchor dialog window is displayed.
   b. Specify a name for the trust anchor configuration in the Trust anchor name field.
   c. Specify a keystore password in the Key store storepass field. The keystore storepass is the password that is required to access the keystore file.
   d. Specify the path to the keystore file in the Key store path field. The key store path is the directory where the keystore resides. Make sure that wherever you deploy your application that the server can locate your keystore file.
   e. Select a key store type from the Key store type field. The key store type that you select must match the keystore file that is specified in the Key store path field.
   f. Click **OK** to save the trust anchor configuration.
5. Expand the Token Consumer section and click **Add** to add a new entry or click **Edit** to edit a selected entry. The Token Consumer Dialog window is displayed.
6. Specify a name in the Token consumer name field. If this token consumer is for an X.509 certificate and is used for signature validation or decryption, the token consumer name is referenced in the Token field of the Key Information dialog window.
7. Select a token consumer class in the Token consumer class field. Select the token consumer class that matches the type of token that you are configuring. For example, if you are configuring a token consumer that processes an X.509 security token in the received message, select the com.ibm.wsspi.wssecurity.token.X509TokenConsumer token consumer class.
8. Select a security token reference in the Security token field. The value in this field references the security token that is configured in the extensions file. If you are configuring this token consumer for an X.509 security token, where the token consumer class is com.ibm.wsspi.wssecurity.token.X509tokenConsumer, leave this field blank.
9. Select the **Use value type** option and select the value type in the Value type field. Select the value type of the security token that matches the type of token consumer that you are configuring. When you select the value type, the assembly tool automatically enters the correct values in the Local name and URI fields depending upon the type of security token that is specified by the value type.
10. **Optional:** Select the **Use jaas.config** option and specify a Java Authentication and Authorization Service (JAAS) configuration name in the jaas.config.name field if a JAAS configuration is required for the security token. The JAAS configuration name that you specify must be for the security token that is specified for this token consumer. The following table lists the JAAS configuration names for the different security tokens specified by the value type.

*Table 11. JAAS configuration names and the corresponding value type*

| jaas.config name | Value type |
| --- | --- |
| system.wssecurity.UsernameToken | Username Token |

*Table 11. JAAS configuration names and the corresponding value type (continued)*

| jaas.config name | Value type |
| --- | --- |
| system.wssecurity.IDAssertionUsernameToken | Username Token (for IDAssertion) |
| system.wssecurity.X509BST | X509 certificate token |
| system.wssecurity.PkiPath | X509 certificates in a PKIPath |
| system.wssecurity.PKCS7 | X509 certificates and CRLs in a PKCS#7 |

11. **Optional:** If a trusted ID evaluator is required for this token consumer, select either the **Use trusted ID evaluator** option to define a new trusted ID evaluator or select the **Use trusted ID evaluator reference** option to select an existing trusted ID evaluator that is defined in a default binding file. A trusted ID evaluator is typically used by the target Web service in a multi-hop environment to determine whether to trust the identity of the intermediary Web service. Complete the following steps if you select the **Use trusted ID evaluator** option:
    a. Specify a trusted ID evaluator implementation in the Trusted ID evaluator class field. The trusted ID evaluators are implemented by specifying a class that implements the com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator interface. WebSphere Application Server Version 6 provides the com.ibm.wsspi.wssecurity.id.TrustedIDEvaluatorImpl default implementation of a trusted ID evaluator.

       The implementation is initialized with a list of trusted identity names. The trusted identities are specified as trustedIDEvaluator properties in the binding file. When a name is evaluated, it is checked against a list of trusted identity names. If the name is in the list, it is trusted and if the name is not in the list, it is not trusted.

    b. Click **Add** under the Trusted ID evaluator property section to add a new entry or click **Remove** to delete a selected entry. Each property entry represents a trusted identity.
    c. Specify `trustedId_trustmode` in the Name field and the identity of the intermediary in the Value field.

    If you select the **Use trusted ID evaluator reference** option, specify the name of an existing Trusted ID evaluator in the Trusted ID evaluator reference field.
12. **Optional:** Click **Add** under Property to add a new property for this token consumer or click **Remove** to delete a selected property. If this token consumer needs to process a nonce and a time stamp that is contained in a username token, define the properties in the following table.

*Table 12. Nonce and time stamp properties*

| Name | Value |
| --- | --- |
| `com.ibm.wsspi.wssecurity.token.Username.verifyNonce` | `true` |
| `com.ibm.wsspi.wssecurity.token.Username.verifyTimestamp` | `true` |

13. **Optional:** Select the **Use certificate path settings** option if you are configuring this token consumer for an X.509 security token.
14. Select either the **Certificate path reference** option or the **Trust any certificate** option if you are configuring this token consumer for an X.509 security token.

    **Important:** When you configure a token consumer for an X.509 certificate token, use caution when you select the **Trust any certificate** option. This option might compromise the security of your Web service application by allowing the SOAP message to be signed or encrypted using any certificate. It is recommended that you use the trust anchor and certificate store list to validate the X.509 certificate embedded in the received SOAP message.

    If you select the **Certificate path reference** option, complete the following steps:

    a. Select a trust anchor reference from the list in the Trust anchor reference field. This reference is the name of the trust anchor that specifies the key store, which contains the trusted root certificate authority (CA) certificates.

b. Select a certificate store from the Certificate store reference field. A certificate store list contains both non-root CA certificates (or intermediary certificates) and certificate revocation lists (CRLs).

15. Click **OK** to save your configuration.

Configure the key information if this token consumer configuration is for an X.509 security token. For more information, see "Configuring key information for the consumer binding with an assembly tool" on page 617.

**Related concepts**

"Trusted ID evaluator" on page 613
A *trusted ID evaluator* (com.ibm.wsspi.wssecurity.id.TrustedIDEvaluatorImpl) is an abstraction of the mechanism that evaluates whether the given ID name is trusted.

**Related tasks**

"Configuring the security token requirement in consumer security constraints" on page 653

"Configuring the security token in generator security constraints" on page 653

"Configuring key information for the consumer binding with an assembly tool" on page 617

***Configuring the key locator for the consumer binding with an assembly tool:***

Prior to completing this task, you must import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF. Before configuring a key locator, you should know which key information configuration references this key locator. For example, if you configure this key locator for the STRREF key information type, select the com.ibm.wsspi.wssecurity.keyinfo.X509TokeyKeyLocator key locator class.

WebSphere Application Server Version 6 provides default key locator implementations that you can choose or you can write your own implementation. Custom key locators must implement the com.ibm.wsspi.wssecurity.keyinfo.KeyLocator interface. Using this implementation, you can locate keys within any data source.

Complete the following steps to configure a key locator for the consumer using an assembly tool. The purpose of the key locators is to find keys or certificates. The key locator information on the consumer side is used to find the key for validating the digital signature in the received SOAP message or for decrypting the encrypted parts of the message. The response consumer is configured for the client and the request consumer is configured for the server. In the following steps, you must configure either the client-side bindings in step 2 or the server-side bindings in step 3.

1. Start the assembly tool and click **Window > Open Perspective > J2EE**.
2. **Optional:** Locate the client-side bindings using the Project Explorer window. The Client Deployment Descriptor window is displayed. This Web service contains the bindings that you need to configure. Complete the following steps to locate the client-side bindings:
   a. Expand the Web Services > Client section and double-click the name of the Web service.
   b. Click the **WS Binding** tab and expand the Security Response Consumer Binding Configuration section.
3. **Optional:** Locate the server-side bindings using the Project Explorer window. The Web Services Editor window is displayed. This Web service contains the bindings that you need to configure. Complete the following steps to locate the server-side bindings:
   a. Expand the Web Services > Services section and double-click the name of the Web service.
   b. Click the **Binding Configurations** tab and expand the Request Consumer Binding Configuration Details section.
4. Expand the Key locators section and click **Add** to add a new entry or click **Edit** to edit a selected entry.
5. Specify a name for this configuration in the Key locator name field. This configuration name is referenced in the Key locator field of the Key Information dialog.

6. Select a key locator implementation in the Key locator class field. Select the key locator class that matches the Key Information configuration that references this key locator. For example, select the com.ibm.wsspi.wssecurity.keyinfo.X509TokenKeyLocator key locator class if the received Simple Object Access Protocol (SOAP) message contains an X.509 certificate that is needed for signature validation. Select the com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator key locator class if the key that is required for signature validation or decryption needs to be specified using a keystore file. The com.ibm.wsspi.wssecurity.keyinfo.SignerCertKeyLocator key locator class is not used on the consumer side. It is typically used in the response generator configuration for encrypting the response message using the signer key from the request message.
7. Select the **Use key store** option if you need to configure a key store for this key locator. Whether you need to configure the key store information for a key locator depends upon the key locator class and your application configuration. For example, if you select the com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator key locator class in the previous step, configure the key store information for this key locator.
   a. Specify a keystore password in the Key store storepass field. The keystore storepass is the password that is required to access the keystore file.
   b. Specify the path to the keystore file in the Key store path field. The key store path is the directory where the keystore resides. Make sure that wherever you deploy your application that the server can locate your keystore file.
   c. Select a key store type from the Key store type field. The key store type that you select must match the keystore file that is specified in the Key store path field. The following keystore types are supported:

      **JKS**
      Use this option if you are not using Java Cryptography Extensions (JCE) and if your keystore file uses the Java Keystore (JKS) format.

      **JCEKS**
      Use this option if you are using Java Cryptography Extensions.

      **PKCS11**
      Use this format if your keystore uses the PKCS#11 file format. Keystores using this format might contain RSA keys on cryptographic hardware or might encrypt keys that use cryptographic hardware to ensure protection.

      **PKCS12**
      Use this option if your keystore uses the PKCS#12 file format.
8. Click **Add** under the Key field to add a key entry from the keystore file that you specified in the previous step. This key is used for signature validation or decryption. The key that you specify must match the key that is used for digital signing or encryption for the generator. Complete the following steps to add a key entry:
   a. Specify an alias name for the key in the Alias field.
   b. Specify the password that is associated with the key in the Key pass field. This password protects the private key of the key pair that is specified by this key.
   c. Specify the key name in the Key name field. The key name specifies the Distinguished Name (DN) for the owner of the key.
9. Click **OK** to save the key locator configuration

After you configure the key locator and any token consumer that you need to configure, you can configure the key information that references this key locator. For more information, see “Configuring key information for the consumer binding with an assembly tool.”

**Related tasks**

“Configuring key information for the consumer binding with an assembly tool”

***Configuring key information for the consumer binding with an assembly tool:***

Prior to completing this task, you must complete the following steps:

1. Import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF.
2. Configure the key locator that is referenced by the key information configuration. For more information, see “Configuring the key locator for the consumer binding with an assembly tool” on page 616.
3. Configure the token consumer that is referenced by the key information configuration. For more information, see “Configuring token consumers with an assembly tool” on page 613

Complete the following steps to configure the key information for the server-side and client-side bindings using an assembly tool. The key information on the consumer side is used for specifying the information about the key that is used for validating the digital signature in the received message or for decrypting the encrypted parts of the message. The response consumer is configured for the client and the request consumer is configured for the server. In the following steps, you must configure either the client-side bindings in step 2 or the server-side bindings in step 3.

1. Start the assembly tool and click **Window > Open Perspective > J2EE**.
2. **Optional:** Locate the client-side bindings using the Project Explorer window. The Client Deployment Descriptor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the client-side bindings:
   a. Expand the Web Services > Client section and double-click the name of the Web service.
   b. Click the **WS Binding** tab and expand the Security Response Consumer Binding Configuration section.
3. **Optional:** Locate the server-side bindings using the Project Explorer window. The Web Services Editor window is displayed. This Web service contains the bindings that you need to configure. Complete the following steps to locate the server-side bindings:
   a. Expand the Web Services > Services section and double-click the name of the Web service.
   b. Click the **Binding Configurations** tab and expand the Request Consumer Binding Configuration Details section.
4. Expand the Key Information section and click **Add** to add a new entry or click **Edit** to edit a selected entry.
5. Specify a name for this configuration in the Key information name field. This configuration name is referenced by the Key information element within the Signing Information and Encryption Information Dialog windows. For more information, see “Configuring signing information for the consumer binding with an assembly tool” on page 620 and “Configuring encryption information for the consumer binding with an assembly tool” on page 637.
6. Select a key information type from the Key information type field. The key information types specify different mechanisms for referencing security tokens. The assembly tools support the following key information types:

   **STRREF**
   This type is the security token reference. The security token is directly referenced using Universal Resource Identifiers (URIs). The following <KeyInfo> element is generated in the SOAP message for this key information type:

```
<ds:KeyInfo>
              <wsse:SecurityTokenReference>
                   <wsse:Reference URI="#mytoken" />
              </wsse:SecurityTokenReference>
         </ds:KeyInfo>
```

   **EMB** This type is the embedded token. The security token is directly embedded within the <SecurityTokenReference> element. The following <KeyInfo> element is generated in the SOAP message for this key information type:

```
<ds:KeyInfo>
   <wsse:SecurityTokenReference>
      <wsse:Embedded wsu:Id="tok1" />
```

```
            ...
        </wsse:Embedded>
    </wsse:SecurityTokenReference>
</ds:KeyInfo>
```

**KEYID**

This type is a key identifier. The security token is referenced using an opaque value that uniquely identifies the token. The algorithm that is used for generating the key identifier value depends upon the token type. For example, a hash of the important elements of the security token is used for generating the KeyIdentifier value. The following `<KeyInfo>` element is generated in the SOAP message for this key information type:

```
<ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <wsse:SecurityTokenReference>
        <wsse:KeyIdentifier ValueType="wsse:X509v3">/62wXO...</wsse:KeyIdentifier>
    </wsse:SecurityTokenReference>
</ds:KeyInfo>
```

**KEYNAME**

This type is the key name. The security token is referenced using a name that matches an asserted identity within the token.

**Note:** Do not use this key type as it might result in multiple security tokens that match the specified name.

The KEYNAME type does not require a token consumer reference. The following `<KeyInfo>` element is generated in the SOAP message for this key information type:

```
<ds:KeyInfo>
    <ds:KeyName>CN=Group1</ds:KeyName>
</ds:KeyInfo>
```

**X509ISSUER**

This type is the X.509 certificate issuer name and serial number. The security token is referenced by an issuer name and issuer serial number of an X.509 certificate. The following `<KeyInfo>` element is generated in the SOAP message for this key information type:

```
<ds:KeyInfo>
    <wsse:SecurityTokenReference>
        <ds:X509Data>
            <ds:X509IssuerSerial>
                <ds:X509IssuerName>CN=Jones, O=IBM, C=US</ds:X509IssuerName>
                <ds:X509SerialNumber>1040152879</ds:X509SerialNumber>
            </ds:X509IssuerSerial>
        </ds:X509Data>
    </wsse:SecurityTokenReference>
</ds:KeyInfo>
```

7. Select the **Use key locator** option. Complete the following steps:
    a. Select the name of a key locator configuration from the Key locator field. The value of this field is a reference to a key locator that specifies how to find keys or certificates. For more information, see “Configuring the key locator for the consumer binding with an assembly tool” on page 616.
    b. Optional: Specify a key name in the Key name field. You do not need to specify the key name when you configure the key information for the consumer.
8. **Optional:** Select the **Use token** option and a token consumer configuration in the Token field if a token consumer is required for the key information configuration. The token that you select specifies a reference to a token consumer that is used for processing the security token within the message. A token consumer is required for all key information types except the KEYNAME type. Before you specify a token reference, you must configure a token consumer. For more information on token consumer configurations, see “Configuring token consumers with an assembly tool” on page 613.

After completing this task, configure the signing information or encryption information that references the key information that is specified by this task. For more information, see "Configuring signing information for the consumer binding with an assembly tool" or "Configuring encryption information for the consumer binding with an assembly tool" on page 637.

**Related tasks**

"Configuring the key locator for the consumer binding with an assembly tool" on page 616

"Configuring token consumers with an assembly tool" on page 613

"Configuring signing information for the consumer binding with an assembly tool"

"Configuring encryption information for the consumer binding with an assembly tool" on page 637

***Configuring signing information for the consumer binding with an assembly tool:***

Prior to completing this task, you must complete the following steps:

1. Import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF.
2. Specify which message parts to digitally sign. For more information, see "Signing message elements in consumer security constraints with keywords" on page 600 or "Signing message elements in consumer security constraints with an XPath expression" on page 606.
3. Configure the key information that is referenced by the Key information element within the Signing information dialog window. For more information, see "Configuring key information for the consumer binding with an assembly tool" on page 617.

Complete the following steps to configure the signing information for the server-side and client-side bindings using an assembly tool. The signing information on the consumer side is used to verify the integrity of the received Simple Object Access Protocol (SOAP) message by validating the message parts that are signed. The response consumer is configured for the client and the request consumer is configured for the server. In the following steps, you must configure either the client-side bindings in step 2 or the server-side bindings in step 3.

1. Start the assembly tool and click **Window > Open Perspective > J2EE**.
2. **Optional:** Locate the client-side bindings using the Project Explorer window. The Client Deployment Descriptor window is displayed. This Web service contains the bindings that you need to configure. Complete the following steps to locate the client-side bindings:
   a. Expand the Web Services > Client section and double-click the name of the Web service.
   b. Click the **WS Binding** tab and expand the Security Response Consumer Binding Configuration section.
3. **Optional:** Locate the server-side bindings using the Project Explorer window. The Web Services Editor window is displayed. This Web service contains the bindings that you need to configure. Complete the following steps to locate the server-side bindings:
   a. Expand the Web Services > Services section and double-click the name of the Web service.
   b. Click the **Binding Configurations** tab and expand the Request Consumer Binding Configuration Details section.
4. Expand the Signing Information section and click **Add** to add a new entry or select an existing entry and click **Edit**. The Signing Information Dialog window is displayed. Complete the following steps to specify the signing information:
   a. Specify a name for the signing information configuration in the Signing information name field.
   b. Select a canonicalization method from the Canonicalization method algorithm field. The canonicalization method algorithm is used to canonicalize the signing information before it is integrated as part of the signature operation. The following preconfigured algorithms are supported:
      - http://www.w3.org/2001/10/xml-exc-c14n#
      - http://www.w3.org/2001/10/xml-exc-c14n#WithComments

- http://www.w3.org/TR/2001/REC-xml-c14n-20010315
- http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments

You must specify the same canonicalization algorithm for both the generator and the consumer. For more information on configuring the signing information for the generator, see “Configuring signing information for the generator binding with an assembly tool” on page 598.

c. Select a signature method algorithm from the Signature method algorithm field. The following preconfigured algorithms are supported:
   - http://www.w3.org/2000/09/xmldsig#rsa-sha1
   - http://www.w3.org/2000/09/xmldsig#dsa-sha1
   - http://www.w3.org/2000/09/xmldsig#hmac-sha1

   You must specify the same signature algorithm for both the generator and the consumer. For more information on configuring the signing information for the generator, see “Configuring signing information for the generator binding with an assembly tool” on page 598.

5. Click **Add** in the Signing Key Information section to add a new key information entry or click **Remove** to delete a selected entry. Complete the following substeps if you are adding a new key information entry.
   a. Specify a name in the Key information name field.
   b. Select a key information reference from the list under the Key information element field. The value in this field references the key information configuration that you specified previously. If you have a key information configuration called `con_signkeyinfo` that you want to use with this signing information configuration, specify `con_signkeyinfo` in the Key information element field. For more information, see “Configuring key information for the consumer binding with an assembly tool” on page 617.
6. **Optional:** Select the **Use key information signature** option if you want to sign the key information within the SOAP message.
7. **Optional:** Select a key information signature type from the Type field if you select the **Use key information signature** option. Select the **keyinfo** value to specify that the entire KeyInfo element must be signed within the SOAP message. Select the **keyinfochildelements** value to specify that the child elements within the KeyInfo element must be signed. However, the KeyInfo element itself does not need to be signed.
8. Click **OK** to save your signing information configuration.
9. Expand the Part References subsection and select the signing information configuration from the Signing Information section.
10. Click **Add** in the Part References subsection to add a new entry or select an existing entry and click **Edit**. The Part References Dialog window is displayed. Complete the following steps to configure a part reference:
    a. Specify a name for the part reference configuration in the Part reference name field.
    b. Select a required integrity part configuration in the RequiredIntegrity part field. The required integrity part configuration specifies the message parts that are required to be signed. For more information on how to configure the required integrity, see “Signing message elements in consumer security constraints with keywords” on page 600 or “Signing message elements in consumer security constraints with an XPath expression” on page 606.
    c. Select the http://www.w3.org/2000/09/xmldsig#sha1 digest method algorithm in the Digest method algorithm field. This digest method algorithm is used to create the digest for each message part that is specified by this part reference.
    d. Click **OK** to save your part reference configuration.
11. Expand the Transforms subsection and the part reference configuration from the Part reference subsection.
12. Click **Add** in the Transforms subsection to add a new entry or select an existing entry and click **Edit**. The Transform dialog window is displayed.

a. Specify a transform name in the Name field.
b. Select a transform algorithm from the Algorithm field. The following transform algorithms are supported:

**http://www.w3.org/2001/10/xml-exc-c14n#**
This algorithm specifies the World Wide Web Consortium (W3C) Exclusive Canonicalization recommendation.

**http://www.w3.org/TR/1999/REC-xpath-19991116**
This algorithm specifies the W3C XML path language recommendation. If you specify this algorithm, you must specify the property name and value by clicking **Add**, which is displayed under Transform properties. For example, you might specify the following information:

**Name** `com.ibm.wsspi.wssecurity.dsig.XPathExpression`

**Value** `not(ancestor-or-self::*[namespace-uri()='http://www.w3.org/2000/09/xmldsig#' and local-name()='Signature'])`

**http://www.w3.org/2002/06/xmldsig-filter2**
This algorithm specifies the XML-Signature XPath Filter Version 2.0 proposed recommendation.

When you use this algorithm, you must specify a set of properties in the Transform property fields. You can use multiple property sets for the XPath Filter Version 2.

**Note:** End your property names with the number of the property set, which is denoted by an asterisk in the following examples:

- To specify an XPath expression for the XPath filter2, you might use:

```
name com.ibm.wsspi.wssecurity.dsig.XPath2Expression_*
```

- To specify a filter type for each XPath, you might use:

```
name com.ibm.wsspi.wssecurity.dsig.XPath2Filter_*
```

  Following this expression, you can have a value, `[intersect]`, `[subtract]`, or `[union]`.

- To specify the processing order for each XPath, you might use:

```
name com.ibm.wsspi.wssecurity.dsig.XPath2Order_*
```

  Following this expression, indicate the processing order of the XPath.

The following is a list of complete examples:

```
com.ibm.wsspi.wssecurity.dsign.XPath2Filter_1 = [intersect]
com.ibm.wsspi.wssecurity.dsign.XPath2Order_1 = [1]
com.ibm.wsspi.wssecurity.dsign.XPath2Expression_2 = [XPath expression#2]
com.ibm.wsspi.wssecurity.dsign.XPath2Filter_2 = [subtract]
com.ibm.wsspi.wssecurity.dsign.XPath2Filter_2 = [1]
```

**http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform**

**http://www.w3.org/2002/07/decrypt#XML**
This algorithm specifies the W3C decryption transform for XML Signature recommendation.

**http://www.w3.org/2000/09/xmldsig#enveloped-signature**
This algorithm specifies the W3C recommendation for XML digital signatures.

c. Click **OK** to save your transforms configuration.

After you complete this task for the consumer binding, you must configure the signing information for generator binding if this task was not previously completed.

**Related tasks**


"Signing message elements in consumer security constraints with keywords" on page 600

"Signing message elements in consumer security constraints with an XPath expression" on page 606

"Configuring key information for the consumer binding with an assembly tool" on page 617

"Configuring signing information for the generator binding with an assembly tool" on page 598


***Encrypting the message elements in generator security constraints with keywords:***

Prior to completing this task, you must import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF.

Complete the following steps to specify which message parts to encrypt when you configure the consumer security constraints for either the request generator or the response generator. The request generator is configured for the client and the response generator is configured for the server. In the following steps, you must configure either the client-side extensions in step 2 or the server-side extensions in step 3.

1. Start the assembly tool and click **Window > Open Perspective > J2EE**.
2. **Optional:** Locate the client-side extensions using the Project Explorer window. The Client Deployment Descriptor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the client-side extensions:
   a. Expand the Web Services > Client section and double-click the name of the Web service.
   b. Click the **WS Extension** tab and expand the Request Generator Configuration section.
3. **Optional:** Locate the server-side extensions using the Project Explorer window. The Web Services Editor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the server-side extensions:
   a. Expand the Web Services > Services section and double-click the name of the Web service.
   b. Click the **Extensions** tab and expand the Response Generator Service Configuration Details section.
4. Expand the Required Confidentiality section. Confidentiality refers to encryption while integrity refers to digital signing. Confidentiality reduces the risk of someone intercepting the message as it moves across a network. With confidentiality specifications, the message is encrypted before it is sent and decrypted when it is received at the intended target. For more information on encryption, see "XML encryption" on page 634.
5. Click **Add** to specify which parts of the message to encrypt. The Confidentiality Dialog window is displayed. Complete the following steps to specify the message parts:
   a. Specify a name for the confidentiality element in the Confidentiality Name field. For example, you might specify `conf_webskey`.
   b. Specify an order in the Order field. The value, which must be a positive integer value, specifies the order in which the encryption is processed. An order value of 1 specifies that the encryption is done first.
6. Click **Add** under Message parts and select the Message parts dialect. The http://www.ibm.com/websphere/webservices/wssecurity/dialect-was dialect specifies which message part is encrypted using keywords. If you select this dialect, you can select one of the following keywords under Message parts keyword:

**bodycontent**
: Specifies the user data portion of the message. If you select this keyword, the body is encrypted.

**usernametoken**
: Specifies a username token that contains the basic authentication information such as a user name and a password. Usually, the username token is encrypted so that the user information is secure. If you select this keyword, the the username token element is encrypted.

**digestvalue**
: Specifies a unique digest value. When a part of the SOAP message is signed, a unique digest value is created and is used by the receiving party to check the integrity of the message. You can encrypt the digestvalue element to secure the digest value.

**Note:** You must have a matching configuration for the consumer side.

In addition to the message parts, you also can specify that WebSphere Application Server encrypt the nonce and timestamp elements. For more information, see the following articles:

- "Adding time stamps for confidentiality to generator security constraints with keywords"
- "Adding time stamps for confidentiality to generator security constraints with an XPath expression" on page 630
- "Adding the nonce for confidentiality to generator security constraints with keywords" on page 627
- "Adding the nonce for confidentiality to generator security constraints with an XPath expression" on page 632

7. Click **OK** to save your configuration.

For example, the following sample is a part of a Simple Object Access Protocol (SOAP) message whose message content is encrypted using the bodycontent keyword and the http://www.ibm.com/websphere/webservices/wssecurity/dialect-was dialect:

```
<soapenv:Body soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <EncryptedData Id="wssecurity_encryption_id_8770799378696212005"
    Type="http://www.w3.org/2001/04/xmlenc#Content" xmlns="http://www.w3.org/2001/04/xmlenc#">
      <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
      <CipherData>
         <CipherValue>nIlF+Uthee0H96HbtRrolJ/tBmOazyryNYRwr/reF4nqtbHqGtNuew==</CipherValue>
      </CipherData>
   </EncryptedData>
</soapenv:Body>
```

After you specify which message parts to encrypt, you must specify which method is used to encrypt the message parts. For more information, see "Configuring encryption information for the generator binding with an assembly tool" on page 648.

**Related tasks**

"Adding time stamps for confidentiality to generator security constraints with an XPath expression" on page 630

"Adding the nonce for confidentiality to generator security constraints with an XPath expression" on page 632

"Configuring encryption information for the generator binding with an assembly tool" on page 648

*Adding time stamps for confidentiality to generator security constraints with keywords:*

Prior to completing this task, you must import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF.

This task is used to specify that a time stamp is embedded in a particular element and that the element is encrypted. Complete the following steps to specify the time stamp for confidentiality using keywords when

you configure the generator security constraints for either the request generator or the response generator. The request generator is configured for the client and the response generator is configured for the server. In the following steps, you must configure either the client-side extensions in step 2 or the server-side extensions in step 3.

1. Start the assembly tool and click **Window > Open Perspective > J2EE**.
2. **Optional:** Locate the client-side extensions using the Project Explorer window. The Client Deployment Descriptor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the client-side extensions:
   a. Expand the Web Services > Client section and double-click the name of the Web service.
   b. Click the **WS Extension** tab and expand the Request Generator Configuration section.
3. **Optional:** Locate the server-side extensions using the Project Explorer window. The Web Services Editor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the server-side extensions:
   a. Expand the Web Services > Services section and double-click the name of the Web service.
   b. Click the **Extensions** tab and expand the Response Generator Service Configuration Details section.
4. Expand the Confidentiality section. Confidentiality refers to encryption while integrity refers to digital signing. Confidentiality reduces the risk of someone intercepting the message as it moves across a network. With confidentiality specifications, the message is encrypted before it is sent and decrypted when it is received at the intended target. For more information on encryption, see “XML encryption” on page 634.
5. Click **Add** to specify a time stamp for confidentiality. The Confidentiality Dialog window is displayed. Complete the following steps to specify a confidentiality configuration:
   a. Specify a name for the confidentiality element in the Confidentiality Name field. For example, you might specify `conf_tmstmp`.
   b. Specify an order in the Order field. The value, which must be a positive integer value, specifies the order in which the encryption is processed. An order value of 1 specifies that the encryption is done first.
6. In the Timestamp section, click **Add** and select the Timestamp dialect. The http://www.ibm.com/websphere/webservices/wssecurity/dialect-was dialect specifies the message part that is encrypted using the keywords. If you select this dialect, you can select one of the following keywords under the Timestamp keyword heading:

   **bodycontent**
   : Specifies the user data portion of the message. If this keyword is selected, the time stamp is embedded in the Simple Object Access Protocol (SOAP) message body and the body is encrypted.

   **usernametoken**
   : Specifies a username token that contains the basic authentication information such as a user name and a password. Usually, the username token is encrypted so that the user information is secure. If you select this keyword, the timestamp element is embedded in the username token element and it is encrypted.

   **digestvalue**
   : Specifies a unique digest value. When a part of the Simple Object Access Protocol (SOAP) message is signed, a unique digest value is created and is used by the receiving party to check the integrity of the message. You can encrypt the digestvalue element to secure the digest value. If you select this keyword, the time stamp is embedded in the digestvalue element and the element is encrypted.
7. Specify an expiration time for the time stamp in the Timestamp expires field. The time stamp helps defend against replay attacks. The lexical representation for the duration is the [ISO 8601] extended format `PnYnMnDTnHnMnS`, where:

   **`P`**
   : Precedes the date and time values.

**nY** Represents the number of years in which the time stamp is in effect. Select a value from 0 to 99 years.

**nM** Represents the number of months in which the time stamp is in effect. Select a value from 0 to 11 months.

**nD** Represents the number of days in which the time stamp is in effect. Select a value from 0 to 30 days.

**T** Separates the date and time values.

**nH** Represents the number of hours in which the time stamp is in effect. Select a value from 0 to 23 hours.

**nM** Represents the number of minutes in which the time stamp is in effect. Select a value from 0 to 59 minutes.

**nS** Represents the number of seconds in which the time stamp is in effect. The number of seconds can include decimal digits to aribrary precision. You can select a value from 0 to 59 for the seconds and from 0 to 9 for tenths of a second.

For example, 1 year, 2 months, 3 days, 10 hours, and 30 minutes is represented as `P1Y2M3DT10H30M`. Typically, you might configure a message time stamp for between 10 and 30 minutes. For example, 10 minutes is represented as `P0Y0M0DT0H10M0S`.

8. In the Message Parts section, click **Add** and select http://www.ibm.com/websphere/webservices/wssecurity/dialect-was in the Message parts dialect field.
9. In the Message Parts section, select the message parts keyword.

   **Important:** You must define at least one message part in Message Parts section in order to specify a times tamp for confidentiality. When you select the message part, you are encrypting the message part in addition to the parent element of the time stamp.
10. Click **OK** to save the configuration changes.

    **Note:** These configurations for the consumer and generator must match.

    In addition to the time stamp, you can specify that the nonce is signed. For more information, see the following articles:
    - "Adding the nonce for confidentiality to generator security constraints with keywords" on page 627
    - "Adding the nonce for confidentiality to generator security constraints with an XPath expression" on page 632

For example, the following example is a part of a SOAP message where a time stamp is inserted into the bodycontent element and is encrypted using bodycontent keyword and the http://www.ibm.com/websphere/webservices/wssecurity/dialect-was dialect.

**Important:** You cannot see the time stamp in the message because it is encrypted.

```
<soapenv:Body soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <EncryptedData Id="wssecurity_encryption_id_4349704672508984224" Type=
    "http://www.w3.org/2001/04/xmlenc#Content" xmlns="http://www.w3.org/2001/04/xmlenc#">
      <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
         <CipherData>
            <CipherValue>IxSuTmF1vAygF/SBLCd8bgu8opPiwHmroIBLzZbENGr9JpxhSFt/0fV0sFun0uxg/
             h/Y+1erE+NaysREuL+E9AQm0lxALNEdBX9zpeVf+ZffUCSzZfXXe9iosQlPe9jG7yTp+rhZGdp/KOp
             26c3DZXCNDr0Wgz31wn3KNm6bGO6RmBzahEOSW8d0wR999DeqSpOY12d8iWJa3HZ8gnGnineCiZ3wr
             Hy9rOC58iijcsNv1fP3lExuA5WkHra6rndhbi8P7jDMhkzf4Odj2yy1M3XURWalOLNYhNJ9YaWACsaY
             CY2ukcKtzw==</CipherValue>
        </CipherData>
   </EncryptedData>
</soapenv:Body>
```

After you specify which message parts to encrypt, you must specify which method is used to encrypt sign the message. For more information, see “Configuring encryption information for the generator binding with an assembly tool” on page 648.

**Related tasks**

“Adding the nonce for confidentiality to generator security constraints with keywords”

“Adding the nonce for confidentiality to generator security constraints with an XPath expression” on page 632

“Configuring encryption information for the generator binding with an assembly tool” on page 648

*Adding the nonce for confidentiality to generator security constraints with keywords:*

Prior to completing this task, you must import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF.

Nonce for confidentiality is used to specify that the nonce is embedded in a particular element within the message and that the element is encrypted. Nonce is a randomly generated, cryptographic token. When you add a nonce to a specific part of a message, it can prevent theft and replay attacks because a generated nonce is unique. For example, without a nonce, the token might be intercepted and used in a replay attack when a user name token is passed from one machine to another machine using a non-secure transport, such as HTTP. The user name token can be stolen even if you use XML digital signature and XML encryption. This situation might be prevented by adding a nonce.

Complete the following steps to specify a nonce for confidentiality using keywords when you configure the generator security constraints for either the request generator or the response generator. The request generator is configured for the client and the response generator is configured for the server. In the following steps, you must configure either the client-side extensions in step 2 or the server-side extensions in step 3.

1. Start the assembly tool and click **Window > Open Perspective > J2EE**.
2. **Optional:** Locate the client-side extensions using the Project Explorer window. The Client Deployment Descriptor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the client-side extensions:
   a. Expand the Web Services > Client section and double-click the name of the Web service.
   b. Click the **WS Extension** tab and expand the Request Generator Configuration section.
3. **Optional:** Locate the server-side extensions using the Project Explorer window. The Web Services Editor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the server-side extensions:
   a. Expand the Web Services > Services section and double-click the name of the Web service.
   b. Click the **Extensions** tab and expand the Response Generator Service Configuration Details section.
4. Expand the Confidentiality section. Confidentiality refers to encryption while integrity refers to digital signing. Confidentiality reduces the risk of someone intercepting the message as it moves across a network. With confidentiality specifications, the message is encrypted before it is sent and decrypted when it is received at the intended target. For more information on encryption, see “XML encryption” on page 634.
5. Click **Add** to specify a nonce for integrity. The Confidentiality Dialog window is displayed. Complete the following steps to specify a configuration:
   a. Specify a name for the confidentiality element in the Confidentiality Name field. For example, you might specify `conf_nonce`.
   b. Specify an order in the Order field. The value, which must be a positive integer value, specifies the order in which the encryption is processed. An order value of 1 specifies that the encyrption is done first.

6. Under Nonce, click **Add** and select the Nonce dialect. The http://www.ibm.com/websphere/webservices/wssecurity/dialect-was dialect specifies the message part to which a nonce is added and encrypted. If you select this dialect, you can select one of the following keywords under Nonce keyword:

   **bodycontent**
   Specifies the user data portion of the message. If this keyword is selected, the nonce is embedded in the Simple Object Access Protocol (SOAP) message body and the body is encrypted.

   **usernametoken**
   Specifies a username token that contains the basic authentication information such as a user name and a password. Usually, the username token is encrypted so that the user information is secure. If you select this keyword, the nonce element is embedded in the username token element and it is encrypted.

   **digestvalue**
   Specifies a unique digest value. When a part of the SOAP message is signed, a unique digest value is created and is used by the receiving party to check the integrity of the message. You can encrypt the digestvalue element to secure the digest value. If you select this keyword, the nonce is embedded in the digestvalue element and the element is encrypted.
7. In the Message Parts section, click **Add** and select http://www.ibm.com/websphere/webservices/wssecurity/dialect-was in the Message parts dialect field.
8. In the Message Parts section, select the message parts keyword.

   **Important:** You must define at least one message part in Message Parts section in order to specify a nonce for confidentiality. When you select the message part, you are encrypting the message part in addition to the parent element of the nonce.
9. Click **OK** to save the configuration changes.

   **Note:** These configurations for the generator and the consumer must match.

   In addition to the nonce, you can specify that the timestamp element is encrypted. For more information, see the following articles:
   - “Adding time stamps for confidentiality to generator security constraints with keywords” on page 624
   - “Adding time stamps for confidentiality to generator security constraints with an XPath expression” on page 630

The following example is a part of a SOAP message where a nonce is inserted into the bodycontent element and it is encrypted using the bodycontent keyword and the http://www.ibm.com/websphere/webservices/wssecurity/dialect-was dialect.

**Important:** You cannot see the nonce in the message because it is encrypted.

```
<soapenv:Body soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <EncryptedData Id="wssecurity_encryption_id_1669600751905274321"
    Type="http://www.w3.org/2001/04/xmlenc#Content" xmlns="http://www.w3.org/2001/04/xmlenc#">
      <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
      <CipherData>
         <CipherValue>pZpVL6Rs6zhvu8UrC7TH3BA2zvOdpPpLeHnwH0dCpmdc7ETz1tUHDdXLFxy143nYu91Mxpzsp
          Wt1rWx2Lx9vFGRIfblRSX5lEpV8+0LvezvhJYY/cbTA04mTMUzCfv28v2TI09AZQ4TjII4u+cPeh5f0prBVK1
          E5hLTql4QMcwf/rq9h+tttrJbR7ub3AUgIVo42ucQs5HZbaDijxmdSuFboBql4lv1Ep24ZfeoB/p7aHzyeWy7p
          YtO0bshpks/oBwO/78vxSklVJKu4sUseFvZa+B7sciFneeNnNuRCqB2JXc/vtH83l3AELUZg60ehd4vqvXkyuv
          SLohZ/kKnF/A5c+BP5Bo1pgvwmDEeJItQ5a7LOKkTavLuc2WGtVo1947fnNGm2TN4C6U/cp9ERT7jAB9Lr/1v/8
          ZqPZYmssyME4pGeSWLy232WrPvk6HEu96GHfRt+YXWpVNvSEt/gZw==</CipherValue>
      </CipherData>
   </EncryptedData>
</soapenv:Body>
```

After you specify which message parts to encrypt, you must specify which method is used to encrypt the message. For more information, see "Configuring encryption information for the generator binding with an assembly tool" on page 648.

**Related concepts**

"XML digital signature" on page 570
XML-Signature Syntax and Processing (XML digital signature) is a specification that defines XML syntax and processing rules to sign and verify digital signatures for digital content. The specification was developed jointly by the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF).

**Related tasks**

"Adding time stamps for confidentiality to generator security constraints with keywords" on page 624

"Adding time stamps for confidentiality to generator security constraints with an XPath expression" on page 630

"Configuring encryption information for the generator binding with an assembly tool" on page 648

***Encrypting the message elements in generator security constraints with an XPath expression:***

Prior to completing this task, you must import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF.

The following information explains the difference between using an XPath expression and using keywords to specify which part of the message to sign:

**XPath expression**
: Specify any part of the message using an XPath expression. XPath is a language that is used to address parts of an XML document. You can find information on XPath syntax at the following Web site: http://www.w3.org/TR/1999/REC-xpath-19991116.

**Keywords**
: Specify only elements within the message using predefined keywords.

Complete the following steps to specify which message parts to encrypt using an XPath expression when you configure the consumer security constraints for either the request generator or the response generator. The request generator is configured for the client and the response generator is configured for the server. In the following steps, you must configure either the client-side extensions in step 2 or the server-side extensions in step 3.

1. Start the assembly tool and click **Window > Open Perspective > J2EE**.
2. **Optional:** Locate the client-side extensions using the Project Explorer window. The Client Deployment Descriptor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to configure the client-side extensions:
   a. Expand the Web Services > Client section and double-click the name of the Web service.
   b. Click the **WS Extension** tab and expand the Request Generator Configuration section.
3. **Optional:** Locate the server-side extensions using the Project Explorer window. The Web Services Editor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to configure the server-side extensions:
   a. Expand the Web Services > Services section and double-click the name of the Web service.
   b. Click the **Extensions** tab and expand the Response Generator Service Configuration Details section.
4. Expand the Confidentiality section. Confidentiality refers to encryption while integrity refers to digital signing. Confidentiality reduces the risk of someone intercepting the message as it moves across a network. With confidentiality specifications, the message is encrypted before it is sent and decrypted when it is received at the intended target. For more information on encryption, see "XML encryption" on page 634.

5. Click **Add** to specify which parts of the message to encrypt. The Confidentiality Dialog window is displayed. Complete the following steps to specify a configuration:
   a. Specify a name for the confidentiality element in the Confidentiality Name field. For example, you might specify `conf_xpath`.
   b. Specify an order in the Order field. The value, which must be a positive integer value, specifies the order in which the encryption is processed. An order value of 1 specifies that the encryption is done first.
6. Click **Add** under the Message parts section of the Confidentiality Dialog window. Complete the following steps to specify the message parts:
   a. Select the message parts dialect from the Message parts section. The http://www.w3.org/TR/1999/REC-xpath-19991116 dialect specifies which message part is encrypted using an XPath expression.
   b. Specify the message part to be encrypted using an XPath expression in the Message parts keyword field. For example, to specify that the body is encrypted, you might add the following expression in the Message parts keyword field as one continuous line:

```
/*[namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/' and local-name()='Envelope']/*
[namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/' and local-name()='Body']
```

   **Note:** These configurations for the generator and the consumer must match.

   In addition to the message parts, you also can specify that WebSphere Application Server encrypt the nonce and timestamp elements. For more information, see the following articles:
   - "Adding time stamps for confidentiality to generator security constraints with keywords" on page 624
   - "Adding time stamps for confidentiality to generator security constraints with an XPath expression"
   - "Adding the nonce for confidentiality to generator security constraints with keywords" on page 627
   - "Adding the nonce for confidentiality to generator security constraints with an XPath expression" on page 632
7. Click **OK** to save your configuration.

After you specify which message parts to encrypt, you must specify which method is used to encrypt the message parts. For more information, see "Configuring encryption information for the generator binding with an assembly tool" on page 648.

**Related tasks**

"Adding time stamps for confidentiality to generator security constraints with keywords" on page 624

"Adding the nonce for confidentiality to generator security constraints with keywords" on page 627

"Configuring encryption information for the generator binding with an assembly tool" on page 648

*Adding time stamps for confidentiality to generator security constraints with an XPath expression:*

Prior to completing this task, you must import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF.

The following information explains the difference between using an XPath expression and using keywords to specify which part of the message to sign:

**XPath expression**
Specify any part of the message using an XPath expression. XPath is a language that is used to address parts of an XML document. You can find information on XPath syntax at the following Web site: http://www.w3.org/TR/1999/REC-xpath-19991116.

**Keywords**
Specify only elements within the message using predefined keywords.

This task is used to specify that a time stamp is embedded in a particular element and that the element is encrypted. Complete the following steps to specify the time stamp for confidentiality using an XPath expression when you configure the generator security constraints for either the request generator or the response generator. The request generator is configured for the client and the response generator is configured for the server. In the following steps, you must configure either the client-side extensions in step 2 or the server-side extensions in step 3.

1. Start the assembly tool and click **Window > Open Perspective > J2EE**.
2. **Optional:** Locate the client-side extensions using the Project Explorer window. The Client Deployment Descriptor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the client-side extensions:
   a. Expand the Web Services > Client section and double-click the name of the Web service.
   b. Click the **WS Extension** tab and expand the Request Generator Configuration section.
3. **Optional:** Locate the server-side extensions using the Project Explorer window. The Web Services Editor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the server-side extensions:
   a. Expand the Web Services > Services section and double-click the name of the Web service.
   b. Click the **Extensions** tab and expand the Response Generator Service Configuration Details section.
4. Expand the Confidentiality section. Confidentiality refers to encryption while integrity refers to digital signing. Confidentiality reduces the risk of someone intercepting the message as it moves across a network. With confidentiality specifications, the message is encrypted before it is sent and decrypted when it is received at the intended target. For more information on encryption, see “XML encryption” on page 634.
5. Click **Add** to specify a time stamp for confidentiality. The Confidentiality Dialog window is displayed. Complete the following information to specify a configuration:
   a. Specify a name for the confidentiality element in the Confidentiality Name field.
   b. Specify an order in the Order field. The value, which must be a positive integer value, specifies the order in which the encryption is processed. An order value of 1 specifies that the encryption is done first.
6. Click **Add** under the Timestamp section of the Confidentiality Dialog window. Complete the following steps to configure the time stamp information:
   a. Select the timestamp dialect from the Timestamp section. The http://www.w3.org/TR/1999/REC-xpath-19991116 dialect specifies which message part is encrypted using an XPath expression.
   b. Specify the message part to which a time stamp is added and encrypted using an XPath expression in the Timestamp keyword field. For example, to specify that a time stamp is added to the bodycontent element and it is encrypted, you might add the following expression in the Timestamp keyword field as one continuous line:

   ```
   /*[namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/' and local-name()='Envelope']/*
   [namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/' and local-name()='Bodycontent']
   ```

   c. Specify an expiration time for the time stamp in the Timestamp expires field. The time stamp helps defend against replay attacks. The lexical representation for the duration is the [ISO 8601] extended format `PnYnMnDTnHnMnS`, where:

   **P** Precedes the date and time values.

   **nY** Represents the number of years in which the time stamp is in effect. Select a value from 0 to 99 years.

   **nM** Represents the number of months in which the time stamp is in effect. Select a value from 0 to 11 months.

   **nD** Represents the number of days in which the time stamp is in effect. Select a value from 0 to 30 days.

   **T** Separates the date and time values.

**nH** Represents the number of hours in which the time stamp is in effect. Select a value from 0 to 23 hours.

**nM** Represents the number of minutes in which the time stamp is in effect. Select a value from 0 to 59 minutes.

**nS** Represents the number of seconds in which the time stamp is in effect. The number of seconds can include decimal digits to aribrary precision. You can select a value from 0 to 59 for the seconds and from 0 to 9 for tenths of a second.

For example, to indicate 1 year, 2 months, 3 days, 10 hours, and 30 minutes, the format is `P1Y2M3DT10H30M`. Typically, you might configure a message time stamp for between 10 and 30 minutes. For example, 10 minutes is represented as `P0Y0M0DT0H10M0S`.

7. In the Message Parts section, click **Add** and select http://www.ibm.com/websphere/webservices/wssecurity/dialect-was in the Message parts dialect field.
8. In the Message Parts section, select the message parts keyword.

   **Important:** You must define at least one message part in Message Parts section in order to specify Timestamp for Confidentiality. The selection of message part here is for encrypting a message part in addition to the parent element of the timestamp
9. Click **OK** to save the configuration changes.

   **Note:** These configurations for the consumer and the generator must match.

   In addition to the time stamp, you can specify that the nonce is signed. For more information, see the following articles:
   - "Adding the nonce for confidentiality to generator security constraints with keywords" on page 627
   - "Adding the nonce for confidentiality to generator security constraints with an XPath expression"

After you specify which message parts to encrypt, you must specify which method is used to encrypt sign the message. For more information, see "Configuring encryption information for the generator binding with an assembly tool" on page 648.

**Related tasks**

"Adding the nonce for confidentiality to generator security constraints with keywords" on page 627
"Adding the nonce for confidentiality to generator security constraints with an XPath expression"
"Configuring encryption information for the generator binding with an assembly tool" on page 648

*Adding the nonce for confidentiality to generator security constraints with an XPath expression:*

Prior to completing this task, you must import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF.

The following information explains the difference between using an XPath expression and using keywords to specify which part of the message to sign:

**XPath expression**
: Specify any part of the message using an XPath expression. XPath is a language that is used to address parts of an XML document. You can find information on XPath syntax at the following Web site: http://www.w3.org/TR/1999/REC-xpath-19991116.

**Keywords**
: Specify only elements within the message using predefined keywords.

Nonce for confidentiality is used to specify that the nonce is embedded in a particular element within the message and that the element is encrypted. Nonce is a randomly generated, cryptographic token. When you add a nonce to a specific part of a message, it can prevent theft and replay attacks because a

generated nonce is unique. For example, without a nonce, the token might be intercepted and used in a replay attack when a user name token is passed from one machine to another machine using a non-secure transport, such as HTTP. The user name token can be stolen even if you use XML digital signature and XML encryption. This situation might be prevented by adding a nonce.

Complete the following steps to specify a nonce for confidentiality using an XPath expression when you configure the generator security constraints for either the request generator or the response generator. The request generator is configured for the client and the response generator is configured for the server. In the following steps, you must configure either the client-side extensions in step 2 or the server-side extensions in step 3.

1. Start the assembly tool and click **Window > Open Perspective > J2EE**.
2. **Optional:** Locate the client-side extensions using the Project Explorer window. The Client Deployment Descriptor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the client-side extensions:
   a. Expand the Web Services > Client section and double-click the name of the Web service.
   b. Click the **WS Extension** tab and expand the Request Generator Configuration section.
3. **Optional:** Locate the server-side extensions using the Project Explorer window. The Web Services Editor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the server-side extensions:
   a. Expand the Web Services > Services section and double-click the name of the Web service.
   b. Click the **Extensions** tab and expand the Response Generator Service Configuration Details section.
4. Expand the Confidentiality section. Confidentiality refers to encryption while integrity refers to digital signing. Confidentiality reduces the risk of someone intercepting the message as it moves across a network. With confidentiality specifications, the message is encrypted before it is sent and decrypted when it is received at the intended target. For more information on encryption, see “XML encryption” on page 634.
5. Click **Add** to specify a nonce for integrity. The Confidentiality Dialog window is displayed. Complete the following steps to specify a configuration:
   a. Specify a name for the confidentiality element in the Confidentiality Name field. For example, you might specify `conf_nonce`.
   b. Specify an order in the Order field. The value, which must be a positive integer value, specifies the order in which the encryption is processed. An order value of 1 specifies that the encryption is done first.
6. Click **Add** under the Nonce section of the Confidentiality Dialog window. Complete the following steps to specify the none dialect and the message part:
   a. Select the nonce dialect from the Nonce section. The http://www.w3.org/TR/1999/REC-xpath-19991116 dialect specifies the message part to which a nonce is added and encrypted using an XPath expression.
   b. Specify the message part to which a nonce is added and encrypted using an XPath expression in the Nonce keyword field. For example, to specify that a nonce is added to the bodycontent element and it is encrypted, you might add the following expression in the Nonce keyword field as one continuous line:

   ```
   /*[namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/' and local-name()='Envelope']/*
   [namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/' and local-name()='Bodycontent']
   ```
7. In the Message Parts section, click **Add** and select http://www.w3.org/TR/1999/REC-xpath-19991116 in the Message parts dialect field.
8. In the Message Parts section, select the message parts keyword.

   **Important:** You must define at least one message part in Message Parts section in order to specify nonce for Confidentiality. The selection of message part here is for encrypting a message part in addition to the parent element of the nonce.

9. Click **OK** to save the configuration changes.

   **Note:** These configurations for the consumer and the generator must match.

   In addition to the nonce, you can specify that the timestamp element is signed. For more information, see the following articles:

   - “Adding time stamps for confidentiality to generator security constraints with keywords” on page 624
   - “Adding time stamps for confidentiality to generator security constraints with an XPath expression” on page 630

After you specify which message parts to encrypt, you must specify which method is used to encrypt sign the message. For more information, see “Configuring encryption information for the generator binding with an assembly tool” on page 648.

**Related concepts**

“XML digital signature” on page 570
XML-Signature Syntax and Processing (XML digital signature) is a specification that defines XML syntax and processing rules to sign and verify digital signatures for digital content. The specification was developed jointly by the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF).

**Related tasks**

“Adding time stamps for confidentiality to generator security constraints with keywords” on page 624

“Adding time stamps for confidentiality to generator security constraints with an XPath expression” on page 630

“Configuring encryption information for the generator binding with an assembly tool” on page 648

***XML encryption:***

XML encryption is a specification developed by World Wide Web (WWW) Consortium (W3C) in 2002 that contains the steps to encrypt data, the steps to decrypt encrypted data, the XML syntax to represent encrypted data, the information used to decrypt the data, and a list of encryption algorithms such as triple DES, AES, and RSA.

You can apply XML encryption to an XML element, XML element content, and arbitrary data, including an XML document. For example, suppose that you need to encrypt the CreditCard element shown in the example 1.

**Example 1: Sample XML document**

```
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <CreditCard Limit='5,000' Currency='USD'>
    <Number>4019 2445 0277 5567</Number>
    <Issuer>Example Bank</Issuer>
    <Expiration>04/02</Expiration>
  </CreditCard>
</PaymentInfo>
```

**Example 2: XML document with a common secret key**

Example 2 shows the XML document after encryption. The EncryptedData element represents the encrypted CreditCard element. The EncryptionMethod element describes the applied encryption algorithm, which is triple DES in this example. The KeyInfo element contains the information to retrieve a decryption key, which is a KeyName element in this example. The CipherValue element contains the ciphertext obtained by serializing and encrypting the CreditCard element.

```
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <EncryptedData Type='http://www.w3.org/2001/04/xmlenc#Element'
```

```
    xmlns='http://www.w3.org/2001/04/xmlenc#'>
   <EncryptionMethod
      Algorithm='http://www.w3.org/2001/04/xmlenc#tripledes-cbc'/>
      <KeyInfo xmlns='http://www.w3.org/2000/09/xmldsig#'>
        <KeyName>John Smith</KeyName>
      </KeyInfo>
      <CipherData>
        <CipherValue>ydUNqHkMrD...</CipherValue>
      </CipherData>
  </EncryptedData>
</PaymentInfo>
```

**Example 3: XML document encrypted with the public key of the recipient**

In example 2, it is assumed that both the sender and recipient have a common secret key. If the recipient has a public and private key pair, which is most likely the case, the CreditCard element can be encrypted as shown in example 3. The EncryptedData element is the same as the EncryptedData element found in Example 2. However, the KeyInfo element contains an EncryptedKey .

```
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <EncryptedData Type='http://www.w3.org/2001/04/xmlenc#Element'
    xmlns='http://www.w3.org/2001/04/xmlenc#'>
    <EncryptionMethod
      Algorithm='http://www.w3.org/2001/04/xmlenc#tripledes-cbc'/>
    <KeyInfo xmlns='http://www.w3.org/2000/09/xmldsig#'>
      <EncryptedKey xmlns='http://www.w3.org/2001/04/xmlenc#'>
        <EncryptionMethod
          Algorithm='http://www.w3.org/2001/04/xmlenc#rsa-1_5'/>
        <KeyInfo xmlns='http://www.w3.org/2000/09/xmldsig#'>
          <KeyName>Sally Doe</KeyName>
        </KeyInfo>
        <CipherData>
          <CipherValue>yMTEyOTA1M...</CipherValue>
        </CipherData>
      </EncryptedKey>
    </KeyInfo>
    <CipherData>
      <CipherValue>ydUNqHkMrD...</CipherValue>
    </CipherData>
  </EncryptedData>
</PaymentInfo>
```

**XML Encryption in the WSS-Core**

WSS-Core specification is under development by Organization for the Advancement of Structured Information Standards (OASIS). The specification describes enhancements to Simple Object Access Protocol (SOAP) messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. The message confidentiality is realized by encryption based on XML Encryption.

The WSS-Core specification supports encryption of any combination of body blocks, header blocks, their sub-structures, and attachments of a SOAP message. The specification also requires that when you encrypt parts of a SOAP message, you mprepend a reference from the security header block to the encrypted parts of the message. The reference can be a clue for a recipient to identify which encrypted parts of the message to decrypt.

The XML syntax of the reference varies according to what information is encrypted and how it is encrypted. For example, suppose that the CreditCard element in example 4 is encrypted with either a common secret key or the public key of the recipient.

**Example 4: Sample SOAP message**

```
<SOAP-ENV:Envelope
  SOAP-ENV:encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
  xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'>
  <SOAP-ENV:Body>
    <PaymentInfo xmlns='http://example.org/paymentv2'>
      <Name>John Smith</Name>
      <CreditCard Limit='5,000' Currency='USD'>
        <Number>4019 2445 0277 5567</Number>
        <Issuer>Example Bank</Issuer>
        <Expiration>04/02</Expiration>
      </CreditCard>
    </PaymentInfo>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The resulting SOAP messages are shown in Examples 5 and 6. In these example, the ReferenceList and EncryptedKey elements are used as references, respectively.

**Example 5: SOAP message encrypted with a common secret key**

```
<SOAP-ENV:Envelope
  SOAP-ENV:encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
  xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'>
  <SOAP-ENV:Header>
    <Security SOAP-ENV:mustUnderstand='1'
      xmlns='http://schemas.xmlsoap.org/ws/2003/06/secext'>
      <ReferenceList xmlns='http://www.w3.org/2001/04/xmlenc#'>
        <DataReference URI='#ed1'/>
      </ReferenceList>
    </Security>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <PaymentInfo xmlns='http://example.org/paymentv2'>
      <Name>John Smith</Name>
      <EncryptedData Id='ed1'
        Type='http://www.w3.org/2001/04/xmlenc#Element'
        xmlns='http://www.w3.org/2001/04/xmlenc#'>
        <EncryptionMethod
          Algorithm='http://www.w3.org/2001/04/xmlenc#tripledes-cbc'/>
        <KeyInfo xmlns='http://www.w3.org/2000/09/xmldsig#'>
          <KeyName>John Smith</KeyName>
        </KeyInfo>
        <CipherData>
          <CipherValue>ydUNqHkMrD...</CipherValue>
        </CipherData>
      </EncryptedData>
    </PaymentInfo>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

**Example 6: SOAP message encrypted with the public key of the recipient**

```
<SOAP-ENV:Envelope
  SOAP-ENV:encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
  xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'>
  <SOAP-ENV:Header>
    <Security SOAP-ENV:mustUnderstand='1'
      xmlns='http://schemas.xmlsoap.org/ws/2003/06/secext'>
      <EncryptedKey xmlns='http://www.w3.org/2001/04/xmlenc#'>
        <EncryptionMethod
          Algorithm='http://www.w3.org/2001/04/xmlenc#rsa-1_5'/>
        <KeyInfo xmlns='http://www.w3.org/2000/09/xmldsig#'>
          <KeyName>Sally Doe</KeyName>
        </KeyInfo>
        <CipherData>
          <CipherValue>yMTEyOTA1M...</CipherValue>
        </CipherData>
```

```
        <ReferenceList>
          <DataReference URI='#ed1'/>
        </ReferenceList>
      </EncryptedKey>
    </Security>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <PaymentInfo xmlns='http://example.org/paymentv2'>
      <Name>John Smith</Name>
      <EncryptedData Id='ed1'
        Type='http://www.w3.org/2001/04/xmlenc#Element'
        xmlns='http://www.w3.org/2001/04/xmlenc#'>
        <EncryptionMethod
          Algorithm='http://www.w3.org/2001/04/xmlenc#tripledes-cbc'/>
        <CipherData>
          <CipherValue>ydUNqHkMrD...</CipherValue>
        </CipherData>
      </EncryptedData>
    </PaymentInfo>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

**Relationship to digital signature**

The WSS-Core specification also provides message integrity, which is realized by a digital signature based on the XML-Signature specification.

A combination of encryption and digital signature over common data introduces cryptographic vulnerabilities.

***Configuring encryption information for the consumer binding with an assembly tool:***

Prior to completing this task, you must complete the following steps:

1. Import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF.
2. Specify which message parts to encrypt. For more information, see “Encrypting message elements in consumer security constraints with keywords” on page 639 or “Encrypting message elements in consumer security constraints with an XPath expression” on page 643.
3. Configure the key information that is referenced by the Key information element within the Encryption information dialog window. For more information, see “Configuring key information for the consumer binding with an assembly tool” on page 617.

Complete the following steps to configure the encryption information for the server-side and client-side bindings using an assembly tool. The encryption information on the consumer side is used for decrypting the encrypted message parts in the incoming Simple Object Access Protocol (SOAP) message. The response consumer is configured for the client and the request consumer is configured for the server. In the following steps, you must configure either the client-side bindings in step 2 or the server-side bindings in step 3.

1. Start the assembly tool and click **Window > Open Perspective > J2EE**.
2. **Optional:** Locate the client-side bindings using the Project Explorer window. The Client Deployment Descriptor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the client-side bindings:
   a. Expand the Web Services > Client section and double-click the name of the Web service.
   b. Click the **WS Binding** tab and expand the Security Response Consumer Binding Configuration section.

3. **Optional:** Locate the server-side bindings using the Project Explorer window. The Web Services Editor window is displayed. This Web service contains the bindings that you need to configure. Complete the following steps to locate the server-side bindings:
   a. Expand the Web Services > Services section and double-click the name of the Web service.
   b. Click the **Binding Configurations** tab and expand the Request Consumer Binding Configuration Details section.
4. Expand the Encryption Information section and click **Add** to add a new entry or select an existing entry and click **Edit**. The Encryption Information dialog window is displayed. Complete the following steps to specify an encryption information configuration:
   a. Specify a name for the encryption information configuration in the Encryption name field.
   b. Select a data encryption algorithm from the Data encryption method algorithm field. The data encryption algorithm is used for encrypting or decrypting parts of a SOAP message such as the SOAP body or the username token. The following pre-configured algorithms are supported:
      - http://www.w3.org/2001/04/xmlenc#tripledes-cbc
      - http://www.w3.org/2001/04/xmlenc#aes128-cbc
      - http://www.w3.org/2001/04/xmlenc#aes256-cbc

        To use this algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.
      - http://www.w3.org/2001/04/xmlenc#aes192-cbc

        To use this algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.

      This algorithm must match the data encryption algorithm that is configured for the generator. For more information on configuring the encryption information for the generator, see “Configuring encryption information for the generator binding with an assembly tool” on page 648.
   c. Select a key encryption algorithm from the Key encryption method algorithm field. The key encryption algorithm is used to encrypt the key that is used for encrypting the message parts within the SOAP message. The following pre-configured algorithms are supported:
      - http://www.w3.org/2001/04/xmlenc#rsa-1_5
      - http://www.w3.org/2001/04/xmlenc#kw-tripledes
      - http://www.w3.org/2001/04/xmlenc#kw-aes128
      - http://www.w3.org/2001/04/xmlenc#kw-aes256

        To use this algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.
      - http://www.w3.org/2001/04/xmlenc#kw-aes192

        To use this algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.

      Select the blank entry if the data encryption key, which is the (key used for encrypting the message parts, is not encrypted. This key encryption algorithm for the consumer must match the key encryption algorithm for the generator. For more information on configuring the encryption information for the generator, see “Configuring encryption information for the generator binding with an assembly tool” on page 648.
5. Click **Add** in the Encryption Key Information section to add a new key information entry or click **Remove** to delete a selected entry. Complete the following substeps if you are adding a new key information entry.
   a. Specify a name in the Key information name field.

b. Select a key information reference from the list under the Encryption key information field. The value in this field references the key information configuration that you specified previously. If you have a key information configuration called `con_enckeyinfo` that you want to use with this encryption information configuration, specify `con_enckeyinfo` in the Key information element field. For more information, see “Configuring key information for the consumer binding with an assembly tool” on page 617.
6. Select a required confidentiality part from the list in the RequiredConfidentiality part field. The value in this field specifies a reference to the message parts for encryption.
7. Click **OK** to save your encryption information configuration.

After you complete this task for the consumer binding, you must configure the encryption information for generator binding if this task was not previously completed. For more information, see “Configuring encryption information for the generator binding with an assembly tool” on page 648.

**Related tasks**

“Encrypting message elements in consumer security constraints with keywords”

“Encrypting message elements in consumer security constraints with an XPath expression” on page 643

“Configuring encryption information for the generator binding with an assembly tool” on page 648

“Configuring key information for the consumer binding with an assembly tool” on page 617

***Encrypting message elements in consumer security constraints with keywords:***

Prior to completing this task, you must import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF.

Complete the following steps to specify which message parts to check for encryption when you configure the consumer security constraints for either the response consumer or the request consumer. The response consumer is configured for the client and the request consumer is configured for the server. In the following steps, you must configure either the client-side bindings in step 2 or the server-side bindings in step 3.

1. Start the assembly tool and click **Window > Open Perspective > J2EE**.
2. **Optional:** Locate the client-side bindings using the Project Explorer window. The Client Deployment Descriptor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the client-side bindings:
   a. Expand the Web Services > Client section and double-click the name of the Web service.
   b. Click the **WS Extension** tab and expand the Response Consumer Configuration section.
3. **Optional:** Locate the server-side bindings using the Project Explorer window. The Web Services Editor window is displayed. This Web service contains the bindings that you need to configure. Complete the following steps to locate the server-side bindings:
   a. Expand the Web Services > Services section and double-click the name of the Web service.
   b. Click the **Extensions** tab and expand the Request Consumer Service Configuration Details section.
4. Expand the Required Confidentiality section. Confidentiality refers to encryption while integrity refers to digital signing. Confidentiality reduces the risk of someone intercepting the message as it moves across a network. With confidentiality specifications, the message is encrypted before it is sent and decrypted when it is received at the intended target. For more information on encryption, see “XML encryption” on page 634.
5. Click **Add** to specify which parts of the message to check for encryption. The Required Confidentiality Dialog window is displayed. Complete the following steps to specify a configuration:
   a. Specify a name for the confidentiality element in the Required Confidentiality Name field.

b. Specify a usage type in the Usage type field. This field specifies the requirement for the confidentiality element. The value of this attribute is either Required or Optional.

6. Click **Add** under Message parts and select the message parts dialect. The http://www.ibm.com/websphere/webservices/wssecurity/dialect-was dialect specifies which message part to be checked for encryption using keywords. If you select this dialect, you can select one of the following keywords under Message parts keyword:

   **bodycontent**
   Specifies the user data portion of the message. If you select this keyword, the body is checked for encryption.

   **usernametoken**
   Specifies a username token that contains the basic authentication information such as a user name and a password. Usually, the username token is encrypted so that the user information is secure. If you select this keyword, the username token element is checked for encryption.

   **digestvalue**
   Specifies a unique digest value. When a part of the Simple Object Access Protocol (SOAP) message is signed, a unique digest value is created and is used by the receiving party to check the integrity of the message. You can encrypt the digestvalue element to secure the digest value. If you select this keyword, the digestvalue is checked for encryption.

   **Note:** You must have a matching configuration for the generator.

   In addition to the message parts, you also can specify that WebSphere Application Server check the encryption of the nonce and timestamp elements. For more information, see the following articles:

   - "Adding time stamps for confidentiality in consumer security constraints with keywords"
   - "Adding time stamps for confidentiality in consumer security constraints with an XPath expression" on page 645
   - "Adding a nonce for confidentiality in consumer security constraints with keywords" on page 642
   - "Adding the nonce for confidentiality in consumer security constraints with an XPath expression" on page 646

7. Click **OK** to save your configuration.

After you specify which message parts to check for encryption, you must specify which method is used to verify the encryption of the message parts. For more information, see "Configuring encryption information for the consumer binding with an assembly tool" on page 637.

**Related tasks**

"Adding time stamps for confidentiality in consumer security constraints with an XPath expression" on page 645

"Adding the nonce for confidentiality in consumer security constraints with an XPath expression" on page 646

"Configuring encryption information for the consumer binding with an assembly tool" on page 637

*Adding time stamps for confidentiality in consumer security constraints with keywords:*

Prior to completing this task, you must import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF.

This task is used to specify that a time stamp embedded in a particular element and encrypted is checked for encryption along with the message parts in the Required Integrity . Complete the following steps to specify the time stamp for confidentiality using keywords when you configure the consumer security constraints for either the response consumer or the request consumer. The response consumer is configured for the client and the request consumer is configured for the server. In the following steps, you must configure either the client-side extensions in step 2 or the server-side extensions in step 3.

1. Start the assembly tool and click **Window > Open Perspective > J2EE**.
2. **Optional:** Locate the client-side extensions using the Project Explorer window. The Client Deployment Descriptor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the client-side extensions:
   a. Expand the Web Services > Client section and double-click the name of the Web service.
   b. Click the **WS Extension** tab and expand the Response Consumer Configuration section.
3. **Optional:** Locate the server-side extensions using the Project Explorer window. The Web Services Editor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the server-side extensions:
   a. Expand the Web Services > Services section and double-click the name of the Web service.
   b. Click the **Extensions** tab and expand the Request Consumer Service Configuration Details section.
4. Expand the Required Confidentiality section. Confidentiality refers to encryption while integrity refers to digital signing. Confidentiality reduces the risk of someone intercepting the message as it moves across a network. With confidentiality specifications, the message is encrypted before it is sent and decrypted when it is received at the intended target. For more information on encryption, see “XML encryption” on page 634.
5. Click **Add** to specify that the element within which a timestamp is added and encrypted, is checked for confidentiality. The Required Confidentiality Dialog window is displayed. Complete the following steps to specify a configuration:
   a. Specify a name for the confidentiality element in the Required Confidentiality Name field.
   b. Specify a usage type in the Usage type field. This field specifies the requirement for the confidentiality element. The value of this attribute is either Required or Optional.
6. In the Timestamp section, click **Add** and select the Timestamp dialect. The http://www.ibm.com/websphere/webservices/wssecurity/dialect-was dialect specifies the message part that is verified for encryption using the keywords. If you select this dialect, you can select one of the following keywords under the Timestamp keyword heading:

   **bodycontent**
      Specifies the user data portion of the message. If this keyword is selected, the body along with the embedded timestamp is checked for confidentiality.

   **usernametoken**
      Specifies a username token that contains the basic authentication information such as a user name and a password. Usually, the username token is encrypted so that the user information is secure. If you select this keyword, the username token along with the embedded timestamp is checked for confidentiality.

   **digestvalue**
      Specifies a unique digest value. When a part of the Simple Object Access Protocol (SOAP) message is signed, a unique digest value is created and is used by the receiving party to check the integrity of the message. You can encrypt the digestvalue element to secure the digest value. If you select this keyword, the digestvalue along with the embedded timestamp is checked for confidentiality.
7. If you have not specified message part(s) in Required Confidentiality, in the Message Parts section click **Add** to add message parts. You must define at least one message part in Required Confidentiality for specifying Timestamp in Required Confidentiality.″
8. In the Message Parts section, select the message parts keyword.
9. Click **OK** to save the configuration changes.

   **Note:** These configurations for the consumer and the generator must match.

   In addition to the time stamp, you can specify that the nonce is checked for confidentiality. For more information, see the following articles:

- “Adding a nonce for integrity in consumer security constraints with keywords” on page 604
- “Adding a nonce for integrity in consumer security constraints with an XPath expression” on page 610

After you specify which message parts to check for encryption, you must specify which method is used to check the encryption. For more information, see “Configuring encryption information for the consumer binding with an assembly tool” on page 637.

**Related tasks**

“Adding a nonce for integrity in consumer security constraints with keywords” on page 604

“Adding a nonce for integrity in consumer security constraints with an XPath expression” on page 610

“Configuring encryption information for the consumer binding with an assembly tool” on page 637

*Adding a nonce for confidentiality in consumer security constraints with keywords:*

Prior to completing this task, you must import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF.

Nonce for confidentiality is used to specify that the nonce is embedded in a particular element within the message and that the element is encrypted. Complete the following steps to check the confidentiality of an element that has nonce embedded in it and is encrypted using keywords when you configure the consumer security constraints for either the response consumer or the request consumer. The response consumer is configured for the client and the request consumer is configured for the server. In the following steps, you must configure either the client-side extensions in step 2 or the server-side extensions in step 3.

1. Start the assembly tool and click **Window > Open Perspective > J2EE**.
2. **Optional:** Locate the client-side extensions using the Project Explorer window. The Client Deployment Descriptor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the client-side extensions:
   a. Expand the Web Services > Client section and double-click the name of the Web service.
   b. Click the **WS Extension** tab and expand the Response Consumer Configuration section.
3. **Optional:** Locate the server-side extensions using the Project Explorer window. The Web Services Editor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the server-side extensions:
   a. Expand the Web Services > Services section and double-click the name of the Web service.
   b. Click the **Extensions** tab and expand the Request Consumer Service Configuration Details section.
4. Expand the Required Confidentiality section. Confidentiality refers to encryption while integrity refers to digital signing. Confidentiality reduces the risk of someone intercepting the message as it moves across a network. With confidentiality specifications, the message is encrypted before it is sent and decrypted when it is received at the intended target. For more information on encryption, see “XML encryption” on page 634.
5. Click **Add** to specify that the element within which a nonce is added and encrypted, is checked for confidentiality. The Required Confidentiality Dialog window is displayed. Complete the following steps to specify a configuration:
   a. Specify a name for the confidentiality element in the Required Confidentiality Name field.
   b. Specify a usage type in the Usage type field. This field specifies the requirement for the confidentiality element. The value of this attribute is either Required or Optional.
6. Under Nonce, click **Add** and select the Nonce dialect. The http://www.ibm.com/websphere/webservices/wssecurity/dialect-was dialect specifies the message part which has an embedded nonce is verified for encryption. If you select this dialect, you can select one of the following keywords under Nonce keyword:

**bodycontent**
: Specifies the user data portion of the message. If this keyword is selected, the nonce is embedded in the Simple Object Access Protocol (SOAP) message body and the body along with the embedded nonce is checked for confidentiality.

**usernametoken**
: Specifies a username token that contains the basic authentication information such as a user name and a password. Usually, the username token is encrypted so that the user information is secure. If you select this keyword, the username token element along with the embedded nonce is checked for confidentiality.

**digestvalue**
: Specifies a unique digest value. When a part of the Simple Object Access Protocol (SOAP) message is signed, a unique digest value is created and is used by the receiving party to check the integrity of the message. You can encrypt the digestvalue element to secure the digest value. If you select this keyword, the digestvalue element along with the embedded nonce is checked for confidentiality.

7. If you have not specified message parts in Required Confidentiality, in the Message Parts section, click **Add** to add message parts. You must define at least one message part in Required Confidentiality for specifying Nonce in Required Confidentiality.
8. In the Message Parts section, select the message parts keyword.
9. Click **OK** to save the configuration changes.

   **Note:** These configurations for the consumer and the generator must match.

   In addition to the nonce, you can specify that the timestamp element is checked for confidentiality. For more information, see the following articles:
   - “Adding time stamps for confidentiality in consumer security constraints with keywords” on page 640
   - “Adding time stamps for confidentiality in consumer security constraints with an XPath expression” on page 645

After you specify which message parts to check for confidentiality, you must specify which method is used to verify the encryption. For more information, see “Configuring encryption information for the consumer binding with an assembly tool” on page 637.

**Related concepts**

“XML digital signature” on page 570
XML-Signature Syntax and Processing (XML digital signature) is a specification that defines XML syntax and processing rules to sign and verify digital signatures for digital content. The specification was developed jointly by the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF).

**Related tasks**

“Adding a stand-alone time stamp in consumer security constraints” on page 651

“Adding time stamps for confidentiality in consumer security constraints with keywords” on page 640

“Adding time stamps for confidentiality in consumer security constraints with an XPath expression” on page 645

“Configuring encryption information for the consumer binding with an assembly tool” on page 637

***Encrypting message elements in consumer security constraints with an XPath expression:***

Prior to completing this task, you must import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF.

The following information explains the difference between using an XPath expression and using keywords to specify which part of the message to sign:

**XPath expression**
: Specify any part of the message using an XPath expression. XPath is a language that is used to address parts of an XML document. You can find information on XPath syntax at the following Web site: http://www.w3.org/TR/1999/REC-xpath-19991116.

**Keywords**
: Specify only elements within the message using predefined keywords.

Complete the following steps to specify which message parts to check for encryption when you configure the consumer security constraints for either the response consumer or the request consumer. The response consumer is configured for the client and the request consumer is configured for the server. In the following steps, you must configure either the client-side extensions in step 2 or the server-side extensions in step 3.

1. Start the assembly tool and click **Window > Open Perspective > J2EE**.
2. **Optional:** Locate the client-side extensions using the Project Explorer window. The Client Deployment Descriptor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the client-side extensions:
   a. Expand the Web Services > Client section and double-click the name of the Web service.
   b. Click the **WS Extension** tab and expand the Response Consumer Configuration section.
3. **Optional:** Locate the server-side extensions using the Project Explorer window. The Web Services Editor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the server-side extensions:
   a. Expand the Web Services > Services section and double-click the name of the Web service.
   b. Click the **Extensions** tab and expand the Request Consumer Service Configuration Details section.
4. Expand the Required Confidentiality section. Confidentiality refers to encryption while integrity refers to digital signing. Confidentiality reduces the risk of someone intercepting the message as it moves across a network. With confidentiality specifications, the message is encrypted before it is sent and decrypted when it is received at the intended target. For more information on encryption, see “XML encryption” on page 634.
5. Click **Add** to specify which parts of the message to check for encryption. The Required Confidentiality Dialog window is displayed. Complete the following steps to specify a configuration:
   a. Specify a name for the confidentiality element in the Required Confidentiality Name field.
   b. Specify a usage type in the Usage type field. This field specifies the requirement for the confidentiality element. The value of this attribute is either Required or Optional.
6. Click **Add** under the Message parts section of the Required Confidentiality Dialog window. Complete the following steps to specify the message part and its associated message parts dialect:
   a. Select the message parts dialect from the Message parts section. The http://www.w3.org/TR/1999/REC-xpath-19991116 dialect specifies which message part is checked for encryption using an XPath expression.
   b. Specify the message part to be checked for encryption using an XPath expression in the Message parts keyword field. For example, to specify that the body is checked for encryption, you might add the following expression in the Message parts keyword field as one continuous line:

   ```
   /*[namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/' and local-name()='Envelope']/*
   [namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/' and local-name()='Body']
   ```

   **Note:** These configurations for the consumer and the generator must match.

   In addition to the message parts, you also can specify that WebSphere Application Server check the nonce and timestamp elements for encryption. For more information, see the following articles:
   - “Adding time stamps for confidentiality in consumer security constraints with keywords” on page 640
   - “Adding time stamps for confidentiality in consumer security constraints with an XPath expression” on page 645

- “Adding a nonce for confidentiality in consumer security constraints with keywords” on page 642
- “Adding the nonce for confidentiality in consumer security constraints with an XPath expression” on page 646

7. Click **OK** to save your configuration.

After you specify which message parts to check for encryption, you must specify which method is used to verify the encryption of the message parts. For more information, see “Configuring encryption information for the consumer binding with an assembly tool” on page 637.

**Related tasks**

“Adding time stamps for confidentiality in consumer security constraints with keywords” on page 640

“Adding a nonce for confidentiality in consumer security constraints with keywords” on page 642

“Configuring encryption information for the consumer binding with an assembly tool” on page 637

*Adding time stamps for confidentiality in consumer security constraints with an XPath expression:*

Prior to completing this task, you must import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF.

The following information explains the difference between using an XPath expression and using keywords to specify which part of the message to sign:

**XPath expression**
: Specify any part of the message using an XPath expression. XPath is a language that is used to address parts of an XML document. You can find information on XPath syntax at the following Web site: http://www.w3.org/TR/1999/REC-xpath-19991116.

**Keywords**
: Specify only elements within the message using predefined keywords.

This task is used to specify that a time stamp embedded in a particular element and encrypted on the generator side is checked for encryption on the consumer side. Complete the following steps to specify the time stamp for confidentiality using an XPath expression when you configure the consumer security constraints for either the response consumer or the request consumer. The response consumer is configured for the client and the request consumer is configured for the server. In the following steps, you must configure either the client-side extensions in step 2 or the server-side extensions in step 3.

1. Start the assembly tool and click **Window > Open Perspective > J2EE**.
2. **Optional:** Locate the client-side extensions using the Project Explorer window. The Client Deployment Descriptor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the client-side extensions:
   a. Expand the Web Services > Client section and double-click the name of the Web service.
   b. Click the **WS Extension** tab and expand the Response Consumer Configuration section.
3. **Optional:** Locate the server-side extensions using the Project Explorer window. The Web Services Editor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the server-side extensions:
   a. Expand the Web Services > Services section and double-click the name of the Web service.
   b. Click the **Extensions** tab and expand the Request Consumer Service Configuration Details section.
4. Expand the Required Confidentiality section. Confidentiality refers to encryption while integrity refers to digital signing. Confidentiality reduces the risk of someone intercepting the message as it moves across a network. With confidentiality specifications, the message is encrypted before it is sent and decrypted when it is received at the intended target. For more information on encryption, see “XML encryption” on page 634.

5. Click **Add** to specify that the element within which a timestamp is added and encrypted, is checked for confidentiality. The Required Confidentiality Dialog window is displayed. Complete the following steps to specify a configuration:
   a. Specify a name for the confidentiality element in the Required Confidentiality Name field.
   b. Specify a usage type in the Usage type field. This field specifies the requirement for the confidentiality element. The value of this attribute is either Required or Optional.
6. Click **Add** under the Timestamp section of the Required Confidentiality Dialog window. Complete the following steps to specify the timestamp dialect and the message part:
   a. Select the timestamp dialect from the Timestamp section. The http://www.w3.org/TR/1999/REC-xpath-19991116 dialect specifies which message part is verified for encryption using an XPath expression.
   b. Specify the message part which has an embedded time stamp and is checked for encryption using an XPath expression in the Timestamp keyword field. For example, to specify that the bodycontent element along with the embedded timestamp is checked for encryption, you might add the following expression in the Timestamp keyword field as one continuous line:

      ```
      /*[namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/' and local-name()='Envelope']/*
      [namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/' and local-name()='Bodycontent']
      ```

      If you have not specified message parts in Required Confidentiality, you can click **Add** in the Message Parts section to add the message parts. You must define at least one message part in Required Confidentiality to specify a time stamp in Required Confidentiality.
7. Click **OK** to save the configuration changes.

   **Note:** These configurations for the consumer and the generator must match.

   In addition to the time stamp, you can specify that the nonce is checked for encryption. For more information, see the following articles:
   - “Adding a nonce for integrity in consumer security constraints with keywords” on page 604
   - “Adding a nonce for integrity in consumer security constraints with an XPath expression” on page 610

After you specify which message parts to check for encryption, you must specify which method is used to verify the encryption. For more information, see “Configuring encryption information for the consumer binding with an assembly tool” on page 637.

**Related tasks**

“Adding a nonce for integrity in consumer security constraints with keywords” on page 604
“Adding a nonce for integrity in consumer security constraints with an XPath expression” on page 610
“Configuring encryption information for the consumer binding with an assembly tool” on page 637

*Adding the nonce for confidentiality in consumer security constraints with an XPath expression:*

Prior to completing this task, you must import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF.

The following information explains the difference between using an XPath expression and using keywords to specify which part of the message to sign:

**XPath expression**
: Specify any part of the message using an XPath expression. XPath is a language that is used to address parts of an XML document. You can find information on XPath syntax at the following Web site: http://www.w3.org/TR/1999/REC-xpath-19991116.

**Keywords**
: Specify only elements within the message using predefined keywords.

Nonce for confidentiality is used to specify that the nonce is embedded in a particular element within the message and that the element is encrypted. Complete the following steps to check the confidentiality of an element that has nonce embedded in it and is encrypted using an XPath expression when you configure the consumer security constraints for either the response consumer or the request consumer. The response consumer is configured for the client and the request consumer is configured for the server. In the following steps, you must configure either the client-side bindings in step 2 or the server-side bindings in step 3.

1. Start the assembly tool and click **Window > Open Perspective > J2EE**.
2. **Optional:** Locate the client-side bindings using the Project Explorer window. The Client Deployment Descriptor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the client-side bindings:
   a. Expand the Web Services > Client section and double-click the name of the Web service.
   b. Click the **WS Extension** tab and expand the Response Consumer Configuration section.
3. **Optional:** Locate the server-side bindings using the Project Explorer window. The Web Services Editor window is displayed. This Web service contains the bindings that you need to configure. Complete the following steps to locate the server-side bindings:
   a. Expand the Web Services > Services section and double-click the name of the Web service.
   b. Click the **Extensions** tab and expand the Request Consumer Service Configuration Details section.
4. Expand the Required Confidentiality section. Confidentiality refers to encryption while integrity refers to digital signing. Confidentiality reduces the risk of someone intercepting the message as it moves across a network. With confidentiality specifications, the message is encrypted before it is sent and decrypted when it is received at the intended target. For more information on encryption, see “XML encryption” on page 634.
5. Click **Add** to specify that the element within which a nonce is added and encrypted is checked for confidentiality. The Required Confidentiality Dialog window is displayed. Complete the following steps to specify a configuration:
   a. Specify a name for the confidentiality element in the Required Confidentiality Name field.
   b. Specify a usage type in the Usage type field. This field specifies the requirement for the confidentiality element. The value of this attribute is either Required or Optional.
6. Click **Add** under the Nonce section of the Required Confidentiality Dialog window. Complete the following steps to specify a nonce dialect and its associate message part:
   a. Select the nonce dialect from the Nonce section. The http://www.w3.org/TR/1999/REC-xpath-19991116 dialect specifies the message part which has an embedded nonce is verified for encryption using an XPath expression.
   b. Specify the message part which has an embedded nonce is verified for encryption using an XPath expression in the Nonce keyword field. For example, to specify that the bodycontent element along with the embedded nonce is checked for confidentiality, you might add the following expression in the Nonce keyword field as one continuous line:
   ```
   /*[namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/' and local-name()='Envelope']/*
   [namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/' and local-name()='Bodycontent']
   ```
7. If you have not specify message parts in Required Confidentiality, you can click **Add** in the Message Parts section to add message parts. You must define at least one message part in Required Confidentiality for specifying Nonce in Required Confidentiality.
8. In the Message Parts section, select the message parts keyword.
9. Click **OK** to save the configuration changes.

   **Note:** These configurations on the consumer side and the generator side must match.

   In addition to the nonce, you can specify that the timestamp element is checked for encryption. For more information, see the following articles:
   - “Adding time stamps for confidentiality in consumer security constraints with keywords” on page 640

- “Adding time stamps for confidentiality in consumer security constraints with an XPath expression” on page 645

After you specify which message parts to check for encryption, you must specify which method is used to verify the encryption. For more information, see “Configuring encryption information for the consumer binding with an assembly tool” on page 637.

**Related concepts**

“XML digital signature” on page 570
XML-Signature Syntax and Processing (XML digital signature) is a specification that defines XML syntax and processing rules to sign and verify digital signatures for digital content. The specification was developed jointly by the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF).

**Related tasks**

“Adding a stand-alone time stamp in consumer security constraints” on page 651

“Adding time stamps for confidentiality in consumer security constraints with keywords” on page 640

“Adding time stamps for confidentiality in consumer security constraints with an XPath expression” on page 645

“Configuring encryption information for the consumer binding with an assembly tool” on page 637

***Configuring encryption information for the generator binding with an assembly tool:***

Prior to completing this task, you must complete the following steps:

1. Import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF.
2. Specify which message parts to encrypt. For more information, see “Encrypting the message elements in generator security constraints with keywords” on page 623 or “Encrypting the message elements in generator security constraints with an XPath expression” on page 629.
3. Configure the key information that is referenced by the Key information element within the Encryption information dialog window. For more information, see “Configuring key information for the generator binding with an assembly tool” on page 596.

Complete the following steps to configure the encryption information for the server-side and client-side bindings using an assembly tool. The encryption information on the generator side is used for encrypting an outgoing Simple Object Access protocol (SOAP) message. The request generator is configured for the client and the response generator is configured for the server. In the following steps, you must configure either the client-side bindings in step 2 or the server-side bindings in step 3.

1. Start the assembly tool and click **Window > Open Perspective > J2EE**.
2. **Optional:** Locate the client-side bindings using the Project Explorer window. The Client Deployment Descriptor window is displayed. This Web service contains the bindings that you need to configure. Complete the following steps to locate the client-side bindings:
   a. Expand the Web Services > Client section and double-click the name of the Web service.
   b. Click the **WS Binding** tab and expand the Security Request Generator Binding Configuration section.
3. **Optional:** Locate the server-side bindings using the Project Explorer window. The Web Services Editor window is displayed. This Web service contains the bindings that you need to configure. Complete the following steps to locate the server-side bindings:
   a. Expand the Web Services > Services section and double-click the name of the Web service.
   b. Click the **Binding Configurations** tab and expand the Response Generator Binding Configuration Details section.
4. Expand the Encryption Information section and click **Add** to add a new entry or select an existing entry and click **Edit**. The Encryption Information Dialog window is displayed. Complete the following steps to specify an encyrption information configuration:

a. Specify a name for the encryption information configuration in the Encryption name field. For example, you might specify `gen_encinfo`.
b. Select a data encryption algorithm from the Data encryption method algorithm field. This specifies the algorithm used to encrypt parts of the message. The following preconfigured algorithms are supported:
   - http://www.w3.org/2001/04/xmlenc#tripledes-cbc
   - http://www.w3.org/2001/04/xmlenc#aes128-cbc
   - http://www.w3.org/2001/04/xmlenc#aes256-cbc

     To use this algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.
   - http://www.w3.org/2001/04/xmlenc#aes192-cbc

     To use this algorithm, you must download the unrestricted JCE policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.

   This algorithm must match the data encryption algorithm that is configured for the consumer. For more information on configuring the encryption information for the consumer, see "Configuring encryption information for the consumer binding with an assembly tool" on page 637.
c. Select a key encryption algorithm from the Key encryption method algorithm field. This algorithm is used to encrypt the keys. The following pre-configured algorithms are supported:
   - http://www.w3.org/2001/04/xmlenc#rsa-1_5
   - http://www.w3.org/2001/04/xmlenc#kw-tripledes
   - http://www.w3.org/2001/04/xmlenc#kw-aes128
   - http://www.w3.org/2001/04/xmlenc#kw-aes256

     To use this algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.
   - http://www.w3.org/2001/04/xmlenc#kw-aes192

     To use this algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.

   Select the blank entry if the data encryption key, which is the key that is used for encrypting the message parts, is not encrypted. The key encryption algorithm for the generator and the consumer must match. For more information on configuring the encryption information for the generator, see "Configuring encryption information for the generator binding with an assembly tool" on page 648.
d. Specify a name in the Key information name field. For example, you might specify `gen_ekeyinfo`.
e. Select a key information element in the Key information element field. The value in this field references the key information configuration that you specified previously. If you have a key information configuration called `gen_enckeyinfo` that you want to use with this encryption information configuration, specify `get_enckeyinfo` in the Key information element field. For more information, see "Configuring key information for the generator binding with an assembly tool" on page 596.
f. Select a confidentiality part in the Confidentiality part field. The value in this field specifies the name of the confidentiality element that is encrypted.

5. Click **OK** to save your encryption information configuration.

After you complete this task for the consumer binding, you must configure the encryption information for consumer binding. For more information, see "Configuring encryption information for the consumer binding with an assembly tool" on page 637.

**Related tasks**

"Encrypting the message elements in generator security constraints with keywords" on page 623

"Encrypting the message elements in generator security constraints with an XPath expression" on page 629

"Configuring key information for the generator binding with an assembly tool" on page 596

"Configuring encryption information for the consumer binding with an assembly tool" on page 637

***Adding a stand-alone time stamp to generator security constraints:***

Prior to completing this task, you must import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF.

The timestamp determines if the message is valid based upon the time that the message is sent by one machine and then received by another machine.

Complete the following steps to specify a stand-alone time stamp when you configure the generator security constraints for either the request generator or the response generator. The request generator is configured for the client and the response generator is configured for the server. In the following steps, you must configure either the client-side extensions in step 2 or the server-side extensions in step 3.

1. Start the assembly tool and click **Window > Open Perspective > J2EE**.
2. **Optional:** Locate the client-side extensions using the Project Explorer window. The Client Deployment Descriptor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the client-side extensions:
   a. Expand the Web Services > Client section and double-click the name of the Web service.
   b. Click the **WS Extension** tab and expand the Request Generator Configuration section.
3. **Optional:** Locate the server-side extensions using the Project Explorer window. The Web Services Editor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the server-side extensions:
   a. Expand the Web Services > Services section and double-click the name of the Web service.
   b. Click the **Extensions** tab and expand the Response Generator Service Configuration Details section.
4. Expand the Add Timestamp section and select the **Use Add Timestamp** option. When you select this option, a time stamp is added to the message that is sent.
5. Specify an expiration time for the time stamp, which helps defend against replay attacks. Complete the following steps to configure the time stamp:
   a. Expand the Expires subsection within the Add Timestamp section.
   b. Select the **Use Expires** option.
   c. Specify an expiration time for the time stamp. The lexical representation for the duration is the [ISO 8601] extended format `PnYnMnDTnHnMnS`, where:

      **P** Precedes the date and time values.

      **nY** Represents the number of years in which the time stamp is in effect. Select a value from 0 to 99 years.

      **nM** Represents the number of months in which the time stamp is in effect. Select a value from 0 to 11 months.

      **nD** Represents the number of days in which the time stamp is in effect. Select a value from 0 to 30 days.

      **T** Separates the date and time values.

      **nH** Represents the number of hours in which the time stamp is in effect. Select a value from 0 to 23 hours.

**nM** Represents the number of minutes in which the time stamp is in effect. Select a value from 0 to 59 minutes.

**nS** Represents the number of seconds in which the time stamp is in effect. The number of seconds can include decimal digits to arbitrary precision. You can select a value from 0 to 59 for the seconds and from 0 to 9 for tenths of a second.

For example, 1 year, 2 months, 3 days, 10 hours, and 30 minutes is represented as `P1Y2M3DT10H30M`. Typically, you might configure a message time stamp for between 10 and 30 minutes. For example, 10 minutes is represented as `P0Y0M0DT0H10M0S`.

***Adding a stand-alone time stamp in consumer security constraints:***

Prior to completing this task, you must import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF.

The timestamp determines if the message is valid based upon the time that the message is sent by one machine and then received by another machine.

Complete the following steps to specify a stand-alone time stamp when you configure the consumer security constraints for either the response consumer or the request consumer. The response consumer is configured for the client and the request consumer is configured for the server. In the following steps, you must configure either the client-side extensions in step 2 or the server-side extensions in step 3.

1. Start the assembly tool and click **Window > Open Perspective > J2EE**.
2. **Optional:** Locate the client-side extensions using the Project Explorer window. The Client Deployment Descriptor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the client-side extensions:
   a. Expand the Web Services > Client section and double-click the name of the Web service.
   b. Click the **WS Extension** tab and expand the Response Consumer Configuration section.
3. **Optional:** Locate the server-side extensions using the Project Explorer window. The Web Services Editor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the server-side extensions:
   a. Expand the Web Services > Services section and double-click the name of the Web service.
   b. Click the **Extensions** tab and expand the Request Consumer Service Configuration Details section.
4. Expand the Add Timestamp section and select **Use Add Timestamp** option. When you select this option, a time stamp is added to the message that is sent.
5. Specify an expiration time for the time stamp, which helps defend against replay attacks. Complete the following steps to configure the time stamp:
   a. Expand the Expires subsection within the Add Timestamp section.
   b. Select the **Use Expires** option.
   c. Specify an expiration time for the time stamp. The lexical representation for the duration is the [ISO 8601] extended format `PnYnMnDTnHnMnS`, where:

      **P** Precedes the date and time values.

      **nY** Represents the number of years in which the time stamp is in effect. Select a value from 0 to 99 years.

      **nM** Represents the number of months in which the time stamp is in effect. Select a value from 0 to 11 months.

      **nD** Represents the number of days in which the time stamp is in effect. Select a value from 0 to 30 days.

      **T** Separates the date and time values.

**nH** Represents the number of hours in which the time stamp is in effect. Select a value from 0 to 23 hours.

**nM** Represents the number of minutes in which the time stamp is in effect. Select a value from 0 to 59 minutes.

**nS** Represents the number of seconds in which the time stamp is in effect. The number of seconds can include decimal digits to aribrary precision. You can select a value from 0 to 59 for the seconds and from 0 to 9 for tenths of a second.

For example, 1 year, 2 months, 3 days, 10 hours, and 30 minutes is represented as `P1Y2M3DT10H30M`. Typically, you might configure a message time stamp for between 10 and 30 minutes. For example, 10 minutes is represented as `P0Y0M0DT0H10M0S`.

***Security token:***

A security token represents a set of claims made by a client that might include a name, password, identity, key, certificate, group, privilege, and so on.

Web services security provides a general-purpose mechanism to associate security tokens with messages for single message authentication. A specific type of security token is not required by Web services security. Web services security is designed to be extensible and support multiple security token formats to accommodate a variety of authentication mechanisms. For example, a client might provide proof of identity and proof of a particular business certification. However, the security token usage for Web services security is defined in separate profiles such as the Username token profile, the X.509 token profile, the Security Assertion Markup Language (SAML) token profile, the eXtensible rights Markup Language (XrML) token profile, the Kerberos token profile and so on.

A security token is embedded in the Simple Object Access Protocol (SOAP) message within the SOAP header. The security token within the SOAP header is propagated from the message sender to the intended message receiver. On the receiving side, the WebSphere Application Server security handler authenticates the security token and sets up the caller identity on the running thread.

WebSphere Application Server Version 6 contains an enhanced security token that has the following features:

- The client can send multiple tokens to downstream servers.
- The receiver can determine which security token to use for authorization based upon the type or signed part for X.509 tokens.
- You can use the custom token for digital signing or encryption.

**Related concepts**

"Username token element" on page 673
You can use the UsernameToken element to propagate a user name and, optionally, password information. Also, you can use this token type to carry basic authentication information. Both a user name and a password are used to authenticate the message. A UsernameToken containing the user name is used in identity assertion, which establishes the identity of the user based on the trust relationship.

"Binary security token" on page 740
The ValueType attribute identifies the type of the security token, for example, a Lightweight Third Party Authentication (LTPA) token. The EncodingType type indicates how the security token is encoded, for example, Base64Binary. The BinarySecurityToken element defines a security token that is binary encoded. The encoding is specified using the EncodingType attribute. The value type and space are specified using the ValueType attribute. The Web services security implementation for WebSphere Application Server, Version 6 supports both LTPA and X.509 certificate binary security tokens.

"XML token" on page 526
XML tokens are offered in two well-known formats called Security Assertion Markup Language (SAML) and eXtensible rights Markup Language (XrML).

***Configuring the security token in generator security constraints:***

Prior to completing this task, you must import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF.

A security token represents a set of claims that are made by a client and might include a name, password, identity, key, certificate, group, privilege, and so on. It is embedded in the Simple Object Access protocol (SOAP) message within the SOAP header. WebSphere Application Server propagates the security token within the SOAP header from the message sender to the intended message receiver. On the receiving side, the security handler for WebSphere Application Server authenticates the security token and sets up the caller identity on the thread.

Complete the following steps to specify the security token when you configure the request generator or the response generator. The request generator is configured for the client and the response generator is configured for the server. In the following steps, you must configure either the client-side extensions in step 2 or the server-side extensions in step 3.

1. Start the assembly tool and click **Window > Open Perspective > J2EE**.
2. **Optional:** Locate the client-side extensions using the Project Explorer window. The Client Deployment Descriptor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the client-side extensions:
   a. Expand the Web Services > Client section and double-click the name of the Web service.
   b. Click the **WS Extension** tab and expand the Request Generator Configuration section.
3. **Optional:** Locate the server-side extensions using the Project Explorer window. The Web Services Editor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the server-side extensions:
   a. Expand the Web Services > Services section and double-click the name of the Web service.
   b. Click the **Extensions** tab and expand the Response Generator Service Configuration Details section.
4. Expand the Security Token section. The Security Token Dialog window is displayed.
5. Click **Add** to configure the security token. Complete the following steps to configure the security token:
   a. Specify a name for the security token in the Name field. For example, you might specify `un_token`.
   b. Select a token type from the Token type field. For example, if you wish to send a username token, select Username for the token type. If you select a token type other than custom token, you do not need to specify values in the Uniform Resource Identifier (URI) and Local name fields. These fields are automatically specified when you select a token type other than custom token.

      For a username token or an X.509 token, you do not need to specify a value in the URI field.
   c. Optional: Specify a value for the URI and Local name fields if you are configuring a custom token. For example, you might specify http://www.ibm.com/custom in the URI field and CustomToken in the Local name field.
6. Click **OK** to save the configuration changes.

When you configure the token generator, select the security token that you created using these steps. For more information, see “Configuring token generators with an assembly tool” on page 587.

**Related concepts**

“Security token” on page 652
A security token represents a set of claims made by a client that might include a name, password, identity, key, certificate, group, privilege, and so on.

**Related tasks**

“Configuring token generators with an assembly tool” on page 587

***Configuring the security token requirement in consumer security constraints:***

Prior to completing this task, you must import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF.

A security token represents a set of claims that are made by a client and might include a name, password, identity, key, certificate, group, privilege, and so on. It is embedded in the Simple Object Access protocol (SOAP) message within the SOAP header. WebSphere Application Server propagates the security token within the SOAP header from the message sender to the intended message receiver. On the receiving side, the security handler for WebSphere Application Server authenticates the security token and sets up the caller identity on the thread.

Complete the following steps to specify the security token when you configure the response consumer or the request consumer. The response consumer is configured for the client and the request consumer is configured for the server. In the following steps, you must configure either the client-side extensions in step 2 or the server-side extensions in step 3.

1. Start the assembly tool and click **Window > Open Perspective > J2EE**.
2. **Optional:** Locate the client-side extensions using the Project Explorer window. The Client Deployment Descriptor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the client-side extensions:
   a. Expand the Web Services > Client section and double-click the name of the Web service.
   b. Click the **WS Extension** tab and expand the Response Consumer Configuration section.
3. **Optional:** Locate the server-side extensions using the Project Explorer window. The Web Services Editor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the server-side extensions:
   a. Expand the Web Services > Services section and double-click the name of the Web service.
   b. Click the **Extensions** tab and expand the Request Consumer Service Configuration Details section.
4. Expand the Required Security Token section. The Required Security Token Dialog window is displayed.
5. Click **Add** to configure the security token. Complete the following steps to configure the security token:
   a. Specify a name for the security token in the Name field. For example, you might specify `un_token`.
   b. Select a token type from the Token type field. For example, if you want to send a username token, select Username for the token type. If you select a token type other than custom token, you do not need to specify values in the Uniform Resource Identifier (URI) and Local name fields. These fields are automatically specified when you select a token type other than custom token.

      For a username token or an X.509 token, you do not need to specify a value in the URI field.
   c. Optional: Specify a value for the URI and Local name fields if you are configuring a custom token. For example, you might specify http://www.ibm.com/custom in the URI field and CustomToken in the Local name field.
   d. Specify a usage type in the Usage type field. This field specifies the requirement for the confidentiality element. The value of this attribute is either Required or Optional.
6. Click **OK** to save the configuration changes.

When you configure the token consumer, select the security token that you created using these steps. For more information, see “Configuring token consumers with an assembly tool” on page 613.

**Related tasks**

“Configuring token consumers with an assembly tool” on page 613

***Configuring the caller in consumer security constraints:***

Prior to completing this task, you must import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF.

The caller is used to identify the token. The run time for Web services security uses this token identity to create the security credential and principal for WebSphere Application Server. The token identity must be in the configured user registry so that the Application Server can use the token identity in Java 2 Platform, Enterprise Edition (J2EE) authorization checks.

Complete the following steps to specify the caller part when you configure the consumer security constraints for either the response consumer or the request consumer. The response consumer is configured for the client and the request consumer is configured for the server. In the following steps, you must configure either the client-side extensions in step 2 or the server-side extensions in step 3.

1. Start the assembly tool and click **Window > Open Perspective > J2EE**.
2. **Optional:** Locate the client-side extensions using the Project Explorer window. The Client Deployment Descriptor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the client-side extensions:
   a. Expand the Web Services > Client section and double-click the name of the Web service.
   b. Click the **WS Extension** tab and expand the Response Consumer Configuration section.
3. **Optional:** Locate the server-side extensions using the Project Explorer window. The Web Services Editor window is displayed. This Web service contains the extensions that you need to configure. Complete the following steps to locate the server-side extensions:
   a. Expand the Web Services > Services section and double-click the name of the Web service.
   b. Click the **Extensions** tab and expand the Request Consumer Service Configuration Details section.
4. Expand the Caller Part section.
5. Click **Add** to specify the caller part. The Caller Part Dialog window is displayed. Complete the following steps to configure the caller part:
   a. Specify the name of the caller in the Name field.
   b. Optional: Specify the name of an integrity or confidentiality part in the Required Integrity or Required Confidentiality part field if you want to select the token that used for either digital signature or encryption as the caller token. For more information on these configurations, see the following tasks:
      - “Signing message elements in consumer security constraints with keywords” on page 600
      - “Signing message elements in consumer security constraints with an XPath expression” on page 606
      - “Encrypting message elements in consumer security constraints with keywords” on page 639
      - “Encrypting message elements in consumer security constraints with an XPath expression” on page 643

      **Important:** Either complete this step or specify a token type in the Token type field in the next step.
   c. Optional: Specify a token type in the Token type field if you want to select a stand-alone security token as the caller token.

      If a stand-alone security token is used for authentication, then the Uniform Resource Identifier (URI) and local name attributes must define the type of security token that is used for authentication. You can select standard or custom security tokens by URI and local name.

      If you specify a token type in the Token type field, complete the following steps:
      1) Specify the namespace URI of the security token that is used for authentication in the URI field.
      2) Specify the local name of the security token that is used for authentication in the Local name field. The following table shows the URI and local name combinations that are supported:

*Table 13. URI and Local name combinations*

| URI | Local name |
|---|---|
| X.509 certificate token | http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3 |
| X.509 certificates in a PKIPath | http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509PKIPathv1 |
| A list of X509 certificates and CRLs in a PKCS#7 | http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#PKCS7 |
| http://www.ibm.com/websphere/appserver/tokentype/5.0.2 | LTPA |

The Custom token requires that you specify both the URI and the Local name.

6. **Optional:** Configure identity assertion. For more information, see "Configuring identity assertion"
7. **Optional:** Click **Add** and specify a Trust method property in the Trust method property section, if necessary.
8. **Optional:** Click **Add** and specify an additional property in the Property section, if necessary.
9. Click **OK** to save the configuration changes.

**Note:** These configurations on the consumer side and the generator side must match.

**Related tasks**

"Signing message elements in consumer security constraints with keywords" on page 600

"Signing message elements in consumer security constraints with an XPath expression" on page 606

"Encrypting message elements in consumer security constraints with keywords" on page 639

"Encrypting message elements in consumer security constraints with an XPath expression" on page 643

"Configuring identity assertion"

***Configuring identity assertion:***

Prior to completing this task, you must import your application into an assembly tool. For information on how to import your application, see ″Importing enterprise applications″ in the ″Developing and deploying applications″ PDF.

Identity assertion is one of the WebSphere Application Server Version 6 enhancements, but it must be used in a secured environment such as a Virtual Private Network (VPN) or HTTPs. In a secure environment, it is possible to send the requester identity without credentials with other trusted credentials such as the server identity. With identity assertion, WebSphere Application server supports the following types of trust modes:

**None** Specifies that a trusted credential is not attached to the Simple Object Access protocol (SOAP) message

**BasicAuth**
Specifies that a username token with a user name and a password is used as a trusted credential

**Signature**
Specifies that an X.509 certificate security token is used in the digital signature

The specific configuration for identity assertion is necessary on the consumer side in a service configuration only. On the generator side, you need to configure two token generators in a client configuration: one for a requester token and one for a token of a trusted party.

Complete the following steps to configure an application for identity assertion. You must configure both the consumer and the generator to complete the configuration.

1. Start the assembly tool and click **Window > Open Perspective > J2EE**.
2. Expand the Web Services > Services section in the Project Explorer and double-click the name of the Web service.
3. Click the **Extensions** tab and expand the Response Consumer Service Configuration Details > Caller Part section to configure the caller token.
4. Configure the caller token for the consumer. Complete the following steps to configure the caller token for the consumer:
   a. Click **Add** to configure the caller part. The Caller Part Dialog window is displayed. In this window, configure both a token that is used as a caller (requester) credential and a token for the trusted party.
   b. Specify a name for the caller token in the Name field.
   c. Select the type of caller token in Token type field. For example, you can select **Username** if a username token is used as the caller token. When you select the token type, the Local name is automatically specified.
   d. Optional: If you select the **Custom token** in the Token type field, you must specify the Local name and the Uniform Resource Identifier (URI) of the custom token. The URI field is used only for a custom token.
   e. Optional: If the caller token is also used as a certificate of a required integrity or confidentiality part, select the name of the part in Integrity or Confidentiality part field. The list contains the names of the integrity and confidentiality parts that are defined in the Required Integrity and Required Confidentiality sections for the consumer. For example, when an X.509 certificate token is used for both a caller token and a signature certificate of the body element, you can select **X.509 certificate token** in the Token type field and select **reqint_body1** in Integrity or Confidentiality part field. This example assumes that **reqint_body1** is a required integrity configuration.
5. Configure a trusted party token for the consumer. Complete the following steps to configure the trusted party token:
   a. Select the **Use IDAssertion** option to associate a trust method with this caller and to verify an asserted identity from the intermediary (caller).
   b. Select the name of the trust method in the Trust method name field. The following selections are supported:

      **None** Select this option to specify that a trusted credential is not attached to the SOAP message.

      **BasicAuth**
      Select this option to specify that a username token with a user name and password is used as a trusted credential.

      **Signature**
      Select this option to specify that an X.509 certificate security token is used in the digital signature.

      When you select either BasicAuth or Signature, the URI and the Local name fields are automatically specified.
   c. Optional: Select a name of an integrity or confidentiality part in the Integrity or Confidentiality part field if you require digital signature or encryption by the trusted party token. For example, if you select **Signature** in the Trust method name field and you require that the trusted party token signs the body element, select **reqint_body2** in Integrity and Confidentiality part field. This example assumed that **reqint_body2** is a required integrity configuration.
6. **Optional:** If you select **BasicAuth** or **Signature** in the Trust method name field, specify a trusted ID evaluator in Token Consumer Dialog window of the binding configuration. Complete the following steps to specify a trusted ID evaluator:
   a. Click **Binding Configurations** in the Web services editor.

b. Expand the Token Consumer section and click **Add**.
c. Click the **Use trusted ID evaluator** option.
d. Specify a class name in the Trusted ID evaluator class field. The class implements the com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator interface and validates a trusted party token. WebSphere Application Server provides the com.ibm.wsspi.wssecurity.id.TrustedIDEvaluatorImpl class, which is a sample implementation of the TrustedIDEvaluator interface. If you use this class, specify `com.ibm.wsspi.wssecurity.id.TrustedIDEvaluatorImpl` in Trusted ID evaluator class field and click **Add** to add the following trusted ID evaluator property:
   - In the name field, specify `trustedid`
   - In the value field, specify `CN=Alice,O=IBM,C=US`

   The value of the property is the distinguished name (DN) of the username or X.509 certificate token of the trusted party token.
e. Click **OK** to save the configuration.

7. Expand the Web Services > Client section in the Project Explorer and double-click the name of the Web service.
8. Click the **WS Extension** tab and expand the Request Generator Configuration > Security Token section.
9. Specify the caller token for the generator. Do not specify a token in the required token if the token is used for signing or encryption. However, you must specify a token in the required token for a stand-alone token. A stand-alone token is a token that is not used for signing or encryption. When the caller token type is a username token or an X.509 certificate token and it is not used for signing or encryption, specify a security token for this caller token.
   a. Click **Add** to configure a security token.
   b. Specify a name for the caller token in the Name field.
   c. Select either the **Username** or **X.509 certificate token** option in the Token type field. After you select one of these two options, a value for the Local name field is automatically defined.
   d. Click **OK** to save the configuration.
   e. Click the **WS Binding** tab and expand the Security Request Generator Binding Configuration > Token Generator section.
   f. Click **Add** and add the token generator configuration for the caller token.
   g. Click **OK** to save the configuration.
10. Configure the trusted party token. When the trust mode, which was specified previously, is None only the caller token is attached and you do not need to specify the security token of the trusted party. When the trust mode is BasicAuth or Signature you need to specify a username token or an X.509 certificate token of the trusted party token. However, if the X.509 certificate token of trusted party is used for digital signing or encryption as well, you do not need to specify the security token of the trusted party. Complete the following steps to configure the trusted party token:
    a. Expand the Web Services > Client section in the Project Explorer and double-click the name of the Web service.
    b. Click the **WS Extension** tab and expand the Request Generator Configuration > Security Token section.
    c. Click **Add** to configure a security token.
    d. Specify a name for the trusted party token in the Name field.
    e. Select either the **Username** or **X.509 certificate token** option in the Token type field. After you select one of these two options, a value for the Local name field is automatically defined.
    f. Click **OK** to save the configuration.
    g. Click the **WS Binding** tab and expand the Security Request Generator Binding Configuration > Token Generator section.
    h. Click **Add** and add the token generator configuration for the trusted party token.

i. Click **OK** to save the configuration.

Your environment is configured for identity assertion.

**Related tasks**

"Configuring token generators with an assembly tool" on page 587

## Configuring trust anchors for the generator binding on the application level

This document describes how to configure trust anchors for the generator binding at the application level. It does not describe how to configure trust anchors at the server or cell level. Trust anchors defined at the application level have a higher precedence over trust anchors defined at the server or cell level. For more information on creating and configuring trust anchors on the server or cell level, see "Configuring trust anchors on the server or cell level" on page 761.

You can configure a trust anchor for the application-level trust anchor using an Application Server Toolkit or the administrative console. This document describes how to configure the application-level trust anchor using the administrative console.

A trust anchor specifies key stores that contain trusted root certificates, which validate the signer certificate. These key stores are used by the request generator and the response generator (when Web services is acting as client) to generate the signer certificate for the digital signature. The keystores are critical to the integrity of the digital signature validation. If they are tampered with, the result of the digital signature verification is doubtful and comprised. Therefore, it is recommended that you secure these keystores. The binding configuration specified for the request generator must match the binding configuration for the response generator.

The trust anchor configuration for the request generator on the client must match the configuration for the request consumer on the server. Also, the trust anchor configuration for the response generator on the server must match the configuration for the response consumer on the client.

Complete the following steps to configure trust anchors for the generator binding on the application level:

1. Locate the trust anchor panel in the administrative console.
   a. Click **Applications > Enterprise applications >** *application_name*.
   b. Under Related Items, click **EJB Modules** or **Web Modules >** *URI_name*.
   c. Under Additional Properties you can access the trust anchor configuration for the following bindings:
      - For the request generator (sender) binding, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom**.
      - For the response generator (sender) binding, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**.
   d. Under Additional properties, click **Trust anchors**.
   e. Click **New** to create a trust anchor configuration, click **Delete** to delete an existing configuration, or click the name of an existing trust anchor configuration to edit its settings. If you are creating a new configuration, enter a unique name in the Trust anchor name field.
2. Specify the keystore password, the keystore location, and the keystore type. Key store files contain public and private keys, root certificate authority (CA) certificates, the intermediate CA certificate, and so on. Keys retrieved from the keystore are used to sign and validate or encrypt and decrypt messages or message parts. If you specified the com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator implementation for the key locator class implementation, you must specify a key store password, location, and type.
   a. Specify a password in the Key store password field. This password is used to access the keystore file.
   b. Specify the location of the key store file in the Key store path field.

c. Select a keystore type from the Key store type field. The Java Cryptography Extension (JCE) used by IBM supports the following key store types:

**JKS**
Use this option if you are not using Java Cryptography Extensions (JCE) and if your keystore file uses the Java Keystore (JKS) format.

**JCEKS**
Use this option if you are using Java Cryptography Extensions.

**PKCS11KS (PKCS11)**
Use this format if your keystore uses the PKCS#11 file format. Keystores using this format might contain RSA keys on cryptographic hardware or might encrypt keys that use cryptographic hardware to ensure protection.

**PKCS12KS (PKCS12)**
Use this option if your keystore uses the PKCS#12 file format.

WebSphere Application Server provides some sample keystore files in the `${USER_INSTALL_ROOT}/etc/ws-security/samples` directory. For example, you might use the `enc-receiver.jceks` keystore file for encryption keys. The password for this file is Storepass and the type is JCEKS.

**Attention:** Do not use these keystore files in a production environment. These samples are provided for testing purposes only.

This task configures trust anchors for the generator binding at the application level.

You must specify a similar trust anchor configuration for the consumer.

***Trust anchor collection:***

Use this page to view a list of keystore objects that contain trusted root certificates. These objects are used for certificate path validation of incoming X.509-formatted security tokens. Keystore objects within trust anchors contain trusted root certificates that are used by the CertPath API to validate the trust of a certificate chain.

To create the keystore file, use the key tool that is located in the *install_dir*`\java\jre\bin\keytool` directory.

To view this administrative console page for trust anchors on the cell level, complete the following steps:

1. Click **Security > Web services**.
2. Under Additional properties, click **Trust anchors**.

To view this administrative console page for trust anchors on the server level, complete the following steps:

1. Click **Servers > Application servers >***server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Trust anchors**.

To view this administrative console page for trust anchors on the application level,

1. Click **Applications > Enterprise applications** > *application_name*.
2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.
3. **6.x application** Under Additional properties, you can access trust anchors information for the following bindings:
   - For the Response consumer (receiver) binding, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**.

- For the Request consumer (receiver) binding, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**.

4. **5.x application** Under Additional properties, you can access the trust anchors information for the following bindings:
   - For the Response receiver binding, click **Web services: Client security bindings**. Under Response receiver binding, click **Edit**.
   - For the Request receiver binding, click **Web services: Server security bindings**. Under Request receiver binding, click **Edit**.
5. Under Additional properties, click **Trust anchors**.

If you click **Update runtime**, the Web services security run time is updated with the default binding information, which is contained in the `ws-security.xml` file that was previously saved. If you make changes on this panel, you must complete the following steps:

1. Save your changes by clicking **Save** at the top of the administrative console. When you click **Save**, you are returned to the administrative console home panel.
2. Return to the Trust anchors collection panel and click **Update runtime**. When you click **Update runtime**, the configuration changes made to the other Web services also are updated in the Web services security run time.

**Related reference**

"Trust anchor configuration settings"
Use this information to configure a trust anchor. Trust anchors point to keystores that contain trusted root or self-signed certificates. This information enables you to specify a name for the trust anchor and the information that is needed to access a keystore. The application binding uses this name to reference a predefined trust anchor definition in the binding file (or the default).

*Trust anchor name:*

Specifies the unique name that is used to identify the trust anchor.

*Key store path:*

Specifies the location of the keystore file that contains the trust anchors.

*Key store type:*

Specifies the type of keystore file.

The value for this field is **JKS**, **JCEKS**, **PKCS11KS (PKCS11)**, or **PKCS12KS (PKCS12)**.

***Trust anchor configuration settings:***

Use this information to configure a trust anchor. Trust anchors point to keystores that contain trusted root or self-signed certificates. This information enables you to specify a name for the trust anchor and the information that is needed to access a keystore. The application binding uses this name to reference a predefined trust anchor definition in the binding file (or the default).

To view this administrative console page for trust anchors on the cell level, complete the following steps:

1. Click **Security > Web services**.
2. Under Additional properties, click **Trust anchors**.
3. Click **New** to create a trust anchor or click the name of an existing configuration to modify its settings.

To view this administrative console page for trust anchors on the server level, complete the following steps:

1. Click **Servers > Application servers >***server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Trust anchors**.
4. Click **New** to create a trust anchor or click the name of an existing configuration to modify its settings.

To view this administrative console page for trust anchors on the application level,

1. Click **Applications > Enterprise applications** > *application_name*.
2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.
3. **6.x application** Under Additional properties, you can access trust anchors information for the following bindings:
   - For the Response consumer (receiver) binding, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**.
   - For the Request consumer (receiver) binding, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**.
4. **5.x application** Under Additional properties, you can access the trust anchors information for the following bindings:
   - For the Response receiver binding, click **Web services: Client security bindings**. Under Response receiver binding, click **Edit**.
   - For the Request receiver binding, click **Web services: Server security bindings**. Under Request receiver binding, click **Edit**.
5. Under Additional properties, click **Trust anchors**.
6. Click **New** to create a trust anchor or click the name of an existing configuration to modify its settings.

**Related reference**

"Trust anchor collection" on page 660
Use this page to view a list of keystore objects that contain trusted root certificates. These objects are used for certificate path validation of incoming X.509-formatted security tokens. Keystore objects within trust anchors contain trusted root certificates that are used by the CertPath API to validate the trust of a certificate chain.

*Trust anchor name:*

Specifies the unique name that is used by the application binding to reference a predefined trust anchor definition in the default binding.

*Key store password:*

Specifies the password that is needed to access the key store file.

*Key store path:*

Specifies the location of the keystore file.

Use ${USER_INSTALL_ROOT} as this path expands to the WebSphere Application Server path on your machine.

*Key store type:*

Specifies the type of keystore file.

Choose from the following options:

**JKS** Use this option if you are not using Java Cryptography Extensions (JCE).

**JCEKS**
Use this option if you are using Java Cryptography Extensions.

**PKCS11KS (PKCS11)**
Use this format if your keystore uses the PKCS#11 file format. Keystores that use this format might contain Rivest Shamir Adleman (RSA) keys on cryptographic hardware or might encrypt keys that use cryptographic hardware to ensure protection.

**PKCS12KS (PKCS12)**
Use this option if your keystore uses the PKCS#12 file format.

| | |
|---|---|
| **Default** | JKS |
| **Range** | JKS, JCEKS, PKCS11KS (PKCS11), PKCS12KS (PKCS12) |

## Configuring the collection certificate store for the generator binding on the application level

A collection certificate store is a collection of non-root, certificate authority (CA) certificates and certificate revocation lists (CRLs). This collection of CA certificates and CRLs is used to check for a valid signature in a digitally signed Simple Object Access Protocol (SOAP) message. Complete the following steps to configure a collection certificate for the generator bindings on the application level:

1. Locate the collection certificate store configuration panel in the administrative console.
   a. Click **Applications > Enterprise applications >** *application_name*.
   b. Under Related Items, click **EJB Modules** or **Web Modules >** *URI_name*.
   c. Under Additional Properties you can access the key information for the request generator and response generator bindings.
      - For the request generator (sender) binding, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom**.
      - For the response generator (sender) binding, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**.
   d. Under Additional properties, click **Collection certificate store**.
2. Specify the Certificate store name. Click **New** to create a collection certificate store configuration, select the box next to the configuration and click **Delete** to delete an existing configuration, .or click the name of an existing collection certificate store configuration to edit its settings. If you are creating a new configuration, enter a name in the Certificate store name field.

   The name of the collection certificate store must be unique to the level of the application server. For example, if you create the collection certificate store for the application level, the store name must be unique to the application level. The name that is specified in the Certificate store name field is used by other configurations to refer to a predefined collection certificate store. WebSphere Application Server searches for the collection certificate store based on proximity.

   For example, if an application binding refers to a collection certificate store named `cert1`, the Application Server searches for `cert1` at the application level before searching the server level.
3. Specify a certificate store provider in the Certificate store provider field. WebSphere Application Server supports the IBMCertPath certificate store provider. To use another certificate store provider, you must define the provider implementation in the provider list within the *install_dir*`/java/jre/lib/security/java.security` file. However, make sure that your provider supports the same requirements of the certificate path algorithm as WebSphere Application Server.
4. Click **OK** and **Save** to save the configuration.
5. Click the name of your certificate store configuration. After you specify the certificate store provider, you must specify either the location of a certificate revocation list or the X.509 certificates. However, you can specify both a certificate revocation list and the X.509 certificates for your certificate store configuration.

6. Under Additional properties, click **Certificate revocation lists**.
7. Click **New** to specify a certificate revocation list path, click **Delete** to delete an existing list reference, or click the name of an existing reference to edit the path. You must specify the fully qualified path to the location where WebSphere Application Server can find your list of certificates that are not valid. For portability reasons, it is recommended that you use the WebSphere Application Server variables to specify a relative path to the certificate revocation lists (CRL). This recommendation is especially important when you are working in a WebSphere Application Server Network Deployment environment. For example, you might use the *USER_INSTALL_ROOT* variable to define a path such as $*USER_INSTALL_ROOT*/mycertstore/mycrl1. For a list of supported variables, click **Environment > WebSphere variables** in the administrative console. The following list provides recommendation for using certificate revocation lists:
   - If CRLs are added to the collection certificate store, add the CRLs for the root certificate authority and each intermediate certificate, if applicable. When the CRL is in the certificate collection store, the certificate revocation status for every certificate in the chain is checked against the CRL of the issuer.
   - When the CRL file is updated, the new CRL does not take effect until you restart the Web service application.
   - Before a CRL expires, you must load a new CRL into the certificate collection store to replace the old CRL. An expired CRL in the collection certificate store results in a certificate path (CertPath) build failure.
8. Click **OK** and **Save** to save the configuration.
9. Return to the collection certificate store configuration panel. To access the panel, complete the following steps:
   a. Click **Applications > Enterprise applications >** *application_name*.
   b. Under Related Items, click **EJB Modules** or **Web Modules >** *URI_name*.
   c. Under Additional Properties you can access the key information for the request generator and response generator bindings.
      - For the request generator (sender) binding, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom**.
      - For the response generator (sender) binding, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**.
   d. Under Additional properties, click **Collection certificate store >** *certificate_store_name*.
10. Under Additional properties, click **X.509 certificates**.
11. Click **New** to create a X.509 certificate configuration, click **Delete** to delete an existing configuration, or click the name of an existing X.509 certificate configuration to edit its settings. If you are creating a new configuration, enter a name in the Certificate store name field.
12. Specify a path in the X.509 certificate path field. This entry is the absolute path to the location of the X.509 certificate. The collection certificate store is used to validate the certificate path of incoming X.509-formatted security tokens.

    You can use the *USER_INSTALL_ROOT* variable as part of path name. For example, you might type: `USER_INSTALL_ROOT/etc/ws-security/samples/intca2.cer`. Do not use this certificate path for production use. You must obtain your own X.509 certificate from a certificate authority before putting your WebSphere Application Server environment into production.

    Click **Environment > WebSphere variables** in the administrative console to configure the *USER_INSTALL_ROOT* variable.
13. Click **OK** and then **Save** to save your configuration.

You have configured the collection certificate store for the generator binding.

You must specify a similar collection certificate store configuration for the consumer.

**Related tasks**

“Configuring the collection certificate store for the consumer binding on the application level” on page 738

***Collection certificate store collection:***

Use this page to view a list of certificate stores that contains untrusted, intermediary certificate files awaiting validation. Validation might consist of checking to see if the certificate is on a certificate revocation list (CRL), checking that the certificate is not expired, and checking that the certificate is issued by a trusted signer.

The following list provides recommendations for using CRLs:
- If CRLs are added to the collection certificate store collection, add the CRLs for the root certificate authority and each intermediate certificate, if applicable. When the CRL is in the certificate collection store, the certificate revocation status for every certificate in the chain is checked against the CRL of the issuer.
- When the CRL file is updated, the new CRL does not take effect until you restart the Web service application.
- Before a CRL expires, you must load a new CRL into the certificate collection store to replace the old CRL. An expired CRL in the collection certificate store results in a certificate path (CertPath) build failure.

To view the administrative console panel for the collection certificate store on the server level, complete the following steps:
1. Click **Servers > Application servers >** *server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Collection certificate store**.

To view this administrative console page for the collection certificate store on the application level, complete the following steps:
1. Click **Applications > Enterprise applications** > *application_name*.
2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.
3. **6.x application** Under Additional properties, you can access collection certificate stores for the following bindings:
   - For the Request generator, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom > Collection certificate store**.
   - For the Request consumer, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom > Collection certificate store**.
   - For the Response generator, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom > Collection certificate store**.
   - For the Response consumer, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom > Collection certificate store**.
4. **5.x application** Under Additional properties, you can access collection certificate stores for the following bindings:
   - For the Request receiver binding, click **Web services: Server security bindings**. Under Response receiver binding, click **Edit > Collection certificate store**.
   - For the Response receiver binding, click **Web services: Client security bindings**. Under Response receiver binding, click **Edit > Collection certificate store**.

Complete the following steps:
1. Click **New** to specify a new certificate store name and certificate store provider.
2. Click **OK** and messages display at the top of the administrative console panel.

3. Within the messages at the top of the administrative console panel, click **Save**.
4. Return to the collection certificate store collection panel and click **Update runtime** to update the Web services security run time with the default binding information, which is found in the `ws_security.xml` file. When you click **Update runtime**, the configuration changes made to the other Web services are also updated in the Web services security run time.

**Related reference**

"Collection certificate store configuration settings"
Use this page to specify the name and the provider for a collection certificate store. A *collection certificate store* is a collection of nonroot, certificate authority (CA) certificates and certificate revocation lists (CRLs). This collection of CA certificates and CRLs is used to check the signature of a digitally signed Simple Object Access Protocol (SOAP) message.

"X.509 certificates collection" on page 668
Use this page to view a list of untrusted, intermediate certificate files. This collection certificate store is used for certificate path validation of incoming X.509-formatted security tokens.

"X.509 certificate configuration settings" on page 669
Use this page to specify a list of untrusted, intermediate certificate files. This collection certificate store is used for certificate path validation of incoming X.509-formatted security tokens.

**6.x application** "Certificate revocation list collection" on page 671
Use this page to determine the location of the certificate revocation lists (CRL) known to WebSphere Application Server. The Application Server checks the CRLs to determine the validity of the client certificate. A certificate that is found in a certificate revocation list might not be expired, but is no longer trusted by the certificate authority (CA) that issued the certificate. The CA might add the certificate to the certificate revocation list if it believes that the client authority is compromised.

**6.x application** "Certificate revocation list configuration settings" on page 672
Use this page to specify a list of certificate revocations that check the validity of a certificate. The application server checks the certificate revocation lists (CRL) to determine the validity of the client certificate. A certificate that is found in a certificate revocation list might not be expired, but is no longer trusted by the certificate authority (CA) that issued the certificate. The CA might add the certificate to the certificate revocation list if it believes that the client authority is compromised.

*Certificate store name:*

Specifies the name of the certificate store.

*Certificate store provider:*

Specifies the provider of the certificate store.

***Collection certificate store configuration settings:***

Use this page to specify the name and the provider for a collection certificate store. A *collection certificate store* is a collection of nonroot, certificate authority (CA) certificates and certificate revocation lists (CRLs). This collection of CA certificates and CRLs is used to check the signature of a digitally signed Simple Object Access Protocol (SOAP) message.

To view the administrative console panel for the collection certificate store on the server level, complete the following steps:

1. Click **Servers > Application servers >** *server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Collection certificate store**.
4. Specify a new collection certificate store by clicking **New** or by clicking the collection certificate store name to modify its settings.

To view this administrative console page for the collection certificate store on the application level, complete the following steps:

1. Click **Applications > Enterprise applications** > *application_name*.
2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.
3. **6.x application** Under Additional properties, you can access collection certificate stores for the following bindings:
   - For the Request generator, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom > Collection certificate store**.
   - For the Request consumer, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom > Collection certificate store**.
   - For the Response generator, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom > Collection certificate store**.
   - For the Response consumer, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom > Collection certificate store**.
4. **5.x application** Under Additional properties, you can access collection certificate stores for the following bindings:
   - For the Request receiver binding click **Edit > Collection certificate store**.
   - For the Response receiver binding, click **Edit > Collection certificate store**.
5. Specify a new collection certificate store by clicking **New** or by clicking the collection certificate store name to modify its settings.

After configuring a collection certificate store, you can select the new configuration under Certificate store on the token generator and token consumer panels. To access these panels, complete the following steps:

1. Click **Security > Web services**.
2. Under Default generator bindings, click **Token generators** or under Default consumer bindings, click **Token consumers**.
3. Click **New** to create a new token generator or token consumer, or click the name of an existing configuration to make modifications.

After you configure your collection certificate store on this panel, you must click **Apply** before configuring either the certificate revocation list or an X.509 certificate. The certificate revocation list configuration is not available for version 5.x applications through the administrative console. After you configure your certificate revocation list or X.509 certificate, complete the following steps:

1. Click **Save**, at the top of the administrative console panel, which returns you to the list of the configured collection certificate stores.
2. Click **Update runtime** to update the Web services security run time with the default binding information, which is found in the `ws_security.xml` file.

   **Related reference**

   "Collection certificate store collection" on page 665
   Use this page to view a list of certificate stores that contains untrusted, intermediary certificate files awaiting validation. Validation might consist of checking to see if the certificate is on a certificate revocation list (CRL), checking that the certificate is not expired, and checking that the certificate is issued by a trusted signer.

   "X.509 certificates collection" on page 668
   Use this page to view a list of untrusted, intermediate certificate files. This collection certificate store is used for certificate path validation of incoming X.509-formatted security tokens.

   "X.509 certificate configuration settings" on page 669
   Use this page to specify a list of untrusted, intermediate certificate files. This collection certificate store is used for certificate path validation of incoming X.509-formatted security tokens.

   "Certificate revocation list collection" on page 671
   Use this page to determine the location of the certificate revocation lists (CRL) known to WebSphere

Application Server. The Application Server checks the CRLs to determine the validity of the client certificate. A certificate that is found in a certificate revocation list might not be expired, but is no longer trusted by the certificate authority (CA) that issued the certificate. The CA might add the certificate to the certificate revocation list if it believes that the client authority is compromised.

"Certificate revocation list configuration settings" on page 672
Use this page to specify a list of certificate revocations that check the validity of a certificate. The application server checks the certificate revocation lists (CRL) to determine the validity of the client certificate. A certificate that is found in a certificate revocation list might not be expired, but is no longer trusted by the certificate authority (CA) that issued the certificate. The CA might add the certificate to the certificate revocation list if it believes that the client authority is compromised.

*Certificate store name:*

Specifies the name for the certificate store.

The name of the collection certificate store must be unique in the scope. For example, the name must be unique at the server level. The name specified in **Certificate store name** field is used by other configurations to refer to a pre-defined collection certificate store. For example, the application binding refers to a collection certificate store that is defined on the server level. WebSphere Application Server looks up the collection certificate store based on proximity. For example, if *cert1* is defined as the name of the certificate store on the cell and server levels and *cert1* is referenced in the application binding, the application server uses the server-level collection certificate store.

*Certificate Store Provider:*

Specifies the provider for the certificate store implementation.

WebSphere Application Server supports the `IBMCertPath` certificate path provider. If you need to use another certificate path provider, define the provider implementation in the provider list within the `java.security` file in the Software Development Kit (SDK).

| | |
|---|---|
| **Data type** | String |
| **Default** | IBMCertPath |

***X.509 certificates collection:***

Use this page to view a list of untrusted, intermediate certificate files. This collection certificate store is used for certificate path validation of incoming X.509-formatted security tokens.

To view the administrative console panel for the collection certificate store on the server level, complete the following steps:

1. Click **Servers > Application servers >** *server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Collection certificate store**.
4. Click the name of a configured collection certificate store or create a new collection certificate store first.
5. Under Additional properties, click **X.509 certificates**.

To view this administrative console page for an X.509 certificate on the application level, complete the following steps:

1. Click **Applications > Enterprise applications** > *application_name*.
2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.

3. **6.x application** Under Additional properties, you can access collection certificate stores for the following bindings:
   - For the Request generator, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom > Collection certificate store**.
   - For the Request consumer, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom > Collection certificate store**.
   - For the Response generator, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom > Collection certificate store**.
   - For the Response consumer, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom > Collection certificate store**.
4. **5.x application** Under Additional properties, you can access the collection certificate stores for the following bindings.
   - For the Response receiver binding, click **Web services: Client security bindings**. Under Response receiver binding, click **Edit > Collection certificate store**.
   - For the Request receiver binding, click **Web services: Server security bindings**. Under Request receiver binding, click **Edit > Collection certificate store**.
5. Click the name of a configured collection certificate store or create a new collection certificate store first.
6. Under Additional properties, click **X.509 certificates**.

   **Related reference**

   "X.509 certificate configuration settings"
   Use this page to specify a list of untrusted, intermediate certificate files. This collection certificate store is used for certificate path validation of incoming X.509-formatted security tokens.

   "Collection certificate store collection" on page 665
   Use this page to view a list of certificate stores that contains untrusted, intermediary certificate files awaiting validation. Validation might consist of checking to see if the certificate is on a certificate revocation list (CRL), checking that the certificate is not expired, and checking that the certificate is issued by a trusted signer.

   "Collection certificate store configuration settings" on page 666
   Use this page to specify the name and the provider for a collection certificate store. A *collection certificate store* is a collection of nonroot, certificate authority (CA) certificates and certificate revocation lists (CRLs). This collection of CA certificates and CRLs is used to check the signature of a digitally signed Simple Object Access Protocol (SOAP) message.

*X.509 certificate path:*

Specifies the location of the X.509 certificate.

***X.509 certificate configuration settings:***

Use this page to specify a list of untrusted, intermediate certificate files. This collection certificate store is used for certificate path validation of incoming X.509-formatted security tokens.

To view the administrative console panel for the collection certificate store on the server level, complete the following steps:

1. Click **Servers > Application servers >** *server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Collection certificate store**.
4. Click the name of a configured collection certificate store or create a new collection certificate store first.
5. Under Additional properties, click **X.509 certificates**.

6. Specify a new X.509 certificate path by clicking **New** or by clicking the X.509 certificate path to modify its settings.

To view this administrative console page for an X.509 certificate on the application level, complete the following steps:

1. Click **Applications > Enterprise applications** > *application_name*.
2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.
3. **6.x application** Under Additional properties, you can access collection certificate stores for the following bindings:
   - For the Request generator, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom > Collection certificate store**.
   - For the Request consumer, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom > Collection certificate store**.
   - For the Response generator, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom > Collection certificate store**.
   - For the Response consumer, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom > Collection certificate store**.
4. **5.x application** Under Additional properties, you can access the collection certificate stores for the following bindings.
   - For the Response receiver binding, click **Web services: Client security bindings**. Under Response receiver binding, click **Edit > Collection certificate store**.
   - For the Request receiver binding, click **Web services: Server security bindings**. Under Request receiver binding, click **Edit > Collection certificate store**.
5. Click the name of a configured collection certificate store or create a new collection certificate store first.
6. Under Additional properties, click **X.509 certificates**.
7. Specify a new X.509 certificate path by clicking **New** or click the X.509 certificate path to modify its settings.

**Related tasks**

“Managing digital certificates” on page 443

**Related reference**

“X.509 certificates collection” on page 668
Use this page to view a list of untrusted, intermediate certificate files. This collection certificate store is used for certificate path validation of incoming X.509-formatted security tokens.

“Collection certificate store collection” on page 665
Use this page to view a list of certificate stores that contains untrusted, intermediary certificate files awaiting validation. Validation might consist of checking to see if the certificate is on a certificate revocation list (CRL), checking that the certificate is not expired, and checking that the certificate is issued by a trusted signer.

“Collection certificate store configuration settings” on page 666
Use this page to specify the name and the provider for a collection certificate store. A *collection certificate store* is a collection of nonroot, certificate authority (CA) certificates and certificate revocation lists (CRLs). This collection of CA certificates and CRLs is used to check the signature of a digitally signed Simple Object Access Protocol (SOAP) message.

*X.509 Certificate Path:*

Specifies the absolute path to the location of the X.509 certificate.

As shown in the following example, you can use the USER_INSTALL_ROOT variable as part of the path name: `{USER_INSTALL_ROOT}/etc/ws-security/samples/intca2.cer`. This X.509 certificate path is not for production use. Obtain your own X.509 from a certificate authority before putting your WebSphere Application Server environment into production.

You can configure the USER_INSTALL_ROOT variable in the administrative console by clicking **Environment > WebSphere Variables**.

***Certificate revocation list collection:***

Use this page to determine the location of the certificate revocation lists (CRL) known to WebSphere Application Server. The Application Server checks the CRLs to determine the validity of the client certificate. A certificate that is found in a certificate revocation list might not be expired, but is no longer trusted by the certificate authority (CA) that issued the certificate. The CA might add the certificate to the certificate revocation list if it believes that the client authority is compromised.

To view the administrative console panel for the collection certificate store on the server level, complete the following steps:

1. Click **Servers > Application servers >** *server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Collection certificate store**.
4. Click the name of a configured collection certificate store or create a new collection certificate store first.
5. Under Additional properties, click **Certificate revocation lists**.

**6.x application** To view this administrative console page for the collection certificate store on the application level, complete the following steps:

1. Click **Applications > Enterprise applications** > *application_name*.
2. Under Related items, click **EJB modules Web modules** > *URI_name*.
3. Under Additional properties, you can access collection certificate stores for the following bindings:
   - For the Request generator, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom > Collection certificate store**.
   - For the Request consumer, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom > Collection certificate store**.
   - For the Response generator, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom > Collection certificate store**.
   - For the Response consumer, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom > Collection certificate store**.
4. Click the name of a configured collection certificate store or create a new collection certificate store first.
5. Under Additional properties, click **Certificate revocation lists**.
6. Under Additional properties, you can access collection certificate stores for the following bindings:
   - For the Response receiver binding, click **Web services: Client security bindings**. Under Response receiver binding, click **Edit**.
7. Under Additional properties, click **Collection certificate store >***certificate_store_name*.
8. Under Additional properties, click **X.509 certificates**.
9. Click **New** and specify the path to the certificate revocation list.

**5.x application** To add a certificate revocation list for a version 5.x application, complete the following steps:

1. Click **Applications > Enterprise applications** > *application_name*.

2. Under Related items, click **EJB modules Web modules** > *URI_name*.

**Related reference**

"Certificate revocation list configuration settings"
Use this page to specify a list of certificate revocations that check the validity of a certificate. The application server checks the certificate revocation lists (CRL) to determine the validity of the client certificate. A certificate that is found in a certificate revocation list might not be expired, but is no longer trusted by the certificate authority (CA) that issued the certificate. The CA might add the certificate to the certificate revocation list if it believes that the client authority is compromised.

"Collection certificate store collection" on page 665
Use this page to view a list of certificate stores that contains untrusted, intermediary certificate files awaiting validation. Validation might consist of checking to see if the certificate is on a certificate revocation list (CRL), checking that the certificate is not expired, and checking that the certificate is issued by a trusted signer.

"Collection certificate store configuration settings" on page 666
Use this page to specify the name and the provider for a collection certificate store. A *collection certificate store* is a collection of nonroot, certificate authority (CA) certificates and certificate revocation lists (CRLs). This collection of CA certificates and CRLs is used to check the signature of a digitally signed Simple Object Access Protocol (SOAP) message.

*Certificate revocation list path:*

Specifies the location where you can find the list of certificates that are not valid.

***Certificate revocation list configuration settings:***

Use this page to specify a list of certificate revocations that check the validity of a certificate. The application server checks the certificate revocation lists (CRL) to determine the validity of the client certificate. A certificate that is found in a certificate revocation list might not be expired, but is no longer trusted by the certificate authority (CA) that issued the certificate. The CA might add the certificate to the certificate revocation list if it believes that the client authority is compromised.

To view the administrative console panel for the collection certificate store on the server level, complete the following steps:

1. Click **Servers > Application servers >** *server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Collection certificate store**.
4. Click the name of a configured collection certificate store or create a new collection certificate store first.
5. Under Additional properties, click **Certificate revocation lists > New** to specify the path to a new list or click the name of a certificate revocation list to modify its path.

To view this administrative console page for the collection certificate store on the application level, complete the following steps:

1. Click **Applications > Enterprise applications** > *application_name*.
2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.
3. Under Additional properties, you can access collection certificate stores for the following bindings:
   - For the Request generator, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom > Collection certificate store**.
   - For the Request consumer, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom > Collection certificate store**.
   - For the Response generator, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom > Collection certificate store**.

- For the Response consumer, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom > Collection certificate store**.

4. Click the name of a configured collection certificate store or create a new collection certificate store first.
5. Under Additional properties, click **Certificate revocation lists > New** to specify the path to a new list or click the name of a certificate revocation list to modify its path.

**Related reference**

"Certificate revocation list collection" on page 671
Use this page to determine the location of the certificate revocation lists (CRL) known to WebSphere Application Server. The Application Server checks the CRLs to determine the validity of the client certificate. A certificate that is found in a certificate revocation list might not be expired, but is no longer trusted by the certificate authority (CA) that issued the certificate. The CA might add the certificate to the certificate revocation list if it believes that the client authority is compromised.

"Collection certificate store collection" on page 665
Use this page to view a list of certificate stores that contains untrusted, intermediary certificate files awaiting validation. Validation might consist of checking to see if the certificate is on a certificate revocation list (CRL), checking that the certificate is not expired, and checking that the certificate is issued by a trusted signer.

"Collection certificate store configuration settings" on page 666
Use this page to specify the name and the provider for a collection certificate store. A *collection certificate store* is a collection of nonroot, certificate authority (CA) certificates and certificate revocation lists (CRLs). This collection of CA certificates and CRLs is used to check the signature of a digitally signed Simple Object Access Protocol (SOAP) message.

*Certificate revocation list path:*

Specifies a fully qualified path to the location where you can find the list of certificates that are not valid.

For portability reasons, it is recommended that you use the WebSphere Application Server variables to specify a relative path to the certificate revocation list. This recommendation is especially important when you are working in a WebSphere Application Server Network Deployment environment. For example, you might use the *USER_INSTALL_ROOT* variable to define a path such as `$USER_INSTALL_ROOT/mycertstore/mycrl` where *mycertstore* represents the name of your certificate store and *mycrl* represents the certificate revocation list. For a list of the supported variables, click **Environment > WebSphere variables** in the administrative console.

The following list provides recommendations for using CRLs:

- If CRLs are added to the collection certificate store collection, add the CRLs for the root certificate authority and each intermediate certificate, if applicable. When the CRL is in the certificate collection store, the certificate revocation status for every certificate in the chain is checked against the CRL of the issuer.
- When the CRL file is updated, the new CRL does not take effect until you restart the Web service application.
- Before a CRL expires, you must load a new CRL into the certificate collection store to replace the old CRL. An expired CRL in the collection certificate store results in a certificate path (CertPath) build failure.

## Username token element

You can use the UsernameToken element to propagate a user name and, optionally, password information. Also, you can use this token type to carry basic authentication information. Both a user name and a password are used to authenticate the message. A UsernameToken containing the user name is used in identity assertion, which establishes the identity of the user based on the trust relationship.

The following example shows the syntax of the UsernameToken element:

```
<wsse:UsernameToken wsu:Id="Example-1">
   <wsse:Username>
   ...
   </wsse:Username>
   <wsse:Password Type="...">
   ...
   </wsse:Password>
   <wsse:Nonce EncodingType="...">
   ...
   </wsse:Nonce>
   <wsu:Created>
   ...
   </wsu:Created>
</wsse:UsernameToken>
```

The Web services security specification defines the following password types:

**wsse:PasswordText (default)**
: This type is the actual password for the user name.

**wsse:PasswordDigest**
: The type is the digest of the password for the user name. The value is a base64-encoded SHA1 hash value of the UTF8-encoded password.

WebSphere Application Server supports the default PasswordText type. However, it does not support password digest because most user registry security policies do not expose the password to the application software.

The following example illustrates the use of the `<UsernameToken>` element:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
            xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
    <S:Header>
            ...
        <wsse:Security>
            <wsse:UsernameToken>
                <wsse:Username>Joe</wsse:Username>
                <wsse:Password>ILoveJava</wsse:Password>
            </wsse:UsernameToken>
        </wsse:Security>
    </S:Header>
</S:Envelope>
```

**Related concepts**

“Nonce, a randomly generated token” on page 567
*Nonce* is a randomly generated, cryptographic token used to prevent replay attacks. Although Nonce can be inserted anywhere in the SOAP message, it is typically inserted in the <UsernameToken> element.

“Binary security token” on page 740
The ValueType attribute identifies the type of the security token, for example, a Lightweight Third Party Authentication (LTPA) token. The EncodingType type indicates how the security token is encoded, for example, Base64Binary. The BinarySecurityToken element defines a security token that is binary encoded. The encoding is specified using the EncodingType attribute. The value type and space are specified using the ValueType attribute. The Web services security implementation for WebSphere Application Server, Version 6 supports both LTPA and X.509 certificate binary security tokens.

“XML token” on page 526
XML tokens are offered in two well-known formats called Security Assertion Markup Language (SAML) and eXtensible rights Markup Language (XrML).

“Security token” on page 652
A security token represents a set of claims made by a client that might include a name, password, identity, key, certificate, group, privilege, and so on.

## Nonce, a randomly generated token

*Nonce* is a randomly generated, cryptographic token used to prevent replay attacks. Although Nonce can be inserted anywhere in the SOAP message, it is typically inserted in the <UsernameToken> element.

Without nonce, when a user name token is passed from one machine to another machine using a non-secure transport, such as HTTP, the token might be intercepted and used in a replay attack. The same password might be reused when the username token is transmitted between the client and the server, which leaves it vulnerable to attack. The user name token can be stolen even if you use XML digital signature and XML encryption.

To help eliminate these replay attacks, the `<wsse:Nonce>` and `<wsu:Created>` elements are generated within the `<wsse:UsernameToken>` element and used to validate the message. The server checks the freshness of the message by verifying that the difference between the nonce creation time, which is specified by the <wsu:Created> element, and the current time falls within a specified time period. Also, the server checks a cache of used nonces to verify that the username token in the received Simple Object Access Protocol (SOAP) message has not been processed within the specified time period. These two features are used to lessen the chance that a user name token is used for a replay attack.

To add nonce for the username token, you can specify it in the token generator for the username token. When the token generator for the username token is specified, you can select the **Add nonce** option if you want to include nonce in the username token.

**Related concepts**

"Distributed nonce caching" on page 763
The *distributed nonce caching* feature enables you to distribute the cache for a nonce to different servers in a cluster.

## Configuring the token generator on the application level

This task describes the steps that are needed to specify the token generators at the application level. The information is used on the generator side to generate the security token.

Complete the following steps to configure the token generator on the application level:

1. Locate the token generator panel in the administrative console.
   a. Click **Applications > Enterprise applications >** *application_name*.
   b. Under Related Items, click **EJB Modules** or **Web Modules >** *URI_name*.
   c. Under Additional Properties you can access the token generators for the following bindings:
      - For the request generator (sender) binding, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom**.
      - For the response generator (sender) binding, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**.
   d. Under Additional properties, click **Token generators**.
   e. Click **New** to create a token generator configuration, select the box next to an existing configuration and click **Delete** to delete an existing configuration, or click the name of an existing token generator configuration to edit its settings. If you are creating a new configuration, enter a unique name in the Token generator name field. For example, you might specify `gen_signtgen`.
2. Specify a class name in the Token generator class name field. The token generator class must implement the com.ibm.wsspi.wssecurity.token.TokenGeneratorComponent interface. The token generator class name for the request generator and the response generator must be similar to the token consumer class name for the request consumer and the response consumer. For example, if your application requires a username token consumer, you can specify the com.ibm.wsspi.wssecurity.token.UsernameTokenConsumer class name on the token consumer panel for the application level and the com.ibm.wsspi.wssecurity.token.UsernameTokenGenerator class name in this field.

3. **Optional:** Select a part reference in the Part reference field. The part reference indicates the name of the security token that is defined in the deployment descriptor.

   **Important:** On the application level, if you do not specify a security token in your deployment descriptor, the Part reference field is not displayed. If you define a security token called `user_tgen` in your deployment descriptor, `user_tgen` is displayed as an option in the Part reference field. You can specify a security token in the deployment descriptor when you assemble your application using an assembly tool.
4. Select either **None** or **Dedicated signing information** for the certificate path. Select **None** when the token generator does not use the PKCS#7 token type. When the token generator uses the PKCS#7 token type and you want to package certificate revocation lists (CRLs) in the security token, select **Dedicated signing information** and select a certificate store. To configure a collection certificate store and certificate revocation lists for the generator bindings on the application level, complete the following steps:
   a. Click **Applications > Enterprise applications >** *application_name*.
   b. Under Related Items, click **EJB Modules** or **Web Modules >** *URI_name*.
   c. Under Additional Properties you can access the collection certificate store configuration for the following bindings:
      - For the request generator (sender) binding, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom**.
      - For the response generator (sender) binding, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**.
   d. Under Additional properties, click **Collection certificate store**.

   For more information about configuring a collection certificate store, see "Configuring the collection certificate store for the generator binding on the application level" on page 663.
5. **Optional:** Select the **Add nonce** option. This option indicates whether a nonce is included in the user name token for the token generator. Nonce is a unique, cryptographic number that is embedded in a message to help stop repeat, unauthorized attacks of user name tokens. The **Add nonce** option is valid only when the generated token type is a user name token and is available only for the request generator binding.

   If you select the **Add nonce** option, you can specify the following properties under Additional properties. These properties are used by the request consumer.

*Table 14. Additional nonce properties*

| Property name | Default value | Explanation |
| --- | --- | --- |
| `com.ibm.ws.wssecurity.config.token.BasicAuth.Nonce.cacheTimeout` | 600 seconds | Specifies the timeout value, in seconds, for the nonce value that is cached on the server. |
| `com.ibm.ws.wssecurity.config.token.BasicAuth.Nonce.clockSkew` | 0 seconds | Specifies the time, in seconds, before the nonce time stamp expires. |
| `com.ibm.ws.wssecurity.config.token.BasicAuth.Nonce.maxAge` | 300 seconds | Specifies the clock skew value, in seconds, to consider when WebSphere Application Server checks the timeliness of the message. |

   On the server level, you can specify these additional properties for a nonce on the Default bindings for Web services security panel within the administrative console. To access the panell, click **Servers > Application servers >***server_name*. Under Security, click **Web services: Default bindings for Web services security**.
6. **Optional:** Select the **Add timestamp** option. This option indicates whether to insert a time stamp into the user name token. The **Add timestamp** option is valid only when the generated token type is a user name token and is available only for the request generator binding.

7. Specify the value type local name in the Local name field. For a user name token and an X.509 certificate security token, WebSphere Application Server provides predefined local names for the value type. When you specify any of the following local names, you do not need to specify a value type URI:

   **http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#UsernameToken**
   This local name specifies a user name token.

   **http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509**
   This local name specifies an X.509 certificate token.

   **http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509PKIPathv1**
   This local name specifies X.509 certificates in a public key infrastructure (PKI) path.

   **http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#PKCS7**
   This local name specifies a list of X.509 certificates and certificate revocation lists in a PKCS#7 format.

   For an LTPA token, you can use `LTPA` for the value type local name and `http://www.ibm.com/websphere/appserver/tokentype/5.0.2` for the value type Uniform Resource Identifier (URI).
8. **Optional:** Specify the value type URI in the URI field. This entry specifies the namespace URI of the value type for the generated token.
9. Click **OK** and **Save** to save the configuration.
10. Click the name of your token generator configuration.
11. Under Additional properties, click **Callback handler**.
12. Specify the settings for the callback handler.
    a. Specify a class name in the Callback handler class name field. This class name is the name of the callback handler implementation class that is used to plug-in a security token framework. The specified callback handler class must implement the javax.security.auth.callback.CallbackHandler interface and must provide a constructor using the following syntax:

       `MyCallbackHandler(String *username*, char[] *password*, java.util.Map *properties*)`

       Where:

       **username**
       Specifies the user name that is passed into the configuration.

       **password**
       Specifies the password that is passed into the configuration.

       **properties**
       Specifies the other configuration properties that are passed into the configuration.

       This constructor is required if the callback handler needs a user name and a password. However, if the callback handler does not need a user name and a password, such as X509CallbackHandler, use a constructor with the following syntax:

       ```
       MyCallbackHandler(java.util.Map properties)
       ```

       WebSphere Application Server provides the following default callback handler implementations:

       **com.ibm.wsspi.wssecurity.auth.callback.GUIPromptCallbackHandler**
       This callback handler uses a login prompt to gather the user name and password information. However, if you specify the user name and password on this panel, a prompt is not displayed and WebSphere Application Server returns the user name and password to the token generator. Use this implementation for a Java 2 Platform, Enterprise Edition (J2EE) application client only. If you use this implementation, you must provide a basic authentication user ID and password on this panel.

**com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler**
This callback handler does not issue a prompt and returns the user name and password if it is specified on this panel. You can use this callback handler when the Web service is acting as a client. If you use this implementation, you must provide a basic authentication user ID and password on this panel.

**com.ibm.wsspi.wssecurity.auth.callback.StdinPromptCallbackHandler**
This callback handler uses a standard-in prompt to gather the user name and password. However, if the user name and password is specified on this panel, WebSphere Application Server does not issue a prompt, but returns the user name and password to the token generator. Use this implementation for a Java 2 Platform, Enterprise Edition (J2EE) application client only. If you use this implementation, you must provide a basic authentication user ID and password on this panel.

**com.ibm.wsspi.wssecurity.auth.callback.LTPATokenCallbackHandler**
This callback handler is used to obtain the Lightweight Third Party Authentication (LTPA) security token from the Run As invocation Subject. This token is inserted in the Web services security header within the SOAP message as a binary security token. However, if the user name and password are specified on this panel, WebSphere Application Server authenticates the user name and password to obtain the LTPA security token rather than obtaining it from the Run As Subject. Use this callback handler only when the Web service is acting as a client on the application server. It is recommended that you do not use this callback handler on a J2EE application client. If you use this implementation, you must provide a basic authentication user ID and password on this panel.

**com.ibm.wsspi.wssecurity.auth.callback.X509CallbackHandler**
This callback handler is used to create the X.509 certificate that is inserted in the Web services security header within the SOAP message as a binary security token. A keystore and a key definition is required for this callback handler. If you use this implementation, you must provide a key store password, path, and type on this panel.

**com.ibm.wsspi.wssecurity.auth.callback.PKCS7CallbackHandler**
This callback handler is used to create X.509 certificates encoded with the PKCS#7 format. The certificate is inserted in the Web services security header in the SOAP message as a binary security token. A keystore is required for this callback handler. You can specify a certificate revocation list (CRL) in the collection certificate store. The CRL is encoded with the X.509 certificate in the PKCS#7 format. If you use this implementation, you must provide a key store password, path, and type on this panel.

**com.ibm.wsspi.wssecurity.auth.callback.PkiPathCallbackHandler**
This callback handler is used to create X.509 certificates encoded with the PkiPath format. The certificate is inserted in the Web services security header within the SOAP message as a binary security token. A keystore is required for this callback handler. A CRL is not supported by the callback handler; therefore, the collection certificate store is not required or used. If you use this implementation, you must provide a key store password, path, and type on this panel.

The callback handler implementation obtains the required security token and passes it to the token generator. The token generator inserts the security token in the Web services security header within the SOAP message. Also, the token generator is a plug-in point for the pluggable security token framework. Service providers can provide their own implementation, but the implementation must use the com.ibm.wsspi.wssecurity.token.TokenGeneratorComponent interface.

b. Optional: Select the **Use identity assertion** option. Select this option if you have identity assertion defined in the IBM extended deployment descriptor. This option indicates that only the identity of the initial sender is required and inserted into the Web services security header within the SOAP message. For example, WebSphere Application Server sends only the user name of the original caller for a username token generator. For an X.509 token generator, the application server sends the original signer certification only.

c. Optional: Select the **Use RunAs identity** option. Select this option if you have identity assertion defined in the IBM extended deployment descriptor and you want to use the Run As identity instead of the initial caller identity for identity assertion in a downstream call. This option is valid only if you have Username TokenGenerator configured as a token generator.

d. Optional: Specify the basic authentication user ID in the Basic authentication user ID field. This entry specifies the user name that is passed to the constructors of the callback handler implementation. The basic authentication user name and password are used if you specified one of the following default callback handler implementations in the Callback handler class name field:
   - com.ibm.wsspi.wssecurity.auth.callback.GUIPromptCallbackHandler
   - com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler
   - com.ibm.wsspi.wssecurity.auth.callback.StdinPromptCallbackHandler
   - com.ibm.wsspi.wssecurity.auth.callback.LTPATokenCallbackHandler

e. Optional: Specify the basic authentication password in the Basic authentication password field. This entry specifies the password that is passed to the constructors of the callback handler implementation.

f. Optional: Specify the key store password in the Key store password field. This entry specifies the password used to access the key store file. The key store and its configuration are used if you select on of the following default callback handler implementations that are provided by WebSphere Application Server:

   **com.ibm.wsspi.wssecurity.auth.callback.PKCS7CallbackHandler**
   The keystore is used to build the X.509 certificate with the certificate path.

   **com.ibm.wsspi.wssecurity.auth.callback.PkiPathCallbackHandler**
   The keystore is used to build the X.509 certificate with the certificate path.

   **com.ibm.wsspi.wssecurity.auth.callback.X509CallbackHandler**
   The keystore is used to retrieve the X.509 certificate.

g. Optional: Specify the key store path in the Path field. It is recommended that you use the `${USER_INSTALL_ROOT}` in the path name as this variable expands to the WebSphere Application Server path on your machine. To change the path used by this variable, click **Environment > WebSphere variables**, and click **USER_INSTALL_ROOT**. This field is required when you use the com.ibm.wsspi.wssecurity.auth.callback.PKCS7CallbackHandler, com.ibm.wsspi.wssecurity.auth.callback.PkiPathCallbackHandler, or com.ibm.wsspi.wssecurity.auth.callback.X509CallbackHandler callback handler implementations.

h. Optional: Select the key store type in the Type field. This selection indicates the format used by the keystore file. You can select one of the following values for this field:

   **JKS** Use this option if the keystore uses the Java Keystore (JKS) format.

   **JCEKS**
   Use this option if the Java Cryptography Extension is configured in the software development kit (SDK). The default IBM JCE is configured in WebSphere Application Server. This option provides stronger protection for stored private keys by using Triple DES encryption.

   **PKCS11KS (PKCS11)**
   Use this format if your keystore uses the PKCS#11 file format. Keystores using this format might contain RSA keys on cryptographic hardware or might encrypt keys that use cryptographic hardware to ensure protection.

   **PKCS12KS (PKCS12)**
   Use this option if your keystore uses the PKCS#12 file format.

13. Click **OK** and then click **Save** to save the configuration.
14. Click the name of your token generator configuration.
15. Under Additional properties, click **Callback handler > Keys**.

16. Specify the key name, key alias, and the key password.
    a. Click **New** to create a key configuration, click **Delete** to delete an existing configuration, or click the name of an existing key configuration to edit its settings. If you are creating a new configuration, enter a unique name in the key name field. For digital signatures, the key name is used by the request generator or response generator signing information to determine which key is used to digitally sign the message. For encryption, the key name is used to determine the key used for encryption. The key name must be a fully qualified, distinguished name. For example, `CN=Bob,O=IBM,C=US`.
    b. Specify the key alias in the Key alias field. The key alias is used by the key locator to find the key within the keystore file.
    c. Specify the key password in the Key password field. This password is needed to access the key object within the keystore file.
17. Click **OK** and **Save** to save the configuration.

You have configured the token generator for the application level.

You must specify a similar token consumer configuration for the application level.

**Related tasks**

"Configuring the collection certificate store for the generator binding on the application level" on page 663

"Configuring the token generator on the application level" on page 675

***Request generator (sender) binding configuration settings:***

Use this page to specify the binding configuration for the request generator.

To view this administrative console page, complete the following steps:
1. Click **Applications > Enterprise applications >** *application_name*.
2. Under Related items, click **EJB modules** or **Web modules**.
3. Click the Uniform Resource Identifier (URI).
4. Under Additional properties, click **Web services: Client security bindings**.
5. Under Request generator (sender) binding, click **Edit custom**.

The security constraints or bindings are defined using the application assembly process before the application is installed. WebSphere Application Server provides assembly tools to assemble your application.

(Note that an assembly tool is not available on the z/OS platform.)

If the security constraints are defined in the application, you must either define the corresponding binding information or select the Use defaults option on this panel and use the default binding information for the cell or server level. The default binding provided by WebSphere Application Server is a sample. Do not use this sample in a production environment without modifying the configuration. The security constraints define what is signed or encrypted in the Web services security message. The bindings define how to enforce the requirements.

**Digital signature security constraint (integrity)**

The following table shows the required and optional binding information when the digital signature security constraint (integrity) is defined in the deployment descriptor.

| Information type | Required or optional |
| --- | --- |
| Signing information | Required |

| Information type | Required or optional |
| --- | --- |
| Key information | Required |
| Key locators | Optional |
| Collection certificate store | Optional |
| Token generator | Optional |
| Properties | Optional |

You can use the key locators and the collection certificate store that are defined at either the server-level or the cell-level.

**Encryption constraint (confidentiality)**

The following table shows the required and optional binding information when the encryption constraint (confidentiality) is defined in the deployment descriptor.

| Information type | Required or optional |
| --- | --- |
| Encryption information | Required |
| Key information | Required |
| Key locators | Optional |
| Collection certificate store | Optional |
| Token generator | Optional |
| Properties | Optional |

You can use the key locators and the collection certificate store that are defined at either the server-level or the cell-level.

**Security token constraint**

The following table shows the required and optional binding information when the security token constraint is defined in the deployment descriptor.

| Information type | Required or optional |
| --- | --- |
| Token generator | Required |
| Collection certificate store | Optional |
| Properties | Optional |

You can use the collection certificate store that is defined at either the server-level or the cell-level.

**Related reference**

"Request consumer (receiver) binding configuration settings" on page 745
Use this page to specify the binding configuration for the request consumer.

"Response generator (sender) binding configuration settings" on page 682
Use this page to specify the binding configuration for the response generator or response sender.

"Response consumer (receiver) binding configuration settings" on page 747
Use this page to specify the binding configuration for the response consumer.

*Use defaults:*

Select this option if you want to use the default binding information from the server or cell level.

*Component:*

Specifies the enterprise bean in an assembled EJB module.

*Port:*

Specifies the port in the Web service that is defined during application assembly.

*Web service:*

Specifies the name of the Web service that is defined during application assembly.

***Response generator (sender) binding configuration settings:***

Use this page to specify the binding configuration for the response generator or response sender.

To view this administrative console page, complete the following steps:
1. Click **Applications > Enterprise applications >***application_name*.
2. Under Related items, click **EJB modules** or **Web modules**.
3. Click the Uniform Resource Identifier (URI).
4. Under Additional properties, click **Web services: Server security bindings**.
5. Under Response generator (sender) binding, click **Edit custom**.

The security constraints or bindings are defined using the application assembly process before the application is installed. WebSphere Application Server provides assembly tools to assemble your application.

**Note:** An assembly tool is not available on the z/OS platform.

If the security constraints are defined in the application, you must either define the corresponding binding information or select the Use defaults option on this panel and use the default binding information for the server or cell level. The default binding that is provided by WebSphere Application Server is a sample. Do not use this sample in a production environment without modifying the configuration. The security constraints define what is signed or encrypted in the Web services security message. The bindings define how to enforce the requirements.

**Digital signature security constraint (integrity)**

The following table shows the required and optional binding information when the digital signature security constraint (integrity) is defined in the deployment descriptor.

| Information type | Required or optional |
|---|---|
| Signing information | Required |
| Key information | Required |
| Key locators | Optional |
| Collection certificate store | Optional |
| Token generator | Optional |
| Properties | Optional |

You can use the key locators and the collection certificate store that are defined at either the server-level or the cell-level.

**Encryption constraint (confidentiality)**

The following table shows the required and optional binding information when the encryption constraint (confidentiality) is defined in the deployment descriptor.

| Information type | Required or optional |
|---|---|
| Encryption information | Required |
| Key information | Required |
| Key locators | Optional |
| Collection certificate store | Optional |
| Token generator | Optional |
| Properties | Optional |

You can use the key locators and the collection certificate store that are defined at either the server-level or the cell-level.

**Security token constraint**

The following table shows the required and optional binding information when the security token constraint is defined in the deployment descriptor.

| Information type | Required or optional |
|---|---|
| Token generator | Required |
| Collection certificate store | Optional |
| Properties | Optional |

You can use the collection certificate store that is defined at either the server-level or the cell-level.

**Related reference**

"Request generator (sender) binding configuration settings" on page 680
Use this page to specify the binding configuration for the request generator.
"Request consumer (receiver) binding configuration settings" on page 745
Use this page to specify the binding configuration for the request consumer.
"Response consumer (receiver) binding configuration settings" on page 747
Use this page to specify the binding configuration for the response consumer.

*Use defaults:*

Select this option if you want to use the default binding information from the server or cell level.

*Port:*

Specifies the port number in the Web service that is defined during application assembly.

*Web service:*

Specifies the name of the Web service that is defined during application assembly.

***Callback handler configuration settings:***

Use this page to specify how to acquire the security token that is inserted in the Web services security header within the Simple Object Access Protocol (SOAP) message. The token acquisition is a pluggable

framework that leverages the Java Authentication and Authorization Service (JAAS) javax.security.auth.callback.CallbackHandler interface for acquiring the security token.

To view this administrative console page for the callback handler on the server level, complete the following steps:

1. Click **Servers > Application servers >** *server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Default generator bindings, click **Token generators >** *token_generator_name*.
4. Under Additional properties, click **Callback handler**.

To view this administrative console page for the callback handler on the application level , complete the following steps:

1. Click **Applications > Enterprise applications** > *application_name*.
2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.
3. Under Additional properties, you can access the callback handler information for the following bindings:
   - For the Request generator (sender) binding, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom**. Under Additional properties, click **Token generator**. Click **New** to create a new token generator configuration or click the name of an existing configuration to modify its settings. Under Additional properties, click **Callback handler**.
   - For the Response generator (sender) binding, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**. Under Additional properties, click **Token generator**. Click **New** to create a new token generator configuration or click the name of an existing configuration to modify its settings. Under Additional properties, click **Callback handler**.

**Related reference**

"Token generator collection" on page 769
Use this page to view the token generators. The information is used on the generator side only to generate the security token.

"Token generator configuration settings" on page 769
Use this page to specify the information for the token generator. The information is used at the generator side only to generate the security token.

*Callback handler class name:*

Specifies the name of the callback handler implementation class that is used to plug in a security token framework.

The specified callback handler class must implement the javax.security.auth.callback.CallbackHandler class. The implementation of the JAAS javax.security.auth.callback.CallbackHandler interface must provide a constructor using the following syntax:

```
MyCallbackHandler(String username, char[] password, java.util.Map properties)
```

Where:

**username**
: Specifies the user name that is passed into the configuration.

**password**
: Specifies the password that is passed into the configuration.

**properties**
: Specifies the other configuration properties that are passed into the configuration.

WebSphere Application Server provides the following default callback handler implementations:

**5.x and 6.x application com.ibm.wsspi.wssecurity.auth.callback.GUIPromptCallbackHandler**
This callback handler uses a login prompt to gather user name and password information. However, if you specify the user name and password on this panel, a prompt is not displayed and WebSphere Application Server returns the user name and password to the token generator if it is specified on this panel. However, use this implementation for a Java 2 Platform, Enterprise Edition (J2EE) application client only.

**5.x and 6.x application com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler**
This callback handler does not issue a prompt and returns the user name and password if it is specified on this panel. You can use this callback handler when the Web service is acting as a client.

**5.x and 6.x application com.ibm.wsspi.wssecurity.auth.callback.StdinPromptCallbackHandler**
This callback handler uses a standard-in prompt to gather the user name and password. However, if the user name and password is specified on this panel, WebSphere Application Server does not issue a prompt, but returns the user name and password to the token generator. However, use this implementation for a Java 2 Platform, Enterprise Edition (J2EE) application client only.

**5.x and 6.x application com.ibm.wsspi.wssecurity.auth.callback.LTPATokenCallbackHandler**
This callback handler is used to obtain the Lightweight Third Party Authentication (LTPA) security token from the Run As invocation Subject. This token is inserted in the Web services security header within the SOAP message as a binary security token. However, if the user name and password are specified on this panel, WebSphere Application Server authenticates the user name and password to obtain the LTPA security token rather than obtaining it from the Run As Subject. Use this callback handler only when the Web service is acting as a client on the application server. It is recommended that you do not use this callback handler on a J2EE application client.

**6.x application com.ibm.wsspi.wssecurity.auth.callback.X509CallbackHandler**
This callback handler is used to create the X.509 certificate that is inserted in the Web services security header within the SOAP message as a binary security token. A keystore and a key definition is required for this callback handler.

**6.x application com.ibm.wsspi.wssecurity.auth.callback.PKCS7CallbackHandler**
This callback handler is used to create X.509 certificates encoded with the PKCS#7 format. The certificate is inserted in the Web services security header in the SOAP message as a binary security token. A keystore is required for this callback handler. You must specify a certificate revocation list (CRL) in the collection certificate store. The CRL is encoded with the X.509 certificate in the PKCS#7 format.

**6.x application com.ibm.wsspi.wssecurity.auth.callback.PkiPathCallbackHandler**
This callback handler is used to create X.509 certificates encoded with the PkiPath format. The certificate is inserted in the Web services security header within the SOAP message as a binary security token. A keystore is required for this callback handler. A CRL is not supported by the callback handler, hence, the collection certificate store is not required or used.

The callback handler implementation obtains the required security token and passes it to the token generator. The token generator inserts the security token in the Web services security header within the SOAP message. Also, the token generator is plug-in point for the pluggable security token framework. Service providers can provide their own implementation, but the implementation must use the com.ibm.wsspi.wssecurity.token.TokenGeneratorComponent interface.

*Use identity assertion:*

Select this option if you have identity assertion defined in the IBM extended deployment descriptor.

This option indicates that only the identity of the initial sender is required and inserted into the Web services security header within the SOAP message. For example, WebSphere Application Server only sends the user name of the original caller for a Username TokenGenerator. For an X.509 token generator, the application server sends the original signer certification only.

*Use RunAs identity:*

Select this option if you have identity assertion defined in the IBM extended deployment descriptor and you want to use the Run As identity instead of the initial caller identity for identity assertion for a downstream call.

This option is valid only if you have Username TokenGenerator configured as a token generator.

*Basic authentication user ID:*

Specifies the user name that is passed to the constructors of the callback handler implementation.

The basic authentication user name and password are used if you select one of the following default callback handler implementations provided by WebSphere Application Server:

- com.ibm.wsspi.wssecurity.auth.callback.GUIPromptCallbackHandler
- com.ibm.wsspi.wssecurity.auth.callback.LTPATokenCallbackHandler
- com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler
- com.ibm.wsspi.wssecurity.auth.callback.StdinPromptCallbackHandler

These implementations are described in detail under the **Callback handler class name** field description in this article.

*Basic authentication password:*

Specifies the password that is passed to the constructor of the callback handler.

The keystore and its related configuration are used if you select one of the following default callback handler implementations provided by WebSphere Application Server:

**com.ibm.wsspi.wssecurity.auth.callback.PKCS7CallbackHandler**
The keystore is used to build the X.509 certificate with the certificate path.

**com.ibm.wsspi.wssecurity.auth.callback.PkiPathCallbackHandler**
The keystore is used to build the X.509 certificate with the certificate path.

**com.ibm.wsspi.wssecurity.auth.callback.X509CallbackHandler**
The keystore is used to retrieve the X.509 certificate.

*Key store password:*

Specifies the password that is used to access the keystore file.

*Key store path:*

Specifies the location of the keystore file.

Use ${USER_INSTALL_ROOT} in the path name because this variable expands to the WebSphere Application Server path on your machine. To change the path used by this variable, click **Environment > WebSphere variables** and click **USER_INSTALL_ROOT**.

*Key store type:*

Specifies the type of keystore file format

You can choose one of the following values for this field:

**JKS** Use this option if the keystore uses the Java Keystore (JKS) format.

**JCEKS**
Use this option if the Java Cryptography Extension is configured in the software development kit (SDK). The default IBM JCE is configured in WebSphere Application Server. This option provides stronger protection for stored private keys by using Triple DES encryption.

**PKCS11KS (PKCS11)**
Use this option if your keystore file uses the PKCS#11 file format. Keystore files that use this format might contain Rivest Shamir Adleman (RSA) keys on cryptographic hardware or might encrypt keys that use cryptographic hardware to ensure protection.

**PKCS12KS (PKCS12)**
Use this option if your keystore file uses the PKCS#12 file format.

***Key collection:***

Use this page to view a list of logical names that is mapped to a key alias in the keystore file.

To view this administrative console page for the key locator collection on the server level, complete the following steps:

1. Click **Servers > Application servers >** *server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Default generator bindings, click **Token Generators >** *token_generator_name*.
4. Under Additional properties, click **Callback handler > Keys**.

Keys are also available by clicking **Key locators >***key_locator_name*. Under Additional properties, click **Keys**.

To use this administrative console page for the key locator collection on the application level, complete the following steps:

1. Click **Applications > Enterprise applications** > *application_name*.
2. Under Related items, click **EJB modules** > *URI_name*.
3. **6.x application** Under Additional properties, you can access key locators for the following bindings:
   - For the Request generator, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom > Key locators**. Under Additional properties, click **Keys**.
   - For the Request consumer, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom > Key locators**. Under Additional properties, click **Keys**.
   - For the Response generator, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom > Key locators**. Under Additional properties, click **Keys**.
   - For the Response consumer, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom > Key locators**.Under Additional properties, click **Keys**.
4. **5.x application** Under Additional properties, you can access key locators for the following bindings:
   - For the Request sender, click **Web services: Client security bindings**. Under Request sender binding, click **Edit > Key locators**. Under Additional properties, click **Keys**.
   - For the Request receiver, click **Web services: Server security bindings**. Under Request receiver binding, click **Edit > Key locators**. Under Additional properties, click **Keys**.

- For the Response sender, click **Web services: Server security bindings**. Under Response sender binding, click **Edit > Key locators**. Under Additional properties, click **Keys**.
- For the Response receiver, click **Web services: Client security bindings**. Under Response receiver binding, click **Edit > Key locators**. Under Additional properties, click **Keys**.

**Related reference**

"Key locator collection" on page 694
Use this page to view a list of key locator configurations that retrieve keys from the keystore for digital signature and encryption. A key locator must implement the com.ibm.wsspi.wssecurity.config.KeyLocator interface.

"Key locator configuration settings" on page 695
Use this page to specify the settings for a key locator configuration. The key locators retrieve keys from the keystore file for digital signature and encryption. WebSphere Application Server enables you to plug in a custom key locator configuration.

"Key configuration settings"
Use this page to define the mapping of a logical name to a key alias in a keystore file.

*Key name:*

Specifies the name of the key object that is found in the keystore file.

*Key alias:*

Specifies an alias for the key object.

The alias is used when the key locator searches for the key objects in the keystore file.

***Key configuration settings:***

Use this page to define the mapping of a logical name to a key alias in a keystore file.

To view this administrative console page for the key locator collection on the server level, complete the following steps:

1. Click **Servers > Application servers >** *server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Default generator bindings, click **Token Generators >** *token_generator_name*.
4. Under Additional properties, click **Callback handler > Keys**.
5. Specify a new key configuration by clicking **New** or by clicking the key configuration name to modify the settings.

Keys are also available by clicking **Key locators >***key_locator_name*. Under Additional properties, click **Keys > New**.

To use this administrative console page for the key locator collection on the application level, complete the following steps:

1. Click **Applications > Enterprise applications** > *application_name*.
2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.
3. **6.x application** Under Additional properties, you can access key locators for the following bindings:
   - For the Request generator, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom > Key locators**. Under Additional properties, click **Keys**.

- For the Request consumer, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom > Key locators**. Under Additional properties, click **Keys**.
- For the Response generator, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom > Key locators**. Under Additional properties, click **Keys**.
- For the Response consumer, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom > Key locators**.Under Additional properties, click **Keys**.

4. **5.x application** Under Additional properties, you can access key locators for the following bindings:
   - For the Request sender, click **Web services: Client security bindings**. Under Request sender binding, click **Edit > Key locators**. Under Additional properties, click **Keys**.
   - For the Request receiver, click **Web services: Server security bindings**. Under Request receiver binding, click **Edit > Key locators**. Under Additional properties, click **Keys**.
   - For the Response sender, click **Web services: Server security bindings**. Under Response sender binding, click **Edit > Key locators**. Under Additional properties, click **Keys**.
   - For the Response receiver, click **Web services: Client security bindings**. Under Response receiver binding, click **Edit > Key locators**. Under Additional properties, click **Keys**.
5. Specify a new key configuration by clicking **New** or by clicking the key configuration name to modify the settings.

   **Related reference**

   "Key collection" on page 687
   Use this page to view a list of logical names that is mapped to a key alias in the keystore file.

*Key name:*

Specifies the name of the key object. For digital signatures, the key name is used by the request sender or request generator signing information to determine which key is used to digitally sign the message. For encryption, the key name is used to determine the key used for encryption.

The key name must be a fully qualified, distinguished name. For example, `CN:Bob,O=IBM,C=US`.

**Note:** If you enter the distinguished name with spaces before or after commas and equal symbols, WebSphere Application Server normalizes the distinguished names automatically during run time by removing these extra spaces.

*Key alias:*

Specifies the alias for the key object, which is used by the key locator to find the key within the keystore file.

*Key password:*

Specifies the password that is needed to access the key object within the keystore file.

***Web services: Client security bindings collection:***

Use this page to view a list of application-level, client-side binding configurations for Web services security. These bindings are used when a Web service is a client to another Web service.

To view this administrative console page, complete the following steps:

1. Click **Applications > Enterprise Applications >***application_name*.

2. Under Related Items, click **EJB module** or **Web Module >** *URI_file_name*.
3. Under Additional properties, click **Web services: Client security bindings**.

**Related reference**

**6.x application** "Request generator (sender) binding configuration settings" on page 680
Use this page to specify the binding configuration for the request generator.

**6.x application** "Response consumer (receiver) binding configuration settings" on page 747
Use this page to specify the binding configuration for the response consumer.

"Web services: Server security bindings collection" on page 691
Use this page to view a list of server-side binding configurations for Web services security.

*Component:*

Specifies the enterprise bean in an assembled Enterprise JavaBeans (EJB) module.

*Port:*

Specifies the port that is used to send messages to a server and receive messages from a server.

*Web service:*

Specifies the name of the Web service that is defined during application assembly.

*Request generator (sender) binding:*

Specifies the binding configuration that is used to send request messages to the request consumer.

Click **Edit custom** to configure the required and additional properties such as signing information, key information, token generators, key locators, and collection certificate stores.

The binding information for the request generator that is specified for the client must match the binding information for the request consumer that is specified for the server.

*Response consumer (receiver) binding:*

Specifies the binding configuration that is used to receive response messages from the response generator.

Click **Edit custom** to configure the required and additional properties such as signing information, key information, token consumers, key locators, collection certificate stores, and trust anchors.

The binding information for the response consumer that is specified for the client must match the binding information for the response generator that is specified for the server.

*Request sender binding:*

Specifies the binding configuration that is used to send request messages to the request receiver.

Click **Edit** to configure the additional properties for the request sender such as signing information, key information, encryption information, key locators, and the login binding.

The binding information for the request sender that is specified for the client must match the binding information for the request receiver that is specified for the server.

*Response receiver binding:*

Specifies the binding configuration that is used to receive response messages from the response sender.

Click **Edit** to configure the additional properties for the response receiver such as signing information, encryption information, trust anchors, collection certificate stores, and key locators.

The binding information for the response receiver that is specified for the client must match the binding information for the response sender that is specified for the server.

*HTTP basic authentication:*

Specifies the user name and password to use for this port with HTTP transport-level basic authentication. You can enable transport-level authentication security independently of message-level security.

Click **Edit** to configure the basic authentication ID and password for transport-level authentication.

*HTTP SSL configuration:*

Enables and configures transport-level Secure Sockets Layer (SSL) security for this port. You can enable transport-level SSL security independently of message-level security.

Click **Edit** to specify the settings for transport-level HTTP SSL configuration for this port.

***Web services: Server security bindings collection:***

Use this page to view a list of server-side binding configurations for Web services security.

To view this administrative console page, complete the following steps:

1. Click **Applications** > **Enterprise applications** >*application_name*.
2. Under Related Items, click **EJB modules** or **Web modules** >*URI_file_name*.
3. Under Additional properties, click **Web services: Server security bindings**.

**Related reference**

**6.x application** "Request consumer (receiver) binding configuration settings" on page 745
Use this page to specify the binding configuration for the request consumer.

**6.x application** "Response generator (sender) binding configuration settings" on page 682
Use this page to specify the binding configuration for the response generator or response sender.

"Web services: Client security bindings collection" on page 689
Use this page to view a list of application-level, client-side binding configurations for Web services security. These bindings are used when a Web service is a client to another Web service.

*Port:*

Specifies the port in which messages are received from the request generator.

*Port:*

Specifies the port in which messages are received from the request sender.

*Web service:*

Specifies the name of the Web service that is defined during application assembly.

*Request consumer (receiver) binding:*

Specifies the binding configuration that is used to receive request messages from the request generator (sender) binding.

Click **Edit custom** to configure the required and additional information such as signing information, key information, token consumers, key locators, intermediate certificates in the collection certificate store, and trust anchors.

The binding information for the request consumer that is specified for the server must match the binding information for the request generator that is specified for the client.

*Response generator (sender) binding:*

Specifies the binding configuration that is used to send request messages to the response consumer.

Click **Edit custom** to configure the required and additional information such as signing information, key information, token generators, key locators, and intermediate certificates in the collection certificate store.

The binding information for the response generator that is specified for the server must match the binding information for the response consumer that is specified for the client.

*Request receiver binding:*

Specifies the binding configuration that is used to receive request messages from the request sender binding.

Click **Edit** to configure additional properties for the request receiver such as signing information, encryption information, trust anchors, collection certificate stores, key locators, trusted ID evaluators, and login mappings.

The binding information for the request receiver that is specified for the server must match the binding information for the request sender that is specified for the client.

*Response sender binding:*

Specifies the binding configuration that is used to send request messages to the response receiver.

Click **Edit** to configure additional properties for the response sender such as signing information, encryption information, and key locators.

The binding information for the response sender that is specified for the server must match the binding information for the response receiver that is specified for the client.

## Configuring the key locator for the generator binding on the application level

The key locator information for the default generator specifies which key locator implementation is used to locate the key used for signature and encryption information. The key locator information for the generator specifies which key locator implementation is used to locate the key that is used for signature validation or encryption. WebSphere Application Server provides default values for the bindings. However, you must modify the defaults for a production environment.

Complete the following steps to configure the key locator for the generator binding on the application level:

1. Locate the encryption information configuration panel in the administrative console.
   a. Click **Applications > Enterprise applications >** *application_name*.
   b. Under Related Items, click **EJB Modules** or **Web Modules >** *URI_name*.
   c. Under Additional Properties you can access the key information for the request generator and response generator bindings.
      - For the request generator (sender) binding, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom**.

- For the response generator (sender) binding, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**.

d. Under Additional properties, click **Key locators**.

e. Click **New** to create a key locator configuration, select the box next to the configuration and click **Delete** to delete an existing configuration, or click the name of an existing key locator configuration to edit its settings. If you are creating a new configuration, enter a unique name in the Key locator name field. For example, you might specify `gen_keyloc`.

2. Specify a class name for the key locator class implementation in the Key locator class name field. Key locators associated with version 6 applications must implement the com.ibm.wsspi.wssecurity.keyinfo.KeyLocator interface. Specify a class name according to the requirements of the application. For example, if the application requires that the key is read from a keystore file, specify the com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator implementation. WebSphere Application Server supports the following default key locator class implementations for version 6 applications that are available to use with the request generator or response generator:

**com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator**
This implementation locates and obtains the key from the specified keystore file.

**com.ibm.wsspi.wssecurity.keyinfo.SignerCertKeyLocator**
This implementation uses the public key from the signer certificate and is used by the response generator.

3. Specify the keystore password, the keystore location, and the keystore type. Key store files contain public and private keys, root certificate authority (CA) certificates, the intermediate CA certificate, and so on. Keys retrieved from the keystore are used to sign and validate or encrypt and decrypt messages or message parts. If you specified the com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator implementation for the key locator class implementation, you must specify a key store password, location, and type.

a. Specify a password in the Key store password field. This password is used to access the keystore file.

b. Specify the location of the key store file in the Key store path field.

c. Select a keystore type from the Type field. The Java Cryptography Extension (JCE) used by IBM supports the following key store types:

**JKS** Use this option if you are not using Java Cryptography Extensions (JCE) and if your keystore file uses the Java Keystore (JKS) format.

**JCEKS**
Use this option if you are using Java Cryptography Extensions.

**PKCS11KS (PKCS11)**
Use this format if your keystore uses the PKCS#11 file format. Keystores using this format might contain RSA keys on cryptographic hardware or might encrypt keys that use cryptographic hardware to ensure protection.

**PKCS12KS (PKCS12)**
Use this option if your keystore uses the PKCS#12 file format.

WebSphere Application Server provides some sample keystore files in the `${USER_INSTALL_ROOT}/etc/ws-security/samples` directory. For example, you might use the `enc-receiver.jceks` keystore file for encryption keys. The password for this file is Storepass and the type is JCEKS.

**Important:** Do not use the sample keystore files in a production environment. These samples are provided for testing purposes only.

4. Click **OK** and then click **Save** to save the configuration.

5. Under Additional properties, click **Keys**.

6. Click **New** to create a key configuration, select the box next to the configuration and click **Delete** to delete an existing configuration, or click the name of an existing key configuration to edit its settings. This entry specifies the name of the key object within the keystore file. If you are creating a new configuration, enter a unique name in the Key name field. For digital signatures, the key name is used by the request generator or the response generator signing information to determine which key is used to digitally sign the message.

   You must use a fully qualified distinguished name for the key name. For example, you might use `CN=Bob,O=IBM,C=US`.

   **Important:** Do not use the sample key files in a production environment. These samples are provided for testing purposes only.
7. Specify an alias in the Key alias field. The key alias is used by the key locator to search for key objects in the keystore.
8. Specify a password in the Key password field. The password is used to access the key object within the keystore file.
9. Click **OK** and then click **Save** to save the configuration.

You have configured the key locator for the generator binding at the application level.

You must specify a similar key information configuration for the consumer.

**Related tasks**

"Configuring the key locator for the consumer binding on the application level" on page 749

***Key locator collection:***

Use this page to view a list of key locator configurations that retrieve keys from the keystore for digital signature and encryption. A key locator must implement the com.ibm.wsspi.wssecurity.config.KeyLocator interface.

To view this administrative console page for the key locator collection on the server level, complete the following steps:
1. Click **Servers > Application servers >** *server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Key locators**.

To use this administrative console page for the key locator collection on the application level, complete the following steps:
1. Click **Applications > Enterprise applications** > *application_name*.
2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.
3. **6.x application** Under Additional properties, you can access key locators for the following bindings:
   - For the Request generator, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom > Key locators**.
   - For the Request consumer, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom > Key locators**.
   - For the Response generator, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom > Key locators**.
   - For the Response consumer, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom > Key locators**.
4. **5.x application** Under Additional properties, you can access key locators for the following bindings:
   - For the Request sender, click **Web services: Client security bindings**. Under Request sender binding, click **Edit > Key locators**.

- For the Request receiver, click **Web services: Server security bindings**. Under Request receiver binding, click **Edit > Key locators**.
- For the Response sender, click **Web services: Server security bindings**. Under Response sender binding, click **Edit > Key locators**.
- For the Response receiver, click **Web services: Client security bindings**. Under Response receiver binding, click **Edit > Key locators**.

**Tip:** The bindings for a version 5.x application has a link that says **Edit** and the bindings for a version 6.x application has a link that says **Edit custom.** This is quick reference to determine which application version you are configuring.

Using this **Key locator collection** panel, complete the following steps:

1. Specify a key locator name and a key locator class name on the panel.
2. Save your changes by clicking **Save** in the messages section at the top of the administrative console. The administrative console home panel is displayed.
3. After saving your changes, update the Web services security run time with the default binding information by clicking **Update runtime**. When you click **Update runtime**, the configuration changes made to the other Web services also are updated in the Web services security run time.
4. After you define key locators, click the key locator name to specify additional properties and keys under **Additional Properties**.

**Related reference**

"Key locator configuration settings"
Use this page to specify the settings for a key locator configuration. The key locators retrieve keys from the keystore file for digital signature and encryption. WebSphere Application Server enables you to plug in a custom key locator configuration.

*Key locator name:*

Specifies the unique name of the key locator.

*Key locator class name:*

Specifies the class name of the key locator, which retrieves the key that is used for digital signing and encryption.

***Key locator configuration settings:***

Use this page to specify the settings for a key locator configuration. The key locators retrieve keys from the keystore file for digital signature and encryption. WebSphere Application Server enables you to plug in a custom key locator configuration.

To view this administrative console page for the key locator collection on the server level, complete the following steps:

1. Click **Servers > Application servers >** *server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Key locators.**
4. Click **New** to create a new configuration or click the name of a configuration to modify its settings.

To use this administrative console page for the key locator collection on the application level, complete the following steps:

1. Click **Applications > Enterprise applications >***application_name*.
2. Under Related items, click **EJB modules** or **Web modules >***URI_name*.

3. **6.x application** Under Additional properties, you can access key locators for the following bindings:
   - For the Request generator, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom > Key locators**.
   - For the Request consumer, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom > Key locators**.
   - For the Response generator, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom > Key locators**.
   - For the Response consumer, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom > Key locators**.
4. **5.x application** Under Additional properties, you can access key locators for the following bindings:
   - For the Request sender, click **Web services: Client security bindings**. Under Request sender binding, click **Edit > Key locators**.
   - For the Request receiver, click **Web services: Server security bindings**. Under Request receiver binding, click **Edit > Key locators**.
   - For the Response sender, click **Web services: Server security bindings**. Under Response sender binding, click **Edit > Key locators**.
   - For the Response receiver, click **Web services: Client security bindings**. Under Response receiver binding, click **Edit > Key locators**.
5. Click **New** to create a new configuration or click the name of a configuration to modify its settings.

   **Related reference**

   “Key locator collection” on page 694
   Use this page to view a list of key locator configurations that retrieve keys from the keystore for digital signature and encryption. A key locator must implement the com.ibm.wsspi.wssecurity.config.KeyLocator interface.

   “Key collection” on page 687
   Use this page to view a list of logical names that is mapped to a key alias in the keystore file.

   “Key configuration settings” on page 688
   Use this page to define the mapping of a logical name to a key alias in a keystore file.

*Key locator name:*

Specifies the name of the key locator.

| | |
|---|---|
| **Data type** | String |

*Key locator class name:*

Specifies the name for the key locator class implementation.

**6.x application** Key locators that are associated with version 6.x applications must implement the com.ibm.wsspi.wssecurity.keyinfo.KeyLocator interface. WebSphere Application Server provides the following default key locator class implementations for version 6.x applications:

**com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator**
: This implementation locates and obtains the key from the specified keystore file.

**com.ibm.wsspi.wssecurity.keyinfo.SignerCertKeyLocator**
: This implementation uses the public key from the certificate of the signer. This class implementation is used by the response generator.

**com.ibm.wsspi.wssecurity.keyinfo.X509TokenKeyLocator**
: This implementation uses the X.509 security token from the sender message for digital signature validation and encryption. This class implementation is used by the request consumer and the response consumer.

**5.x application** Key locators that are associated with version 5.x applications must implement the com.ibm.wsspi.wssecurity.config.KeyLocator interface. WebSphere Application Server provides the following default key locator class implementations for version 5.x applications

**com.ibm.wsspi.wssecurity.config.WSIdKeyStoreMapKeyLocator**
This implementation maps an authenticated identity to a key and is used by the response sender. If encryption is used, this class is used to locate a key to encrypt the response message. The com.ibm.wsspi.wssecurity.config.WSIdKeyStoreMapKeyLocator class can map an authenticated identity from the invocation credential of the current thread to a key that is used to encrypt the message. If an authenticated identity is present on the current thread, the class maps the ID to the mapped name. For example, user1 is mapped to mappedName_1. Otherwise, `name="default"`. When a matching key is not found, the authenticated identity is mapped to the default key that is specified in the binding file. This implementation supports the following formats: JKS, JCEKS, and PKCS12.

**com.ibm.wsspi.wssecurity.config.KeyStoreKeyLocator**
This implementation maps a name to an alias and is used by the response receiver, request sender, and request receiver. The encryption process uses this class to obtain a key to encrypt a message, and the digital signature process uses this class to obtain a key to sign a message. The com.ibm.wsspi.wssecurity.config.KeyStoreKeyLocator class maps a logical name to a key alias in the keystore file. For example, key #105115176771 is mapped to `CN=Alice, O=IBM, c=US`.

**com.ibm.wsspi.wssecurity.config.CertInRequestKeyLocator**
This implementation uses the signer certificate to encrypt the response. This class implementation is used by the response sender and response receiver.

| | |
|---|---|
| **Data type** | String |

*Key store password:*

Specifies the password that is used to access the keystore file.

*Key store path:*

Specifies the location of the keystore file.

*Key store type:*

Specifies the type of keystore file.

**JKS** Use this option if you are not using Java Cryptography Extensions (JCE) and if your keystore file uses the Java Keystore (JKS) format.

**JCEKS**
Use this option if you are using Java Cryptography Extensions.

**PKCS11KS (PKCS11)**
Use this format if your keystore file uses the PKCS#11 file format. Keystores files that use this format might contain Rivest Shamir Adleman (RSA) keys on cryptographic hardware or might encrypt keys that use cryptographic hardware to ensure protection.

**PKCS12KS (PKCS12)**
Use this option if your keystore file uses the PKCS#12 file format.

| | |
|---|---|
| **Default** | JKS |
| **Range** | JKS, JCEKS, PKCS11KS (PKCS11), PKCS12KS (PKCS12) |

***Web services security property collection:***

Use this page to a view a list of additional properties for the configuration.

You can view a Web services security property collection panel in several ways. Complete the following steps to view one of these administrative console pages:

1. Click **Security > Web services**.
2. Under Default generator bindings or Default consumer bindings, click **Properties**.
3. Click **New** to create a new property.
4. Click **Delete** to a delete a property that you specified previously.

**Related reference**

"Web services security property configuration settings"
Use this page to configure additional properties.

*Property name:*

Specifies the name of the property.

*Property value:*

Specifies the value for the property.

***Web services security property configuration settings:***

Use this page to configure additional properties.

You can view a Web services security property configuration settings panel in several wasys. Complete the following steps to view one of these administrative console pages:

1. Click **Security > Web services**.
2. Under Default generator bindings or Default consumer bindings, click **Properties > New**.

**Related reference**

"Web services security property collection" on page 697
Use this page to a view a list of additional properties for the configuration.

*Property Name:*

Specifies the name of the property.

**Data type:** String

*Property Value:*

Specifies the value for the property.

**Data type:** String

The following table lists the properties that you can configure using the Web services security property panels.

| Configuration panel name | Property name | Property value | Description |
|---|---|---|---|
| JAAS configuration | `com.ibm.wsspi.wssecurity.token.X509.issuerName` | Specify the SubjectDN or the IssuerDN of the issuer for the X.509 certificate. | This property is used to specify the issuer of the certificate in the token consumer component. |

| Configuration panel name | Property name | Property value | Description |
| --- | --- | --- | --- |
| JAAS configuration | com.ibm.wsspi.wssecurity.token.X509.issuerSerial | Specify the serial number of the X.509 certificate. | This property is used to specify the serial number of the certificate in the token consumer component. |
| Key information | com.ibm.wsspi.wssecurity.keyinfo.EncodingNS | Specify the namespace Uniform Resource Identifier (URI) for the qualified name (QName). | This property is used to specify the namespace URI part of the QName that represents the encoding method. |
| Request generator and Response generator | com.ibm.wsspi.wssecurity.timestamp.SOAPHeaderElement | Specify 1 or true. | This property is used with the **Add nonce** option to set the mustUnderstand flag in the deployment descriptor. |
| Request generator and Response generator | com.ibm.wsspi.wssecurity.timestamp.dialect | | |
| Signing information | com.ibm.wsspi.wssecurity.dsig.dumpPath | Specify the path used to locate the output file. | This property is used to specify an output file for dumping the target UTF-8 binary data before signing and verifying messages. |
| Token generator | com.ibm.wsspi.wssecurity.token.username.timestampExpires | Specify 1 or true. | This property is used to specify an expiration date for the user name token. |
| Transform algorithms | com.ibm.wsspi.wssecurity.dsig.XPathExpression | not(ancestor-or-self::*[namespace-uri()='http://www.w3.org/2000/09/xmldsig#' and local-name()='Signature']) | This property is used with the http://www.w3.org/TR/1999/REC-xpath-19991116 algorithm. |

## Configuring the key information for the generator binding on the application level

Before you begin this task, configure the key locators and the token consumers that are referenced by the Key locator reference and Token reference fields within the key information panel.

This task provides the steps needed for configuring the key information for the request generator (client side) and the response generator (server side) bindings at the application level. The key information is used to specify the configuration needed to generate the key for digital signature and encryption. The signing information and the encryption information configurations can share the key information, which is why they are both defined at the same level.

Complete the following information to configure the key information for the generator binding on the application level:

1. Locate the key information configuration panel in the administrative console.
   a. Click **Applications > Enterprise applications >** *application_name*.
   b. Under Related Items, click **EJB Modules** or **Web Modules >** *URI_name*.
   c. Under Additional Properties you can access the key information for the request generator and response generator bindings.
      - For the request generator (sender) binding, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom**.
      - For the response generator (sender) binding, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**.

d. Under Required properties, click **Key information**.

e. Click **New** to create a key information configuration, select the box next to an existing configuration and click **Delete** to delete the configuration, or click the name of an existing signing information configuration to edit its settings. If you are creating a new configuration, enter a name in the Key information name field. For example, you might specify `gen_signkeyinfo`.

2. Select a key information type from the Key information type field. The key information type specifies how to reference the security tokens. WebSphere Application Server supports the following key information types:

**Key identifier**

The security token is referenced using an opaque value that uniquely identifies the token. The algorithm that is used for generating the <KeyIdentifier> element value depends upon the token type. For example, a hash of the important elements of the security token is used for generating the <KeyIdentifier> element value. The following <KeyInfo> element is generated in the Simple Object Access Protocol (SOAP) message for this key information type:

```
<ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
   <wsse:SecurityTokenReference>
      <wsse:KeyIdentifier ValueType="wsse:X509v3">/62wXO...</wsse:KeyIdentifier>
   </wsse:SecurityTokenReference>
</ds:KeyInfo>
```

**Key name**

The security token is referenced using a name that matches an identity assertion within the token. It is recommended that you do not use this key type as it might result in multiple security tokens that match the specified name. The following <KeyInfo> element is generated in the SOAP message for this key information type:

```
<ds:KeyInfo>
   <ds:KeyName>CN=Group1</ds:KeyName>
</ds:KeyInfo>
```

**Security token reference**

The security token is directly referenced using Universal Resource Identifiers (URIs). The following <KeyInfo> element is generated in the SOAP message for this key information type:

```
<ds:KeyInfo>
   <wsse:SecurityTokenReference>
      <wsse:Reference URI="#mytoken" />
   </wsse:SecurityTokenReference>
</ds:KeyInfo>
```

**Embedded token**

The security token is directly embedded within the <SecurityTokenReference> element. The following <KeyInfo> element is generated in the SOAP message for this key information type:

```
<ds:KeyInfo>
   <wsse:SecurityTokenReference>
      <wsse:Embedded wsu:Id="tok1" />
         ...
      </wsse:Embedded>
   </wsse:SecurityTokenReference>
</ds:KeyInfo>
```

**X509 issuer name and issuer serial**

The security token is referenced by an issuer name and an issuer serial number of an X.509 certificate. The following <KeyInfo> element is generated in the SOAP message for this key information type:

```
<ds:KeyInfo>
   <wsse:SecurityTokenReference>
     <ds:X509Data>
        <ds:X509IssuerSerial>
           <ds:X509IssuerName>CN=Jones, O=IBM, C=US</ds:X509IssuerName>
           <ds:X509SerialNumber>1040152879</ds:X509SerialNumber>
```

```
          </ds:X509IssuerSerial>
       </ds:X509Data>
     </wsse:SecurityTokenReference>
  </ds:KeyInfo>
```

Each type of key information is described in the Web Services Security: SOAP Message Security 1.0 (WS-Security 2004) OASIS standard, which is located at: http://www.oasis-open.org/home/index.php under Web services security.

3. Select a key locator reference from the Key locator reference field. This reference specifies a key locator that WebSphere Application Server uses to locate the keys that are used for digital signature and encryption. Before you can select a key locator, you must have configured a key locator. For more information on configuring a key locator, see the following articles:
   - "Configuring the key locator for the generator binding on the application level" on page 692
   - "Configuring the key locator for the consumer binding on the application level" on page 749
4. Click **Get keys** to view a list of key name references. After you click **Get keys**, the key names that are defined in the sig_klocator element are shown in the key name reference menu. If you change the key locator reference, you must click **Get keys** again to display the list of key names associated with the new key locator.
5. Select a key name reference from the Key name reference field. This reference specifies the name of a key that is used for generating a digital signature and for encryption. The list of key names provided comes from the key locator specified with the key locator reference.
6. Select a token reference from the Token reference field. This token reference specifies the name of token generator that is used for processing the security token. However, WebSphere Application Server requires this field only when you select Security token reference or Embedded token in the Key information type field. Before specifying a token reference, you must configure a token generator. For more information on configuring a token generator, see "Configuring the token generator on the application level" on page 675.
7. **Optional:** If you select Key identifier as the key information type on this panel, you must specify an encoding method, calculation method, value type namespace URI, and a value type local name.
   a. Select an encoding method from the Encoding method field. The encoding method specifies the encoding format for the key identifier. WebSphere Application Server supports the following encoding methods:
      - http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary
      - http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#HexBinary

   b. Select a calculation method from the Calculation method field. WebSphere Application Server supports the following calculation methods:
      - http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#ITSHA1
      - http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#IT60SHA1

   c. Specify a value type namespace Uniform Resource Identifier (URI) in the Namespace URI field. In this field, specify the namespace URI of the value type for a security token that is referenced by the key identifier. When you specify the X.509 certificate token, you do not need to specify this option. If you want to specify another token, you must specify the URI of the qualified name (QName) for value type.

   d. Specify a value type local name. This name is the local name of the value type for a security token that is referenced by the key identifier. When this local name is used in conjunction with the corresponding namespace URI, the information is called the value type qualified name or QName.

      When you specify the X.509 certificate token, it is recommended that you use the predefined local names. When you specify the predefined local names, you do not need to specify the namespace URI of the value type. However, if you do not use one of the predefined local names, you must specify both the uniform resource identifier (URI) and the local name. WebSphere Application Server provides the following predefined local names:

**X.509 certificate token**
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3

**X.509 certificates in a PKIPath**
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509PKIPathv1

**A list of X509 certificates and CRLs in a PKCS#7**
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#PKCS7

**LTPA** http://www.ibm.com/websphere/appserver/tokentype/5.0.2

8. Click **OK** and then click **Save** to save the configuration.

You have configured the key information for the generator binding at the application level

You must specify a similar key information configuration for the consumer.

**Related tasks**

"Configuring the key information for the consumer binding on the application level" on page 750
"Configuring the signing information for the generator binding on the application level" on page 712
"Configuring the key locator for the generator binding on the application level" on page 692
"Configuring the token generator on the application level" on page 675

***Key information collection:***

Use this page to view the configurations that are currently available for generating or consuming the key for XML digital signatures and XML encryption.

To view this administrative console page on the cell level for the key information references, complete the following steps:

1. Click **Security > Web services**.
2. Under Default generator bindings or the Default consumer bindings, click **Key information**.

To view this administrative console page on the server level for the key information references, complete the following steps:

1. Click **Servers > Application Servers >***server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Default generator bindings or the Default consumer bindings, click **Key information**.

To view this administrative console page on the application level for the key information references, complete the following steps. This option is available on the application level for version 6.x applications.

1. Click **Applications > Enterprise applications >***application_name*.
2. Under Related items, click **EJB modules** or **Web modules >***URI_name*.
3. Under Additional properties, you can access the signing information for the following bindings:
   - For the Request generator (sender) binding, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom**.
   - For the Request consumer (receiver) binding, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**.
   - For the Response generator (sender) binding, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**.
   - For the Response consumer (receiver) binding, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**.
4. Under Required properties, click **Key information**.

**Related reference**

"Token generator collection" on page 769
Use this page to view the token generators. The information is used on the generator side only to generate the security token.

"Token consumer collection" on page 791
Use this page to view the token consumer. The information is used on the consumer side only to process the security token.

"Key information configuration settings"
Use this page to specify the related configuration need to specify the key for XML digital signature or XML encryption.

*Key information name:*

Specifies the name that is given for the key configuration.

*Key information class name:*

Specifies the class name that is used for the key information type.

*Key information type:*

Specifies the type of mechanism used to reference the security token. The type corresponds to the class name that is specified in the Key information class name field.

***Key information configuration settings:***

Use this page to specify the related configuration need to specify the key for XML digital signature or XML encryption.

To view this administrative console page on the cell level for the key information references, complete the following steps:
1. Click **Security > Web services**.
2. Under Default generator bindings or the Default consumer bindings, click **Key information**.

To view this administrative console page on the server level for the key information references, complete the following steps:
1. Click **Servers > Application Servers >***server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Default generator bindings or the Default consumer bindings, click **Key information**.
4. Click **New** to create a new configuration or click the configuration name to modify its contents.

To view this administrative console page on the application level for the key information references, complete the following steps. This option is available on the application level for version 6.x applications.
1. Click **Applications > Enterprise applications** > *application_name*.
2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.
3. Under Additional properties, you can access the signing information for the following bindings:
   - For the Request generator (sender) binding, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom**.
   - For the Request consumer (receiver) binding, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**.
   - For the Response generator (sender) binding, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**.
   - For the Response consumer (receiver) binding, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**.

4. Under Required properties, click **Key information**.
5. Click **New** to create a new configuration or click the configuration name to modify its contents.

Before clicking **Properties** under Additional properties, you must enter a value in the Key information name field and select an option for the Key information type and Key locator reference options.

**Related reference**

"Token generator collection" on page 769
Use this page to view the token generators. The information is used on the generator side only to generate the security token.

"Token consumer collection" on page 791
Use this page to view the token consumer. The information is used on the consumer side only to process the security token.

"Key information collection" on page 702
Use this page to view the configurations that are currently available for generating or consuming the key for XML digital signatures and XML encryption.

*Key information name:*

Specifies a name for the key information configuration.

*Key information type:*

Specifies the type of key information. The key information type specifies how to reference security tokens.

WebSphere Application Server supports the following types of key information. Each type of key information is described in the Web Services Security: SOAP Message Security 1.0 (WS-Security 2004) OASIS standard, which is located at: http://www.oasis-open.org/home/index.php under Web services security.

| Type | Description |
| --- | --- |
| **Key identifier** | The security token is referenced using an opaque value that uniquely identifies the token. |
| **Key name** | The security token is referenced using a name that matches an identity assertion within the token. |
| **Security token reference** | With this type, the security token is directly referenced. |
| **Embedded token** | With this type, the security token reference is embedded. |
| **X509 issuer name and issuer serial** | With this type, the security token is referenced by an issuer and serial number of an X.509 certificate |

The X.509 issuer name and issuer serial is described in Web Services Security: X509 Token Profile Version 1.0 located at: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf. The other types are described in Web Services Security: SOAP Message Security Version 1.0 located at: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf.

If you select **Key identifier** for the key information type, you can specify values in the following fields on this panel:

- Encoding method
- Calculation method
- Value type namespace URI
- Value type local name

*Key locator reference:*

Specifies the reference that is used to retrieve the key for digital signature and encryption.

Before specifying a key locator reference, you must configure a key locator. You can specify a signing key configuration for the following bindings:

| **Binding name** | **Cell level, server level, or application level** | **Path** |
|---|---|---|
| Default generator binding | Server level | 1. Click **Servers > Application servers >***server_name*.<br>2. Under Security, click **Web services: Default bindings for Web services security**.<br>3. Under Additional properties, click **Key locators**.<br>4. Click **New** to create a new key locator or click the name of a configured key locator to modify its configuration. |
| Default consumer binding | Server level | 1. Click **Servers > Application servers >***server_name*.<br>2. Under Security, click **Web services: Default bindings for Web services security**.<br>3. Under Additional properties, click **Key locators**.<br>4. Click **New** to create a new key locator or click the name of a configured key locator to modify its configuration. |
| Request sender binding | Application level | 1. Click **Applications > Enterprise applications** > *application_name*.<br>2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.<br>3. Click **Web services: Client security bindings**. Under Request sender binding, click **Edit**.<br>4. Under Additional properties, click **Key locators**.<br>5. Click **New** to create a new key locator or click the name of a configured key locator to modify its configuration. |

| **Binding name** | **Cell level, server level, or application level** | **Path** |
|---|---|---|
| Response receiver binding | Application level | 1. Click **Applications > Enterprise applications** > *application_name*.<br>2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.<br>3. Click **Web services: Client security bindings**. Under Response receiver binding, click **Edit**.<br>4. Under Additional properties, click **Key locators**.<br>5. Click **New** to create a new key locator or click the name of a configured key locator to modify its configuration. |
| Request receiver binding | Application level | 1. Click **Applications > Enterprise applications** > *application_name*.<br>2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.<br>3. Click **Web services: Server security bindings**. Under Request receiver binding, click **Edit**.<br>4. Under Additional properties, click **Key locators**.<br>5. Click **New** to create a new key locator or click the name of a configured key locator to modify its configuration. |
| Response sender binding | Application level | 1. Click **Applications > Enterprise applications** > *application_name*.<br>2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.<br>3. Click **Web services: Server security bindings**. Under Response sender binding, click **Edit**.<br>4. Under Additional properties, click **Key locators**.<br>5. Click **New** to create a new key locator or click the name of a configured key locator to modify its configuration. |

| Binding name | Cell level, server level, or application level | Path |
| --- | --- | --- |
| Request generator (sender) binding | Application level | 1. Click **Applications > Enterprise applications** > *application_name*.<br>2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.<br>3. Click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit**.<br>4. Under Additional properties, click **Key locators**.<br>5. Click **New** to create a new key locator or click the name of a configured key locator to modify its configuration. |
| Response consumer (receiver) binding | Application level | 1. Click **Applications > Enterprise applications** > *application_name*.<br>2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.<br>3. Click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**.<br>4. Under Additional properties, click **Key locators**.<br>5. Click **New** to create a new key locator or click the name of a configured key locator to modify its configuration. |
| Request consumer (receiver) binding | Application level | 1. Click **Applications > Enterprise applications** > *application_name*.<br>2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.<br>3. Click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**.<br>4. Under Additional properties, click **Key locators**.<br>5. Click **New** to create a new key locator or click the name of a configured key locator to modify its configuration. |

| Binding name | Cell level, server level, or application level | Path |
| --- | --- | --- |
| Response generator (sender) binding | Application level | 1. Click **Applications > Enterprise applications** > *application_name*.<br>2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.<br>3. Click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**.<br>4. Under Additional properties, click **Key locators**.<br>5. Click **New** to create a new key locator or click the name of a configured key locator to modify its configuration. |

*Key name reference:*

Specifies the name of the key that is used for generating digital signature and encryption.

This field is displayed for the default generator and is also displayed for the request generator and response generator for Version 6.x applications.

| Binding name | Cell level, server level, or application level | Path |
| --- | --- | --- |
| Default generator binding | Server level | 1. Click **Servers > Application servers >***server_name*.<br>2. Under Security, click **Web services: Default bindings for Web services security**.<br>3. Under Additional properties, click **Key locators**.<br>4. Click **New** to create a new key locator or click the name of a configured key locator to modify its configuration. |
| Request generator (sender) binding | Application level | 1. Click **Applications > Enterprise applications** > *application_name*.<br>2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.<br>3. Click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit**.<br>4. Under Additional properties, click **Key locators**.<br>5. Click **New** to create a new key locator or click the name of a configured key locator to modify its configuration. |

| Binding name | Cell level, server level, or application level | Path |
| --- | --- | --- |
| Response generator (sender) binding | Application level | 1. Click **Applications > Enterprise applications** > *application_name*.<br>2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.<br>3. Click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**.<br>4. Under Additional properties, click **Key locators**.<br>5. Click **New** to create a new key locator or click the name of a configured key locator to modify its configuration. |

*Token reference:*

Specifies the name of a token generator or token consumer that is used for processing a security token.

WebSphere Application Server requires this field only when you specify Security token reference or Embedded token in the Key information type field. The Token reference field is also required when you specify a key identifier type for the consumer. Before specifying a token reference, you must configure a token generator or token consumer. You can specify a token configuration for the following bindings on the following levels:

| Binding name | Cell level, server level, or application level | Path |
| --- | --- | --- |
| Default generator binding | Server level | 1. Click **Servers > Application servers >***server_name*.<br>2. Under Security, click **Web services: Default bindings for Web services security**.<br>3. Under Default generator bindings, click **Token generator**.<br>4. Click **New** to create a new token generator or click the name of a configured token generator to modify its configuration. |
| Default consumer binding | Server level | 1. Click **Servers > Application servers >***server_name*.<br>2. Under Security, click **Web services: Default bindings for Web services security**.<br>3. Under Default consumer bindings, click **Token consumer**.<br>4. Click **New** to create a new token consumer or click the name of a configured token consumer to modify its configuration. |

| Binding name | Cell level, server level, or application level | Path |
| --- | --- | --- |
| Request generator (sender) binding | Application level | 1. Click **Applications > Enterprise applications** > *application_name*.<br>2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.<br>3. Click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom**.<br>4. Under Additional properties, click **Token generators**.<br>5. Click **New** to create a new token consumer or click the name of a configured token consumer to modify its configuration. |
| Response consumer (receiver) binding | Application level | 1. Click **Applications > Enterprise applications** > *application_name*.<br>2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.<br>3. Click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**.<br>4. Under Required properties, click **Token consumers**.<br>5. Click **New** to create a new token consumer or click the name of a configured token consumer to modify its configuration. |
| Request consumer (receiver) binding | Application level | 1. Click **Applications > Enterprise applications** > *application_name*.<br>2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.<br>3. Click **Web services: Sender security bindings**. Under Request consumer (receiver) binding, click **Edit custom**.<br>4. Under Required properties, click **Token consumers**.<br>5. Click **New** to create a new token consumer or click the name of a configured token consumer to modify its configuration. |

| Binding name | Cell level, server level, or application level | Path |
| --- | --- | --- |
| Response generator (sender) binding | Application level | 1. Click **Applications > Enterprise applications** > *application_name*.<br>2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.<br>3. Click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**.<br>4. Under Additional properties, click **Token generators**.<br>5. Click **New** to create a new token consumer or click the name of a configured token consumer to modify its configuration. |

*Encoding method:*

Specifies the encoding method that indicates the encoding format for the key identifier.

This field is valid when you specify Key identifier in the Key information type field. WebSphere Application Server supports the following encoding methods:

- http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary
- http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#HexBinary

This field is available for the default generator binding only.

*Calculation method:*

This field is valid when you specify Key identifier in the Key information type field. WebSphere Application Server supports the following calculation methods:

- http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#ITSHA1
- http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#IT60SHA1

This field is available for the generator binding only.

*Value type namespace URI:*

Specifies the namespace Uniform Resource Identifier (URI) of the value type for a security token that is referenced by the key identifier.

This field is valid when you specify Key identifier in the Key information type field. When you specify the X.509 certificate token, you do not need to specify this option. If you want to specify another token, specify the URI of QName for value type.

WebSphere Application Server provides the following predefined value type URI for the Lightweight Third Party Authentication (LTPA) token: http://www.ibm.com/websphere/appserver/tokentype/5.0.2

This field is available for the generator binding only.

*Value type local name:*

Specifies the local name of the value type for a security token that is referenced by the key identifier.

When this local name is used with the corresponding namespace URI, the information is called the *value type qualified name* or *QName*.

This field is valid when you specify Key identifier in the Key information type field. When you specify the X.509 certificate token, it is recommended that you use the predefined local names. When you specify the predefined local names, you do not need to specify the URI of the value type. WebSphere Application Server provides the following predefined local names:

**X.509 certificate token**
: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3

**X.509 certificates in a PKIPath**
: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509PKIPathv1

**A list of X509 certificates and CRLs in a PKCS#7**
: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#PKCS7

**LTPA**

**Attention:** For LTPA, the value type local name is `LTPA`. If you enter LTPA for the local name, you must specify the http://www.ibm.com/websphere/appserver/tokentype/5.0.2 URI value in the Value type URI field as well. For the other predefined value types (User name token, X509 certificate token, X509 certificates in a PKIPath, and a list of X509 certificates and CRLs in a PKCS#7), the value for the local name field begins with `http://`. For example, if you are specifying the user name token for the value type, enter `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#UsernameToken` in the value type local name field and then you do not need to enter a value in the value type URI field.

When you specify a custom value type for custom tokens, you can specify the local name and the URI of the quality name (QName) of the value type. For example, you might specify `Custom` for the local name and `http://www.ibm.com/custom` for the URI.

This field is also available for the generator binding only.

## Configuring the signing information for the generator binding on the application level

In the server-side extensions file (`ibm-webservices-ext.xmi`) and the client-side deployment descriptor extensions file (`ibm-webservicesclient-ext.xmi`), you must specify which parts of the message are signed. Also, you need to configure the key information that is referenced by the key information references on the signing information panel within the administrative console.

This task explains the required steps to configure the signing information for the client-side request generator and the server-side response generator bindings at the application level. WebSphere Application Server uses the signing information for the default generator to sign parts of the message including the body, time stamp, and user name token. The Application Server provides default values for bindings. However, an administrator must modify the defaults for a production environment. Complete the following steps to configure the signing information for the generator sections of the bindings files on the application level:

1. Locate the signing information configuration panel in the administrative console.
   a. Click **Applications > Enterprise applications >** *application_name*.
   b. Under Related Items, click **EJB Modules** or **Web Modules >** *URI_name*.
   c. Under Additional properties, you can access the signing information for the request generator and the response generator bindings.

- For the request generator (sender) binding, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom**.
- For the response generator (sender) binding, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**.

d. Under Required properties, click **Signing information**.

e. Click **New** to create a signing information configuration, select the box next to the configuration and click **Delete** to delete an existing configuration, or click the name of an existing signing information configuration to edit its settings. If you are creating a new configuration, enter a name in the Signing information name field. For example, you might specify `gen_signinfo`.

2. Select a signature method algorithm from the Signature method field. The algorithm that is specified for the generator, which is either the request generator or the response generator configuration, must match the algorithm that is specified for the consumer, which is either the request consumer or response consumer configuration. WebSphere Application Server supports the following pre-configured algorithms:
   - http://www.w3.org/2000/09/xmldsig#rsa-sha1
   - http://www.w3.org/2000/09/xmldsig#dsa-sha1
   - http://www.w3.org/2000/09/xmldsig#hmac-sha1
3. Select a canonicalization method from the Canonicalization method field. The canonicalization algorithm that you specify for the generator must match the algorithm for the consumer. WebSphere Application Server supports the following pre-configured algorithms:
   - http://www.w3.org/2001/10/xml-exc-c14n#
   - http://www.w3.org/2001/10/xml-exc-c14n#WithComments
   - http://www.w3.org/TR/2001/REC-xml-c14n-20010315
   - http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments
4. Select a key information signature type from the Key information signature type field. WebSphere Application Server supports the following signature types:

   **None** Specifies that the KeyInfo element is not signed.

   **Keyinfo**
   Specifies that the entire KeyInfo element is signed.

   **Keyinfochildelements**
   Specifies that the child elements of the KeyInfo element are signed.

   The key information signature type for the generator must match the signature type for the consumer. You might encounter the following situations:
   - If you do not specify one of the previous signature types, WebSphere Application Server uses keyinfo, by default.
   - If you select Keyinfo or Keyinfochildelements and you select http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform as the transform algorithm in a subsequent step, WebSphere Application Server also signs the referenced token.
5. Select a signing key information reference from the Signing key information field. This selection is a reference to the signing key that the Application Server uses to generate digital signatures.
6. Click **OK** and **Save** to save the configuration.
7. Click the name of the new signing information configuration. This configuration is the one that you specified in a previous step.
8. Specify the part reference, digest algorithm, and transform algorithm. The part reference specifies which parts of the message to digitally sign.

   a. Under Additional properties, click **Part references > New** to create a new part reference, click **Part references > Delete** to delete an existing part reference, or click a part name to edit an existing part reference.

b. Specify a unique part name for this part reference. For example, you might specify `reqint`.
c. Select a part reference from the Part reference field.

The part reference refers to the message part that is digitally signed. The part attribute refers to the name of the <Integrity> element in the deployment descriptor when the <PartReference> element is specified for the signature. You can specify multiple <PartReference> elements within the <SigningInfo> element. The <PartReference> element has two child elements when it is specified for the signature: <DigestTransform> and <Transform>.
d. Select a digest method algorithm from the menu. The digest method algorithm specified within the <DigestMethod> element is used in the <SigningInfo> element. WebSphere Application Server supports the http://www.w3.org/2000/09/xmldsig#sha1 algorithm.
e. Click **OK** to save the configuration.
f. Click the name of the new part reference configuration. This configuration is the one that you specified in a previous step.
g. Under Additional Properties, click **Transforms > New** to create a new transform, click **Transforms > Delete** to delete a transform, or click a transform name to edit an existing transform. If you create a new transform configuration, specify a unique name. For example, you might specify `reqint_body_transform1`.
h. Select a transform algorithm from the menu. The transform algorithm is that is specified within the <Transform> element and specifies the transform algorithm for the signature. WebSphere Application Server supports the following algorithms:
   - http://www.w3.org/2001/10/xml-exc-c14n#
   - http://www.w3.org/TR/1999/REC-xpath-19991116
   - http://www.w3.org/2002/06/xmldsig-filter2
   - http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform
   - http://www.w3.org/2002/07/decrypt#XML
   - http://www.w3.org/2000/09/xmldsig#enveloped-signature

   The transform algorithm that you select for the generator must match the transform algorithm that you select for the consumer.

   **Important:** If both of the following conditions are true, WebSphere Application Server signs the referenced token:
   - You previously selected the Keyinfo or the Keyinfochildelements option from the Key information signature type field on the signing information panel.
   - You select http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform as the transform algorithm.

9. Click **OK**.
10. Click **Save** at the top of the panel to save your configuration.

After completing these steps, the signing information is configured for the generator on the application level.

You must specify a similar signing information configuration for the consumer.

**Related tasks**

"Configuring the signing information for the consumer binding on the application level" on page 753

***Signing information collection:***

Use this page to view a list of signing parameters. Signing information is used to sign and validate parts of a message including the body, time stamp, and user name token. You can also use these parameters for

X.509 validation when the authentication method is IDAssertion and the ID type is X509Certificate in the server-level configuration. In such cases, you must fill in the certificate path fields only.

**Note:** Use Internet Explorer if you experience difficulties in the Signing Information panel using Netscape 4.7.9.

To view this administrative console page on the cell level for signing information, complete the following steps:

To view this administrative console page on the server level for signing information, complete the following steps:

1. Click **Servers > Application Servers >** *server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Default generator bindings or Default consumer bindings, click **Signing information**.
4. Click **New** to create a signing parameter. Click **Delete** to delete a signing parameter.

To view this administrative console page on the application level for signing information, complete the following steps:

1. Click **Applications > Enterprise applications** > *application_name*.
2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.
3. **6.x application** Under Additional properties, you can access the signing information for the following bindings:
   - For the Request generator (sender) binding, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom**.
   - For Response consumer (receiver) binding, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**.
   - For the Request consumer (receiver) binding, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**.
   - For the Response generator (sender) binding, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**.
4. **6.x application** Under Required properties, click **Signing information**.
5. **5.x application** Under Additional properties, you can use this panel to configure the following bindings:
   - For the Request receiver binding, click **Web services: Server security bindings**. Under Request receiver binding, click **Edit**.
   - For the Response receiver binding, click **Web services: Client security bindings**. Under Response receiver binding, click **Edit**.
6. **5.x application** Under Additional properties, click **Signing information**.
7. Click **New** to create a signing parameter. Click **Delete** to delete a signing parameter.

   **Related reference**

   "Signing information configuration settings" on page 716
   Use this page to configure new signing parameters.

*Signing information name:*

Specifies the unique name that is assigned to the signing configuration.

*Signature method:*

Specifies the signature method algorithm that is chosen for the signing configuration.

*Canonicalization method:*

Specifies the canonicalization method algorithm that is chosen for the signing configuration.

***Signing information configuration settings:***

Use this page to configure new signing parameters.

The specifications that are listed on this page for the signature method, digest method, and canonicalization method are located in the World Wide Web Consortium (W3C) document entitled, *XML Signature Syntax and Specification: W3C Recommendation 12 Feb 2002*.

To view this administrative console page on the cell level for signing information, complete the following steps:

To view this administrative console page on the server level for signing information, complete the following steps:

1. Click **Servers > Application Servers >** *server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Default generator bindings or Default consumer bindings, click **Signing information**.
4. Click **New** to create a signing parameter or click the name of an existing configuration to modify its settings.

To view this administrative console page on the application level for signing information, complete the following steps:

1. Click **Applications > Enterprise applications** > *application_name*.
2. Under Related items, click **EJB modules** or **Web modules** >*URI_name*.
3. **6.x application** Under Additional properties, you can access the signing information for the following bindings:
   - For the Request generator (sender) binding, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom**.
   - For Response consumer (receiver) binding, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**.
   - For the Request consumer (receiver) binding, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**.
   - For the Response generator (sender) binding, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**.
4. **6.x application** Under Required properties, click **Signing information**.
5. **5.x application** Under Additional properties, you can access the signing information for the following bindings:
   - For the Request receiver binding, click **Web services: Server security bindings**. Under Request receiver binding, click **Edit**.
   - For the Response receiver binding, click **Web services: Client security bindings**. Under Response receiver binding, click **Edit**.
6. **5.x application** Under Additional properties, click **Signing information**.
7. Click **New** to create a signing parameter or click the name of an existing configuration to modify its settings.

   **Related reference**

   "Signing information collection" on page 714
   Use this page to view a list of signing parameters. Signing information is used to sign and validate parts of a message including the body, time stamp, and user name token. You can also use these

parameters for X.509 validation when the authentication method is IDAssertion and the ID type is X509Certificate in the server-level configuration. In such cases, you must fill in the certificate path fields only.

"Trust anchor configuration settings" on page 661
Use this information to configure a trust anchor. Trust anchors point to keystores that contain trusted root or self-signed certificates. This information enables you to specify a name for the trust anchor and the information that is needed to access a keystore. The application binding uses this name to reference a predefined trust anchor definition in the binding file (or the default).

"Collection certificate store configuration settings" on page 666
Use this page to specify the name and the provider for a collection certificate store. A *collection certificate store* is a collection of nonroot, certificate authority (CA) certificates and certificate revocation lists (CRLs). This collection of CA certificates and CRLs is used to check the signature of a digitally signed Simple Object Access Protocol (SOAP) message.

*Signing information name:*

Specifies the name that is assigned to the signing configuration.

*Signature method:*

Specifies the algorithm Uniform Resource Identifiers (URI) of the signature method.

The following pre-configured algorithms are supported:

- **5.x and 6.x application** http://www.w3.org/2000/09/xmldsig#rsa-sha1
- **5.x and 6.x application** http://www.w3.org/2000/09/xmldsig#dsa-sha1
- **6.x application** http://www.w3.org/2000/09/xmldsig#hmac-sha1

For Version 6.x applications, you can specify additional signature methods on the Algorithm URI panel. To access the Algorithm URI panel, complete the following steps:

1. Click **Security** > **Web services**.
2. Under Additional properties, click **Algorithm mappings** > *algorithm_factory_engine_class_name* > **Algorithm URI** > **New**.

When you specify the Algorithm URI, you also must specify an algorithm type. To have the algorithm display as a selection in the Signature method field on the Signing information panel, you must select **Signature** as the algorithm type.

This field is available for Version 6.x applications and for the request receiver and response receiver bindings for Version 5.x applications.

*Digest method:*

Specifies the algorithm URI of the digest method.

The http://www.w3.org/2000/09/xmldsig#sha1 algorithm is supported.

This field is available for the request receiver and response receiver bindings for Version 5.x applications.

*Canonicalization method:*

Specifies the algorithm URI of the canonicalization method.

The following pre-configured algorithms are supported:

- http://www.w3.org/2001/10/xml-exc-c14n#
- http://www.w3.org/2001/10/xml-exc-c14n#WithComments
- http://www.w3.org/TR/2001/REC-xml-c14n-20010315
- http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments

This field is for Version 6.x applications and for the request receiver and response receiver bindings for Version 5.x applications.

*Key information signature type:*

Specifies how to sign a KeyInfo element if `dsigkey` or `enckey` is specified for the signing part in the deployment descriptor.

WebSphere Application Server supports the following keywords:

**keyinfo (default)**
Specifies that the entire KeyInfo element is signed.

**keyinfochildelements**
Specifies that the child elements of the KeyInfo element is signed.

If you do not specify a keyword, WebSphere Application Server uses the keyinfo value, by default.

The Key information signature type field is available for the token consumer binding.

**6.x application** For Version 6.x applications, the field is also available for the default consumer, request consumer, and response consumer bindings.

*Signing key information:*

Specifies a reference to the key information that WebSphere Application Server uses to generate the digital signature.

You can specify one signing key only for the default generator binding on the server level. However, you can specify multiple signing keys for the default consumer bindings. The signing keys for the default consumer bindings are specified using the Key Information references link under Additional properties on the Signing information panel.

On the application level, you can specify only one signing key for the request generator and the response generator. You can specify multiple signing keys for the request consumer and response generator. The signing keys for the request consumer and the response consumer are specified using the Key information references link under Additional properties.

You can specify a signing key configuration for the following bindings on the following levels:

| Binding name | Cell level, server level, or application level | Path |
| --- | --- | --- |
| Default generator binding | Server level | 1. Click **Servers > Application Servers** >*server_name*.<br>2. Under Security, click **Web services: Default bindings for Web services security** .<br>3. Under Default generator binding, click **Key information**. |

| Binding name | Cell level, server level, or application level | Path |
| --- | --- | --- |
| Default consumer binding | Server level | 1. Click **Servers > Application Servers** >*server_name*.<br>2. Under Security, click **Web services: Default bindings for Web services security** .<br>3. Under Default consumer binding, click **Key information**. |

*Certificate path:*

Specifies the settings for the certificate path validation. When you select **Trust any**, this validation is skipped and all incoming certificates are trusted.

The certificate path options are available on the application level.

**Trust anchor**

WebSphere Application Server searches for trust anchor configurations on the application and server levels and lists the configurations in this menu.

**5.x application** You can specify trust anchors as an additional property for the response receiver binding and the request receiver binding.

You can specify a trust anchor configuration for the following bindings on the following levels:

| Binding name | Cell level, server level, or application level | Path |
| --- | --- | --- |
| Default generator binding | Server level | 1. Click **Servers > Application servers** >*server_name*.<br>2. Under Security, click **Web services: Default bindings for Web services security**.<br>3. Under Additional properties, click **Trust anchors > New**. |
| Default consumer binding | Server level | 1. Click **Servers > Application servers** >*server_name*.<br>2. Under Security, click **Web services: Default bindings for Web services security**.<br>3. Under Additional properties, click **Trust anchors > New**. |
| Response receiver | Application level for Version 5.x applications | 1. Click **Applications > Enterprise applications** >*application_name*.<br>2. Under Related items, click **Web modules** or **EJB modules** >*URI_name*.<br>3. Click **Web services: Client security bindings**.<br>4. Under the Response receiver binding, click **Edit**.<br>5. Under Additional properties, click **Trust anchors > New**. |

| Binding name | Cell level, server level, or application level | Path |
|---|---|---|
| Request receiver | Application level for Version 5.x applications | 1. Click **Applications > Enterprise applications >***application_name*.<br>2. Under Related items, click **Web modules** or **EJB modules >***URI_name*.<br>3. Click **Web services: Server security bindings**.<br>4. Under the Request receiver binding, click **Edit**.<br>5. Under Additional properties, click **Trust anchors > New**. |

For an explanation of the fields on the trust anchor panel, see "Trust anchor configuration settings" on page 661.

### Certificate store

WebSphere Application Server searches for certificate store configurations on the application and server levels and lists the configurations in this menu.

You can specify a certificate store configuration for the following bindings on the following levels:

| Binding name | Cell level, server level, or application level | Path |
|---|---|---|
| Default generator binding | Server level | 1. Click **Servers > Application servers >***server_name*.<br>2. Under Security, click **Web services: Default bindings for Web services security**.<br>3. Under Additional properties, click **Collection certificate store > New**. |
| Default consumer binding | Server level | 1. Click **Servers > Application servers >***server_name*.<br>2. Under Security, click **Web services: Default bindings for Web services security**.<br>3. Under Additional properties, click **Collection certificate store > New**. |
| Response receiver | Application level for Version 5.x applications | 1. Click **Applications > Enterprise applications >***application_name*.<br>2. Under Related items, click **Web modules** or **EJB modules >***URI_name*.<br>3. Click **Web services: Client security bindings**.<br>4. Under the Response receiver binding, click **Edit**.<br>5. Under Additional properties, click **Collection certificate store > New**. |

| Binding name | Cell level, server level, or application level | Path |
|---|---|---|
| Request receiver | Application level for Version 5.x applications | 1. Click **Applications > Enterprise applications >***application_name*.<br>2. Under Related items, click **Web modules** or **EJB modules >***URI_name*.<br>3. Click **Web services: Server security bindings**.<br>4. Under the Request receiver binding, click **Edit**.<br>5. Under Additional properties, click **Collection certificate store > New**. |

For an explanation of the fields on the collection certificate store panel, see "Collection certificate store configuration settings" on page 666.

***Part reference collection:***

Use this page to view the message part references for signature and encryption that are defined in the deployment descriptors.

To view this administrative console page on the cell level for signing information, complete the following steps:
1. Click **Security > Web services**.
2. Under Default generator bindings or Default consumer bindings, click **Signing information >** *signing_information_name*.
3. Under Additional properties, click **Part references**.

To view this administrative console page on the server level for signing information, complete the following steps:
1. Click **Servers > Application Servers >***server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Default generator bindings or Default consumer bindings, click **Signing information >** *signing_information_name*.
4. Under Additional properties, click **Part references**.

To view this administrative console page on the application level for signing information, complete the following steps. Part references are available through the administrative console using Version 6.x applications only.
1. Click **Applications > Enterprise applications** > *application_name*.
2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.
3. Under Additional properties, you can access the signing information for the following bindings:
   - For the Request generator (sender) binding, click **Web services: Client security bindings**. Under Request generator (sending) binding, click **Edit custom**.
   - For Response consumer (receiver) binding, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**.
   - For the Request consumer (receiver) binding, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**.
   - For the Response generator (sender) binding, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**.

4. Under Required properties, click **Signing information >***signing_information_name*.
5. Under Additional properties, click **Part references**.

**Related reference**

"Part reference configuration settings"
Use this page to specify a reference to the message parts for signature and encryption that are defined in the deployment descriptors.

"Transforms collection" on page 724
Use this page to view the transform algorithm that is used for processing the Web services security message.

"Transforms configuration settings" on page 725
Use this page to specify the transform algorithm that is used for processing the Web services security message.

*Part name:*

Specifies the name that is assigned to the part reference configuration.

*Part reference:*

Specifies the name of the signed part that is defined in the deployment descriptor.

The Part reference field is specified in the application binding configuration only.

*Digest method algorithm:*

Specifies the algorithm URI of the digest method that is used for the signed part that is specified by the part reference.

***Part reference configuration settings:***

Use this page to specify a reference to the message parts for signature and encryption that are defined in the deployment descriptors.

To view this administrative console page on the cell level for signing information, complete the following steps:
1. Click **Security > Web services**.
2. Under Default generator bindings or Default consumer bindings, click **Signing information >** *signing_information_name*.
3. Under Additional properties, click **Part references**.
4. Click **New** to create a part reference or click the name of an existing configuration to modify its settings.

To view this administrative console page on the server level for signing information, complete the following steps:
1. Click **Servers > Application Servers >***server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Default generator bindings or Default consumer bindings, click **Signing information >** *signing_information_name*.
4. Under Additional properties, click **Part references**.
5. Click **New** to create a part reference or click the name of an existing configuration to modify its settings.

To view this administrative console page on the application level for signing information, complete the following steps. Part references are available through the administrative console using Version 6.x applications only.

1. Click **Applications > Enterprise applications** > *application_name*.
2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.
3. Under Additional properties, you can access the signing information for the following bindings:
   - For the Request generator (sender) binding, click **Web services: Client security bindings**. Under Request generator (sending) binding, click **Edit custom**.
   - For Response consumer (receiver) binding, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**.
   - For the Request consumer (receiver) binding, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**.
   - For the Response generator (sender) binding, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**.
4. Under Required properties, click **Signing information >***signing_information_name*.
5. Under Additional properties, click **Part references**.
6. Click **New** to create a part reference or click the name of an existing configuration to modify its settings.

You must specify a part name and select a part reference before specifying additional properties. Before specifying the digest method properties that are accessible under Additional properties, specify a digest method algorithm on this panel. If you specify **none** and click Digest method, an error message is displayed.

**Related reference**

"Part reference collection" on page 721
Use this page to view the message part references for signature and encryption that are defined in the deployment descriptors.

"Transforms collection" on page 724
Use this page to view the transform algorithm that is used for processing the Web services security message.

"Transforms configuration settings" on page 725
Use this page to specify the transform algorithm that is used for processing the Web services security message.

*Part name:*

Specifies the name that is assigned to the part reference configuration.

*Part reference:*

Specifies the name of the <integrity> or <requiredIntegrity> element for the signed part of the message or it specifies the name of the <confidentiality> or <requiredConfidentiality> element for the encrypted part of the message in the deployment descriptor.

The part names that are defined in the deployment descriptor are listed as options in this field. This field is displayed for the binding configuration on the application level only.

*Digest method algorithm:*

Specifies the algorithm URI of the digest method that is used for the signed part that is specified by the part reference.

WebSphere Application Server provides the following predefined algorithm URI: `http://www.w3.org/2000/09/xmldsig#sha1`. If you want to specify a custom algorithm, you must configure the custom algorithm in the Algorithm URI panel before setting the digest method algorithm.

To access the Algorithm URI panel, complete the following steps for the server level:

1. Click **Servers > Application servers >***server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Algorithm mappings >** *algorithm_factory_engine_class_name* **> Algorithm URI > New**.

The specified algorithms are listed as options for this field.

When you specify the Algorithm URI, you also must specify an algorithm type. To have the algorithm display as a selection in the Digest method algorithm field on the Part reference panel, you must select **Digest value calculation (Message digest)** as the algorithm type.

***Transforms collection:***

Use this page to view the transform algorithm that is used for processing the Web services security message.

To view this administrative console page for the cell level, complete the following steps:

1. Click **Security > Web services**.
2. Under Default generator bindings or Default consumer bindings, click **Signing information >** *signing_information_name*.
3. Under Additional properties, click **Part references >***part_reference_name*.
4. Under Additional properties, click **Transforms**.

To view this administrative console page for the server level, complete the following steps:

1. Click **Application Servers > Servers >***server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Default generator bindings or Default consumer bindings, click **Signing information >** *signing_information_name*.
4. Under Additional properties, click **Part references >***part_reference_name*.
5. Under Additional properties, click **Transforms**.

**6.x application** To view this administrative console page for the application level, complete the following steps. This option is available for Version 6.x applications only.

1. Click **Applications > Enterprise applications >***application_name*.
2. Under Related items, click **EJB Modules** or **Web modules >***URI_name*.
3. Under Additional properties, you can access the transforms information for the following bindings:
   - For the Request generator (sender) binding, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom**.
   - For the Request consumer (receiver) binding, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**.
   - For the Response generator (sender) binding, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**.
   - For the Response consumer (receiver) binding, click **Web services: Client security bindings**. Under Request consumer (receiver) binding, click **Edit custom**.
4. Under Required properties, click **Signing information >***signing_information_name*.
5. Under Additional properties, click **Part references >***part_name* **> Transforms**.

**Related reference**

"Transforms configuration settings"
Use this page to specify the transform algorithm that is used for processing the Web services security message.

"Part reference collection" on page 721
Use this page to view the message part references for signature and encryption that are defined in the deployment descriptors.

"Part reference configuration settings" on page 722
Use this page to specify a reference to the message parts for signature and encryption that are defined in the deployment descriptors.

*Transform name:*

Specifies the name that is assigned to the transform algorithm.

*Transform algorithm:*

Specifies the algorithm URI of the transform algorithm.

***Transforms configuration settings:***

Use this page to specify the transform algorithm that is used for processing the Web services security message.

To view this administrative console page for the cell level, complete the following steps:
1. Click **Security > Web services**.
2. Under Default generator bindings or Default consumer bindings, click **Signing information >** *signing_information_name*.
3. Under Additional properties, click **Part references >***part_reference_name*.
4. Under Additional properties, click **Transforms**.
5. Click **New** to create a transform configuration or click the name of an existing configuration to modify its settings.

To view this administrative console page for the server level, complete the following steps:
1. Click **Application Servers > Servers >***server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Default generator bindings or Default consumer bindings, click **Signing information >** *signing_information_name*.
4. Under Additional properties, click **Part references >***part_reference_name*.
5. Under Additional properties, click **Transforms**.
6. Click **New** to create a transform configuration or click the name of an existing configuration to modify its settings.

**6.x application** To view this administrative console page for the application level, complete the following steps. This option is available for version 6.x applications only.
1. Click **Applications > Enterprise applications >***application_name*.
2. Under Related items, click **EJB modules** or **Web modules >***URI_name*.
3. Under Additional properties, you can access the transforms information for the following bindings:
   - For the Request generator (sender) binding, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom**.

- For the Request consumer (receiver) binding, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**.
- For the Response generator (sender) binding, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**.
- For the Response consumer (receiver) binding, click **Web services: Client security bindings**. Under Request consumer (receiver) binding, click **Edit custom**.

4. Under Required properties, click **Signing information >***signing_information_name*.
5. Under Additional properties, click **Part references >***part_name* **> Transforms**.
6. Click **New** to create a transform configuration or click the name of an existing configuration to modify its settings.

You must specify a transform name and select a transform algorithm before specifying additional properties.

**Related reference**

"Transforms collection" on page 724
Use this page to view the transform algorithm that is used for processing the Web services security message.

*Transform name:*

Specifies the name that is assigned to the transform algorithm.

*Transform algorithm:*

Specifies the algorithm Uniform Resource Identifier (URI) of the transform algorithm.

WebSphere Application Server supports the following algorithms:

**http://www.w3.org/2001/10/xml-exc-c14n#**
This algorithm specifies the World Wide Web Consortium (W3C) Exclusive Canonicalization recommendation.

**http://www.w3.org/TR/1999/REC-xpath-19991116**
This algorithm specifies the W3C XML path language recommendation. If you specify this algorithm, you must specify the property name and value by clicking **Properties**, which is displayed under Additional properties. For example, you might specify the following information:

**Property**
`com.ibm.wsspi.wssecurity.dsig.XPathExpression`

**Value** `not(ancestor-or-self::*[namespace-uri()='http://www.w3.org/2000/09/xmldsig#' and local-name()='Signature'])`

**http://www.w3.org/2002/06/xmldsig-filter2**
This algorithm specifies the XML-Signature XPath Filter Version 2.0 proposed recommendation.

When you use this algorithm, you must specify a set of properties. You can use multiple property sets for the XPath Filter Version 2. Therefore, it is recommended that your property names end with the number of the property set, which is denoted by an asterisk in the following examples:

- To specify an XPath expression for the XPath filter2, you might use:

```
name com.ibm.wsspi.wssecurity.dsig.XPath2Expression_*
```

- To specify a filter type for each XPath, you might use:

```
name com.ibm.wsspi.wssecurity.dsig.XPath2Filter_*
```

Following this expression, you can have a value, `[intersect]`, `[subtract]`, or `[union]`.

- To specify the processing order for each XPath, you might use:

```
name com.ibm.wsspi.wssecurity.dsig.XPath2Order_*
```

  Following this expression, indicate the processing order of the XPath.

The following is a list of complete examples:

```
com.ibm.wsspi.wssecurity.dsign.XPath2Filter_1 = [intersect]
com.ibm.wsspi.wssecurity.dsign.XPath2Order_1 = [1]
com.ibm.wsspi.wssecurity.dsign.XPath2Expression_2 = [XPath expression#2]
com.ibm.wsspi.wssecurity.dsign.XPath2Filter_2 = [subtract]
com.ibm.wsspi.wssecurity.dsign.XPath2Filter_2 = [1]
```

**http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform**

**http://www.w3.org/2002/07/decrypt#XML**
This algorithm specifies the W3C decryption transform for XML Signature recommendation.

**http://www.w3.org/2000/09/xmldsig#enveloped-signature**
This algorithm specifies the W3C recommendation for XML digital signatures.

## Configuring the encryption information for the generator binding on the application level

Before you begin this task, you must configure the key information that is referenced by the key information references in the encryption information panel.

This task provides the steps that are needed for configuring encryption information for the request generator (client side) and the response generator (server side) bindings at the application level. This encryption information is used to specify how the generators (senders) encrypt outgoing messages.

Complete the following steps to configure the encryption information for the request generator or response generator section of the bindings file on the application level:

1. Locate the encryption information configuration panel in the administrative console.
   a. Click **Applications > Enterprise applications >***application_name*.
   b. Under Related Items, click **EJB Modules** or **Web Modules >** *URI_name*.
   c. Under Additional properties, you can access the key information for the request generator and response generator bindings.
      - For the request generator (sender) binding, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom**.
      - For the response generator (sender) binding, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**.
   d. Under Required properties, click **Encryption information**.
   e. Click **New** to create an encryption information configuration. Click **Delete** to delete an existing configuration or click the name of an existing encryption information configuration to edit its settings. If you are creating a new configuration, enter a name in the Encryption information name field. For example, you might specify `gen_encinfo`.
2. Select a data encryption algorithm from the Data encryption algorithm field. The selection specifies the algorithm that is used to encrypt parts of the message. WebSphere Application Server supports the following pre-configured algorithms:
   - http://www.w3.org/2001/04/xmlenc#tripledes-cbc
   - http://www.w3.org/2001/04/xmlenc#aes128-cbc
   - http://www.w3.org/2001/04/xmlenc#aes256-cbc

To use this algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.

- http://www.w3.org/2001/04/xmlenc#aes192-cbc

  To use this algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.

The data encryption algorithm that you select for the generator side must match the data encryption method that you select for the consumer side.

3. Select a key encryption algorithm from the Key encryption algorithm field. This selection specifies the algorithm that is used to encrypt keys. WebSphere Application Server supports the following pre-configured algorithms:
   - http://www.w3.org/2001/04/xmlenc#rsa-1_5
   - http://www.w3.org/2001/04/xmlenc#kw-tripledes
   - http://www.w3.org/2001/04/xmlenc#kw-aes128
   - http://www.w3.org/2001/04/xmlenc#kw-aes256

     To use this algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.
   - http://www.w3.org/2001/04/xmlenc#kw-aes192

     To use this algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.

   The key encryption algorithm that you select for the generator side must match the key encryption method that you select for the consumer side.
4. Select an encryption key information reference from the Encryption key information menu. This selection is a reference to the encryption key that is used to encrypt parts of the message. To configure the key information, see "Configuring the key information for the generator binding on the application level" on page 699.
5. Select a part reference from the Part reference field. This field specifies the name of the part reference for the generator binding element in the deployment descriptor.
6. Click **OK** and then click **Save** to save the configuration.

The encryption information is configured for the generator binding at the application level

You must specify a similar encryption information configuration for the consumer.

**Related tasks**

"Configuring the encryption information for the consumer binding on the application level" on page 757

"Configuring the key information for the generator binding on the application level" on page 699

**Related information**

IBM developer kit: Security information

Web Services Security: SOAP Message Security Version 1.0

***Encryption information collection:***

Use this page to specify the configuration for the encrypting and decrypting parameters. This configuration is used to encrypt and decrypt parts of the message, including the body and user name token.

To view the administrative console panel for the encryption information on the server level, complete the following steps:

1. Click **Servers > Application servers >** *server_name*.

2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under either Default generator bindings or Default consumer bindings, click **Encryption information.**

To view this administrative console page for the collection certificate store on the application level, complete the following steps:
1. Click **Applications > Enterprise applications** > *application_name*.
2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.
3. Under Additional properties, you can access encryption information for the following bindings:
   - For the Request generator, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom**. Under Required properties, click **Encryption information**.
   - For the Request consumer, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**. Under Required properties, click **Encryption information**.
   - For the Response generator, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**. Under Required properties, click **Encryption information**.
   - For the Response consumer, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**. Under Required properties, click **Encryption information**.
4. **5.x application** Under Additional properties, you can access encryption information for the following bindings:
   - For the Request receiver, click **Web services: Server security bindings**. Under Request receiver binding, click **Edit**. Under Additional properties, click **Encryption information**.
   - For the Response receiver, click **Web services: Client security bindings**. Under Response receiver binding, click **Edit**. Under Additional properties, click **Encryption information**.

**Related reference**

"Encryption information configuration settings"
Use this page to configure the encryption and decryption parameters. You can use these parameters to encrypt and decrypt various parts of the message, including the body and user name token.

"Web services: Server security bindings collection" on page 691
Use this page to view a list of server-side binding configurations for Web services security.

*Encryption information name:*

Specifies the name of the encryption information.

*Key locator reference:*

Specifies the name of the key locator configuration that retrieves the key for XML digital signature and XML encryption.

*Key encryption algorithm:* Specifies the algorithm that is used to encrypt and decrypt keys.

*Data encryption algorithm:* Specifies the algorithm that is used to encrypt and decrypt data.

***Encryption information configuration settings:***

Use this page to configure the encryption and decryption parameters. You can use these parameters to encrypt and decrypt various parts of the message, including the body and user name token.

To view the administrative console panel for the encryption information on the server level, complete the following steps:

1. Click **Servers > Application servers >** *server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under either Default generator bindings or Default consumer bindings, click **Encryption information**.
4. Click either **New** to create a new encryption configuration or click the name of an existing encryption configuration.

To view this administrative console page for the collection certificate store on the application level, complete the following steps:

1. Click **Applications > Enterprise applications** > *application_name*.
2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.
3. Under Additional properties, you can access encryption information for the following bindings:
   - For the Request generator, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom**. Under Required properties, click **Encryption information**.
   - For the Request consumer, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**. Under Required properties, click **Encryption information**.
   - For the Response generator, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**. Under Required properties, click **Encryption information**.
   - For the Response consumer, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**. Under Required properties, click **Encryption information**.
4. **5.x application** Under Additional properties, you can access encryption information for the following bindings:
   - For the Request receiver, click **Web services: Server security bindings**. Under Request receiver binding, click **Edit**. Under Additional properties, click **Encryption information**.
   - For the Response receiver, click **Web services: Client security bindings**. Under Response receiver binding, click **Edit**. Under Additional properties, click **Encryption information**.
5. Click either **New** to create a new encryption configuration or click the name of an existing encryption configuration.

   **Related reference**

   "Encryption information collection" on page 728
   Use this page to specify the configuration for the encrypting and decrypting parameters. This configuration is used to encrypt and decrypt parts of the message, including the body and user name token.

   "Key locator collection" on page 694
   Use this page to view a list of key locator configurations that retrieve keys from the keystore for digital signature and encryption. A key locator must implement the com.ibm.wsspi.wssecurity.config.KeyLocator interface.

   "Encryption information configuration settings" on page 734
   Use this page to configure the encryption and decryption parameters.

*Encryption information name:*

Specifies the name for the encryption information.

| **Data type** | String |
|---|---|

*Data encryption algorithm:*

Specifies the algorithm URI of the data encryption method.

The following algorithms are supported:
- http://www.w3.org/2001/04/xmlenc#tripledes-cbc
- http://www.w3.org/2001/04/xmlenc#aes128-cbc
- http://www.w3.org/2001/04/xmlenc#aes256-cbc. To use this algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html. For more information, see "Encryption information configuration settings" on page 734.
- http://www.w3.org/2001/04/xmlenc#aes192-cbc. To use this algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html. For more information, see "Encryption information configuration settings" on page 734.

By default, the Java Cryptography Extension (JCE) is shipped with restricted or limited strength ciphers. To use 192-bit and 256- bit Advanced Encryption Standard (AES) encryption algorithms, you must apply unlimited jurisdiction policy files. For more information, see the Key encryption algorithm field description.

*Key locator reference:*

Specifies the name of the key locator configuration that retrieves the key for XML digital signature and XML encryption.

The Key locator reference field is displayed for the request receiver and response receiver bindings, which are used by Version 5.x applications.

You can configure these key locator reference options on the server level and the application level. The configurations that are listed in the field are a combination of the configurations on these two levels.

You can specify an encryption key configuration for the following bindings on the following levels:

| Binding name | Cell level, server level, or application level | Path |
| --- | --- | --- |
| Default generator binding | Server level | 1. Click **Servers > Application servers >***server_name*. 2. Under Security, click **Web services: Default bindings for Web services security**. 3. Under Additional properties, click **Key locators**. |
| Default consumer binding | Server level | 1. Click **Servers > Application servers >***server_name*. 2. Under Security, click **Web services: Default bindings for Web services security**. 3. Under Additional properties, click **Key locators**. |
| Request sender | Application level | 1. Click **Applications > Enterprise applications** > *application_name*. 2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*. 3. Click **Web services: Client security bindings**. Under Request sender binding, click **Edit**. 4. Under Additional properties, click **Key locators**. |

| Binding name | Cell level, server level, or application level | Path |
|---|---|---|
| Request receiver | Application level | 1. Click **Applications > Enterprise applications** > *application_name*.<br>2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.<br>3. Click **Web services: Server security bindings**. Under Request receiver binding, click **Edit**.<br>4. Under Additional properties, click **Key locators**. |
| Response sender | Application level | 1. Click **Applications > Enterprise applications** > *application_name*.<br>2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.<br>3. Click **Web services: Server security bindings**. Under Response sender binding, click **Edit**.<br>4. Under Additional properties, click **Key locators**. |
| Response receiver | Application level | 1. Click **Applications > Enterprise applications** > *application_name*.<br>2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.<br>3. Click **Web services: Client security bindings**. Under Response receiver binding, click **Edit**.<br>4. Under Additional properties, click **Key locators**. |

*Key encryption algorithm:*

Specifies the algorithm Uniform Resource Identifier (URI) of the key encryption method.

The following algorithms are provided by WebSphere Application Server:
- http://www.w3.org/2001/04/xmlenc#rsa-1_5
- http://www.w3.org/2001/04/xmlenc#kw-tripledes
- http://www.w3.org/2001/04/xmlenc#kw-aes128
- http://www.w3.org/2001/04/xmlenc#kw-aes192
- http://www.w3.org/2001/04/xmlenc#kw-aes256

By default, the Java Cryptography Extension (JCE) ships with restricted or limited strength ciphers. To use 192-bit and 256- bit Advanced Encryption Standard (AES) encryption algorithms, you must apply unlimited jurisdiction policy files. Before downloading these policy files, back up the existing policy files (`local_policy.jar` and `US_export_policy.jar` in the *`WAS_HOME`*`/jre/lib/security/` directory) prior to overwriting them in case you want to restore the original files later. To download the policy files, complete either of the following sets of steps:
- For WebSphere Application Server platforms using IBM Developer Kit, Java Technology Edition Version 1.4.2, including the AIX, Linux, and Windows platforms, you can obtain unlimited jurisdiction policy files by completing the following steps:
  1. Go to the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html

2. Click **JAVA 1.4.2 material > IBM SDK Policy files**.
3. Register, if necessary, and log into the Web site.
4. Locate the correct version of the Java Cryptography Extension (JCE) policy file and click **Download now**.

   The `unrestrict.zip` file is downloaded onto your machine.

After following either of these sets of steps, two Java archive (JAR) files are placed in the Java virtual machine (JVM) `jre/lib/security/` directory.

To specify custom algorithms on the server level, complete the following steps:
1. Click **Servers > Application servers >** *server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Algorithm mappings**.
4. Click **New** to specify a new algorithm mapping or click the name of an existing configuration to modify its settings.
5. Under Additional properties, click **Algorithm URI**.
6. Click **New** to create a new algorithm URI. You must specify **Key encryption** in the **Algorithm type** field to have the configuration display in the **Key encryption algorithm** field on the Encryption information configuration settings panel.

*Encryption key information:*

Specifies the name of the key information reference that is used for encryption. This reference is resolved to the actual key by the specified key locator and defined in the key information.

**6.x application** You must specify either one or no encryption key configurations for the request generator and response generator bindings.

**6.x application** For the response consumer and the request consumer bindings, you can configure multiple encryption key references. To create a new encryption key reference, under Additional properties, click **Key information references**.

You can specify an encryption key configuration for the following bindings on the following levels:

| Binding name | Cell level, server level, or application level | Path |
| --- | --- | --- |
| Default generator binding | Server level | 1. Click **Servers > Application servers >** *server_name*.<br>2. Under Security, click **Web services: Default bindings for Web services security**.<br>3. Under Default generator binding, click **Key information**. |
| Default consumer binding | Server level | 1. Click **Servers > Application servers >** *server_name*.<br>2. Under Security, click **Web services: Default bindings for Web services security**.<br>3. Under Default consumer binding, click **Key information**. |

| Binding name | Cell level, server level, or application level | Path |
| --- | --- | --- |
| Request generator (sender) binding | Application level | 1. Click **Applications > Enterprise applications** > *application_name*.<br>2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.<br>3. Under Additional properties, click **Web services: Client security bindings**.<br>4. Under Request generator (sender) binding, click **Edit custom**.<br>5. Under Required properties, click **Key information**. |
| Response generator (sender) binding | Application level | 1. Click **Applications > Enterprise applications** > *application_name*.<br>2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.<br>3. Under Additional properties, click **Web services: Server security bindings**.<br>4. Under Response generator (sender) binding, click **Edit custom**.<br>5. Under Required properties, click **Key information**. |

*Part Reference:*

Specifies the name of the <confidentiality> element for the generator binding or the <requiredConfidentiality> element for the consumer binding element in the deployment descriptor.

This field is available on the application level only.

***Encryption information configuration settings:***

Use this page to configure the encryption and decryption parameters.

The specifications that are listed on this page for the signature method, digest method, and canonicalization method are located in the World Wide Web Consortium (W3C) document entitled, *XML Encryption Syntax and Processing: W3C Recommendation 10 Dec 2002*.

To view this administrative console page, complete the following steps:

1. Click **Applications > Enterprise Applications >***application_name* and complete one of the following steps:
   - Under Related Items, click **EJB modules** or **Web modules >***URI_file_name* > **Web Services: Client Security Bindings**. Under Request sender binding, click **Edit**. Under Additional properties, click **Encryption Information**.
   - Under Related Items, click **EJB modules** or **Web modules >***URI_file_name* > **Web Services: Server Security Bindings**. Under Response sender binding, click **Edit**. Under Additional properties, click **Encryption Information**.
2. Select **None** or **Dedicated encryption information**. WebSphere Application Server can have either one or no encryption configurations for the request sender and the response sender bindings. If you are not using encryption, select **None**. To configure encryption for either of these two bindings, select **Dedicated encryption information** and specify the configuration settings using the fields that are described in this article.

**Related reference**

"Encryption information collection" on page 728
Use this page to specify the configuration for the encrypting and decrypting parameters. This configuration is used to encrypt and decrypt parts of the message, including the body and user name token.

"Key locator collection" on page 694
Use this page to view a list of key locator configurations that retrieve keys from the keystore for digital signature and encryption. A key locator must implement the com.ibm.wsspi.wssecurity.config.KeyLocator interface.

*Encryption information name:*

Specifies the name of the key locator configuration that retrieves the key for XML digital signature and XML encryption.

*Key locator reference:*

Specifies the name that is used to reference the key locator.

You can configure these key locator reference options on the server level and the application level. The configurations that are listed in the field are a combination of the configurations on these two levels.

To configure the key locators on the server level, complete the following steps:

1. Click **Servers > Application servers >***server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Key locators**.

To configure the key locators on the application level, complete the following steps:

1. Click **Applications > Enterprise applications** > *application_name*.
2. Under Related items, click **EJB modules** > *URI_name*.
3. Under Additional properties, you can access the key locators for the following bindings:
   - For the Request sender, click **Web services: Client security bindings**. Under Request sender binding, click **Edit**. Under Additional properties, click **Key locators**.
   - For the Request receiver, click **Web services: Server security bindings**. Under Request receiver binding, click **Edit**. Under Additional properties, click **Key locators**.
   - For the Response sender, click **Web services: Server security bindings**. Under Response sender binding, click **Edit**. Under Additional properties, click **Key locators**.
   - For the Response receiver, click **Web services: Client security bindings**. Under Response receiver binding, click **Edit**. Under Additional properties, click **Key locators**.

*Encryption key name:*

Specifies the name of the encryption key that is resolved to the actual key by the specified key locator.

| **Data type** | String |
| --- | --- |

*Key encryption algorithm:*

Specifies the algorithm uniform resource identifier (URI) of the key encryption method.

The following algorithms are supported:

- http://www.w3.org/2001/04/xmlenc#rsa-1_5.
- http://www.w3.org/2001/04/xmlenc#kw-tripledes.

- http://www.w3.org/2001/04/xmlenc#kw-aes128.
- http://www.w3.org/2001/04/xmlenc#kw-aes192.

  To use this algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file.
- http://www.w3.org/2001/04/xmlenc#kw-aes256.

  To use this algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file.

By default, the Java Cryptography Extension (JCE) is shipped with restricted or limited strength ciphers. To use 192-bit and 256- bit Advanced Encryption Standard (AES) encryption algorithms, you must apply unlimited jurisdiction policy files. Before downloading these policy files, back up the existing policy files (`local_policy.jar` and `US_export_policy.jar` in the *WAS_HOME*/jre/lib/security/ directory) prior to overwriting them in case you want to restore the original files later. To download the policy files, complete either of the following sets of steps:

- For WebSphere Application Server platforms using IBM Developer Kit, Java Technology Edition Version 1.4.2, including the AIX, Linux, and Windows platforms, you can obtain unlimited jurisdiction policy files by completing the following steps:
  1. Go to the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html
  2. Click **Java 1.4.2 material**
  3. Click **IBM SDK Policy files**.
  4. Select **Unrestricted JCE Policy files for SDK 1.4.2**
  5. Enter your user ID and password or register with IBM to download the policy files. The policy files are downloaded onto your machine.
- For WebSphere Application Server platforms using the Sun-based Java Development Kit (JDK) Version 1.4.2, including the Solaris environments and the HP-UX platform, you can obtain unlimited jurisdiction policy files by completing the following steps:
  1. Go to the following Web site: http://java.sun.com/j2se/1.4.2/download.html
  2. Click **Archive area**.
  3. Locate the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 1.4.2 information and click **Download**. The `jce_policy-1_4_1.zip` file is downloaded onto your machine.

After following either of these sets of steps, two Java archive (JAR) files are placed in the Java virtual machine (JVM) `jre/lib/security/` directory.

*Data encryption algorithm:*

Specifies the algorithm Uniform Resource Identifiers (URI) of the data encryption method.

The following algorithms are supported:

- http://www.w3.org/2001/04/xmlenc#tripledes-cbc
- http://www.w3.org/2001/04/xmlenc#aes128-cbc
- http://www.w3.org/2001/04/xmlenc#aes192-cbc
- http://www.w3.org/2001/04/xmlenc#aes256-cbc

By default, the JCE ships with restricted or limited strength ciphers. To use 192-bit and 256- bit AES encryption algorithms, you must apply unlimited jurisdiction policy files. For more information, see the Key encryption algorithm field description.

## Configuring trust anchors for the consumer binding on the application level

This document describes how to configure trust anchors for the consumer binding at the application level. It does not describe how to configure trust anchors at the server or cell level. Trust anchors that are

defined at the application level have a higher precedence over trust anchors that are defined at the server or cell level. For more information on creating and configuring trust anchors on the server or cell level, see “Configuring trust anchors on the server or cell level” on page 761.

You can configure a trust anchor for the trust anchor using an assembly tool or the administrative console. This document describes how to configure the application-level trust anchor using the administrative console.

A trust anchor specifies key stores that contain trusted root Certificate Authority (CA) certificates, which validate the signer certificate. These keystores are used by the request consumer (as defined in the `ibm-webservices-bnd.xmi` file) and the response consumer (as defined in the `ibm-webservicesclient-bnd.xmi` file when a Web service is acting as a client) to validate the X.509 certificate in the Simple Object Access protocol (SOAP) message. The keystores are critical to the integrity of the digital signature validation. If the keystores are tampered with, the result of the digital signature verification is doubtful and comprised. Therefore, it is recommended that you secure these keystores. The binding configuration specified for the request consumer in the `ibm-webservices-bnd.xmi` file must match the binding configuration for the response consumer in the `ibm-webservicesclient-bnd.xmi` file. The trust anchor configuration for the request consumer on the server side must match the request generator configuration on the client side. Also, the trust anchor configuration for the response consumer on the client side must match the response generator configuration on the server side.

Complete the following steps to configure trust anchors for the consumer binding on the application level:

1. Locate the trust anchor panel in the administrative console.
   a. Click **Applications > Enterprise applications >***application_name*.
   b. Under Related Items, click **EJB Modules** or **Web Modules >** *URI_name*.
   c. Under Additional properties you can access the trust anchor configuration for the following bindings:
      - For the request consumer (receiver) binding, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**.
      - For the response consumer (receiver) binding, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**.
   d. Under Additional properties, click **Trust anchors**.
   e. Click **New** to create a trust anchor configuration. Select the box next to a configuration and click **Delete** to delete an existing configuration or click the name of an existing trust anchor configuration to edit its settings. If you are creating a new configuration, enter a unique name in the Trust anchor name field.
2. Specify the keystore password, the keystore location, and the keystore type. A trust anchor keystore file contains the trusted root Certificate Authority (CA) certificates that are used for validating the X.509 certificate that is used in digital signature or XML encryption.
   a. Specify a password in the Key store password field. This password is used to access the keystore file.
   b. Specify the location of the keystore file in the Key store path field.
   c. Select a keystore type from the Key store type field. The Java Cryptography Extension (JCE) that is used by IBM supports the following keystore types:

      **JKS**
      Use this option if you are not using Java Cryptography Extensions (JCE) and if your keystore file uses the Java Keystore (JKS) format.

      **JCEKS**
      Use this option if you are using Java Cryptography Extensions.

      **PKCS11KS (PKCS11)**
      Use this format if your keystore file uses the PKCS#11 file format. Keystore files that use this format might contain RSA keys on cryptographic hardware or might encrypt keys that use cryptographic hardware to ensure protection.

**PKCS12KS (PKCS12)**
Use this option if your keystore file uses the PKCS#12 file format.

WebSphere Application Server provides some sample keystore files in the `${USER_INSTALL_ROOT}/etc/ws-security/samples` directory. For example, you might use the `enc-receiver.jceks` keystore file for encryption keys. The password for this file is `storepass` and the type is JCEKS.

**Attention:** Do not use these keystore files in a production environment. These samples are provided for testing purposes only.

This task configures trust anchors for the consumer binding at the application level

You must specify a similar trust anchor information for the generator.

**Related tasks**

"Configuring trust anchors for the generator binding on the application level" on page 659

"Configuring trust anchors on the server or cell level" on page 761

**Related reference**

"Trust anchor collection" on page 660
Use this page to view a list of keystore objects that contain trusted root certificates. These objects are used for certificate path validation of incoming X.509-formatted security tokens. Keystore objects within trust anchors contain trusted root certificates that are used by the CertPath API to validate the trust of a certificate chain.

"Trust anchor configuration settings" on page 661
Use this information to configure a trust anchor. Trust anchors point to keystores that contain trusted root or self-signed certificates. This information enables you to specify a name for the trust anchor and the information that is needed to access a keystore. The application binding uses this name to reference a predefined trust anchor definition in the binding file (or the default).

## Configuring the collection certificate store for the consumer binding on the application level

A collection certificate store is a collection of non-root, certificate authority (CA) certificates and certificate revocation lists (CRLs). This collection of CA certificates and CRLs is used to check for a valid signature in a digitally signed Simple Object Access Protocol (SOAP) message. Complete the following steps to configure a collection certificate for the consumer bindings on the application level:

1. Locate the collection certificate store configuration panel in the administrative console.
   a. Click **Applications** > **Enterprise applications** >*application_name*.
   b. Under Related Items, click **EJB Modules** or **Web Modules** > *URI_name*.
   c. Under Additional properties, click **Collection certificate store**.
2. Click **New** to create a collection certificate store configuration, click **Delete** to delete an existing configuration, or click the name of an existing collection certificate store configuration to edit its settings. If you are creating a new configuration, enter a name in the Certificate store name field.

   The name of the collection certificate store must be unique to the level of the application server. For example, if you create the collection certificate store for the application level, the store name must be unique to the application level. The name that is specified in the Certificate store name field is used by other configurations to refer to a predefined collection certificate store. WebSphere Application Server searches for the collection certificate store based on proximity.

   For example, if an application binding refers to a collection certificate store named `cert1`, the Application Server searches for `cert1` at the application level before searching the server level.
3. Specify a certificate store provider in the Certificate store provider field. WebSphere Application Server supports the IBMCertPath certificate store provider. To use another certificate store provider, you must define the provider implementation in the provider list within the

*install_dir*/java/jre/lib/security/java.security file. However, make sure that your provider supports the same requirements of the certificate path algorithm as WebSphere Application Server.

4. Click **OK** and **Save** to save the configuration.
5. Click the name of your certificate store configuration. After you specify the certificate store provider, you must specify either the location of a certificate revocation list or the X.509 certificates. However, you can specify both a certificate revocation list and the X.509 certificates for your certificate store configuration.
6. Under Additional properties, click **Certificate revocation lists**.
7. Click **New** to specify a certificate revocation list path, click **Delete** to delete an existing list reference, or click the name of an existing reference to edit the path. You must specify the fully qualified path to the location where WebSphere Application Server can find your list of certificates that are not valid. For portability reasons, it is recommended that you use the WebSphere Application Server variables to specify a relative path to the certificate revocation lists (CRL). This recommendation is especially important when you are working in a WebSphere Application Server Network Deployment environment. For example, you might use the *USER_INSTALL_ROOT* variable to define a path such as $*USER_INSTALL_ROOT*/mycertstore/mycrl1. For a list of supported variables, click **Environment > WebSphere variables** in the administrative console. The following list provides recommendation for using certificate revocation lists:
   - If CRLs are added to the collection certificate store, add the CRLs for the root certificate authority and each intermediate certificate, if applicable. When the CRL is in the certificate collection store, the certificate revocation status for every certificate in the chain is checked against the CRL of the issuer.
   - When the CRL file is updated, the new CRL does not take effect until you restart the Web service application.
   - Before a CRL expires, you must load a new CRL into the certificate collection store to replace the old CRL. An expired CRL in the collection certificate store results in a certificate path (CertPath) build failure.
8. Click **OK** and **Save** to save the configuration.
9. Return to the Collection certificate store configuration panel. See the first few steps of this article to locate the collection certificate store panel.
10. Under Additional properties, click **X.509 certificates**.
11. Click **New** to create a new configuration for X.509 certificates, click **Delete** to delete an existing configuration, or click the name of an existing X.509 certificate configuration to edit its settings. If you are creating a new configuration, enter a name in the Certificate store name field.
12. Specify a path in the X.509 certificate path field. This entry is the absolute path to the location of the X.509 certificates. The collection certificate store is used to validate the certificate path of incoming X.509-formatted security tokens.

    You can use the *USER_INSTALL_ROOT* variable as part of the path name. For example, you might type: *USER_INSTALL_ROOT*/etc/ws-security/samples/intca2.cer. Do not use this certificate path for production use. You must obtain your own X.509 certificate from a certificate authority before putting your WebSphere Application Server environment into production.

    Click **Environment > WebSphere variables** in the administrative console to configure the *USER_INSTALL_ROOT* variable.
13. Click **OK** and then **Save** to save your configuration.

You have configured the collection certificate store for the consumer binding.

You must configure a token consumer configuration that references this certificate store configuration.

**Related tasks**

"Configuring the collection certificate store for the generator binding on the application level" on page 663

## Binary security token

The ValueType attribute identifies the type of the security token, for example, a Lightweight Third Party Authentication (LTPA) token. The EncodingType type indicates how the security token is encoded, for example, Base64Binary. The BinarySecurityToken element defines a security token that is binary encoded. The encoding is specified using the EncodingType attribute. The value type and space are specified using the ValueType attribute. The Web services security implementation for WebSphere Application Server, Version 6 supports both LTPA and X.509 certificate binary security tokens.

A binary security token has the following attributes that are used for interpretation:

- Value type
- Encoding type

The following example depicts an LTPA binary security token in a Web services security message header:

```
<wsse:BinarySecurityToken xmlns:ns7902342339871340177=
     "http://www.ibm.com/websphere/appserver/tokentype/5.0.2"
     EncodingType="wsse:Base64Binary"
     ValueType="ns7902342339871340177:LTPA">
            MIZ6LGPt2CzXBQfio9wZTo1VotWov0NW3Za6lU5K7Li78DSnIK6iHj3hxXgrUn6p4wZI
            8Xg26havepvmSJ8XxiACMihTJuh1t3ufsrjbFQJOqh5VcRvI+AKEaNmnEgEV65jUYAC9
            C/iwBBWk5U/6DIk7LfXcTT0ZPAd+3D3nCS0f+6tnqMou8EG9mtMeTKccz/pJVTZjaRSo
            msu0sewsOKfl/WPsjW0bR/2g3NaVvBy18VlTFBpUbGFVGgzHRjBKAGo+ctkl80nlVLIk
            TUjt/XdYvEpOr6QoddGi4okjDGPyyoDxcvKZnReXww5UsoqlpfXwN4KG9as=
</wsse:BinarySecurityToken>
</wsse:Security>
</soapenv:Header>
```

As shown in the example, the token is Base64Binary encoded.

## Configuring token consumer on the application level

This task describes the steps that are needed to specify the token consumer on the application level. The information is used on the consumer side to incorporate the security token.

Complete the following steps to configure the token consumer on the application level:

1. Locate the token consumer panel in the administrative console.
   a. Click **Applications > Enterprise applications >***application_name*.
   b. Under Related Items, click **EJB Modules** or **Web Modules >** *URI_name*.
   c. Under Additional properties you can access the token consumer for the following bindings:
      - For the request consumer (receiver) binding, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**.
      - For the response consumer (receiver) binding, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**.
   d. Under Required properties, click **Token consumer**.
   e. Click **New** to create a token consumer configuration, click **Delete** to delete an existing configuration, or click the name of an existing token consumer configuration to edit its settings. If you are creating a new configuration, enter a unique name in the Token consumer name field. For example, you might specify `con_signtgen`.
2. Specify a class name in the Token consumer class name field. The token consumer class must implement the com.ibm.wsspi.wssecurity.token.TokenConsumerComponent interface. The token consumer class name for the request consumer and the response consumer must be similar to the token generator class name for the request generator and the response generator. For example, if your application requires a user name token consumer, you can specify the

com.ibm.wsspi.wssecurity.token.UsernameTokenGenerator class name on the Token generator panel for application level and the com.ibm.wsspi.wssecurity.token.UsernameTokenConsumer class name in this field.

3. **Optional:** Select a part reference in the Part reference field. The part reference indicates the name of the security token that is defined in the deployment descriptor. For example, if you receive a username token in your request message, you might want to reference the token in the username token consumer.

   **Important:** On the application level, if you do not specify a security token in your deployment descriptor, the Part reference field is not displayed. If you define a security token called `user_tcon` in your deployment descriptor, `user_tcon` is displayed as an option in the Part reference field.

4. **Optional:** In the certificate path section of the panel, select a certificate store type and indicate the trust anchor and certificate store name, if necessary. These options and fields are necessary when you specify com.ibm.wsspi.wssecurity.token.X509TokenConsumer as the token consumer class name. The names of the trust anchor and the collection certificate store are created in the certificate path under your token consumer. You can select one of the following options:

   **None** If you select this option, the certificate path is not specified.

   **Trust any**
   If you select this option, any certificate is trusted. When the received token is consumed, the Application Server does not validate the certificate path.

   **Dedicated signing information**
   If you select this option, you can select a trust anchor and a certificate store configuration. When you select the trust anchor or the certificate store of a trusted certificate, you must configure the trust anchor and the certificate store before setting the certificate path.

   **Trust anchor**
   A trust anchor specifies a list of key store configurations that contain trusted root certificates. These configurations are used to validate the certificate path of incoming X.509-formatted security tokens. Keystore objects within trust anchors contain trusted root certificates that are used by the CertPath API to validate the trustworthiness of a certificate chain. You must create the keystore file using the key tool utility, which is located in the *install_dir*/java/jre/bin/keytool file.

   You can configure trust anchors for the application level by completing the following steps:

   a. Click **Applications > Enterprise applications >***application_name*.
   b. Under Related Items, click **EJB Modules** or **Web Modules >** *URI_name*.
   c. Access the token consumer from the following bindings:
      - For the request consumer (receiver) binding, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**.
      - For the response consumer (receiver) binding, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**.
   d. Under Additional properties, click **Trust anchors**.

   **Collection certificate store**
   A collection certificate store includes a list of untrusted, intermediary certificates and certificate revocation lists (CRLs). The collection certificate store is used to validate the certificate path of the incoming X.509-formatted security tokens. You can configure the collection certificate store for the application level by completing the following steps:

   a. Click **Applications > Enterprise applications >***application_name*.

b. Under Related Items, click **EJB Modules** or **Web Modules >** *URI_name*.
c. Access the token consumer from the following bindings:
   - For the request consumer (receiver) binding, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**.
   - For the response consumer (receiver) binding, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**.
d. Under Additional properties, click **Collection certificate store**.

5. **Optional:** Specify a trusted ID evaluator. The trusted ID evaluator is used to determine whether to trust the received ID. You can select one of the following options:

**None** If you select this option, the trusted ID evaluator is not specified.

**Existing evaluator definition**
If you select this option, you can select one of the configured trusted ID evaluators. For example, you can select the SampleTrustedIDEvaluator, which is provided by WebSphere Application Server as an example.

**Binding evaluator definition**
If you select this option, you can configure a new trusted ID evaluator by specifying a trusted ID evaluator name and class name.

**Trusted ID evaluator name**
Specifies the name that is used by the application binding to refer to a trusted identity (ID) evaluator that is defined in the default bindings.

**Trusted ID evaluator class name**
Species the class name of the trusted ID evaluator. The specified trusted ID evaluator class name must implement the com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator interface. The default TrustedIDEvaluator class is com.ibm.wsspi.wssecurity.id.TrustedIDEvaluatorImpl. When you use this default TrustedIDEvaluator class, you must specify the name and value properties for the default trusted ID evaluator to create the trusted ID list for evaluation. To specify the name and value properties, complete the following steps:

a. Under Additional properties, click **Properties > New**.
b. Specify the trusted ID evaluator name in the Property field. You must specify the name in the the form, `trustedId_n` where `_n` is an integer from 0 to n.
c. Specify the trusted ID in the Value field.

For example:

```
property name="trustedId_0", value="CN=Bob,O=ACME,C=US"
property name="trustedId_1, value="user1"
```

If the distinguished name (DN) is used, the space is removed for comparison. See the programming model information in the documentation for an explanation of how to implement the com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator interface. For more information, see “Default implementations of the Web services security service provider programming interfaces” on page 558.

**Note:** Define the trusted ID evaluator on the server level instead of the application level. To define the trusted ID evaluator on the server level, complete the following steps:

a. Click **Servers > Application servers >***server_name*.
b. Under Security, click **Web services: Default bindings for Web services security**.

c. Under Additional properties, click **Trusted ID evaluators**.
d. Click **New** to define a new trusted ID evaluator.

The trusted ID evaluator configuration is available only for the token consumer on the server-side application level.

6. **Optional:** Select the **Verify nonce** option. This option indicates whether to verify a nonce in the user name token if it is specified for the token consumer. Nonce is a unique, cryptographic number that is embedded in a message to help stop repeat, unauthorized attacks of user name tokens. The **Verify nonce** option is valid only when the incorporated token type is a user name token.
7. **Optional:** Select the **Verify timestamp** option. This option indicates whether to verify a time stamp in the user name token. The **Verify nonce** option is valid only when the incorporated token type is a user name token.
8. Specify the value type local name in the Local name field. This field specifies the local name of the value type for the consumed token. For a user name token and an X.509 certificate security token, WebSphere Application Server provides predefined local names for the value type. When you specify any of the following local names, you do not need to specify a value type URI:

**http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#UsernameToken**
This local name specifies a user name token.

**http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509**
This local name specifies an X.509 certificate token.

**http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509PKIPathv1**
This local name specifies X.509 certificates in a public key infrastructure (PKI) path.

**http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#PKCS7**
This local name specifies a list of X.509 certificates and certificate revocation lists in a PKCS#7 format.

**LTPA** This local name specifies a Lightweight Third Party Authentication (LTPA) token.

**Important:** When you specify LTPA for the value type local name, you do not need to specify the value type Uniform Resource Identifier (URI), which is http://www.ibm.com/websphere/appserver/tokentype/5.0.2.

9. **Optional:** Specify the value type URI in the URI field. This entry specifies the namespace URI of the value type for the consumed token.

**Remember:** If you specify the token consumer for a username token or an X.509 certificate security token, you do not need to specify a value type URI.

If you want to specify another token, you must specify both the local name and the URI. For example, if you have an implementation of your own custom token, you can specify `CustomToken` in the Local name field and `http://www.ibm.com/custom`

10. Click **OK** and **Save** to save the configuration.
11. Click the name of your token consumer configuration.
12. Under Additional properties, click **JAAS configuration**. The Java Authentication and Authorization Service (JAAS) configuration specifies the name of the JAAS configuration that is defined in the JAAS login panel. The JAAS configuration specifies how the token logs in on the consumer side.
13. Select a JAAS configuration from the JAAS configuration name field. The field specifies the name of the JAAS system of application login configuration. You can specify additional JAAS system and application configurations by clicking **Security > Global security**. Under Authentication, click **JAAS configuration** and either **Application logins > New** or **System logins > New**. Do not remove the predefined system or application login configurations. However, within these configurations, you can add module class names and specify the order in which WebSphere Application Server loads each module. WebSphere Application Server provides the following predefined JAAS configurations:

**ClientContainer**
This selection specifies the login configuration that is used by the client container applications. The configuration uses the CallbackHandler application programming interface (API) that is defined in the deployment descriptor for the client container. To modify this configuration, see the JAAS configuration panel for application logins.

**WSLogin**
This selection specifies whether all of the applications can use the WSLogin configuration to perform authentication for the security run time. To modify this configuration, see the JAAS configuration panel for application logins.

**DefaultPrincipalMapping**
This selection specifies the login configuration that is used by Java 2 Connectors (J2C) to map users to principals that are defined in the J2C authentication data entries. To modify this configuration, see the JAAS configuration panel for application logins.

**system.wssecurity.IDAssertion**
This selection enables a Version 5.x application to use identity assertion to map a user name to a WebSphere Application Server credential principal. To modify this configuration, see the JAAS configuration panel for system logins.

**system.wssecurity.Signature**
This selection enables a Version 5.x application to map a distinguished name (DN) in a signed certificate to a WebSphere Application Server credential principal. To modify this configuration, see the JAAS configuration panel for system logins.

**system.LTPA_WEB**
This selection processes login requests that are used by the Web container such as servlets and JavaServer Pages (JSPs) files. To modify this configuration, see the JAAS configuration panel for system logins.

**system.WEB_INBOUND**
This selection handles login requests for Web applications, which include servlets and JavaServer Pages (JSP) files. This login configuration is used by WebSphere Application Server Version 5.1.1. To modify this configuration, see the JAAS configuration panel for system logins.

**system.RMI_INBOUND**
This selection handles logins for inbound Remote Method Invocation (RMI) requests. This login configuration is used by WebSphere Application Server Version 5.1.1. To modify this configuration, see the JAAS configuration panel for system logins.

**system.DEFAULT**
This selection handles the logins for inbound requests that are made by internal authentications and most of the other protocols except Web applications and RMI requests. This login configuration is used by WebSphere Application Server Version 5.1.1. To modify this configuration, see the JAAS configuration panel for system logins.

**system.RMI_OUTBOUND**
This selection processes RMI requests that are sent outbound to another server when either the com.ibm.CSI.rmiOutboundLoginEnabled or the com.ibm.CSIOutboundPropagationEnabled properties are `true`. These properties are set in the CSIv2 authentication panel.

To access the panel, click **Security > Global security**. Under Authentication, click **Authentication protocol > CSIv2 outbound authentication**. To set the com.ibm.CSI.rmiOutboundLoginEnabled property, select the **Custom outbound mapping** option. To set the com.ibm.CSIOutboundPropagationEnabled property, select the **Security attribute propagation** option. To modify this JAAS login configuration, see the JAAS configuration panel for system logins.

**system.wssecurity.X509BST**
: This selection verifies an X.509 binary security token (BST) by checking the validity of the certificate and the certificate path. To modify this configuration, see the JAAS configuration panel for system logins.

**system.wssecurity.PKCS7**
: This selection verifies an X.509 certificate within a PKCS7 object that might include a certificate chain, a certificate revocation list, or both. To modify this configuration, see the JAAS configuration panel for system logins.

**system.wssecurity.PkiPath**
: This section verifies an X.509 certificate with a public key infrastructure (PKI) path. To modify this configuration, see the JAAS configuration panel for system logins.

**system.wssecurity.UsernameToken**
: This selection verifies the basic authentication (user name and password) data. To modify this configuration, see the JAAS configuration panel for system logins.

**system.wssecurity.IDAssertionUsernameToken**
: This selection supports the use of identity assertion in Version 6 applications to map a user name to a WebSphere Application Server credential principal. To modify this configuration, see the JAAS configuration panel for system logins.

**None** With this selection, you do not specify a JAAS login configuration.

14. Click **OK** and then click **Save** to save the configuration.

You have configured the token consumer for the application level.

You must specify a similar token generator configuration for the application level.

**Related concepts**

"Default implementations of the Web services security service provider programming interfaces" on page 558

**Related tasks**

"Configuring the collection certificate store for the consumer binding on the application level" on page 738

"Configuring token consumer on the application level" on page 740

***Request consumer (receiver) binding configuration settings:***

Use this page to specify the binding configuration for the request consumer.

To view this administrative console page, complete the following steps:

1. Click **Applications > Enterprise applications >***application_name*.
2. Under Related items, click **EJB modules** or **Web modules**.
3. Click the Uniform Resource Identifier (URI).
4. Under Additional properties, click **Web services: Server security bindings**.
5. Under Request consumer (receiver) binding, click **Edit custom**.

The security constraints or bindings are defined using the application assembly process before the application is installed. WebSphere Application Server provides assembly tools to assemble your application.

If the security constraints are defined in the application, you must either define the corresponding binding information or select the Use defaults option on this panel and use the default binding information for the cell or server level. The default binding that is provided by WebSphere Application Server is a sample. Do

not use this sample in a production environment without modifying the configuration. The security constraints define what is signed or encrypted in the Web services security message. The bindings define how to enforce the requirements.

### Digital signature security constraint (integrity)

The following table shows the required and optional binding information when the digital signature security constraint (integrity) is defined in the deployment descriptor.

| Information type | Required or optional |
|---|---|
| Signing information | Required |
| Key information | Required |
| Token consumer | Required |
| Key locators | Optional |
| Collection certificate store | Optional |
| Trust anchors | Optional |
| Properties | Optional |

You can use the key locators, collection certificate stores, and trust anchors that are defined at either the server level or the cell level.

### Encryption constraint (confidentiality)

The following table shows the required and optional binding information when the encryption constraint (confidentiality) is defined in the deployment descriptor.

| Information type | Required or optional |
|---|---|
| Encryption information | Required |
| Key information | Required |
| Token consumer | Required |
| Key locators | Optional |
| Collection certificate store | Optional |
| Trust anchors | Optional |
| Properties | Optional |

You can use the key locators, collection certificate store, and trust anchors that are defined at either the server level or the cell level.

### Security token constraint

The following table shows the required and optional binding information when the security token constraint is defined in the deployment descriptor.

| Information type | Required or optional |
|---|---|
| Token consumer | Required |
| Collection certificate store | Optional |
| Trust anchors | Optional |
| Properties | Optional |

You can use the collection certificate store and trust anchors that are defined at the server level or the cell level.

**Related reference**

"Request generator (sender) binding configuration settings" on page 680
Use this page to specify the binding configuration for the request generator.

"Response generator (sender) binding configuration settings" on page 682
Use this page to specify the binding configuration for the response generator or response sender.

"Response consumer (receiver) binding configuration settings"
Use this page to specify the binding configuration for the response consumer.

*Use defaults:*

Select this option if you want to use the default binding information from the server or cell level.

*Port:*

Specifies the port in the Web service that is defined during application assembly.

*Web service:*

Specifies the name of the Web service that is defined during application assembly.

***Response consumer (receiver) binding configuration settings:***

Use this page to specify the binding configuration for the response consumer.

To view this administrative console page, complete the following steps:

1. Click **Applications > Enterprise applications >***application_name*.
2. Under Related items, click **EJB modules** or **Web modules**.
3. Click the Uniform Resource Identifier (URI).
4. Under Additional properties, click **Web services: Client security bindings**.
5. Under Response consumer (receiver) binding, click **Edit custom**.

The security constraints or bindings are defined using the application assembly process before the application is installed. WebSphere Application Server provides assembly tools to assemble your application.

If the security constraints are defined in the application, you must either define the corresponding binding information or select the Use defaults option on this panel and use the default binding information for the server or cell level. The default binding that is provided by WebSphere Application Server is a sample. Do not use this sample in a production environment without modifying the configuration. The security constraints define what is signed or encrypted in the Web services security message. The bindings define how to enforce the requirements.

**Digital signature security constraint (integrity)**

The following table shows the required and optional binding information when the digital signature security constraint (integrity) is defined in the deployment descriptor.

| Information type | Required or optional |
| --- | --- |
| Signing information | Required |
| Key information | Required |
| Token consumer | Optional |

| Information type | Required or optional |
| --- | --- |
| Key locators | Optional |
| Collection certificate store | Optional |
| Trust anchors | Optional |
| Properties | Optional |

You can use the key locators, collection certificate stores, and trust anchors that are defined at either the server level or the cell level.

**Encryption constraint (confidentiality)**

The following table shows the required and optional binding information when the encryption constraint (confidentiality) is defined in the deployment descriptor.

| Information type | Required or optional |
| --- | --- |
| Encryption information | Required |
| Key information | Required |
| Token consumer | Optional |
| Key locators | Optional |
| Collection certificate store | Optional |
| Trust anchors | Optional |
| Properties | Optional |

You can use the key locators, collection certificate store, and trust anchors that are defined at the application level, server level, or the cell level.

**Security token constraint**

The following table shows the required and optional binding information when the security token constraint is defined in the deployment descriptor.

| Information type | Required or optional |
| --- | --- |
| Token consumer | Required |
| Collection certificate store | Optional |
| Trust anchors | Optional |
| Properties | Optional |

You can use the collection certificate store and trust anchors that are defined at the application level, server level, or the cell level.

**Related reference**

"Request generator (sender) binding configuration settings" on page 680
Use this page to specify the binding configuration for the request generator.

"Request consumer (receiver) binding configuration settings" on page 745
Use this page to specify the binding configuration for the request consumer.

"Response generator (sender) binding configuration settings" on page 682
Use this page to specify the binding configuration for the response generator or response sender.

*Use defaults:*

Select this option if you want to use the default binding information from the cell or server level.

*Component:*

Specifies the enterprise bean in an assembled Enterprise JavaBeans (EJB) module.

*Port:*

Specifies the port in the Web service that is defined during application assembly.

*Web service:*

Specifies the name of the Web service that is defined during application assembly.

## Configuring the key locator for the consumer binding on the application level

The key locator information for the consumer at the application level specifies which key locator implementation is used to locate the key that is used to validate the digital signature or the encryption information by the application. Complete the following steps to configure the key locator for the consumer binding on the application level:

1. Locate the key locator configuration panel in the administrative console.
   a. Click **Applications > Enterprise applications >***application_name*.
   b. Under Related Items, click **EJB Modules** or **Web Modules >** *URI_name*.
   c. Under Additional properties, you can access the key information for the request consumer and response consumer bindings.
      - For the response consumer (receiver) binding, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**.
      - For the response consumer (receiver) binding, click **Web services: Server security bindings**. Under Response consumer (receiver) binding, click **Edit custom**.
   d. Under Additional properties, click **Key locators**.
   e. Click **New** to create a key locator configuration, click **Delete** and select the box next to the configuration to delete an existing configuration, or click the name of an existing key locator configuration to edit its settings. If you are creating a new configuration, enter a unique name in the Key locator name field. For example, you might specify `klocator`.
2. Specify a name for the key locator class implementation. Key locators that are associated with Version 6 applications must implement the com.ibm.wsspi.wssecurity.keyinfo.KeyLocator interface. Specify a class name according to the requirements of the application. For example, if the application requires that the key is read from a keystore file, specify the com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator implementation. WebSphere Application Server provides the following default key locator class implementations for Version 6 applications that are available to use with the request consumer or response consumer:

   **com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator**
   This implementation locates and obtains the key from the specified keystore file.

   **com.ibm.wsspi.wssecurity.keyinfo.X509TokenKeyLocator**
   This implementation uses the X.509 security token from the sender message for digital signature validation and encryption. This class implementation is used by the request consumer and the response consumer.
3. Specify the keystore password, the keystore location, and the keystore type. Keystore files contain public and private keys, root certificate authority (CA) certificates, the intermediate CA certificate, and so on. Keys that are retrieved from the keystore files are used to sign and validate or encrypt and decrypt messages or message parts. If you specified the com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator implementation for the key locator class implementation, you must specify a keystore password, location, and type.

a. Specify a password in the Key store password field. This password is used to access the keystore file.
b. Specify the location of the keystore file in the Key store path field.
c. Select a keystore type from the Key store type field. The Java Cryptography Extension (JCE) that is used by IBM supports the following keystore types:

**JKS** Use this option if you are not using Java Cryptography Extensions (JCE) and if your keystore file uses the Java Keystore (JKS) format.

**JCEKS**
Use this option if you are using Java Cryptography Extensions.

**PKCS11KS (PKCS11)**
Use this format if your keystore file uses the PKCS#11 file format. Keystore files that use this format might contain RSA keys on cryptographic hardware or might encrypt keys that use cryptographic hardware to ensure protection.

**PKCS12KS (PKCS12)**
Use this option if your keystore uses the PKCS#12 file format.

WebSphere Application Server provides some sample keystore files in the `${USER_INSTALL_ROOT}/etc/ws-security/samples` directory. For example, you might use the `enc-receiver.jceks` keystore file for encryption keys. The password for this file is `Storepass` and the type is JCEKS.

**Attention:** Do not use these keystore files in a production environment. These samples are provided for testing purposes only.

4. Click **OK** and **Save** to save the configuration.
5. Under Additional properties, click **Keys**.
6. Click **New** to create a key configuration, click **Delete** and select the box next to the configuration to delete an existing configuration, or click the name of an existing key configuration to edit its settings. This entry specifies the name of the key object within the keystore file. If you are creating a new configuration, enter a unique name in the Key name field.

   It is recommended that you use a fully qualified distinguished name for the key name. For example, you might use `CN=Bob,O=IBM,C=US`.
7. Specify an alias in the Key alias field. The key alias is used by the key locator to search for key objects in the keystore file.
8. Specify a password in the Key password field. The password is used to access the key object within the keystore file.
9. Click **OK** and then click **Save** to save the configuration.

You have configured the key locator for the consumer binding at the application level.

You must specify a similar key information configuration for the generator.

**Related tasks**

"Configuring the key locator for the generator binding on the application level" on page 692

## Configuring the key information for the consumer binding on the application level

Configure the key locators and the token consumers that are referenced by the Key locator reference and the Token reference fields within the key information panel.

This task provides the steps that are needed for configuring the key information for the request consumer (server side) and the response consumer (client side) bindings at the application level. The key information on the consumer side is used for specifying the information about the key, which is used for validating the

digital signature in the received message or for decrypting the encrypted parts of the message. Complete the following steps to configure the key information for consumer binding on the application level.

1. Locate the key information configuration panel in the administrative console.
   a. Click **Applications > Enterprise applications >***application_name*.
   b. Under Related Items, click **EJB Modules** or **Web Modules >** *URI_name*.
   c. Under Additional properties, you can access the key information for the request consumer and response consumer bindings.
      - For the request consumer (receiver) binding, click **Web services: Server security bindings**. Under request consumer (receiver) binding, click **Edit custom**.
      - For the response consumer (receiver) binding, click **Web services: Client security bindings**. Under response consumer (receiver) binding, click **Edit custom**.
   d. Under Required properties, click **Key information**.
   e. Click **New** to create a key information configuration, click **Delete** and select the box next to the configuration to delete an existing configuration, or click the name of an existing key information configuration to edit its settings. If you are creating a new configuration, enter a name in the Key information name field. For example, you might specify `con_signkeyinfo`.
2. Select a key information type from the Key information type field. The key information types specify different mechanisms for referencing security tokens using the <wsse:SecurityTokenReference> element within the <ds:KeyInfo> element. WebSphere Application Server supports the following key information types:

**Key identifier**
The security token is referenced using an opaque value that uniquely identifies the token. The algorithm that is used for generating the <KeyIdentifier> element value depends upon the token type. For example, you can use the identifier for the public keys that are defined in the Internet Engineering Task Force (IETF) Request for Comment (RFC) 3280. The following <KeyInfo> element is generated in the Simple Object Access Protocol (SOAP) message for this key information type:

```
<ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
   <wsse:SecurityTokenReference>
      <wsse:KeyIdentifier ValueType="http://docs.oasis-open.org/wss/2004/01
      /oasis-200401-wss-x509-token-profile-1.0#X509v3SubjectKeyIdentifier">/62wXO...
      </wsse:KeyIdentifier>
   </wsse:SecurityTokenReference>
</ds:KeyInfo>
```

**Key name**
The security token is referenced using a name that matches an identity assertion within the token. It is recommended that you do not use this key type as it might result in multiple security tokens that match the specified name. The following <KeyInfo> element is generated in the SOAP message for this key information type:

```
<ds:KeyInfo>
   <ds:KeyName>CN=Group1</ds:KeyName>
</ds:KeyInfo>
```

In general, use a key name when you use a Key-Hashing Message Authentication Code (HMAC) digital signature algorithm, such as http://www.w3.org/2000/09/xmldsig#hmac-sha1.

**Security token reference**
The security token is directly referenced using Universal Resource Identifiers (URIs). The following <KeyInfo> element is generated in the SOAP message for this key information type:

```
<ds:KeyInfo>
   <wsse:SecurityTokenReference>
      <wsse:Reference URI='#SomeCert'
```

```
      ValueType="http://docs.oasis-open.org/wss/2004/01/
         oasis-200401-wss-x509-token-profile-1.0#X509v3" />
   </wsse:SecurityTokenReference>
</ds:KeyInfo>
```

**Attention:** As stated in the Web services Interoperability Organization (WS-I) Basic Security Profile Version 1 draft and shown in the previous example, the wsse:Reference element in a SECURE_ENVELOPE must have a ValueType attribute.

**Embedded token**

The security token is directly embedded within the <SecurityTokenReference> element. The following <KeyInfo> element is generated in the SOAP message for this key information type:

```
<ds:KeyInfo>
   <wsse:SecurityTokenReference>
      <wsse:Embedded wsu:Id="tok1" />
         ...
      </wsse:Embedded>
   </wsse:SecurityTokenReference>
</ds:KeyInfo>
```

**X509 issuer name and issuer serial**

The security token is referenced by an issuer name and an issuer serial number of an X.509 certificate. The following <KeyInfo> element is generated in the SOAP message for this key information type:

```
<ds:KeyInfo>
   <wsse:SecurityTokenReference>
     <ds:X509Data>
        <ds:X509IssuerSerial>
           <ds:X509IssuerName>CN=Jones, O=IBM, C=US</ds:X509IssuerName>
           <ds:X509SerialNumber>1040152879</ds:X509SerialNumber>
        </ds:X509IssuerSerial>
     </ds:X509Data>
   </wsse:SecurityTokenReference>
</ds:KeyInfo>
```

Each type of key information is described in the Web Services Security: SOAP Message Security 1.0 (WS-Security 2004) OASIS standard, which is located at: http://www.oasis-open.org/home/index.php under Web services security.

3. Select a key locator reference from the Key locator reference field. The value of this field is a reference to a key locator that WebSphere Application Server uses to locate the keys that are used for digital signature and encryption. Before you can select a key locator, you must configure a key locator. For more information on configuring a key locator, see “Configuring the key locator for the consumer binding on the application level” on page 749.
4. Select a token reference from the Token reference field. The token reference specifies a reference to a token consumer that is used for processing the security token in the message. However, WebSphere Application Server requires this field only when you select Security token reference or Embedded token in the Key information type field. Before specifying a token reference, you must configure a token consumer. For more information on configuring a token consumer, see “Configuring token consumer on the application level” on page 740.

   Select **(none)** if a token consumer is not required for this key information configuration.
5. Click **OK** and **Save** to save this configuration.

You have configured the key information for the generator binding at the application level

If you have not configured the key information for the generator binding. You must specify a similar key information configuration for the generator. After you configure the key information for both the consumer and the generator, configure the signing information or encryption information, which references the key information that is specified in this key information task.

**Related tasks**

"Configuring the key information for the generator binding on the application level" on page 699
"Configuring the signing information for the consumer binding on the application level"
"Configuring the key locator for the consumer binding on the application level" on page 749
"Configuring token consumer on the application level" on page 740

## Configuring the signing information for the consumer binding on the application level

In the server-side extensions file and the client-side deployment descriptor extensions file, you must specify which parts of the message are signed. Also, you need to configure the key information that is referenced by the key information references on the signing information panel within the administrative console.

This task explains the steps that are needed for you to configure the signing information for the server-side request consumer and the client-side response consumer bindings at the application level. WebSphere Application Server uses the signing information on the consumer side to verify the integrity of the received Simple Object Access Protocol (SOAP) message by validating that the message parts are signed. Complete the following steps to configure the signing information for the server-side request consumer and client-side response consumer sections of the bindings files on the application level:

1. Access the administrative console. To access the administrative console, enter `http://localhost:9060/ibm/console` in your Web browser unless you have changed the port number.
2. Click **Applications > Enterprise applications >***application_name*.
3. Under Related Items, click either **Web modules** or **EJB Modules >***URI_name*.
4. Access the server security bindings or the client security bindings.
   - To configure the request consumer signing information, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**.
   - To configure the response consumer signing information, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**.
5. Under Required properties, click **Signing information**.
6. Click **New** to create a signing information configuration, click **Delete** to delete an existing configuration, or click the name of an existing signing information configuration to edit its settings. If you are creating a new configuration, enter a name in the Signing information name field.
7. Select a signature method algorithm from the Signature method field. The signature method is the algorithm that is used to convert the canonicalized <SignedInfo> element in the binding file into the <SignatureValue> element. The algorithm that is specified for the consumer, which is either the request consumer or the response consumer configuration, must match the algorithm specified for the generator, which is either the request generator or response generator configuration. WebSphere Application Server supports the following pre-configured algorithms:
   - http://www.w3.org/2000/09/xmldsig#rsa-sha1
   - http://www.w3.org/2000/09/xmldsig#dsa-sha1
   - http://www.w3.org/2000/09/xmldsig#hmac-sha1
8. Select a canonicalization method from the Canonicalization method field. The canonicalization method algorithm is used to canonicalize the <SignedInfo> element before it is incorporated as part of the digital signature operation. The canonicalization algorithm that you specify for the generator must match the algorithm for the consumer. WebSphere Application Server supports the following pre-configured algorithms:
   - http://www.w3.org/2001/10/xml-exc-c14n#
   - http://www.w3.org/2001/10/xml-exc-c14n#WithComments
   - http://www.w3.org/TR/2001/REC-xml-c14n-20010315
   - http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments

9. Select a key information signature type from the Key information signature type field. The key information signature type specifies how the <KeyInfo> element in the SOAP message is digitally signed. WebSphere Application Server supports the following signature types:

 **None** Specifies that the key is not signed.

 **Keyinfo**
 Specifies that the entire KeyInfo element is signed.

 **Keyinfochildelements**
 Specifies that the child elements of the KeyInfo element are signed.

 If you do not specify one of the previous signature types, WebSphere Application Server uses keyinfo, by default. The key information signature type for the consumer must match the signature type for the generator.
10. Under Additional properties, click **Key information references**.
 a. Click **New** to create a key information reference or click the name of an existing entry to edit its configuration. The Key information references panel is displayed.
 b. Enter a name in the Name field.
 c. Select a key information reference in the Key information reference field. This reference is the key information configuration name that specifies the key information that is used by this signing information configuration.
11. Return to the Signing information panel. Under Additional properties, click **Part references**. On the Part references panel, you can specify references to the message parts that are defined in the deployment descriptor extensions file.
 a. Click **New** to create a new Part reference or click the name of an existing part reference to edit its configuration. The Part reference panel is displayed.
 b. Enter a name in the Part name field. This name is the name of the required integrity configuration in the deployment descriptor extensions file and specifies the message parts that must be digitally signed.
 c. Select a digest method algorithm from the Digest method algorithm field. WebSphere Application Server supports the following pre-configured algorithm: http://www.w3.org/200/09/xmldsig#sha1
12. Under Additional properties, click **Transforms**.
 a. Click **New** to create a new transform or click the name of an existing transform to edit its configuration.
 b. Enter a name in the Transform name field.
 c. Select a transform algorithm from the Transform algorithm field. WebSphere Application Server supports the following pre-configured algorithms:
 - http://www.w3.org/2001/10/xml-exc-c14n#
 - http://www.w3.org/TR/1999/REC-xpath-19991116
 - http://www.w3.org/2002/06/xmldsig-filter2
 - http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform
 - http://www.w3.org/2002/07/decrypt#XML
 - http://www.w3.org/2000/09/xmldsig#enveloped-signature

 The transform algorithm that you select for the consumer must match the transform algorithm that you select for the generator. For each part reference in the signing information, specify both a digest method algorithm and a transform algorithm.
13. Click **OK**.
14. Click **Save** at the top of the panel to save your configuration.

After completing these steps, you have configured the signing information for the consumer.

You must specify a similar signing information configuration for the generator.

**Related tasks**

"Configuring the signing information for the generator binding on the application level" on page 712

***Key information references collection:***

Use this page to view the key information references that are needed for encryption or signing.

To view this administrative console page on the cell level, complete the following steps. On the cell level, you can configure the key information references for the default consumer bindings only.

1. Click **Security > Web services**.
2. Under Default consumer bindings, click either of the following links:
   - Click **Encryption information >** *encryption_information_name*.
   - Click **Signing information >** *signing_information_name*.
3. Under Additional properties, click **Key information references**.

To view this administrative console page on the server level, complete the following steps. On the server level, you can configure the key information references for the default consumer bindings only.

1. Click **Servers > Application Servers >***server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Default consumer bindings, click either of the following links:
   - Click **Encryption information >** *encryption_information_name*.
   - Click **Signing information >** *signing_information_name*.
4. Under Additional properties, click **Key information references**.

To view this administrative console page on the application level, complete the following steps. On the application level, you can configure the key information reference for the consumer bindings only.

1. Click **Applications > Enterprise applications** > *application_name*.
2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.
3. Under Required properties, you can access the signing information for the following bindings:
   - For the Response consumer (receiver) binding, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**. Under Required properties, click **Encryption information**. Click **New** to create a new encryption configuration or click the name of a configuration to modify its settings.
   - For the Request consumer (receiver) binding, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**. Under Required properties, click **Encryption information**. Click **New** to create a new encryption configuration or click the name of a configuration to modify its settings.

**Related reference**

"Key information configuration settings" on page 703
Use this page to specify the related configuration need to specify the key for XML digital signature or XML encryption.

*Name:*

Specifies the name of the Key information reference.

*Key information reference:*

Specifies a reference to the message parts that are signed or encrypted.

The value of this field is the name of the <requiredIntegrity> or the <requiredConfidentiality> element in the deployment descriptor.

***Key information reference configuration settings:***

Use this page to specify a reference to the message parts for signature and encryption that is defined in the deployment descriptors.

To view this administrative console page on the cell level for the key information references, complete the following steps. On the cell level, you can configure the key information references for the default consumer bindings only.

1. Click **Security > Web services**.
2. Under Default consumer bindings, click either of the following links:
   - Click **Encryption information >***encryption_information_name*.
   - Click **Signing information >** *signing_information_name*.
3. Under Additional properties, click **Key information references**.
4. Click **New** to create a key information reference or click the name of an existing configuration to modify its settings.

To view this administrative console page on the server level for the key information references, complete the following steps. On the server level, you can configure the key information references for the default consumer bindings only.

1. Click **Servers > Application Servers >***server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Default consumer bindings, click either of the following links:
   - Click **Encryption information >***encryption_information_name*.
   - Click **Signing information >** *signing_information_name*.
4. Under Additional properties, click **Key information references**.
5. Click **New** to create a key information reference or click the name of an existing configuration to modify its settings.

To view this administrative console page on the application level, complete the following steps:

1. Click **Applications > Enterprise applications >***application_name*.
2. Under Related items, click **EJB modules** or **Web modules >** *URI_name*.
3. Under Additional properties, you can access the key information references for the following bindings:
   - For the Response consumer (sender) binding, click **Web services: Client security bindings**. Under Response consumer (sender) binding, click **Edit custom**. Under Required properties, click **Encryption information >***encryption_information_name*. Under Additional properties, click **Key information references**.
   - For the Request consumer (receiver) binding, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**. Under Required properties, click **Encryption information >***encryption_information_name*. Under Additional properties, click **Key information references**.
4. Click **New** to create a key information reference or click the name of an existing configuration to modify its settings.

   **Related reference**

   "Key information references collection" on page 755
   Use this page to view the key information references that are needed for encryption or signing.

*Name:*

Specifies the name of the key information reference.

*Key information reference:*

Specifies a reference to the message parts that are signed or encrypted.

The value of this field is the name of the <requiredIntegrity> or the <requiredConfidentiality> element in the deployment descriptor. You can specify a signing key configuration for the following bindings:

| Binding name | Cell level, Server level, or application level | Path |
| --- | --- | --- |
| Default consumer binding | Server level | 1. Click **Servers > Application Servers >***server_name*.<br>2. Under Security, click **Web services: Default bindings for Web services security**.<br>3. Under Default consumer binding, click **Key information**. |
| Response consumer (receiver) binding | Application level | 1. Click **Applications > Enterprise applications >***application_name*.<br>2. Under Related items, click **EJB modules** or **Web modules >** *URI_name*.<br>3. Under Additional properties, click **Web services: Client security bindings**.<br>4. Under Response consumer (receiver) binding, click **Edit custom**.<br>5. Under Required properties, click **Key information**. |
| Request consumer (receiver) binding | Application level | 1. Click **Applications > Enterprise applications >***application_name*.<br>2. Under Related items, click **EJB modules** or **Web modules >** *URI_name*.<br>3. Under Additional properties, click **Web services: Server security bindings**.<br>4. Under Resquest consumer (receiver) binding, click **Edit custom**.<br>5. Under Required properties, click **Key information**. |

## Configuring the encryption information for the consumer binding on the application level

Before you begin this task, you must configure the key information that is referenced by the key information references in the encryption information panel. For more information, see “Configuring the key information for the consumer binding on the application level” on page 750.

This task provides the steps that are needed for configuring the encryption information for the request consumer (server side) and response consumer (client side) bindings at the application level. The encryption information on the consumer side is used for decrypting the encrypted message parts in the incoming Simple Object Access Protocol (SOAP) message.

Complete the following steps to configure the encryption information for the request consumer or response consumer section of the bindings file on the application level:

1. Locate the Encryption information configuration panel in the administrative console.

a. Click **Applications > Enterprise applications >***application_name*.
b. Under Related Items, click **EJB Modules** or **Web Modules >** *URI_name*.
c. Under Additional properties you can access the encryption information for the request consumer and response consumer bindings.
   - For the request consumer (receiver) binding, click **Web services: Server security bindings**. Under Request consumer (receiver) binding, click **Edit custom**.
   - For the response consumer (receiver) binding, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**.
d. Under Required properties, click **Encryption information**.
e. Click **New** to create an encryption information configuration, click **Delete** to delete an existing configuration, or click the name of an existing encryption information configuration to edit its settings. If you are creating a new configuration, enter a name in the Encryption information name field. For example, you might specify `cons_encinfo`.

2. Select a data encryption algorithm from the Data encryption algorithm field. The data encryption algorithm is used for encrypting or decrypting parts of a SOAP message such as the SOAP body or the username token. WebSphere Application Server supports the following pre-configured algorithms:
   - http://www.w3.org/2001/04/xmlenc#tripledes-cbc
   - http://www.w3.org/2001/04/xmlenc#aes128-cbc
   - http://www.w3.org/2001/04/xmlenc#aes256-cbc

     To use this algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.
   - http://www.w3.org/2001/04/xmlenc#aes192-cbc

     To use this algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.

   The data encryption algorithm that you select for the consumer side must match the data encryption method that you select for the generator side.
3. Select a key encryption algorithm from the Key encryption algorithm field. The key encryption algorithm is used for encrypting the key that is used for encrypting the message parts within the SOAP message. Select **(none)** if the data encryption key, which is the key that is used for encrypting the message parts, is not encrypted. WebSphere Application Server supports the following pre-configured algorithms:
   - http://www.w3.org/2001/04/xmlenc#rsa-1_5
   - http://www.w3.org/2001/04/xmlenc#kw-tripledes
   - http://www.w3.org/2001/04/xmlenc#kw-aes128
   - http://www.w3.org/2001/04/xmlenc#kw-aes256

     To use the http://www.w3.org/2001/04/xmlenc#aes256-cbc algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.
   - http://www.w3.org/2001/04/xmlenc#kw-aes192

     To use the http://www.w3.org/2001/04/xmlenc#kw-aes192 algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.

   The key encryption algorithm that you select for the consumer side must match the key encryption method that you select for the generator side.
4. **Optional:** Select a part reference in the Part reference field. The part reference specifies the name of the message part that is encrypted and is defined in the deployment descriptor. For example, you can encrypt the bodycontent message part in the deployment descriptor. The name of this Required Confidentiality part is `conf_con`. This message part is shown as an option in the Part reference field.

5. Under Additional properties, click **Key information references**.
6. Click **New** to create a key information configuration, click **Delete** to delete an existing configuration, or click the name of an existing key information configuration to edit its settings. If you are creating a new configuration, enter a name in the name field. For example, you might specify `con_ekeyinfo`. This entry is the name of the <encryptionKeyInfo> element in the binding file.
7. Select a key information reference from the Key information reference field. This reference is the value of the keyinfoRef attribute of the <encryptionKeyInfo> element and it is the name of the <keyInfo> element that is referenced by this key information reference. Each key information reference entry generates an <encryptionKeyInfo> element under the <encryptionInfo> element in the binding configuration file. For example, if you enter `con_ekeyinfo` in the Name field and `dec_keyinfo` in the Key information reference field, the following <encryptionKeyInfo> element is generated in the binding file:
```
<encryptionKeyInfo xmi:id="EncryptionKeyInfo_1085092248843"
keyinfoRef="dec_keyinfo"  name="con_ekeyinfo"/>
```
8. Click **OK** and then click **Save** to save the configuration.

You have configured the encryption information for the consumer binding at the application level

You must specify a similar encryption information configuration for the generator.

**Related tasks**

"Configuring the encryption information for the generator binding on the application level" on page 727

"Configuring the key information for the consumer binding on the application level" on page 750

## Retrieving tokens from the JAAS Subject in a server application

In WebSphere Application Server Version 6, the security handlers are responsible for propagating security tokens. These security tokens are embedded in the Simple Object Access Protocol (SOAP) security header and passed to downstream servers. The security tokens are encapsulated in the implementation classes for the com.ibm.wsspi.wssecurity.auth.token.Token interface. You can retrieve the security token data from either a server application or a client application.

With a server application, the application acts as the request consumer and the response generator, is deployed, and runs in the Java 2 Platform, Enterprise Edition (J2EE) container. The consumer component for Web services security stores the security tokens that it receives in the Java Authentication and Authorization Service (JAAS) Subject of the current thread. You can retrieve the security tokens from the JAAS Subject that is maintained as a local thread in the container. Complete the following steps to retrieve the security token data from a server application:

1. Obtain the JAAS Subject of the current thread using the WSSubject utility class. If you enable Java 2 security on the Global security panel in the administrative console, access to the JAAS Subject is denied if the application code is not granted the javax.security.auth.AuthPermission("wssecurity.getCallerAsSubject") permission. The following code sample shows how to obtain the JAAS subject:
```
javax.security.auth.Subject subj;
try {
subj = com.ibm.websphere.security.auth.WSSubject.getCallerSubject();
} catch (com.ibm.websphere.security.WSSecurityException e) {
  ...
}
```
2. Obtain a set of private credentials from the Subject. For more information, see the application programming interface (API) com.ibm.websphere.security.auth.WSSubject class through the information center . To access this information within the information center, click **Reference > Developer > API Documentation > Application Programming Interfaces**. In the Application Programming Interfaces article, click **com.ibm.websphere.security.auth > WSSubject**.

   **Attention:** When Java 2 security is enabled, you might need to use the AccessController class to avoid a security violation that is caused by operating the security objects in the J2EE container.

The following code sample shows how to set the AccessController class and obtain the private credentials:

```
Set s = (Set) AccessController.doPrivileged(
new PrivilegedAction() {
public Object run() {
return subj.getPrivateCredentials();
}
   });
```

3. Search the targeting token class in the private credentials. You can search the targeting token class by using the java.util.Iterator interface. The following example shows how to retrieve a username token with a certain token ID value in the security header. You can also use other method calls to retrieve security tokens. For more information, see the application programming interface (API) documents for the com.ibm.wsspi.wssecurity.auth.token.Token interface or custom token classes.

```
com.ibm.wsspi.wssecurity.auth.token.UsernameToken unt;
Iterator it = s.iterator();
while (it.hasNext()) {
  Object obj = it.next();
  if (obj != null &&
obj instanceOf com.ibm.wsspi.wssecurity.auth.token.UsernameToken) {
    unt =(com.ibm.wsspi.wssecurity.auth.token.UsernameToken) obj;
if (unt.getId().equals("...")) break;
else continue;
  }
}
```

After completing these steps, you have retrieved the security tokens from the JAAS Subject in a server application

**Related concepts**

"Security token" on page 652
A security token represents a set of claims made by a client that might include a name, password, identity, key, certificate, group, privilege, and so on.

## Retrieving tokens from the JAAS Subject in a client application

In WebSphere Application Server Version 6, the security handlers are responsible for propagating security tokens. These security tokens are embedded in the Simple Object Access Protocol (SOAP) security header and passed to downstream servers. The security tokens are encapsulated in the implementation classes for the com.ibm.wsspi.wssecurity.auth.token.Token interface. You can retrieve the security token data from either a server application or a client application.

With a client application, the application serves as the request generator and the response consumer and runs as the Java 2 Platform, Enterprise Edition (J2EE) client application. The consumer component for Web services security stores the security tokens that it receives in one of the properties of the MessageContext object for the current Web services call. You can retrieve a set of token objects through the javax.xml.rpc.Stub interface of that Web Services call. You must know which security tokens to retrieve and their token IDs in case multiple security tokens are included in the SOAP security header. Complete the following steps to retrieve the security token data from a client application:

1. Use the com.ibm.wsspi.wssecurity.token.tokenPropergation key string to obtain the Hashtable for the tokens through a property value in the javax.xml.rpc.Stub interface. The following example shows how to obtain the Hashtable:

```
java.util.Hashtable t;
javax.xml.rpc.Service serv = ...;
MyWSPortType pt = (MyWSPortType)serv.getPort(MyWSPortType.class);
t = (Hashtable)((javax.xml.rpc.Stub)pt)._getProperty(
com.ibm.wsspi.wssecurity.Constants.WSSECURITY_TOKEN_PROPERGATION);
```

2. Search the targeting token objects in the Hashtable. Each token object in the Hashtable is set with its token ID as a key. You must have prior knowledge of the security token IDs to retrieve the security tokens. The following example shows how to retrieve a username token from the security header with a certain token ID value:

```
com.ibm.wsspi.wssecurity.auth.token.UsernameToken unt;
if (t != null) {
  unt = (com.ibm.wsspi.wssecuty.auth.token.UsernameToken)t.get("...");
}
```

After completing these steps, you have retrieved the security tokens from the JAAS Subject in a client application

**Related concepts**

"Security token" on page 652
A security token represents a set of claims made by a client that might include a name, password, identity, key, certificate, group, privilege, and so on.

## Configuring trust anchors on the server or cell level

Prior to completing the steps to configure trust anchors, you must create the keystore file using the key tool. WebSphere Application Server provides the key tool in the `install_dir/java/jre/bin/keytool` file.

This task provides the steps that are needed to configure a list of keystore objects that contain trusted root certificates. These objects are used for certificate path validation of incoming X.509-formatted security tokens. Keystore objects within trust anchors contain trusted root certificates that are used by the CertPath application programming interface (API) to determine whether to trust a certificate chain.

Complete the following steps to configure the trust anchors on the server level:

1. Access the default bindings for the server level.
   a. Click **Servers > Application servers >***server_name*.
   b. Under Security, click **Web services: Default bindings for Web services security**.
2. Under Additional properties, click **Trust anchors**.
3. Click **New** to create a trust anchor configuration, click **Delete** to delete an existing configuration, or click the name of an existing trust anchor configuration to edit its settings. If you are creating a new configuration, enter a unique name for the trust anchor in the Trust anchor name field.
4. Specify a password in the Key store password field that is used to access the keystore file.
5. Specify the absolute location of the keystore file in the Key store path field. It is recommended that you use the *USER_INSTALL_ROOT* variable as a portion of the keystore path. To change this predefined variable, click **Environment > WebSphere variables**. The *USER_INSTALL_ROOT* variable might display on the second page of variables.
6. Specify the type of keystore file in the key store type field. WebSphere Application Server supports the following keystore types:

   **JKS** Use this option if you are not using Java Cryptography Extensions (JCE) and your keystore file uses the Java Key Store (JKS) format.

   **JCEKS**
   Use this option if you are using Java Cryptography Extensions.

   **PKCS11KS (PKCS11)**
   Use this option if your keystore file uses the PKCS#11 file format. Keystore files that use this format might contain Rivest Shamir Adleman (RSA) keys on cryptographic hardware or might encrypt keys that use cryptographic hardware to ensure protection.

   **PKCS12KS (PKCS12)**
   Use this option if your keystore file uses the PKCS#12 file format.
7. Click **OK** and **Save** to save your configuration.

You have configured trust anchors at the server or cell level.

## Configuring the collection certificate store on the server or cell-level bindings

Collection certificate stores contain untrusted, intermediary certificate files awaiting validation. Validation might consist of checking for a valid signature in a digitally signed Simple Object Access Protocol (SOAP) message to see if the certificate is on a certificate revocation list (CRLs), checking that the certificate is not expired, and checking that the certificate is issued by a trusted signer.

Complete the following steps to configure a collection certificate on the server level:

1. Access the default bindings for the server level.
   a. Click **Servers > Application servers >***server_name*.
   b. Under Security, click **Web services: Default bindings for Web services security**.
2. Under Additional properties, click **Collection certificate store**.
3. Click **New** to create a collection certificate store configuration, click **Delete** to delete an existing configuration, or click the name of an existing collection certificate store configuration to edit its settings. If you are creating a new configuration, enter a name in the Certificate store name field. For example, you might name your certificate store `sig_certstore`.

   The name of the collection certificate store must be unique to the level of the application server. For example, if you create the collection certificate store for the server level, the store name must be unique to the server level. The name that is specified in the Certificate store name field is used by other configurations to refer to a predefined collection certificate store. WebSphere Application Server searches for the collection certificate store based on proximity.

   For example, if an application binding refers to a collection certificate store named `cert1`, the Application Server searches for `cert1` at the application level before searching the server level.
4. Specify a certificate store provider in the Certificate store provider field. WebSphere Application Server supports the IBMCertPath certificate store provider. To use another certificate store provider, you must define the provider implementation in the provider list within the *install_dir*`/java/jre/lib/security/java.security` file. However, make sure that your provider supports the same requirements of the certificate path algorithm as WebSphere Application Server.
5. Click **OK** and **Save** to save the configuration.
6. Click the name of your certificate store configuration. After you specify the certificate store provider, you must specify either the location of a certificate revocation list or the X.509 certificates. However, you can specify both a certificate revocation list and the X.509 certificates for your certificate store configuration.
7. Under Additional properties, click **Certificate revocation lists**. For the generator binding, a certificate revocation list (CRL) is used when it is included in a generated security token. For example, a security token might be wrapped in a PKCS#7 format with a CRL. For more information on certificate revocation lists, see “Certificate revocation list” on page 585.
8. Click **New** to specify a certificate revocation list path, click **Delete** to delete an existing list reference, or click the name of an existing reference to edit the path. You must specify the fully qualified path to the location where WebSphere Application Server can find your list of certificates that are not valid. WebSphere Application Server uses the certificate revocation list to check the validity of the sender certificate.

   For portability reasons, it is recommended that you use the WebSphere Application Server variables to specify a relative path to the certificate revocation lists. This recommendation is especially important when you are working in a WebSphere Application Server Network Deployment environment.

   For example, you might use the *USER_INSTALL_ROOT* variable to define a path such as $*USER_INSTALL_ROOT*/*mycertstore*/*mycrl1* where *mycertstore* represents the name of your certificate store and *mycrl1* represents the certificate revocation list. For a list of supported variables, click **Environment > WebSphere variables** in the administrative console. The following list provides recommendations for using certificate revocation lists:

- If CRLs are added to the collection certificate store, add the CRLs for the root certificate authority and each intermediate certificate, if applicable. When the CRL is in the certificate collection store, the certificate revocation status for every certificate in the chain is checked against the CRL of the issuer.
- When the CRL file is updated, the new CRL does not take effect until you restart the Web service application.
- Before a CRL expires, you must load a new CRL into the certificate collection store to replace the old CRL. An expired CRL in the collection certificate store results in a certificate path (CertPath) build failure.

9. Click **OK** and then **Save** to save the configuration.
10. Return to the Collection certificate store configuration panel. To access the panel, complete the following steps:
    a. Click **Servers > Application servers >***server_name*.
    b. Under Security, click **Web services: Default bindings for Web services security**.
    c. Under Additional properties, click **Collection certificate store >***certificate_store_name*.
11. Under Additional properties, click **X.509 certificates**. The X.509 certificate configuration specifies intermediate certificate files that are used for certificate path validation of incoming X.509-formatted security tokens.
12. Click **New** to create an X.509 certificate configuration, click **Delete** to delete an existing configuration, or click the name of an existing X.509 certificate configuration to edit its settings. If you are creating a new configuration, enter a name in the Certificate store name field.
13. Specify a path in the X.509 certificate path field. This entry is the absolute path to the location of the X.509 certificate. The collection certificate store is used to validate the certificate path of the incoming X.509-formatted security tokens.

    You can use the *USER_INSTALL_ROOT* variable as part of path name. For example, you might type: `$USER_INSTALL_ROOT/etc/ws-security/samples/intca2.cer`. Do not use this certificate path for production use. You must obtain your own X.509 certificate from a certificate authority before putting your WebSphere Application Server environment into production.

    Click **Environment > WebSphere variables** in the administrative console to configure the *USER_INSTALL_ROOT* variable.
14. Click **OK** and then **Save** to save your configuration.
15. Return to the Collection certificate store collection panel and click **Update run time** to update the Web services security run time with the default binding information, which is located in the `ws-security.xml` file. When you click **Update run time**, the configuration changes made to other Web services are also updated in the run time for Web services security.

You have configured the collection certificate store for the server or cell level.

**Related concepts**

"Certificate revocation list" on page 585
A *certificate revocation list* is a time-stamped list of certificates that have been revoked by a certificate authority (CA).

## Distributed nonce caching

The *distributed nonce caching* feature enables you to distribute the cache for a nonce to different servers in a cluster.

In previous releases of WebSphere Application Server, the nonce was cached locally. To use this feature, you must complete the following actions:

- Configure cache replication.
- Verify that you created an appropriate domain setting when you form a cluster.

- Verify that replication domain is properly secured. The nonce cache is crucial to the integrity of the nonce validation process. If the nonce cache is compromised, then you cannot trust the result of the validation process.
- In the administrative console for the server level, select the **Distribute nonce caching** option. You can enable the option by completing the following steps:
  1. Click **Security > Web services**.
  2. Select the **Distribute nonce caching** option.
- Restart the servers within your cluster.

When you select the **Distribute nonce caching** option in the administrative console, the nonce is propagated to other servers in your environment. However, the nonce might be subject to a one-second delay in propagation and subject to any network congestion.

For more information on distributed nonce caching, see "Web services security enhancements" on page 531.

**Related concepts**

"Nonce, a randomly generated token" on page 567
*Nonce* is a randomly generated, cryptographic token used to prevent replay attacks. Although Nonce can be inserted anywhere in the SOAP message, it is typically inserted in the <UsernameToken> element.

"Web services security enhancements" on page 531

## Configuring a nonce on the server or cell level

*Nonce* is a randomly generated, cryptographic token that is used to prevent replay attacks of user name tokens that are used with Simple Object Access Protocol (SOAP) messages. Typically, nonce is used with the user name token.

This task provides instructions on how to configure nonce for the cell level using the WebSphere Application Server administrative console. You can configure nonce at the application level, the server level, and the cell level. However, you must consider the order of precedence. The following list shows the order of precedence:

1. Application level

   The application level settings for the nonce maximum age and nonce clock skew fields are specified through the additional properties.
2. Server level

If you configure nonce on the application level and the server level, the values that are specified for the application level take precedence over the values that are specified for the server level. Likewise, the values that are specified for the application level take precedence over the values specified for the server level. Complete the following steps to configure nonce on the server level:

Complete the following steps to configure a nonce on the server or cell level:

1. Access the default bindings for the server level.
   a. Click **Servers > Application servers >***server_name*.
   b. Under Security, click **Web services: Default bindings for Web services security**.
2. Specify a value, in seconds, for the Nonce cache timeout field. The value that is specified for the Nonce cache timeout field indicates how long the nonce remains cached before it is discarded. You must specify a minimum of 300 seconds. However, if you do not specify a value, the default is 600 seconds. This field is optional on the server level, but required on the cell level.
3. Specify a value, in seconds, for the Nonce maximum age field. The value that is specified for the Nonce maximum age field indicates how long the nonce is valid. You must specify a minimum of 300

seconds, but the value cannot exceed the number of seconds that is specified for the Nonce cache timeout field in the previous step. If you do not specify a value, the default is 300 seconds.

In a Network Deployment environment, this field is optional on the server level, but it is required on the cell level.

4. Specify a value, in seconds, for the Nonce clock skew field. The value that is specified for the Nonce clock skew field specifies the amount of time, in seconds, to consider when the message receiver checks the freshness of the value. Consider the following information when you set this value:
   - Difference in time between the message sender and the message receiver, if the clocks are not synchronized.
   - Time that is needed to encrypt and transmit the message.
   - Time that is needed to get through network congestion.

   At a minimum, you must specify 0 seconds in this field. However, the maximum value cannot exceed the number of seconds indicated in the Nonce maximum age field. If you do not specify a value, the default is 0 seconds. This field is optional on the server level, but required on the cell level.
5. Select the **Distribute nonce caching** option. This option enables you to distribute the caching for a nonce using a Data Replication Service (DRS). In previous releases of WebSphere Application Server, the nonce was cached locally. By selecting this option, the nonce is propagated to other servers in your environment. However, the nonce might be subject to a one-second delay in propagation and subject to any network congestion.
6. Restart the server. If you change the Nonce cache timeout value and do not restart the server, the change is not recognized by the server.

## Configuring token generators on the server or cell level

The token generator on the server or cell level is used to specify the information for the token generator if these bindings are not defined at the application level. The signing information and the encryption information can share the token generator information, which is why they are all defined at the same level. WebSphere Application Server provides default values for bindings. You must modify the defaults for a production environment.

Complete the following steps to configure the token generators on the server level:

1. Access the default bindings for the server level.
   a. Click **Servers > Application servers >***server_name*.
   b. Under Security, click **Web services: Default bindings for Web services security**.
2. Under Default generator bindings, click **Token generators**.
3. Click **New** to create a token generator configuration, click **Delete** to delete an existing configuration, or click the name of an existing token generator configuration to edit its settings. If you are creating a new configuration, enter a unique name for the token generator configuration in the Token generator name field. For example, you might specify `sig_tgen`. This field specifies the name of the token generator element.
4. Specify a class name in the Token generator class name field. The token generator class must implement the com.ibm.wsspi.wssecurity.token.TokenGeneratorComponent interface. The token generator class name must be similar to the token consumer class name. For example, if your application requires an X.509 certificate token consumer, you can specify the com.ibm.wsspi.wssecurity.token.X509TokenConsumer class name on the Token consumer panel and the com.ibm.wsspi.wssecurity.token.X509TokenGenerator class name in this field. WebSphere Application Server provides the following default token generator class implementations:

   **com.ibm.wsspi.wssecurity.token.UsernameTokenGenerator**
   This implementation generates a username token.

   **com.ibm.wsspi.wssecurity.token.X509TokenGenerator**
   This implementation generates an X.509 certificate token.

**com.ibm.wsspi.wssecurity.token.LTPATokenGenerator**
This implementation generates a Lightweight Third Party Authentication (LTPA) token.

5. Select a certificate path option. The certificate path specifies the certificate revocation list (CRL), which is used for generating a security token that is wrapped in a PKCS#7 with a CRL. WebSphere Application Server provides the following certificate path options:

   **None** Select this option in case the CRL is not used for generating a security token. You must select this option when the token generator does not use the PKCS#7 token type.

   **Dedicated signing information**
   If the CRL is wrapped in a security token, select **Dedicated signing information** and select a collection certificate store name from the Certificate store field. The Certificate store field shows the names of collection certificate stores already defined. To define a collection certificate store on the cell level, see “Configuring the collection certificate store on the server or cell-level bindings” on page 762.
6. Select the **Add nonce** option to include a nonce in the user name token for the token generator. Nonce is a unique cryptographic number that is embedded in a message to help stop repeat, unauthorized attacks of user name tokens. The **Add nonce** option is available if you specify a user name token for the token generator.
7. Select the **Add timestamp** option to include a time stamp in the user name token for the token generator.
8. Specify a value type local name in the Local name field. This entry specifies the local name of the value type for a security token that is referenced by the key identifier. This attribute is valid when **Key identifier** is selected as Key information type. To specify the Key information type, see “Configuring the key information for the generator binding on the server or cell level” on page 778. WebSphere Application Server provides the following predefined X.509 certificate token configurations:

   **X.509 certificate token**
   http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3

   **X.509 certificates in a PKIPath**
   http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509PKIPathv1

   **A list of X.509 certificates and CRLs in a PKCS#7**
   http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#PKCS7

   **LTPA** For LTPA, the value type local name is LTPA. If you enter `LTPA` for the local name, you must specify the http://www.ibm.com/websphere/appserver/tokentype/5.0.2 URI value in the Value type URI field as well.

   For example, when an X.509 certificate token is specified, you can use `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3` for the local name.
9. Specify the value type uniform resource identifier (URI) in the URI field. This entry specifies the namespace URI of the value type for a security token that is referenced by the key identifier. This attribute is valid when **Key identifier** is selected as Key information type on the Key information panel for the default generator. When the X.509 certificate token is specified, you do not need to specify the namespace URI. If another token is specified, you must specify the namespace URI of the value type.
10. Click **OK** and then **Save** to save the configuration.
11. Click the name of your token generator configuration.
12. Under Additional properties, click **Callback handler** to configure the callback handler properties. The callback handler specifies how to acquire the security token that is inserted in the Web services security header within the Simple Object Access Protocol (SOAP) message. The token acquisition is a pluggable framework that leverages the Java Authentication and Authorization Service (JAAS) javax.security.auth.callback.CallbackHandler interface for acquiring the security token.

a. Specify a callback handler class implementation in the Callback handler class name field. This attribute specifies the name of the Callback handler class implementation that is used to plugin a security token framework. The specified callback handler class must implement the javax.security.auth.callback.CallbackHandler class. WebSphere Application Server provides the following default callback handler implementations:

**com.ibm.wsspi.wssecurity.auth.callback.GUIPromptCallbackHandler**
This callback handler uses a login prompt to gather the user name and password information. However, if you specify the user name and password on this panel, a prompt is not displayed and WebSphere Application Server returns the user name and password to the token generator. Use this implementation for a Java 2 Platform, Enterprise Edition (J2EE) application client only.

**com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler**
This callback handler does not issue a prompt and returns the user name and password if it is specified in the basic authentication section of this panel. You can use this callback handler when the Web service is acting as a client.

**com.ibm.wsspi.wssecurity.auth.callback.StdinPromptCallbackHandler**
This callback handler uses a standard-in prompt to gather the user name and password. However, if the user name and password is specified in the basic authentication section of this panel, WebSphere Application Server does not issue a prompt, but returns the user name and password to the token generator. Use this implementation for a Java 2 Platform, Enterprise Edition (J2EE) application client only.

**com.ibm.wsspi.wssecurity.auth.callback.LTPATokenCallbackHandler**
This callback handler is used to obtain the Lightweight Third Party Authentication (LTPA) security token from the Run As invocation Subject. This token is inserted in the Web services security header within the SOAP message as a binary security token. However, if the user name and password are specified in the basic authentication section of this panel, WebSphere Application Server authenticates the user name and password to obtain the LTPA security token. It obtains the security token this way rather than obtaining it from the Run As Subject. Use this callback handler only when the Web service is acting as a client on the application server. It is recommended that you do not use this callback handler on a J2EE application client.

**com.ibm.wsspi.wssecurity.auth.callback.X509CallbackHandler**
This callback handler is used to create the X.509 certificate that is inserted in the Web services security header within the SOAP message as a binary security token. A keystore file and a key definition are required for this callback handler.

**com.ibm.wsspi.wssecurity.auth.callback.PKCS7CallbackHandler**
This callback handler is used to create X.509 certificates that are encoded with the PKCS#7 format. The certificate is inserted in the Web services security header in the SOAP message as a binary security token. A keystore file is required for this callback handler. You must specify a certificate revocation list (CRL) in the collection certificate store. The CRL is encoded with the X.509 certificate in the PKCS#7 format. For more information on configuring the collection certificate store, see “Configuring the collection certificate store on the server or cell-level bindings” on page 762.

**com.ibm.wsspi.wssecurity.auth.callback.PkiPathCallbackHandler**
This callback handler is used to create X.509 certificates that are encoded with the PkiPath format. The certificate is inserted in the Web services security header within the SOAP message as a binary security token. A keystore file is required for this callback handler. A CRL is not supported by the callback handler; therefore, the collection certificate store is not required or used.

For an X.509 certificate token, you might specify the com.ibm.wsspi.wssecurity.auth.callback.X509CallbackHandler implementation.

b. Optional: Select the **Use identity assertion** option. Select this option if you have identity assertion that is defined in the IBM extended deployment descriptor. This option indicates that only the identity of the initial sender is required and inserted into the Web services security header within the SOAP message. For example, WebSphere Application Server sends only the user name of the original caller for a user name token generator. For an X.509 token generator, the application server sends the original signer certification only.

c. Optional: Select the **Use RunAs identity** option. Select this option if the following conditions are true:
   - You have identity assertion defined in the IBM extended deployment descriptor.
   - You want to use the Run As identity instead of the initial caller identity for identity assertion for a downstream call.

d. Optional: Specify a basic authentication user ID and password in the User ID and password fields. This entry specifies the user name and password that is passed to the constructors of the callback handler implementation. The basic authentication user ID and password are used if you specify one of the following default callback handler implementations that are provided by WebSphere Application Server:
   - com.ibm.wsspi.wssecurity.auth.callback.GUIPromptCallbackHandler
   - com.ibm.wsspi.wssecurity.auth.callback.LTPATokenCallbackHandler
   - com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler
   - com.ibm.wsspi.wssecurity.auth.callback.StdinPromptCallbackHandler

e. Optional: Specify a keystore password and path. The keystore and its related information are necessary when the key or certificate is used for generating a token. For example, the keystore information is required if you select one of the following default callback handler implementations that are provided by WebSphere Application Server:
   - com.ibm.wsspi.wssecurity.auth.callback.PKCS7CallbackHandler
   - com.ibm.wsspi.wssecurity.auth.callback.PkiPathCallbackHandler
   - com.ibm.wsspi.wssecurity.auth.callback.X509CallbackHandler

   The keystore files contain public and private keys, root certificate authority (CA) certificates, intermediate CA certificates, and so on. Keys that are retrieved from the keystore file are used to sign and validate or encrypt and decrypt messages or message parts. To retrieve a key from a keystore file, you must specify the keystore password, the keystore path, and the keystore type.

13. Select a keystore type from the type field. WebSphere Application Server provides the following options:

    **JKS** Use this option if you are not using Java Cryptography Extensions (JCE) and if your keystore file uses the Java Key Store (JKS) format.

    **JCEKS**
    Use this option if you are using Java Cryptography Extensions.

    **PKCS11KS (PKCS11)**
    Use this format if your keystore file uses the PKCS#11 file format. Key store files using this format might contain RSA keys on cryptographic hardware or might encrypt the keys that use cryptographic hardware to ensure protection.

    **PKCS12KS (PKCS12)**
    Use this option if your keystore file uses the PKCS#12 file format.

14. Click **OK** and then **Save** to save the configuration.
15. Click the name of your token generator configuration.
16. Under Additional properties, click **Callback handler > Keys**.

17. Click **New** to create a key configuration, click **Delete** to delete an existing configuration, or click the name of an existing key configuration to edit its settings. If you are creating a new configuration, enter a unique name for the key configuration in the Key name field. This name refers to the name of the key object that is stored within the keystore file.
18. Specify an alias for the key object in the Key alias field. The alias is used when the key locator searches for the key objects in the keystore.
19. Specify the password that is associated with the key in the Key password field.
20. Click **OK** and **Save** to save the configuration.

You have configured the token generators at the server or the cell level.

You must specify a similar token consumer configuration.

**Related tasks**

"Configuring the collection certificate store on the server or cell-level bindings" on page 762

***Token generator collection:***

Use this page to view the token generators. The information is used on the generator side only to generate the security token.

To view this administrative console page for the cell level, complete the following steps:

1. Click **Security > Web services**.
2. Under Default generator bindings, click **Token generators**.

To view this administrative console page for the server level, complete the following steps:

1. Click **Servers > Application servers >***server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Default generator bindings, click **Token generators**.

**Related reference**

"Token consumer collection" on page 791
Use this page to view the token consumer. The information is used on the consumer side only to process the security token.

"Token consumer configuration settings" on page 792
Use this page to specify the information for the token consumer. The information is used at the consumer side only to process the security token.

"Token generator configuration settings"
Use this page to specify the information for the token generator. The information is used at the generator side only to generate the security token.

*Token generator name:*

Specifies the name of the token generator configuration.

*Token generator class name:*

Specifies the name of the token generator implementation class.

This class must implement the com.ibm.wsspi.wssecurity.token.TokenGeneratorComponent interface.

***Token generator configuration settings:***

Use this page to specify the information for the token generator. The information is used at the generator side only to generate the security token.

To view this administrative console page for the server level, complete the following steps:

1. Click **Servers > Application servers >***server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Default generator bindings, click **Token generators >***token_generator_name* or click **New** to create a new token generator.
1. Click **Applications > Enterprise applications** > *application_name*.
2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.
3. Under Additional properties, you can access the token generator information for the following bindings:
   - For the Request generator (sender) binding, click **Web services: Client security bindings**. Under Request generator (sender) binding, click **Edit custom**.
   - For the Response generator (sender) binding, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**.
4. Click **New** to create a new token generator or click the name of an existing token generator name to specify its settings.

To view this administrative console page for the application level, complete the following steps:

1. Click **Applications > Enterprise applications** > *application_name*.
2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.
3. Under Additional properties, click **Web services: Client security bindings**.
4. Under Request generator (sender) binding, click **Edit custom**.
5. Under Additional properties, click **Token generators > New**.

Before specifying additional properties, specify a value in the **Token generator name** and the **Token generator class name** fields.

**Related reference**

"Token consumer collection" on page 791
Use this page to view the token consumer. The information is used on the consumer side only to process the security token.

"Token consumer configuration settings" on page 792
Use this page to specify the information for the token consumer. The information is used at the consumer side only to process the security token.

"Token generator collection" on page 769
Use this page to view the token generators. The information is used on the generator side only to generate the security token.

*Token generator name:*

Specifies the name of the token generator configuration.

*Token generator class name:*

Specifies the name of the token generator implementation class.

This class must implement the com.ibm.wsspi.wssecurity.token.TokenGeneratorComponent interface.

*Certificate path:*

Specifies the certificate revocation list (CRL) that is used for generating a security token wrapped in a PKCS#7 token type with CRL.

When the token generator is not for a PKCS#7 token type, you must select **None**. When the token generator is for the PKCS#7 token type and you want to package CRL in the security token, select **Dedicated signing information** and specify the CRL for the collection certificate store.

You can specify a certificate store configuration for the following bindings on the following levels:

| Binding name | Cell level, server level, or application level | Path |
| --- | --- | --- |
| Default generator bindings | Server level | 1. Click **Servers > Application servers >** *server_name*.<br>2. Under Security, click **Web services: Default bindings for Web services security**.<br>3. Under Additional properties, click **Collection certificate store**. |

Using the collection certificate store, you can configure a related certificate revocation list by clicking **Certificate revocation list** under Additional properties.

*Add nonce:*

Indicates whether nonce is included in the user name token for the token generator. *Nonce* is a unique cryptographic number that is embedded in a message to help stop repeat, unauthorized attacks of user name tokens.

On the application level, if you select the **Add nonce** option, you can specify the following properties under Additional properties:

*Table 15. Additional nonce properties*

| Property name | Default value | Explanation |
| --- | --- | --- |
| `com.ibm.ws.wssecurity.config.token.BasicAuth.Nonce.cacheTimeout` | 600 seconds | Specifies the timeout value, in seconds, for the nonce value that is cached on the server. |
| `com.ibm.ws.wssecurity.config.token.BasicAuth.Nonce.clockSkew` | 0 seconds | Specifies the time, in seconds, before the nonce time stamp expires. |
| `com.ibm.ws.wssecurity.config.token.BasicAuth.Nonce.maxAge` | 300 seconds | Specifies the clock skew value, in seconds, to consider when WebSphere Application Server checks the timeliness of the message. |

These properties are available on the administrative console at the cell and server level. However, on the application level, you can configure the properties under Additional properties.

This option is displayed on the cell, server, and application levels. This option is valid only when the generated token type is a user name token.

*Add timestamp:*

Specifies whether to insert the time stamp into the user name token.

This option is displayed on the cell, server, and application levels. This option is valid only when the generated token type is a user name token.

*Value type local name:*

Specifies the local name of the value type for the generated token.

For a user name token and an X.509 certificate security token, WebSphere Application Server provides predefined value types. When you specify the following local names, you do not need to specify the URI of value type.

**Username token**
: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#UsernameToken

**X509 certificate token**
: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509

**X509 certificates in a PKIPath**
: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509PKIPathv1

**A list of X509 certificates and CRLs in a PKCS#7**
: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#PKCS7

**LTPA**

**Important:** For LTPA, the value type local name is `LTPA`. If you enter `LTPA` for the local name, you must specify the http://www.ibm.com/websphere/appserver/tokentype/5.0.2 URI value in the Value type URI field as well. For the other predefined value types (Username token, X509 certificate token, X509 certificates in a PKIPath, and a list of X509 certificates and CRLs in a PKCS#7), the value for the local name field begins with `http://`. For example, if you are specifying the user name token for the value type, enter `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#UsernameToken` in the Value type local name field and then you do not need to enter a value in the Value type URI field.

When you specify a custom value type for custom tokens, you can specify the local name and the URI of the quality name (QName) of the value type. For example, you might specify `Custom` for the local name and `http://www.ibm.com/custom` for the URI.

*Value type URI:*

Specifies the namespace URI of the value type for the generated token.

When you specify the token generator for the user name token or the X.509 certificate security token, you do not need to specify this option. If you want to specify another token, specify the URI of the QName of the value type.

WebSphere Application Server provides the following predefined value type URI for the LTPA token: http://www.ibm.com/websphere/appserver/tokentype/5.0.2

***Algorithm URI collection:***

Use this page to view a list of uniform resource identifier (URI) algorithms for XML digital signature or XML encryption that are mapped to an algorithm factory engine class. With algorithm mappings, service providers can use other cryptographic algorithms for digest value calculation, digital signature signing and verification, data encryption and decryption, and key encryption and decryption.

To view this administrative console page on the cell level, complete the following steps:
1. Click **Security > Web services**.
2. Under Additional properties, click **Algorithm mappings**.
3. Under Additional properties, click **Algorithm URI**.

To view administrative console page on the server level, complete the following steps:
1. Click **Servers > Application servers >***server_name*.

2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Algorithm mappings**.
4. Under Additional properties, click **Algorithm URI**.

   **Related reference**

   "Algorithm URI configuration settings"
   Use this page to specify the algorithm uniform resource identifier (URI) and its usage type.

   "Algorithm mapping collection" on page 775
   Use this page to view a list of custom uniform resource identifier (URI) algorithms for digest value calculation, signature, key encryption, and data encryption. WebSphere Application Server maps these algorithms to an implementation of the algorithm factory engine interface. With algorithm mappings, service providers can extend the cryptographic algorithms for XML digital signature and XML encryption.

   "Algorithm mapping collection" on page 775
   Use this page to view a list of custom uniform resource identifier (URI) algorithms for digest value calculation, signature, key encryption, and data encryption. WebSphere Application Server maps these algorithms to an implementation of the algorithm factory engine interface. With algorithm mappings, service providers can extend the cryptographic algorithms for XML digital signature and XML encryption.

*Algorithm URI:*

Specifies the algorithm uniform resource identifier (URI) for the specified algorithm type.

*Algorithm type:*

Specifies the algorithm type.

***Algorithm URI configuration settings:***

Use this page to specify the algorithm uniform resource identifier (URI) and its usage type.

WebSphere Application Server supports the following algorithm URI types:

**Message digest**
    Specifies the algorithm URI that is used for digest value calculation.

**Signature**
    Specifies the algorithm URI that is used for digital signature, including both signature and signing verification.

**Data encryption**
    Specifies the algorithm URI that is used for both encrypting and decrypting data.

**Key encryption**
    Specifies the algorithm URI that is used for encrypting and decrypting the encryption key.

If the URI is used for multiple usage types, then you must define a mapping of the URI to each usage type.

To view this administrative console page on the server level, complete the following steps:

1. Click **Servers > Application servers >***server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Algorithm mappings**.
4. Under Additional properties, click **Algorithm URI >** *algorithm_URI_name* or click **New**.

   **Related reference**

"Algorithm URI collection" on page 772
Use this page to view a list of uniform resource identifier (URI) algorithms for XML digital signature or XML encryption that are mapped to an algorithm factory engine class. With algorithm mappings, service providers can use other cryptographic algorithms for digest value calculation, digital signature signing and verification, data encryption and decryption, and key encryption and decryption.

"Algorithm mapping collection" on page 775
Use this page to view a list of custom uniform resource identifier (URI) algorithms for digest value calculation, signature, key encryption, and data encryption. WebSphere Application Server maps these algorithms to an implementation of the algorithm factory engine interface. With algorithm mappings, service providers can extend the cryptographic algorithms for XML digital signature and XML encryption.

"Algorithm mapping collection" on page 775
Use this page to view a list of custom uniform resource identifier (URI) algorithms for digest value calculation, signature, key encryption, and data encryption. WebSphere Application Server maps these algorithms to an implementation of the algorithm factory engine interface. With algorithm mappings, service providers can extend the cryptographic algorithms for XML digital signature and XML encryption.

"Signing information configuration settings" on page 716
Use this page to configure new signing parameters.

"Part reference configuration settings" on page 722
Use this page to specify a reference to the message parts for signature and encryption that are defined in the deployment descriptors.

"Encryption information configuration settings" on page 729
Use this page to configure the encryption and decryption parameters. You can use these parameters to encrypt and decrypt various parts of the message, including the body and user name token.

*Algorithm URI:*

Specifies the algorithm uniform resource identifier (URI) for the specified algorithm type.

The algorithm URI that is defined on this page is available to the various binding configurations. For example, if you specify an algorithm URI and select **Signature** from the Algorithm type field, the URI displays in the Signature method field on the signing information panel.

*Algorithm type:*

Specifies the type of algorithm that is specified in the Algorithm URI field.

The following types of algorithms are supported by WebSphere Application Server. The following list shows where configurations that are specified on this panel are displayed for a binding configuration:

| Algorithm type | Explanation | Location of the configuration |
| --- | --- | --- |
| Signature | This algorithm type is used for digital signatures. | This configuration displays in the Signature method field on the Signing information panel. For information on how to access the Signing information panel, see "Signing information configuration settings" on page 716. |
| Digest value calculation (message digest) | This algorithm type is used for calculating the digest value. | This configuration displays in the Digest method algorithm field on the Part references panel. For information on how to access the Part references panel, see "Part reference configuration settings" on page 722. |

| Algorithm type | Explanation | Location of the configuration |
| --- | --- | --- |
| Data encryption | This algorithm type is used for encrypting data. | This configuration displays in the Data encryption algorithm field on the Encryption information panel. For information on how to access the Encryption information panel, see "Encryption information configuration settings" on page 729. |
| Key encryption | This algorithm type is used for encrypting the key that is used for data encryption. | This configuration displays in the Key encryption algorithm field on the Encryption information panel. For information on how to access the Encryption information panel, see "Encryption information configuration settings" on page 729. |

The actual implementation of the algorithm is done in the implementation class for the engine factory.

***Algorithm mapping collection:***

Use this page to view a list of custom uniform resource identifier (URI) algorithms for digest value calculation, signature, key encryption, and data encryption. WebSphere Application Server maps these algorithms to an implementation of the algorithm factory engine interface. With algorithm mappings, service providers can extend the cryptographic algorithms for XML digital signature and XML encryption.

To view this administrative console page on the cell level, complete the following steps:

1. Click **Security > Web services**.
2. Under Additional properties, click **Algorithm mappings**.

To view this administrative console page on the server level, complete the following steps:

1. Click **Servers > Application servers >***server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Algorithm mappings**.

   **Related reference**

   "Algorithm URI collection" on page 772
   Use this page to view a list of uniform resource identifier (URI) algorithms for XML digital signature or XML encryption that are mapped to an algorithm factory engine class. With algorithm mappings, service providers can use other cryptographic algorithms for digest value calculation, digital signature signing and verification, data encryption and decryption, and key encryption and decryption.

   "Algorithm URI configuration settings" on page 773
   Use this page to specify the algorithm uniform resource identifier (URI) and its usage type.

   "Algorithm mapping collection"
   Use this page to view a list of custom uniform resource identifier (URI) algorithms for digest value calculation, signature, key encryption, and data encryption. WebSphere Application Server maps these algorithms to an implementation of the algorithm factory engine interface. With algorithm mappings, service providers can extend the cryptographic algorithms for XML digital signature and XML encryption.

*Algorithm factory engine class:*

Specifies the custom class that implements the engine factory implementation class for the algorithm factory engine.

The implementation class for the engine factory implements the cryptographic functions of the defined uniform resource identifier (URI).

### *Algorithm mapping configuration settings:*

Use this page to view a list of custom uniform resource identifier (URI) algorithms for digest value calculation, signature, key encryption, and data encryption. WebSphere Application Server maps these algorithms to an implementation of the algorithm factory engine interface. With algorithm mappings, service providers can extend the cryptographic algorithms for XML digital signature and XML encryption.

To view this administrative console page on the cell level, complete the following steps:

To view this administrative console page on the server level, complete the following steps:

1. Click **Servers > Application servers >***server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Algorithm mappings >** *algorithm_factory_engine_class_name* or click **New**.

**Related reference**

"Algorithm URI collection" on page 772
Use this page to view a list of uniform resource identifier (URI) algorithms for XML digital signature or XML encryption that are mapped to an algorithm factory engine class. With algorithm mappings, service providers can use other cryptographic algorithms for digest value calculation, digital signature signing and verification, data encryption and decryption, and key encryption and decryption.

"Algorithm URI configuration settings" on page 773
Use this page to specify the algorithm uniform resource identifier (URI) and its usage type.

"Algorithm mapping collection" on page 775
Use this page to view a list of custom uniform resource identifier (URI) algorithms for digest value calculation, signature, key encryption, and data encryption. WebSphere Application Server maps these algorithms to an implementation of the algorithm factory engine interface. With algorithm mappings, service providers can extend the cryptographic algorithms for XML digital signature and XML encryption.

*Algorithm factory engine class:*

Specifies the custom class that implements the engine factory interface.

To use this algorithm mapping feature, you must specify a custom algorithm class in the Algorithm factory engine class field for digital signature, data encryption, digest value calculation, and key encryption. The algorithm factory engine provides a plug-in point for service providers to provide their implementation for digest value calculation, digital signature, key encryption, and data encryption that is based on a specified algorithm uniform resource identifier (URI). By clicking **Algorithm URI** under Additional properties, you can specify the algorithm URI and its usage type. WebSphere Application Server supports the following algorithm types:

**Message digest**
Specifies the algorithm URI that is used for digest value calculation.

**Signature**
Specifies the algorithm URI that is used for digital signatures including both signing and signature verification.

**Data encryption**
Specifies the algorithm URI that is used for both encrypting and decrypting data.

**Key encryption**
Specifies the algorithm URI that is used for both encrypting and decrypting the encryption key.

If the URI is used for multiple usage types, then you must define a mapping of the URI to each usage type. The actual implementation of the algorithm is provided by the custom class that implements the engine factory interface. For more information, refer to the information center documentation on how to implement a factory class.

## Configuring the key locator on the server or cell level

The key locator information for the default generator bindings specifies which key locator implementation is used to locate the key that is used for signature and encryption information if these bindings are not defined at the application level. The key locator information for the default consumer bindings specifies which key locator implementation is used to locate the key that is used for signature validation or decryption if these bindings are not defined at the application level. WebSphere Application Server provides default values for the bindings. However, you must modify the defaults for a production environment.

Complete the following steps to configure the key locator on the server or cell level:

1. Access the default bindings for the server level.
   a. Click **Servers > Application servers >***server_name*.
   b. Under Security, click **Web services: Default bindings for Web services security**.
2. Under Additional properties, click **Key locator**. You can configure the key locator configurations for both the default generator and the default consumer in this location.
3. Click **New** to create a key locator configuration, click **Delete** to delete an existing configuration, or click the name of an existing key locator configuration to edit its settings. If you are creating a new configuration, enter a unique name for the key locator configuration in the Key locator name field. For example, you might specify `sig_klocator`.
4. Specify a name for the key locator class implementation in the Key locator class name field. The key locators that are associated with Version 6 applications must implement the com.ibm.wsspi.wssecurity.keyinfo.KeyLocator interface. WebSphere Application Server provides the following default key locator class implementations for Version 6 applications:

   **com.ibm.wsspi.wssecurity.keyinfo.KeyStoreLeyLocator**
   This implementation locates and obtains the key from a specified keystore file.

   **com.ibm.wsspi.wssecurity.keyinfo.SignerCertKeyLocator**
   This implementation uses the public key from the certificate of the signer. This class implementation is used by the response generator.

   **com.ibm.wsspi.wssecurity.keyinfo.X509TokenKeyLocator**
   This implementation uses the X.509 security token from the sender message for digital signature validation and encryption. This class implementation is used by the request consumer and the response consumer.

   For example, you might specify the com.ibm.wsspi.wssecurity.keyinfo.KeyStoreLeyLocator implementation if you need the configuration to be the key locator for signing information.
5. Specify the keystore password, the keystore location, and the keystore type. Keystore files contain public and private keys, root certificate authority (CA) certificates, the intermediate CA certificate, and so on. Keys that are retrieved from the keystore file are used to sign and validate or encrypt and decrypt messages or message parts. If you specified the com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator implementation for the key locator class implementation, you must specify a key store password, location, and type.
   a. Specify a password in the Key store password field. This password is used to access the keystore file.
   b. Specify the location of the keystore file in the Key store path field.
   c. Select a keystore type from the Key store type field. The Java Cryptography Extension (JCE) that is used supports the following key store types:

**JKS** Use this option if you are not using Java Cryptography Extensions (JCE) and if your keystore file uses the Java Keystore (JKS) format.

**JCEKS**
Use this option if you are using Java Cryptography Extensions.

**PKCS11**
Use this format if your keystore file uses the PKCS#11 file format. Keystore files that use this format might contain Rivest Shamir Adleman (RSA) keys on cryptographic hardware or might encrypt keys that use cryptographic hardware to ensure protection.

**PKCS12**
Use this option if your keystore file uses the PKCS#12 file format.

WebSphere Application Server provides some sample keystore files in the `${USER_INSTALL_ROOT}/etc/ws-security/samples` directory. For example, you might use the `enc-receiver.jceks` keystore file for encryption keys. The password for this file is `storepass` and the type is JCEKS.

**Attention:** Do not use these keystore files in a production environment. These samples are provided for testing purposes only.

6. Click **OK** and **Save** to save the configuration.
7. Under Additional properties, click **Keys**.
8. Click **New** to create a key configuration, click **Delete** to delete an existing configuration, or click the name of an existing key configuration to edit the settings. This entry specifies the name of the key object within the keystore file. If you are creating a new configuration, enter a unique name in the Key name field.

   You must use a fully qualified distinguished name for the key name. For example, you might use `CN=Bob,O=IBM,C=US`.
9. Specify an alias in the Key alias field. The key alias is used by the key locator to search for key objects in the keystore file.
10. Specify a password in the Key password field. The password is used to access the key object within the keystore file.
11. Click **OK** and then click **Save** to save the configuration.

You have configured the key locator for the server or cell level.

Configure the key information for the default generator and the default consumer bindings that reference this key locator.

## Configuring the key information for the generator binding on the server or cell level

Use the key information for the default generator to specify the key that is used by the signing or the encryption information configurations if these bindings are not defined at the application level. The signing and encryption information configurations can share the same key information, which is why they are both defined on the same level. WebSphere Application Server provides default values for these bindings. However, an administrator must modify these values for a production environment.

Complete the following steps to configure the key information for the generator binding on the server or cell level:

1. Access the default bindings for the server level.
   a. Click **Servers > Application servers >***server_name*.
   b. Under Security, click **Web services: Default bindings for Web services security**.
2. Under Default generator bindings, click **Key information**.

3. Click **New** to create a key information configuration, click **Delete** to delete an existing configuration, or click the name of an existing key information configuration to edit the settings. If you are creating a new configuration, enter a unique name for the key configuration in the Key information name field. For example, you might specify `sig_keyinfo`.
4. Select a key information type from the Key information type field. WebSphere Application Server supports the following types of key information:

   **Key identifier**
   : This key information type is used when two parties agree on how to create a key identifier. For example, a field of X.509 certificates can be used for the key identifier according to the X.509 profile.

   **Key name**
   : This key information type is used when the sender and receiver agree on the name of the key.

   **Security token reference**
   : This key information type is typically used when an X.509 certificate is used for digital signature.

   **Embedded token**
   : This key information type is used to embed a security token in an embedded element.

   **X509 issuer name and issuer serial**
   : This key information type specifies an X.509 certificate with its issuer name and serial number.

   Select **Security token reference** if you are using an X.509 certificate for the digital signature. In these steps, it is assumed that **Security token reference** is selected for this field.

   **Important:** This key information type must match the key information type that is specified for the consumer.
5. Select a key locator reference from the Key locator reference menu. In these steps, assume that the key locator reference is called sig_klocator. The key locator reference is the name of the key locator that is used to generate the key for digital signature. You must configure a key locator before you can select it in this field. For more information on configuring the key locator, see "Configuring the key locator on the server or cell level" on page 777.
6. Click **Get keys** to view a list of key name references. After you click **Get keys**, the key names that are defined in the sig_klocator element are shown in the key name reference menu. If you change the key locator reference, you must click **Get keys** again to display the list of key names that are associated with the new key locator.
7. Select a key name reference from the Key name reference menu. The key name reference specifies the name of the key that is used for generating the digital signature or for encryption. The Key name reference menu displays a list of key names that are defined for the selected key locator in the Key locator reference field. For example, select **signerkey**. It is assumed that signerkey is a key name that is defined for the sig_klocator key locator.
8. Select a token reference from the Token reference field. The token reference refers to the name of a configured token generator. When a security token is required in the deployment descriptor, the token reference attribute is required. If you select **Security token reference** in the Key information type field, the token reference is required and you can specify an X.509 token generator. To specify an X.509 token generator, you must have an X.509 token generator configured. To configure an X.509 token generator, see "Configuring token generators on the server or cell level" on page 765. For the remaining steps, it is assumed that an X.509 token generator that is named gen_tcon is already configured.
9. **Optional:** Select an encoding method from the Encoding method field This field specifies the encoding format for the key identifier. The encoding method attribute is valid when you select **Key identifer** as the key information type. WebSphere Application Server supports the following encoding methods:

- http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary
- http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#HexBinary

10. **Optional:** Select a calculation method from the Calculation method field. The calculation method specifies the calculation algorithm that is used for the key identifier. This attribute is valid when you select **Key identifier** as the key information type. WebSphere Application Server supports the following calculation methods:
    - http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#ITSHA1
    - http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#IT60SHA1
11. **Optional:** Specify a URI of the value type for a security token from the Namespace URI field. The namespace URI is referenced by the key identifier. This attribute is valid when you select **Key identifier** as the key information type. When you specify the X.509 certificate token, you do not need to specify the namespace URI. If another token is specified, you must specify the namespace URI. For example, you can specify `http://www.ibm.com/websphere/appserver/tokentype/5.0.2` for the Lightweight Third Party Authentication (LTPA) token.
12. **Optional:** Specify the local name of the value type for a security token in the Local name field. The local name is referenced by the key identifier. This attribute is valid when you select **Key identifier** as the key information type. WebSphere Application Server supports the following local names:

    **For an X.509 certificate token**
    http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3

    **For X.509 certificates in a PKIPath**
    http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509PKIPathv1

    **For a list of X.509 certificates and CRLs in a PKCS#7**
    http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#PKCS7

    **For LTPA**
    LTPA
13. Click **OK** and **Save** to save the configuration.

You have configured the key information for the generator binding at the server or cell level.

You must specify a similar key information configuration for the consumer.

**Related tasks**

“Configuring the key information for the consumer binding on the server or cell level” on page 795
“Configuring the key locator on the server or cell level” on page 777
“Configuring token generators on the server or cell level” on page 765

## Configuring the signing information for the generator binding on the server or cell level

In the server-side extensions file (`ibm-webservices-ext.xmi`) and the client-side deployment descriptor extensions file (`ibm-webservicesclient-ext.xmi`), you must specify which parts of the message are signed. Also, you need to configure the key information that is referenced by the key information references on the Signing information panel within the administrative console.

This task explains the steps that are needed for you to configure the signing information for the client-side request generator and the server-side response generator bindings at the server or cell level. WebSphere Application Server uses the signing information for the default generator to sign parts of the message that include the body, time stamp, and user name token if these bindings are not defined at the application level. The Application Server provides default values for bindings. However, an administrator must modify the defaults for a production environment.

1. Access the default bindings for the server level.
   a. Click **Servers > Application servers >***server_name*.
   b. Under Security, click **Web services: Default bindings for Web services security**.
2. Under Default generator bindings, click **Signing information**.
3. Click **New** to create a signing information configuration, click **Delete** to delete an existing configuration, or click the name of an existing signing information configuration to edit the settings. If you are creating a new configuration, enter a unique name for the signing configuration in the Signing information name field. For example, you might specify `gen_signinfo`.
4. Select a signature method algorithm from the Signature method field. The algorithm that is specified for the default generator must match the algorithm that is specified for the default consumer. WebSphere Application Server supports the following pre-configured algorithms:
   - http://www.w3.org/2000/09/xmldsig#rsa-sha1
   - http://www.w3.org/2000/09/xmldsig#dsa-sha1
   - http://www.w3.org/2000/09/xmldsig#hmac-sha1
5. Select a canonicalization method from the Canonicalization method field. The canonicalization algorithm that you specify for the generator must match the algorithm for the consumer. WebSphere Application Server supports the following pre-configured canonical XML and exclusive XML canonicalization algorithms:
   - http://www.w3.org/2001/10/xml-exc-c14n#
   - http://www.w3.org/2001/10/xml-exc-c14n#WithComments
   - http://www.w3.org/TR/2001/REC-xml-c14n-20010315
   - http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments
6. Select a key information signature type from the Key information signature type field. The key information signature type determines how to digitally sign the key. WebSphere Application server supports the following signature types:

   **None** Specifies that the KeyInfo element is not signed.

   **Keyinfo**
   Specifies that the entire KeyInfo element is signed.

   **Keyinfochildelements**
   Specifies that the child elements of the KeyInfo element are signed.

   The key information signature type for the generator must match the signature type for the consumer. You might encounter the following situations:
   - If you do not specify one of the previous signature types, WebSphere Application Server uses keyinfo, by default.
   - If you select Keyinfo or Keyinfochildelements and you select http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform as the transform algorithm in a subsequent step, WebSphere Application Server also signs the referenced token.
7. Select a signing key information reference from the Signing key information field. This selection is a reference to the signing key that the Application Server uses to generate digital signatures. In the binding files, this information is specified within the <signingKeyInfo> tag. The key that is used for signing is specified by the key information element, which is defined at the same level as the signing information. For more information, see "Configuring the key information for the generator binding on the server or cell level" on page 778.
8. Click **OK** to save the configuration.
9. Click the name of the new signing information configuration. This configuration is the one that you specified in the previous steps.
10. Specify the part reference, digest algorithm, and transform algorithm. The part reference specifies which parts of the message to digitally sign.

a. Under Additional Properties, click **Part references > New** to create a new part reference, click **Part references > Delete** to delete an existing part reference, or click a part name to edit an existing part reference.
b. Specify a unique part name for the message part that needs signing. This message part is specified on both the server side and the client side. You must specify an identical part name for both the server side and the client side. For example, you might specify `reqint` for both the generator and the consumer.

   **Important:** You do not need to specify a value for the Part reference in the default bindings like you specify on the application level because the part reference on the application level points to a particular part of the message that is signed. Because the default bindings for the server level is applicable to all of the services that are defined on a particular server, you cannot specify this value.
c. Select a digest method algorithm in the Digest method algorithm field. The digest method algorithm that is specified in the binding files within the <DigestMethod> element is used in the <SigningInfo> element. WebSphere Application Server supports the http://www.w3.org/2000/09/xmldsig#sha1 algorithm.
d. Click **OK** and **Save** to save the configuration.
e. Click the name of the new part reference configuration. This configuration is the one that you specified in the previous steps.
f. Under Additional properties, click **Transforms > New** to create a new transform, click **Transforms > Delete** to delete a transform, or click a transform name to edit an existing transform. If you create a new transform configuration, specify a unique name. For example, you might specify `reqint_body_transform1`.
g. Select a transform algorithm from the menu. The transform algorithm is specified within the <Transform> element. This algorithm element specifies the transform algorithm for the digital signature. WebSphere Application Server supports the following algorithms:
   - http://www.w3.org/2001/10/xml-exc-c14n#
   - http://www.w3.org/TR/1999/REC-xpath-19991116
   - http://www.w3.org/2002/06/xmldsig-filter2
   - http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform
   - http://www.w3.org/2002/07/decrypt#XML
   - http://www.w3.org/2000/09/xmldsig#enveloped-signature

   The transform algorithm that you select for the generator must match the transform algorithm that you select for the consumer.

   **Important:** If both of the following conditions are true, WebSphere Application Server signs the referenced token:
   - You previously selected the Keyinfo or the Keyinfochildelements option from the Key information signature type field on the signing information panel.
   - You select http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform as the transform algorithm.

11. Click **OK**.
12. Click **Save** at the top of the panel to save your configuration.

After completing these steps, you have configured the signing information for the generator on the server level.

You must specify a similar signing information configuration for the consumer.

**Related tasks**

“Configuring the signing information for the consumer binding on the server or cell level” on page 796

## Configuring the encryption information for the generator binding on the server or cell level

The encryption information for the default generator specifies how to encrypt the information on the sender side if these bindings are not defined at the application level. WebSphere Application Server provides default values for the bindings. However, an administrator must modify the defaults for a production environment.

Complete the following steps to configure the encryption information for the generator binding on the server level:

1. Access the default bindings for the server level.
   a. Click **Servers > Application servers >***server_name*.
   b. Under Security, click **Web services: Default bindings for Web services security**.
2. Under Default generator bindings, click **Encryption information**.
3. Click **New** to create an encryption information configuration, click **Delete** to delete an existing configuration, or click the name of an existing encryption information configuration to edit the settings. If you are creating a new configuration, enter a unique name for the encryption configuration in the Encryption information name field. For example, you might specify `gen_encinfo`.
4. Select a data encryption algorithm from the Data encryption algorithm field. This algorithm is used to encrypt the data. WebSphere Application Server supports the following pre-configured algorithms:
   - http://www.w3.org/2001/04/xmlenc#tripledes-cbc
   - http://www.w3.org/2001/04/xmlenc#aes128-cbc
   - http://www.w3.org/2001/04/xmlenc#aes256-cbc

     To use this algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.
   - http://www.w3.org/2001/04/xmlenc#aes192-cbc

     To use this algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.

   The data encryption algorithm that you select for the generator side must match the data encryption algorithm that you select for the consumer side.
5. Select a key encryption algorithm from the Key encryption algorithm field. This algorithm is used to encrypt the key. WebSphere Application Server supports the following pre-configured algorithms:
   - http://www.w3.org/2001/04/xmlenc#rsa-1_5
   - http://www.w3.org/2001/04/xmlenc#kw-tripledes
   - http://www.w3.org/2001/04/xmlenc#kw-aes128
   - http://www.w3.org/2001/04/xmlenc#kw-aes256

     To use this algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.
   - http://www.w3.org/2001/04/xmlenc#kw-aes192

     To use this algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.

   If you select **None**, the key is not encrypted.

   The key encryption algorithm that you select for the generator side must match the key encryption algorithm that you select for the consumer side.

6. Select a encryption key configuration from the Encryption key information field. This attribute specifies the name of the key that is used to encrypt the message. To configure the key information, see "Configuring the key information for the generator binding on the server or cell level" on page 778.
7. Click **OK** and then click **Save** to save the configuration.

You have configured the encryption information for the generator binding at the server or cell level.

You must specify a similar encryption information configuration for the consumer.

**Related tasks**

"Configuring the encryption information for the consumer binding on the server or cell level" on page 799

"Configuring the key information for the generator binding on the server or cell level" on page 778

## Configuring trusted ID evaluators on the server or cell level

This task provides the steps that are needed to configure trusted identity (ID) evaluators. The trusted ID evaluator determines whether to trust the identity-asserting authority. After the ID is trusted, the WebSphere Application Server issues the proper credentials based on the identity, which are used in a downstream call to another server for invoking resources. The trusted ID evaluator implements the com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator interface.

Complete the following steps to configure the trusted ID evaluators on the server level:

1. Access the default bindings for the server level.
   a. Click **Servers > Application servers >***server_name*.
   b. Under Security, click **Web services: Default bindings for Web services security**.
2. Under Additional properties, click **Trusted ID evaluators**.
3. Click **New** to create a trusted ID evaluator configuration, click **Delete** to delete an existing configuration, or click the name of an existing configuration to edit the settings. If you are creating a new configuration, enter a unique name for the trusted ID evaluator configuration in the Trusted ID evaluator name field. This field specifies the name that is used by the application binding to refer to a trusted identity (ID) evaluator that is defined in the default binding.
4. Specify a class name in the Trusted ID evaluator class name field. The default class name is com.ibm.wsspi.wssecurity.id.TrustedIDEvaluatorImpl. The specified trusted ID evaluator class name must implement the com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator class. When you use the default TrustedIDEvaluator class, you must specify the name and value properties for the default trusted ID evaluator to create the trusted ID list for evaluation.
5. Under Additional properties, click **Properties > New**.
6. Specify the trusted ID evaluator name as a property name. You must specify the trusted ID evaluator name in the form, `trustedId_`*n*, where *_n* is an integer from zero (0) to n.
7. Specify the trusted ID as a property value.

   ```
   property name="trustedId_0", value="CN=Bob,O=ACME,C=US"
   property name="trustedId_1, value="user1"
   ```

   If a distinguished name (DN) is used, the space is removed for comparison.
8. Click **OK** and then **Save**.

You have configured the trusted ID evaluators at the server or cell level.

***Trusted ID evaluator collection:***

Use this page to view a list of trusted identity (ID) evaluators. The trusted ID evaluator determines whether to trust the identity-asserting authority. After the ID is trusted, WebSphere Application Server issues the proper credentials based on the identity, which are used in a downstream call for invoking resources. The trusted ID evaluator implements the com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator interface.

To view this administrative console page for trusted ID evaluators on the server level, complete the following steps:

1. Click **Servers > Application servers >***server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Trusted ID evaluators**.
4. Click **New** to create a trusted ID evaluator or click **Delete** to a delete a trusted ID evaluator.

**6.x application** To view this administrative console page for trusted ID evaluators on the application level, complete the following steps:

1. Click **Applications > Enterprise applications >***application_name*.
2. Under Related items, click **EJB modules** or **Web modules**.
3. Click *URI_name*.
4. Under Additional properties, click **Web services: Server security bindings**.
5. Under Request consumer (receiver) binding, click **Edit custom**.
6. Click **Trusted ID evaluators**.
7. Click **New** to create a trusted ID evaluator or click **Delete** to delete a trusted ID evaluator.

**5.x application** To view this administrative console page for trusted ID evaluators on the application level, complete the following steps:

1. Click **Applications > Enterprise applications** > *application_name*.
2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.
3. Under Additional properties, click **Web services: Server security bindings**.
4. Click **Edit** under Request receiver binding.
5. Click **Trusted ID evaluators**.
6. Click **New** to create a trusted ID evaluator or click **Delete** to delete a trusted ID evaluator.

**Important:** Trusted ID evaluators are only required for the request receiver (Version 5.x applications) and the request consumer (Version 6.x applications), if identity assertion is configured.

Using this trusted ID evaluator collection panel, complete the following steps:

1. Specify a trusted ID evaluator name and a trusted ID evaluator class name.
2. Save your changes by clicking **Save** in the messages section at the top of the administrative console.
3. Click **Update run time** to update the Web services security run time with the default binding information, which is found in the `ws_security.xml` file. The configuration changes made to the other Web services also are updated in the Web services security run time.

   **Related reference**

   "Trusted ID evaluator configuration settings"
   Use this information to configure trust identity (ID) evaluators.

*Trusted ID evaluator name:*

Specifies the unique name of the trusted ID evaluator.

*Trusted ID evaluator class name:*

Specifies the class name of the trusted ID evaluator.

***Trusted ID evaluator configuration settings:***

Use this information to configure trust identity (ID) evaluators.

To view this administrative console page for trusted ID evaluators on the server level, complete the following steps:

1. Click **Servers > Application servers >***server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Trusted ID evaluators**.
4. Click **New** to create a trusted ID evaluator or click the name of an existing configuration to modify the settings.

**6.x application** To view this administrative console page for trusted ID evaluators on the application level, complete the following steps:

1. Click **Applications > Enterprise applications >***application_name*.
2. Under Related items, click **EJB modules** or **Web modules**.
3. Click the *URI_name*.
4. Under Additional properties, click **Web services: Server security bindings**.
5. Under Request consumer (receiver) binding, click **Edit custom**.
6. Click **Trusted ID evaluators**.
7. Click **New** to create a trusted ID evaluator or click **Delete** to delete a trusted ID evaluator.

**5.x application** To view this administrative console page for trusted ID evaluators on the application level, complete the following steps:

1. Click **Applications > Enterprise applications** > *application_name*.
2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.
3. Under Additional properties, click **Web services: Server security bindings**.
4. Click **Edit** under Request receiver binding.
5. Click **Trusted ID evaluators**.
6. Click **New** to create a trusted ID evaluator or click **Delete** to delete a trusted ID evaluator.

**Important:** Trusted ID evaluators are only required for the request receiver (Version 5.x applications) and the request consumer (Version 6.x applications), if identity assertion is configured.

You can specify one of the following options:

**None** Choose this option if you are not specifying a trusted ID evaluator.

**Existing evaluator definition**
Choose this option to specify a currently defined trusted ID evaluator.

**Binding evaluator definition**
Choose this option to specify a new trusted ID evaluator. A description of the required fields follows.

**Related reference**

"Trusted ID evaluator collection" on page 784
Use this page to view a list of trusted identity (ID) evaluators. The trusted ID evaluator determines whether to trust the identity-asserting authority. After the ID is trusted, WebSphere Application Server issues the proper credentials based on the identity, which are used in a downstream call for invoking resources. The trusted ID evaluator implements the com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator interface.

*Trusted ID evaluator name:*

Specifies the name that is used by the application binding to refer to a trusted identity (ID) evaluator that is defined in the default binding.

*Trusted ID evaluator class name:*

Specifies the class name of the trusted ID evaluator.

The specified trusted ID evaluator class name must implement the com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator interface. The default TrustedIDEvaluator class is com.ibm.wsspi.wssecurity.id.TrustedIDEvaluatorImpl. When you use this default TrustedIDEvaluator class, you must specify the name and the value properties for the default trusted ID evaluator to create the trusted ID list for evaluation.

To specify the name and value properties, complete the following steps:
1. Under Additional properties, click **Properties > New**.
2. Specify the trusted ID evaluator name as a property name. You must specify the trusted ID evaluator name in the form, `trustedId_n`, where_`n` is an integer from zero (0) to n.
3. Specify the trusted ID as a property value.

For example:

```
property name="trustedId_0", value="CN=Bob,O=ACME,C=US"
property name="trustedId_1", value="user1"
```

If a distinguished name (DN) is used, the space is removed for comparison.

| **Default** | com.ibm.wsspi.wssecurity.id.TrustedIDEvaluatorImpl |
|---|---|

See the programming model information in the documentation for an explanation of how to implement the com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator interface.

## Configuring token consumers on the server or cell level

The token consumer on the server or cell level is used to specify the information that is needed to process the security token if it is not defined at the application level. WebSphere Application Server provides default values for bindings. You must modify the defaults for a production environment.

Complete the following steps to configure the token consumers on the server level.
1. Access the default bindings for the server level.
   a. Click **Servers > Application servers >***server_name*.
   b. Under Security, click **Web services: Default bindings for Web services security**.
2. Under Default consumer bindings, click **Token consumers**.
3. Click **New** to create a token consumer configuration, click **Delete** to delete an existing configuration, or click the name of an existing token consumer configuration to edit its settings. If you are creating a new configuration, enter a unique name for the token consumer configuration in the Token consumer name field. For example, you might specify `sig_cgen`. This field specifies the name of the token consumer element.
4. Specify a class name in the Token consumer class name field. The token consumer class must implement the com.ibm.wsspi.wssecurity.token.TokenConsumerComponent interface. The token consumer class name must be similar to the token generator class name.

   For example, if your application requires an X.509 certificate token consumer, you can specify the com.ibm.wsspi.wssecurity.token.X509TokenGenerator class name on the Token generator panel and the com.ibm.wsspi.wssecurity.token.X509TokenConsumer class name in this field. WebSphere Application Server provides the following default token consumer class implementations:

   **com.ibm.wsspi.wssecurity.token.UsernameTokenConsumer**
   This implementation integrates a user name token.

**com.ibm.wsspi.wssecurity.token.X509TokenConsumer**
This implementation integrates an X.509 certificate token.

**com.ibm.wsspi.wssecurity.token.LTPATokenConsumer**
This implementation integrates a Lightweight Third Party Authentication (LTPA) token.

**com.ibm.wsspi.wssecurity.token.IDAssertionUsernameTokenConsumer**
This implementation integrates an IDAssertionUsername token.

A corresponding token generator class does not exist for this implementation.

5. Select a certificate path option. The certificate path specifies the certificate revocation list (CRL) that is used for generating a security token wrapped in a PKCS#7 with a CRL. WebSphere Application Server provides the following certificate path options:

   **None** If you select this option, the certificate path is not specified.

   **Trust any**
   If you select this option, any certificate is trusted. When the received token is consumed, the certificate path validation is not processed.

   **Dedicated signing information**
   If you select this option, you can specify a trust anchor and a certificate store. When you select the trust anchor or the certificate store of a trusted certificate, you must configure the collection certificate store before setting the certificate path. To define a collection certificate store on the server or cell level, see “Configuring the collection certificate store on the server or cell-level bindings” on page 762.

   a. Select a trust anchor in the Trust anchor field. WebSphere Application Server provides two sample trust anchors. However, it is recommended that you configure your own trust anchors for a production environment. For information on configuring a trust anchor, see “Configuring trust anchors on the server or cell level” on page 761.
   b. Select a collection certificate store in the Certificate store field. WebSphere Application Server provides a sample collection certificate store. If you select **None**, the collection certificate store is not specified. For information on specifying a list of certificate stores that contain untrusted, intermediary certificate files awaiting validation, see “Configuring trusted ID evaluators on the server or cell level” on page 784.
6. Select a trusted ID evaluator from the Trusted ID evaluation reference field. This field specifies a reference to the Trusted ID evaluator class name that is defined in Trusted ID evaluators panel. The trusted ID evaluator is used for evaluating whether the received ID is trusted. If you select **None**, the trusted ID evaluator is not referenced in this token consumer configuration. To configure a trusted ID evaluator, see “Configuring trusted ID evaluators on the server or cell level” on page 784.
7. Select the **Verify nonce** option if a nonce is included in a user name token on the generator side. Nonce is a unique cryptographic number that is embedded in a message to help stop repeat, unauthorized attacks of user name tokens. The **Verify nonce** option is available if you specify a user name token for the token consumer and nonce is added to the user name token on the generator side.
8. Select the **Verify timestamp** option if a time stamp is included in the user name token on the generator side. The **Verify Timestamp** option is available if you specify a user name token for the token consumer and a time stamp is added to the user name token on the generator side.
9. Specify the local name of the value type for the integrated token. This entry specifies the local name of the value type for a security token that is referenced by the key identifier. This attribute is valid when **Key identifier** is selected as the key information type. To specify the key information type, see “Configuring the key information for the consumer binding on the server or cell level” on page 795. WebSphere Application Server has predefined value type local names for the user name token and the X.509 certificate security token. Enter one of the following local names for the user name token and the X.509 certificate security token. When you specify the following local names, you do not need to specify the URI of the value type:

**Username token**
: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#UsernameToken

**X.509 certificate token**
: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3

**X.509 certificates in a PKIPath**
: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509PKIPathv1

**A list of X.509 certificates and CRLs in a PKCS#7**
: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#PKCS7

**Note:** To specify Lightweith Third Party Authentication (LTPA), you must specify both the value type local name and the Uniform Resource Identifier (URI). Specify `LTPA` for the local name and `http://www.ibm.com/websphere/appserver/tokentype/5.0.2` for the URI.

For example, when an X.509 certificate token is specified, you can use `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3` for the local name. When you specify the local name of another token, you must specify a value type Qname. For example: uri=`http://www.ibm.com/custom`, localName=`CustomToken`

10. Specify the value type uniform resource identifier (URI) in the URI field. This entry specifies the namespace URI of the value type for a security token that is referenced by the key identifier. This attribute is valid when **Key identifier** is selected as the key information type on the Key information panel for the default generator. When you specify the token consumer for the user name token or an X.509 certificate security token, you do not need to specify this option. If you specify another token, you need to specify the URI of the QName for the value type.
11. Click **OK** and then **Save** to save the configuration. After saving the token generator configuration, you can specify a Java Authentication and Authorization Service (JAAS) configuration for your token consumer.
12. Click the name of your token generator configuration.
13. Under Additional properties, click **JAAS configuration**.
14. Select a JAAS configuration from the JAAS configuration name field. The field specifies the name of the JAAS system for application login configuration. You can specify additional JAAS system and application configurations by clicking **Security > Global security**. Under Authentication, click **JAAS configuration** and either **Application logins > New** or **System logins > New**. Do not remove the predefined system or application login configurations. However, within these configurations, you can add module class names and specify the order in which WebSphere Application Server loads each module. WebSphere Application Server provides the following predefined JAAS configurations:

**ClientContainer**
: This selection specifies the login configuration that is used by the client container applications. The configuration uses the CallbackHandler application programming interface (API) that is defined in the deployment descriptor for the client container. To modify this configuration, see the JAAS configuration panel for application logins.

**WSLogin**
: This selection specifies whether all of the applications can use the WSLogin configuration to perform authentication for the security run time. To modify this configuration, see the JAAS configuration panel for application logins.

**DefaultPrincipalMapping**
: This selection specifies the login configuration that is used by Java 2 Connectors (J2C) to map users to principals that are defined in the J2C authentication data entries. To modify this configuration, see the JAAS configuration panel for application logins.

**system.wssecurity.IDAssertion**
This selection enables a Version 5.x application to use identity assertion to map a user name to a WebSphere Application Server credential principal. To modify this configuration, see the JAAS configuration panel for system logins.

**system.wssecurity.Signature**
This selection enables a Version 5.x application to map a distinguished name (DN) in a signed certificate to a WebSphere Application Server credential principal. To modify this configuration, see the JAAS configuration panel for system logins.

**system.LTPA_WEB**
This selection processes login requests that are used by the Web container such as servlets and JavaServer Pages (JSP) files. To modify this configuration, see the JAAS configuration panel for system logins.

**system.WEB_INBOUND**
This selection handles login requests for Web applications, which include servlets and JavaServer Pages (JSP) files. This login configuration is used by WebSphere Application Server Version 5.1.1. To modify this configuration, see the JAAS configuration panel for system logins.

**system.RMI_INBOUND**
This selection handles logins for inbound Remote Method Invocation (RMI) requests. This login configuration is used by WebSphere Application Server Version 5.1.1. To modify this configuration, see the JAAS configuration panel for system logins.

**system.DEFAULT**
This selection handles the logins for inbound requests that are made by internal authentications and most of the other protocols except Web applications and RMI requests. This login configuration is used by WebSphere Application Server Version 5.1.1. To modify this configuration, see the JAAS configuration panel for system logins.

**system.RMI_OUTBOUND**
This selection processes RMI requests that are sent outbound to another server when either the com.ibm.CSI.rmiOutboundLoginEnabled or the com.ibm.CSIOutboundPropagationEnabled properties are `true`. These properties are set in the CSIv2 authentication panel. To access the panel, click **Security > Global security**. Under Authentication, click **Authentication protocol > CSIv2 outbound authentication**. To set the com.ibm.CSI.rmiOutboundLoginEnabled property, select the **Custom outbound mapping** option. To set the com.ibm.CSIOutboundPropagationEnabled property, select the **Security attribute propagation** option. To modify this JAAS login configuration, see the JAAS configuration panel for system logins.

**system.wssecurity.X509BST**
This section verifies an X.509 binary security token (BST) by checking the validity of the certificate and the certificate path. To modify this configuration, see the JAAS configuration panel for system logins.

**system.wssecurity.PKCS7**
This selection verifies an X.509 certificate with a certificate revocation list in a PKCS7 object. To modify this configuration, see the JAAS configuration panel for system logins.

**system.wssecurity.PkiPath**
This section verifies an X.509 certificate with a public key infrastructure (PKI) path. To modify this configuration, see the JAAS configuration panel for system logins.

**system.wssecurity.UsernameToken**
This selection verifies the basic authentication (user name and password) data. To modify this configuration, see the JAAS configuration panel for system logins.

**system.wssecurity.IDAssertionUsernameToken**
This selection enables Version 6 applications to use identity assertion to map a user name to a WebSphere Application Server credential principal. To modify this configuration, see the JAAS configuration panel for system logins.

**None** With this selection, you do not specify a JAAS login configuration.

15. Click **OK** and then **Save** to save the configuration.

You have configured the token consumer at the server or cell level.

You must specify a similar token generator configuration for the server or cell level.

**Related tasks**

"Configuring trusted ID evaluators on the server or cell level" on page 784

"Configuring the collection certificate store on the server or cell-level bindings" on page 762

"Configuring trust anchors on the server or cell level" on page 761

***Token consumer collection:***

Use this page to view the token consumer. The information is used on the consumer side only to process the security token.

To view this administrative console page for the server level, complete the following steps:

1. Click **Servers > Application servers >***server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Default generator bindings, click **Token consumers**.

To view this administrative console page for Version 6.x applications on the application level, complete the following steps:

1. Click **Applications > Enterprise applications >***application_name*.
2. Under Related items, click **EJB modules** or **Web modules >** *URI_name*.
3. Under Additional properties, you can access the signing information for the following bindings:
   - For the Response generator (sender) binding, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**. Under Required properties, click **Token consumers**.
   - For the Response consumer (receiver) binding, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**. Under Required properties, click **Token consumers**.

**Related reference**

"Token consumer configuration settings" on page 792
Use this page to specify the information for the token consumer. The information is used at the consumer side only to process the security token.

"Token generator collection" on page 769
Use this page to view the token generators. The information is used on the generator side only to generate the security token.

"Token generator configuration settings" on page 769
Use this page to specify the information for the token generator. The information is used at the generator side only to generate the security token.

*Token consumer name:*

Specifies the name of the token consumer configuration.

*Token consumer class name:*

Specifies the name of the token consumer implementation class.

This class must implement the com.ibm.wsspi.wssecurity.token.TokenConsumerComponent interface.

***Token consumer configuration settings:***

Use this page to specify the information for the token consumer. The information is used at the consumer side only to process the security token.

To view this administrative console page for the server level, complete the following steps:
1. Click **Servers > Application servers >***server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Default consumer bindings, click **Token consumers >** *token_consumer_name* or click **New** to create a new token consumer.

To view this administrative console page for Version 6.x applications on the application level, complete the following steps:
1. Click **Applications > Enterprise applications >***application_name*.
2. Under Related items, click **EJB modules** or **Web modules >** *URI_name*.
3. Under Additional properties, you can access the signing information for the following bindings:
   - For the Response generator (sender) binding, click **Web services: Server security bindings**. Under Response generator (sender) binding, click **Edit custom**. Under Required properties, click **Token consumers**.
   - For the Response consumer (receiver) binding, click **Web services: Client security bindings**. Under Response consumer (receiver) binding, click **Edit custom**. Under Required properties, click **Token consumers**.
4. Click **New** to specify a new configuration or click the name of an existing configuration to modify its settings.

Before specifying additional properties, specify a value in the Token consumer name, the Token consumer class name, and the Value type local name fields.

**Related reference**

"Token consumer collection" on page 791
Use this page to view the token consumer. The information is used on the consumer side only to process the security token.

"Token generator collection" on page 769
Use this page to view the token generators. The information is used on the generator side only to generate the security token.

"Token generator configuration settings" on page 769
Use this page to specify the information for the token generator. The information is used at the generator side only to generate the security token.

*Token consumer name:*

Specifies the name of the token consumer configuration.

*Token consumer class name:*

Specifies the name of the token consumer implementation class.

This class must implement the com.ibm.wsspi.wssecurity.token.TokenConsumerComponent interface.

*Part reference:*

Specifies a reference to the name of the security token that is defined in the deployment descriptor.

On the application level, when the security token is not specified in the deployment descriptor, the Part reference field is not displayed.

*Certificate path:*

Specifies the trust anchor and the certificate store.

You can select the following options:

**None** If you select this option, the certificate path is not specified.

**Trust any**
If you select this option, any certificate is trusted. When the received token is incorporated, the certificate path validation is not processed.

**Dedicated signing information**
If you select this option, you can specify the trust anchor and the certificate store. When you select the trust anchor or the certificate store of a trusted certificate, you must configure the collection certificate store before setting the certificate path.

**Trust anchor**

You can specify a trust anchor for the following bindings on the following levels:

| Binding name | Cell level, server level, or application level | Path |
| --- | --- | --- |
| Default consumer binding | Server level | Click **Servers > Application servers >** *server_name*.<br><br>Under Security, click **Web services: Default bindings for Web services security**.<br><br>Under Additional properties, click **Trust anchors**. |

**Certificate store**

You can specify a certificate path configuration for the following bindings on the following levels:

| Binding name | Cell level, server level, or application level | Path |
| --- | --- | --- |
| Default consumer binding | Server level | 1. Click **Servers > Application servers >** *server_name*.<br>2. Under security, click **Web services: Default bindings for Web services security**.<br>3. Under Additional properties, click **Collection certificate store**. |

*Trusted ID evaluator reference:*

Specifies the reference to the Trusted ID evaluator class name that is defined in the Trusted ID evaluators panel. The trusted ID evaluator is used for determining whether the received ID is trusted.

You can select the following options:

**None** If you select this option, the trusted ID evaluator is not specified.

**Existing evaluator definition**
If you select this option, you can select one of the configured trusted ID evaluators.

You can specify a certificate path configuration for the following bindings on the following levels:

| Binding name | Cell level, server level, or application level | Path |
|---|---|---|
| Default consumer binding | Server level | 1. Click **Servers > Application servers >** *server_name*.<br>2. Under security, click **Web services: Default bindings for Web services security**.<br>3. Under Additional properties, click **Trusted ID evaluators**. |

**Binding evaluator definition**
If you select this option, you can specify a new trusted ID evaluator and its class name.

When you select a trusted ID evaluator reference, you must configure the trusted ID evaluators before setting the token consumer.

The Trusted ID evaluator field is displayed in the default binding configuration and the application server binding configuration.

*Verify nonce:*

Specifies whether the nonce of the user name token is verified.

This option is displayed on the cell, server, and application levels. This option is valid only when the type of incorporated token is the user name token.

*Verify timestamp:*

Specifies whether the time stamp of user name token is verified.

This option is displayed on the cell, server, and application levels. This option is valid only when the type of incorporated token is the user name token.

*Value type local name:*

Specifies the local name of value type for the consumed token.

WebSphere Application Server has predefined value type local names for the user name token and the X.509 certificate security token. Use the following local names for the user name token and the X.509 certificate security token. When you specify the following local names, you do not need to specify the Uniform Resource Identifier (URI) of the value type:

**Username token**
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#UsernameToken

**X509 certificate token**
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509

**# X509 certificates in a PKIPath**
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509PKIPathv1

**A list of X509 certificates and CRLs in a PKCS#7**
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#PKCS7

**LTPA**

**Important:** For Lightweight Third Party Authentication (LTPA), the value type local name is `LTPA`. If you enter LTPA for the local name, you must specify the http://www.ibm.com/websphere/appserver/tokentype/5.0.2 URI value in the Value type URI field as well. For the other predefined value types (Username token, X509 certificate token, X509 certificates in a PKIPath, and a list of X509 certificates and CRLs in a PKCS#7), the value for the local name field begins with `http://`. For example, if you are specifying the username token for the value type, enter `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#UsernameToken` in the value type local name field and then you do not need to enter a value in the value type URI field.

When you specify a custom value type for custom tokens, you can specify the local name and the URI of the Quality name (QName) of the value type. For example, you might specify `Custom` for the local name and `http://www.ibm.com/custom` for the URI.

*Value type URI:*

Specifies the namespace URI of the value type for the integrated token.

When you specify the token consumer for the user name token or the X.509 certificate security token, you do not need to specify this option. If you want to specify another token, specify the URI of the QName for the value type.

WebSphere Application Server provides the following predefined value type URI for the LTPA token: http://www.ibm.com/websphere/appserver/tokentype/5.0.2

## Configuring the key information for the consumer binding on the server or cell level

The key information for the default consumer is used to specify the key that is used by the signing or the encryption information configurations if these bindings are not defined at the application level. The signing and encryption information configurations can share the same key information, which is why they are both defined on the same level. WebSphere Application Server provides default values for these bindings. However, an administrator must modify these values for a production environment.

Complete the following steps to configure the key information for the consumer binding on the server or cell level:

1. Access the default bindings for the server level.
   a. Click **Servers > Application servers >***server_name*.
   b. Under Security, click **Web services: Default bindings for Web services security**.
2. Under Default consumer bindings, click **Key information**.
3. Click **New** to create a key information configuration, click **Delete** to delete an existing configuration, or click the name of an existing key information configuration to edit the settings. If you are creating a new configuration, enter a unique name for the key configuration in the Key information name field. For example, you might specify `con_signkeyinfo`.
4. Select a key information type from the Key information type field. WebSphere Application Server supports the following types of key information:

**Key identifier**
: This key information type is used when two parties agree on how to create a key identifier. For example, a field of X.509 certificates can be used for the key identifier according to the X.509 profile.

**Key name**
: This key information type is used when the sender and receiver agree on the name of the key.

**Security token reference**
: This key information type is typically used when an X.509 certificate is used for digital signature.

**Embedded token**
: This key information type is used to embed a security token in an embedded element.

**X509 issuer name and issuer serial**
: This key information type specifies an X.509 certificate with its issuer name and serial number.

Select **Security token reference** if you are using an X.509 certificate for the digital signature. In these steps, it is assumed that **Security token reference** is selected for this field.

**Important:** This key information type must match the key information type that is specified for the generator.

5. Select a key locator reference from the Key locator reference menu. In these steps, assume that the key locator reference is called sig_klocator. You must configure a key locator before you can select it in this field. For more information on configuring the key locator, see “Configuring the key locator on the server or cell level” on page 777.
6. Select a token reference from the Token reference field. The token reference refers to the name of a configured token consumer. When a security token is required in the deployment descriptor, the token reference attribute is required. If you select **Security token reference** in the Key information type field, the token reference is required and you can specify an X.509 token consumer. To specify an X.509 token consumer, you must have an X.509 token consumer configured. To configure an X.509 token consumer, see “Configuring token consumers on the server or cell level” on page 787.
7. Click **OK** and **Save** to save the configuration.

You have configured the key information for the consumer binding at the server or cell level.

You must specify a similar key information configuration for the generator

**Related tasks**

“Configuring the key information for the generator binding on the server or cell level” on page 778

“Configuring the key locator on the server or cell level” on page 777

“Configuring token consumers on the server or cell level” on page 787

## Configuring the signing information for the consumer binding on the server or cell level

In the server-side extensions file (`ibm-webservices-ext.xmi`) and the client-side deployment descriptor extensions file (`ibm-webservicesclient-ext.xmi`), you must specify which parts of the message are signed. Also, you need to configure the key information that is referenced by the key information references on the signing information panel within the administrative console.

This task explains the steps that are needed for you to configure the signing information for the client-side request generator and server-side response generator bindings at the server or cell level. WebSphere Application Server uses the signing information for the default generator to sign parts of the message including the body, time stamp, and user name token, if these bindings are not defined at the application level. The Application Server provides default values for bindings. However, an administrator must modify the defaults for a production environment.

Complete the following steps to configure the signing information for the consumer sections of the bindings files on the server level:

1. Access the default bindings for the server level.
   a. Click **Servers > Application servers >***server_name*.
   b. Under Security, click **Web services: Default bindings for Web services security**.
2. Under Default consumer bindings, click **Signing information**.
3. Click **New** to create a signing information configuration, click **Delete** to delete an existing configuration, or click the name of an existing signing information configuration to edit the settings. If you are creating a new configuration, enter a unique name for the signing configuration in the Signing information name field. For example, you might specify `gen_signinfo`.
4. Select a signature method algorithm from the Signature method field. The algorithm that is specified for the default consumer must match the algorithm that is specified for the default generator. WebSphere Application Server supports the following pre-configured algorithms:
   - http://www.w3.org/2000/09/xmldsig#rsa-sha1
   - http://www.w3.org/2000/09/xmldsig#dsa-sha1
   - http://www.w3.org/2000/09/xmldsig#hmac-sha1
5. Select a canonicalization method from the Canonicalization method field. The canonicalization algorithm that you specify for the generator must match the algorithm for the consumer. WebSphere Application Server supports the following pre-configured canonical XML and exclusive XML canonicalization algorithms:
   - http://www.w3.org/2001/10/xml-exc-c14n#
   - http://www.w3.org/2001/10/xml-exc-c14n#WithComments
   - http://www.w3.org/TR/2001/REC-xml-c14n-20010315
   - http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments
6. Select a key information signature type from the Key information signature type field. The key information signature type determines how to digitally sign the key. WebSphere Application Server supports the following signature types:

   **None** Specifies that the KeyInfo element is not signed.

   **Keyinfo**
   Specifies that the entire KeyInfo element is signed.

   **Keyinfochildelements**
   Specifies that the child elements of the KeyInfo element are signed.

   The key information signature type for the consumer must match the signature type for the generator. You might encounter the following situations:
   - If you do not specify one of the previous signature types, WebSphere Application Server uses keyinfo, by default.
   - If you select Keyinfo or Keyinfochildelements and you select http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform as the transform algorithm in a subsequent step, WebSphere Application Server also signs the referenced token.
7. Click **OK** to save the configuration.
8. Click the name of the new signing information configuration. This configuration is the one that you specified in the previous steps.
9. Specify the key information reference, part reference, digest algorithm, and transform algorithm.
   a. Under Additional properties, click **Key information references > New** to create a new reference, click **Key information references > Delete** to delete an existing reference, or click a reference name to edit an existing key information reference.
   b. Enter a name for the configuration in the Name field. For example, enter `con_skeyinfo`.

c. Select a key information reference from the Key information reference field. The key Information reference points to the key that WebSphere Application Server uses for digital signing. In the binding files, the reference is specified within the <signingKeyInfo> element. The key that is used for signing is specified by the Key information element, which is defined at the same level as the signing information. For more information, see "Configuring the key information for the consumer binding on the application level" on page 750.
d. Click **OK** and **Save** to save the configuration.
e. Under Additional Properties, click **Part references > New** to create a new part reference, click **Part references > Delete** to delete an existing part reference, or click a part name to edit an existing part reference. The part reference specifies which parts of the message to digitally sign. The part attribute refers to the name of the <RequiredIntegrity> element in the deployment descriptor when <PartReference> is specified for the digital signature. WebSphere Application Server enables you to specify multiple <PartReference> elements for the <SigningInfo> element. The `<PartReference>` element has two child elements: <DigestMethod> and <Transform>
f. Specify a unique part name for this part reference. For example, you might specify `reqint`.

**Important:** We do not need to specify a value for the Part reference field like you specify on the application level because the part reference on the application level points to a particular part of the message that is signed. Because the default bindings for the server level is applicable to all of the services that are defined on a particular server, you cannot specify this value.

g. Select a digest method algorithm in the Digest method algorithm field. The digest method algorithm specified within the <DigestMethod> element that is used in the <SigningInfo> element. WebSphere Application Server supports the http://www.w3.org/2000/09/xmldsig#sha1 algorithm.
h. Click **OK** and **Save** to save the configuration.
i. Click the name of the new part reference configuration. This configuration is the one that you specified in the previous steps.
j. Under Additional properties, click **Transforms > New** to create a new transform, click **Transforms > Delete** to delete a transform, or click a transform name to edit an existing transform. If you create a new transform configuration, specify a unique name. For example, you might specify `reqint_body_transform1`.
k. Select a transform algorithm from the menu. The transform algorithm is specified within the <Transform> element. It specifies the transform algorithm for the signature. WebSphere Application Server supports the following algorithms:
   - http://www.w3.org/2001/10/xml-exc-c14n#
   - http://www.w3.org/TR/1999/REC-xpath-19991116
   - http://www.w3.org/2002/06/xmldsig-filter2
   - http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform
   - http://www.w3.org/2002/07/decrypt#XML
   - http://www.w3.org/2000/09/xmldsig#enveloped-signature

   The transform algorithm that you select for the consumer must match the transform algorithm that you select for the generator.

   **Important:** If both of the following conditions are true, WebSphere Application Server signs the referenced token:
   - You previously selected the Keyinfo or the Keyinfochildelements option from the Key information signature type field on the signing information panel.
   - You select http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform as the transform algorithm.

10. Click **OK**.

11. Click **Save** at the top of the panel to save your configuration.

After completing these steps, you have configured the signing information for the consumer on the server level.

You must specify a similar signing information configuration for the generator.

**Related tasks**

"Configuring the signing information for the generator binding on the server or cell level" on page 780

## Configuring the encryption information for the consumer binding on the server or cell level

The encryption information for the default consumer specifies how to process the encryption information on the receiver side if these bindings are not defined at the application level. WebSphere Application Server provides default values for the bindings. However, an administrator must modify the defaults for a production environment.

Complete the following steps to configure the encryption information for the consumer binding on the server level:

1. Access the default bindings for the server level.
   a. Click **Servers > Application servers >***server_name*.
   b. Under Security, click **Web services: Default bindings for Web services security**.
2. Under Default consumer bindings, click **Encryption information**.
3. Click **New** to create an encryption information configuration, click **Delete** to delete an existing configuration, or click the name of an existing encryption information configuration to edit the settings. If you are creating a new configuration, enter a unique name for the encryption configuration in the Encryption information name field. For example, you might specify `con_encinfo`.
4. Select a data encryption algorithm from the Data encryption algorithm field. This algorithm is used to encrypt the data. WebSphere Application Server supports the following pre-configured algorithms:
   - http://www.w3.org/2001/04/xmlenc#tripledes-cbc
   - http://www.w3.org/2001/04/xmlenc#aes128-cbc
   - http://www.w3.org/2001/04/xmlenc#aes256-cbc

     To use this algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.
   - http://www.w3.org/2001/04/xmlenc#aes192-cbc

     To use this algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.

   The data encryption algorithm that you select for the consumer side must match the data encryption algorithm that you select for the generator side.
5. Select a key encryption algorithm from the Key encryption algorithm field. This algorithm is used to encrypt the key. WebSphere Application Server supports the following pre-configured algorithms:
   - http://www.w3.org/2001/04/xmlenc#rsa-1_5
   - http://www.w3.org/2001/04/xmlenc#kw-tripledes
   - http://www.w3.org/2001/04/xmlenc#kw-aes128
   - http://www.w3.org/2001/04/xmlenc#kw-aes256

     To use this algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.
   - http://www.w3.org/2001/04/xmlenc#kw-aes192

To use this algorithm, you must download the unrestricted Java Cryptography Extension (JCE) policy file from the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.

If you select **None**, the key is not encrypted.

The key encryption algorithm that you select for the consumer side must match the key encryption algorithm that you select for the generator side.

6. Under Additional properties, click **Key information references**.
7. Click **New** to create a key information configuration, click **Delete** to delete an existing configuration, or click the name of an existing key information configuration to edit the settings. If you are creating a new configuration, enter a unique name for the key information configuration in the name field. For example, you might specify `con_enckeyinfo`.
8. Select a key information reference from the Key information reference field. This selection refers to the name of the key information that is used for encryption. For more information, see “Configuring the key information for the consumer binding on the server or cell level” on page 795.
9. Click **OK** and **Save** to save the configuration.

You have configured the encryption information for the consumer binding at the server or cell level

You must specify a similar encryption information configuration for the generator.

**Related tasks**

“Configuring the encryption information for the generator binding on the server or cell level” on page 783

## Tuning Web services security

The Java Cryptography Extension (JCE) is integrated into the software development kit (SDK) version 1.4.x and is no longer an optional package. However, due to export and import regulations, the default Java Cryptography Extension (JCE) jurisdiction policy file shipped with the SDK enables you to use strong, but limited, cryptography only. To enforce this default policy, WebSphere Application Server uses a JCE jurisdiction policy file that might introduce a performance impact. The default JCE jurisdiction policy might have a performance impact on the cryptographic functions that are supported by Web services security. If you have Web services applications that use transport level security for XML encryption or digital signatures, you might encounter performance degradation over previous releases of WebSphere Application Server. However, IBM and Sun Microsystems provide versions of these jurisdiction policy files that do not have restrictions on cryptographic strengths. If you are permitted by your governmental import and export regulations, download one of these jurisdiction policy files. After downloading one of these files, the performance of JCE and Web Services security might improve substantially.

For WebSphere Application Server platforms using IBM Developer Kit, Java Technology Edition Version 1.4.2, including the AIX, Linux, and Windows platforms, you can obtain unlimited jurisdiction policy files by completing the following steps:

1. Go to the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html
2. Click **Java 1.4.2 material**
3. Click **IBM SDK Policy files**.
4. Select **Unrestricted JCE Policy files for SDK 1.4.2**
5. Enter your user ID and password or register with IBM to download the policy files. The policy files are downloaded onto your machine.

For WebSphere Application Server platforms using the Sun-based Java Development Kit (JDK) Version 1.4.2, including the Solaris environments and the HP-UX platform, you can obtain unlimited jurisdiction policy files by completing the following steps:

1. Go to the following Web site: http://java.sun.com/j2se/1.4.2/download.html

2. Click **Archive area**.
3. Locate the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 1.4.2 information and click **Download**. The policy files are downloaded onto your machine.

After following either of these sets of steps, two Java Archive (JAR) files are placed in the JVM `jre/lib/security/` directory.

## Securing Web services for version 5.x applications based on WS-Security

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

Web services security for WebSphere Application Server is based on standards included in the Web services security (WS-Security) specification. These standards address how to provide protection for messages exchanged in a Web service environment. The specification defines the core facilities for protecting the integrity and confidentiality of a message and provides mechanisms for associating security-related claims with the message. Web services security is a message-level standard based on securing Simple Object Access Protocol (SOAP) messages through XML digital signature, confidentiality through XML encryption, and credential propagation through security tokens.

Use the deprecated ″Securing Apache SOAP Web services″ topics in the WebSphere Application Server, Version 5 documentation if you are still using Apache SOAP Version 2.3.

To secure Web services, you must consider a broad set of security requirements, including authentication, authorization, privacy, trust, integrity, confidentiality, secure communications channels, federation, delegation, and auditing across a spectrum of application and business topologies. One of the key requirements for the security model in today's business environment is the ability to interoperate between formerly incompatible security technologies (such as public key infrastructure, Kerberos and so on.) in heterogeneous environments (such as Microsoft .NET and Java 2 Platform, Enterprise Edition (J2EE)). The complete Web services security protocol stack and technology roadmap is described in Security in a Web Services World: A Proposed Architecture and Roadmap.

Specification: Web Services Security (WS-Security) proposes a standard set of SOAP extensions that you can use to build secure Web services. These standards confirm integrity and confidentiality, which are generally provided with digital signature and encryption technologies. In addition, Web services security provides a general purpose mechanism for associating security tokens with messages. A typical example of the security token is a user name and password token, in which a user name and password are included as text. Web services security defines how to encode binary security tokens using methods such as X.509 certificates and Kerberos tickets.

To establish a managed environment and to enforce constraints for Web services security, you must perform a Java Naming and Directory Interface (JNDI) lookup on the client to resolve the service reference. For more information on the recommended client programming model, see ″Service lookup″ in the Java Specification Request (JSR) 109 specification available at: ftp://www-126.ibm.com/pub/jsr109/spec/1.0/websvcs-1_0-fr.pdf.

An administrator can use any of the following methods to integrate message-level security into a WebSphere Application Server environment:
- “Securing Web services for version 5.x applications using XML digital signature” on page 829
- “Securing Web services for version 5.x applications using XML encryption” on page 882
- “Securing Web services for version 5.x applications using basicauth authentication” on page 899

- “Securing Web services for version 5.x applications using identity assertion authentication” on page 907
- “Securing Web services for version 5.x applications using signature authentication” on page 913
- “Securing Web services for version 5.x applications using a pluggable token” on page 925

## Web services security specification-a chronology

This document describes the process used to develop the Web services security specifications.

**Non-OASIS activities**

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

In April 2002, IBM, Microsoft, and VeriSign proposed the *Web Services Security (WS-Security) specification* on their Web sites. This specification included the basic ideas of security token, XML signature, and XML encryption. The specification also defined the format for username tokens and encoded binary security tokens. After some discussion and an interoperability test based on the specification, the following issues were noted:

- The specification requires that the Web services security processors understand the schema correctly so that the processor distinguishes between the ID attribute for XML signature and XML encryption.
- The freshness of the message, which indicates whether the message complies with predefined time constraints, cannot be determined.
- Digested password strings do not strengthen security.

In August 2002, IBM, Microsoft, and VeriSign published the *Web Services Security Addendum*, which attempted to address the previously listed issues. The following solutions were put in the addendum:

- Require a global ID attribute for XML signature and XML encryption
- Use time stamp header elements that indicate the time of the creation, receipt, or expiration of the message
- Use password strings that are digested with a time stamp and nonce (randomly generated token)

**OASIS activities**

In June 2002, the Organization for the Advancement of Structured Information Standards (OASIS) received a proposed Web services security specification from IBM, Microsoft, and Verisign. The Web Services Security Technical Committee (WSS TC) was organized at OASIS soon after the submission. The technical committee included many companies including IBM, Microsoft, VeriSign, Sun Microsystems, and BEA Systems.

In September 2002, WSS TC published its first specification, *Web Services Security Core Specification, Working Draft 01*. This specification included the contents of both the original Web services security specification and its addendum.

The coverage of the technical committee became larger as the discussion proceeded. Since the Web Services Security Core Specification allows arbitrary types of security tokens, proposals were published as profiles. The profiles described the method for embedding tokens, including Security Assertion Markup Language (SAML) tokens and Kerberos tokens imbedded into the Web services security messages. Subsequently, the definitions of the usage for user name tokens and X.509 binary security tokens, which were defined in the original Web Services Security Specification, were divided into the profiles.

WebSphere Application Server supports the following specifications:

- Web Services Security: SOAP Message Security Draft 13 (formerly Web Services Security Core Specification)

- Web Services Security: Username Token Profile Draft 2

The following figure shows the various Web services security-related specifications. As indicated in the figure, the current support level for Web services security: SOAP message security is based on Draft 13 from May 2003. The current support level for Web services security User name token profiles, is based on Draft 2 from February 2003.

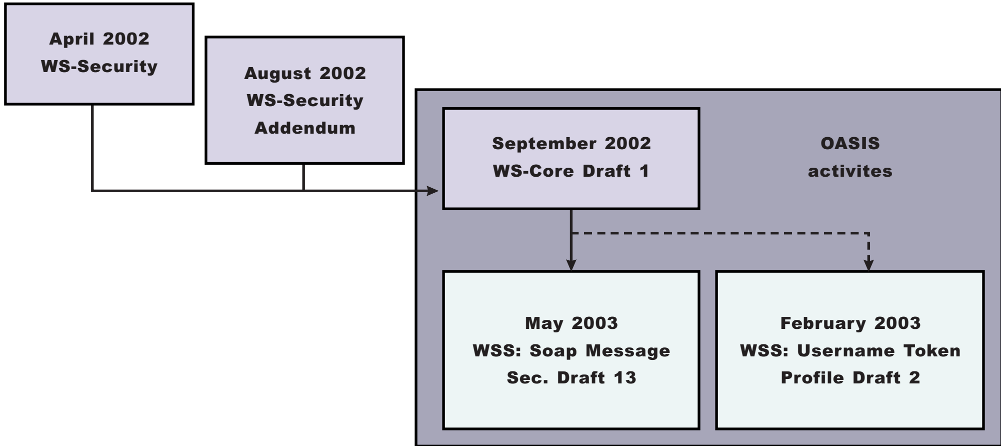



*Figure 3. Web services security specification support*

## Web services security support

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

WebSphere Application Server, Versions 4.x, 5, and 5.0.1 support digital signature for Apache Simple Object Access Protocol (SOAP) Version 2.x. Beginning with WebSphere Application Server, Version 5.0.2, IBM supports *Web services security*, which is an extension of the IBM Web services engine to provide a quality of service. The IBM implementation is based on the Web services security specification, ″Web Services Security (WS-Security)″, originally proposed by IBM, Microsoft, and VeriSign in April 2002. Early versions of the proposed draft specification can be found in Web Services Security (WS-Security) Version 1.0 05 April 2002 and Web Services Security Addendum 18 August 2002. The WebSphere Application Server implementation is based on the Organization for the Advancement of Structured Information Standards (OASIS) working Draft 13 specification. (See the OASIS Web Services Security TC Web site for the latest working specification.) However, not all the features in the OASIS working Draft 13 specification are implemented.

WebSphere Application Server security infrastructure fully integrates Web services security with Java 2 Platform, Enterprise Edition (J2EE) security. Web services security is not supported in a pure Java or non-managed client. When a user ID and password are embedded in a request message, authentication is performed with the user ID and password. If authentication is successful, a user identity is established and further resource access is authorized based on that identity. After the user ID and password are authenticated by the Web services security run time, a J2EE container performs authorization.

WebSphere Application Server provides an implementation of the key features of Web services security based on the following specifications:

- Specification: Web Services Security (WS-Security) Version 1.0 05 April 2002
- Web Services Security Addendum 18 August 2002

- Web Services Security: SOAP Message Security Working 13 May 2003
- Web Services Security: Username Token Profile Draft

The following table provides a summary of Web services security elements supported by WebSphere Application Server:

*Table 16. Web services security elements*

| Element | Notes |
|---|---|
| `UsernameToken` | Both the user name and password for the `BasicAuth` authentication method and the user name for the identity assertion authentication method are supported. WebSphere Application Server supports nonce, a randomly generated value. |
| `BinarySecurityToken` | X.509 certificates and Lightweight Third Party Authentication (LTPA) can be embedded, but there is no implementation to embed Kerberos tickets. However, the binary token generation and validation are pluggable and are based on the Java Authentication and Authorization Service (JAAS) Application Programming Interfaces (APIs). You can extend this implementation to generate and validate other types of binary security tokens. |
| `Signature` | The X.509 certificate is embedded as a binary security token and can be referenced by the `SecurityTokenReference`. WebSphere Application Server does not support shared, key-based signature. |
| `Encryption` | Both the `EncryptedKey` and `ReferenceList` XML tags are supported. `KeyIdentifier` specifies public keys and `KeyName` identifies the secret keys. WebSphere Application Server has the capability to map an authenticated identity to a key for encryption or use the signer certificate to encrypt the response message. |
| `Timestamp` | WebSphere Application Server supports the Created and Expires attributes. The freshness of the message, which indicates whether the message complies with predefined time constraints, is checked only if the Expires attribute is present in the message. WebSphere Application Server does not support the Received attribute, which is defined in the addendum. Instead, WebSphere Application Server uses the TimestampTrace Received attribute, which is defined in the OASIS specification. |
| `XML based token` | You can insert and validate an arbitrary format of XML tokens into a message. This format mechanism is based on the JAAS APIs. |

Signing and encrypting attachments is not supported by WebSphere Application Server. However, WebSphere Application Server signs and encrypts the following elements for the request message.

| Method | Element |
|---|---|
| XML digital signature | • Body<br>• Securitytoken<br>• Timestamp |
| XML encryption | • Bodycontent<br>• Usernametoken |

| Method | Element |
| --- | --- |
| AuthMethod | • BasicAuth<br>• IDAssertion (From WebSphere Application Server to another WebSphere Application Server<br>• Signature<br>• Lightweight Third Party Authentication (LTPA) on the server side<br>• Other customer tokens |

WebSphere Application Server signs and encrypts the following elements for the response message:

| Method | Element |
| --- | --- |
| XML digital signature | • Body<br>• Timestamp |
| XML encryption | • Bodycontent |

The namespaces used for sending a message were published by OASIS in draft 13 (http://schemas.xmlsoap.org/ws/2003/06/secext).

**April 2002 specification**
: http://schemas.xmlsoap.org/ws/2002/04/secext

**August 2002 addendum**
: http://schemas.xmlsoap.org/ws/2002/07/secext

http://schemas.xmlsoap.org/ws/2002/07/utility

**OASIS draft published on draft 13 May 2003**
: http://schemas.xmlsoap.org/ws/2003/06/secext

http://schemas.xmlsoap.org/ws/2003/06/utility

**Note:** WebSphere Application Server only uses the previously mentioned two name spaces for sending out requests and responses. However, the product can process all other mentioned name spaces for incoming requests and responses.

WebSphere Application Server provides the following capabilities for Web services security:
- Integrity of the message
- Authenticity of the message
- Confidentiality of the message
- Privacy of the message
- Transport level security: provided by Secure Sockets Layer (SSL)
- Security token propagation (pluggable)
- Identity assertion

See the previous table titled, ″Web services security elements,″ for a description of capabilities that are not supported.

## Web services security and Java 2 Platform, Enterprise Edition security relationship

This document describes the relationship between Web services security (message level security) and Java 2 Platform, Enterprise Edition (J2EE) platform security.

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

WebSphere Application Server supports Java Specification Requests (JSR) 101 and JSR 109. For more information, see Developing Web services applications. JSRs 101 and 109 define Web services for the Java 2 Platform, Enterprise Edition (J2EE) architecture so that you can develop and run Web services on the J2EE component architecture. ″Web services security″ refers to the ″Web services security: SOAP Message Security″ specification. For more information, see “Web services security support” on page 803.

**Important**

**Note:** ″Web services security″ refers to the ″Web services security: SOAP Message Security″ specification (see Web services security support for more information).

**Securing Web services with WebSphere Application Server security (J2EE role-based security)**

You can secure Web services using the existing security infrastructure of WebSphere Application Server, J2EE role-based security, and Secure Sockets Layer (SSL) transport level security.

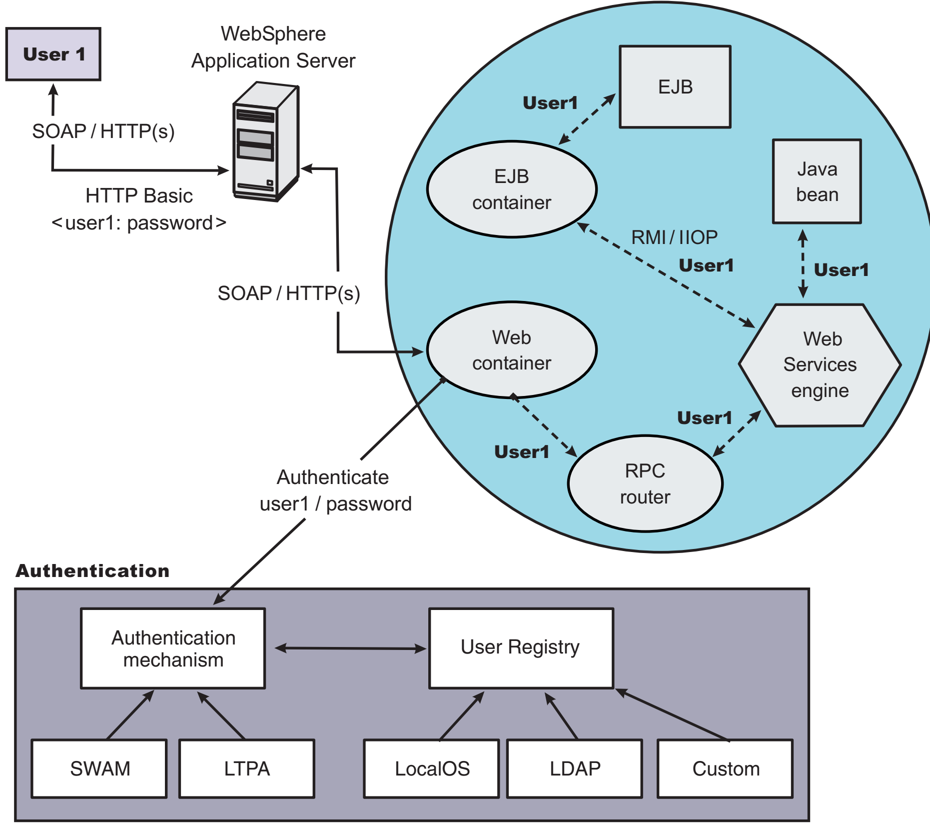



*Figure 4. Simple object access protocol message flow using existing security infrastructure of WebSphere Application Server*

The Web services port can be secured using J2EE role-based security. The Web services sender sends the basic authentication data using the HTTP header. SSL (HTTPS) can be used to secure the transport. When the WebSphere Application Server receives the SOAP message, the Web container authenticates the user (in this example, `user1`) and sets the security context for the call. After the security context is set, the SOAP router servlet sends the request to the implementation of the Web services (the implementation

can be JavaBeans or enterprise bean files). For enterprise bean implementations, the EJB container performs an authorization check against the identity of `user1`.

The Web services port also can be secured using the J2EE role. Then, authorization is performed by the Web container before the SOAP request is dispatched to the Web services implementation.

**Securing Web services with Web services security at the message level**

You can also secure Web services using Web services security at the message level. In this case, you can digitally sign or encrypt a certain part of the message. Web services security also supports security token propagation within the SOAP message. The following scenario assumes that the Web services port is not secured with J2EE role-based security and the enterprise bean is secured with J2EE role-based security.

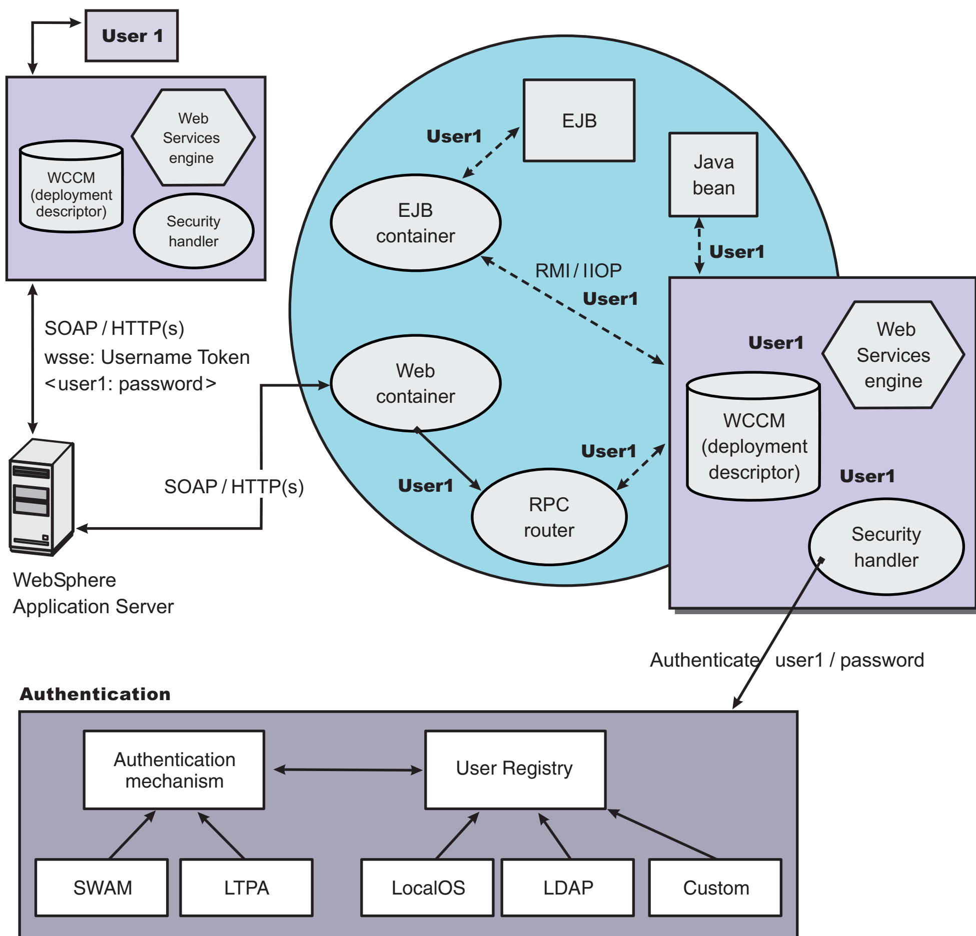



*Figure 5. Simple Object Access Protocol message flow using Web services security*

In this case, the Web services port is not secured with J2EE role-based security. The Web services engine processes the SOAP message before the client sends the message to the Web services port. The Web services security run time acts on the security constraints, such as digitally signing, encrypting, or generating (and inserting) a security token in the SOAP header. In this case `<wsse:UsernameToken>` is generated using `user1` and the password. On the server-side (receiving), the Web services process the incoming message and Web services security enforces security constraints. This enforcement includes making sure that messages are properly signed, properly encrypted, and decrypted, authenticating the security token, and setting up the security context with the authenticated identity. (In this case, `user1` is the

authenticated identity.) Finally, the SOAP message is dispatched to the Web services implementation (if the implementation is an enterprise beans file, the EJB container performs an authorization check against `user1`). SSL also might be used in this scenario.

**Mixing the two**

The second scenario shows that Web services security can complement J2EE role-based security. For example, SSL can be enabled at the transport level to provide a secure channel. Furthermore, if the Web services implementation is an enterprise beans file, you can leverage the EJB role-based authorization by performing authorization checks. Web services security run time leverages the security infrastructure to set the authenticated identity in the security context. The authenticated identity can be used in the downstream call to J2EE resources (or other resource types).

There are subtle consequences of combining the two scenarios. For example, if the HTTP transport is sending basic authentication data with `user1` and password in the HTTP header, but `<wsse:UsernameToken>` with `user99` and `letmein` also is inserted into the SOAP header. In the previous scenarios, there are two authentications performed. One authentication is performed by the Web container for authenticating `user1`, and the other is performed by Web services security for authenticating `user99`. The Web services security run time runs after the Web container runs and `user99` is the authenticated identity that is set in the security context.

Web services security can also propagate security tokens from the sender to the receiver for SOAP over a Java Message Service (JMS) transport.

## Web services security model in WebSphere Application Server

The Web services security model used by WebSphere Application Server is the declarative model. WebSphere Application Server does not include any application programming interfaces (APIs) for programmatically interacting with Web services security. However, a few Server Provider Interfaces (SPIs) are available for extending some security-related behaviors.

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

Enterprise application 1

EJB module

Web services implemented as an EJB file

Security handler

ibm-webserviecesclient-ext.xmi
ibm-webservicesclient-bnd.xmi

Enterprise application 2

EJB module

Web services implemented as an EJB file

Security handler

ibm-webservices-ext.xmi
ibm-webservices-bnd.xmi

*Figure 6. Web services security model*

The security constraints for Web services security are specified in IBM deployment descriptor extensions for Web services. The Web services security run time acts on the constraints to enforce Web services security for the Simple Object Access Protocol (SOAP) message. The scope of the IBM deployment descriptor extension is at the enterprise bean (EJB) or Web module level. Bindings are associated with each of the following IBM deployment descriptor extensions:

**Client (Might be either a J2EE Client (Application Client Container) or Web services acting as a client)**

`ibm-webservicesclient-ext.xmi`

`ibm-webservicesclient-bnd.xmi`

**Server**

`ibm-webservices-ext.xmi`

`ibm-webservices-bnd.xmi`

It is recommended that you use the tools provided by IBM (the Application Server Toolkit and Rational Web Developer) to create the IBM deployment descriptor extension and bindings. After the bindings are created, you can use the administrative console or an assembly tool to specify the bindings.

### Important

**Note:** The binding information is collected after application deployment rather than during application deployment. The alternative is to specify the required binding information before deploying your application.

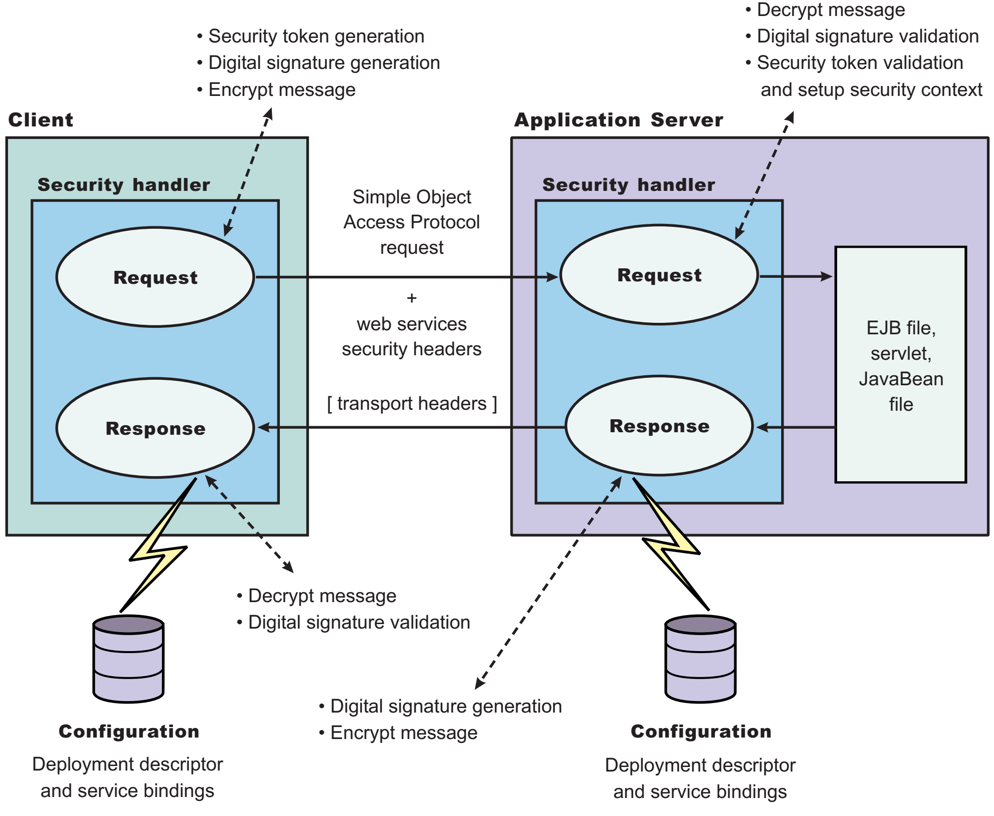



*Figure 7. Web services security message interpretation*

The Web services security run time enforces Web services security based on the defined security constraints in the deployment descriptor and binding files. Web services security has the following four points where it intercepts the message and acts on the security constraints defined:

| Message points | Description |
|---|---|
| Request sender (defined in the `ibm-webservicesclient-ext.xmi` and `ibm-webservicesclient-bnd.xmi` files) | • Applies the appropriate security constraints to the SOAP message (such as signing or encryption) before the message is sent, generating the time stamp or the required security token. |
| Request receiver (defined in the `ibm-webservices-ext.xmi` and `ibm-webservices-bnd.xmi` files) | • Verifies that the Web services security constraints are met.<br>• Verifies the freshness of the message based on the time stamp. The freshness of the message indicates whether the message complies with predefined time constraints.<br>• Verifies the required signature.<br>• Verifies that the message is encrypted and decrypts the message if encrypted.<br>• Validates the security tokens and sets up the security context for the downstream call. |
| Response sender (defined in the `ibm-webservices-ext.xmi` and `ibm-webservices-bnd.xmi` files) | • Applies the appropriate security constraints to the SOAP message response, like signing the message, encrypting the message, or generating the time stamp. |

| Message points | Description |
| --- | --- |
| Response receiver (defined in the `ibm-webservicesclient-ext.xmi` or `ibm-webservicesclient-bnd.xmi` files) | • Verifies that the Web services security constraints are met.<br>• Verifies the freshness of the message based on the time stamp. The freshness of the message indicates whether the message complies with predefined time constraints.<br>• Verifies the required signature.<br>• Verifies that the message is encrypted and decrypts the message, if encrypted. |

## Web services: Default bindings for the Web services security collection

Use this page to configure the settings for nonce on the server level and to manage the default bindings for the signing information, encryption information, key information, token generators, token consumers, key locators, collection certificate store, trust anchors, trusted ID evaluators, algorithm mappings, and login mappings.

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

To view this administrative console page for the server level, complete the following steps:

1. Click **Servers > Application Servers >***server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.

Read the Web services documentation before you begin defining the default bindings for Web services security.

*Nonce* is a unique cryptographic number that is embedded in a message to help stop repeat, unauthorized attacks of user name tokens. In WebSphere Application Server and WebSphere Application Server Express, you must specify values for the Nonce cache timeout, Nonce maximum age, and Nonce clock skew fields for the server-level.

The default binding configuration provides a central location where reusable binding information is defined. The application binding file can reference the information that is contained in the default binding configuration.

***Nonce cache timeout:***

Specifies the timeout value, in seconds, for the nonce cached on the server. Nonce is a randomly generated value.

The Nonce cache timeout field is required on the server level

If you make changes to the value for the Nonce cache timeout field, you must restart WebSphere Application Server for the changes to take effect.

| | |
| --- | --- |
| **Default** | 600 seconds |
| **Minimum** | 300 seconds |

***Nonce maximum age:***

Specifies the default time, in seconds, before the nonce timestamp expires. Nonce is a randomly generated value.

The maximum value cannot exceed the number of seconds that is specified in the Nonce cache timeout field for the server level.

The Nonce maximum age field is required on the server level.

| | |
|---|---|
| **Default** | 300 seconds |
| **Range** | 300 to the value that is specified, in seconds, in the Nonce cache timeout field. |

***Nonce clock skew:***

Specifies the default clock skew value, in seconds, to consider when WebSphere Application Server checks the timeliness of the message. Nonce is a randomly generated value.

The maximum value cannot exceed the number of seconds that is specified in the Nonce maximum age field.

The Nonce clock skew field is required.

| | |
|---|---|
| **Default** | 0 seconds |
| **Range** | 0 to the value that is specified, in seconds, in the Nonce maximum age field. |

***Distribute nonce caching:***

Enables distributed caching for the nonce value using a Data Replication Service (DRS).

In previous releases of WebSphere Application Server, the nonce value cached locally. By selecting this option, the nonce value is propagated to other servers in your environment. However, the nonce value might be subject to a one-second delay in propagation and subject to any network congestion.

## Usage scenario for propagating security tokens

### Important distinction between Version 5.x and Version 6 applications

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

### A sample scenario

This document describes a usage scenario for Web services security.

In scenario 1, Client 1 invokes Web services 1. Then Web services 1 calls EJB file 2. EJB file 2 calls Web services 3 and Web services 3 calls Web services 4.

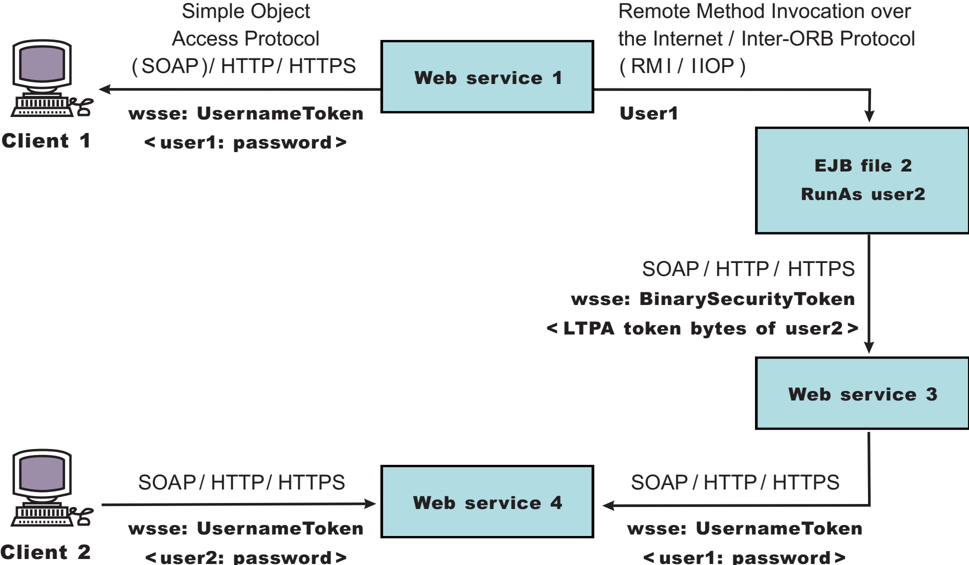



*Figure 8. Propagating security tokens*

The previous scenario shows how to propagate security tokens using Web services security, the security infrastructure of the WebSphere Application Server, and Java 2 Platform, Enterprise Edition (J2EE) security. Web services 1 is configured to accept `<wsse:UsernameToken>` only and use the BasicAuth authentication method. However, Web services 4 is configured to accept either `<wsse:UsernameToken>` using the BasicAuth authentication method or Lightweight Third Party Authentication (LTPA) as `<wsse:BinarySecurityToken>`. The following steps describe the scenario shown in the previous figure:

1. Client 1 sends a SOAP message to Web services 1 with `user1` and `password` in the `<wsse:UsernameToken>` element.
2. The `user1` and `password` values are authenticated by the Web services security run time and set in the current security context as the Java Authentication and Authorization Service (JAAS) Subject.
3. Web services 1 invokes EJB file 2 using the Remote Method Invocation over the Internet Inter-ORB Protocol (RMI/IIOP) protocol.
4. The `user1` identity is propagated to the downstream call.
5. The EJB container of EJB file 2 performs an authorization check against `user1`.
6. EJB file 2 calls Web services 3 and Web services 3 is configured to accept LTPA tokens.
7. The RunAs role of EJB file 2 is set to `user2`.
8. The LTPA CallbackHandler implementation extracts the LTPA token from the current JAAS Subject in the security context and Web services security run time inserts the token as `<wsse:BinarySecurityToken>` in the SOAP header.
9. The Web services security run time in Web services 3 calls the JAAS login configuration to validate the LTPA token and set it in the current security context as the JAAS Subject.
10. Web services 3 is configured to send LTPA security to Web services 4. In this case, assume that the RunAs role is not configured for Web services 3. The LTPA token of `user2` is propagated to Web services 4.
11. Client 2 uses the `<wsse:UsernameToken>` element to propagate the basic authentication data to Web services 4.

Web services security complements the WebSphere Application Server security run time and the J2EE role-based security. This scenario demonstrates how to propagate security tokens across multiple resources such as Web services and EJB files.

## Configurations

The Web services security model used by WebSphere Application Server is the declarative model.

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

No Application Programming Interfaces (APIs) exist in WebSphere Application Server for programmatically interacting with Web services security. However, Service Provider Programming Interfaces (SPIs) are available for extending some security run-time behaviors. You can secure an application with Web services security by defining security constraints in the IBM extension deployment descriptors and in IBM extension bindings.

The development life cycle of a Web services security-enabled application is similar to the Java 2 Platform, Enterprise Edition (J2EE) model. See the following figure for more details.

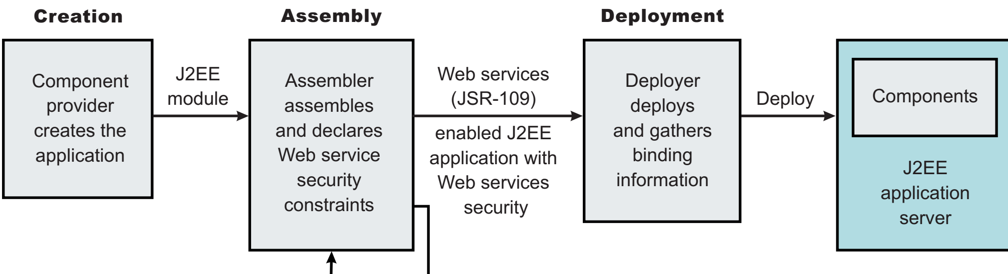



*Figure 9. Application development life cycle*

The Web services security constraints are defined by the assembler during the application assembly phase if the J2EE application is Web services-enabled. Create, define, and edit the Web services security constraints with an assembly tool. For more information, see Assembly tools.

**Web services security constraints**

The security constraints for Web services security are specified in the IBM deployment descriptor extension for Web services. The assembler defines these constraints during the application assembly phase, if the J2EE application is Web services enabled. Define the Web services security constraints using an assembly tool. For more information, see Assembling applications.

The Web services security run time acts on the constraints to enforce Web services security for the SOAP message. The scope of the IBM deployment descriptor extension is at the EJB module or Web module level. There also are bindings associated with each of the following IBM deployment descriptor extensions:

**Client** (might be either a J2EE client (application client container) or Web services acting as a client)
- `ibm-webservicesclient-ext.xmi`
- `ibm-webservicesclient-bnd.xmi`

**Server**
- `ibm-webservices-ext.xmi`
- `ibm-webservices-bnd.xmi`

The IBM extension deployment descriptor and bindings are associated with each EJB module or Web module. See Figure 2 for more information. If Web services is acting as a client, then it contains the client IBM extension deployment descriptors and bindings in the EJB module or Web module.

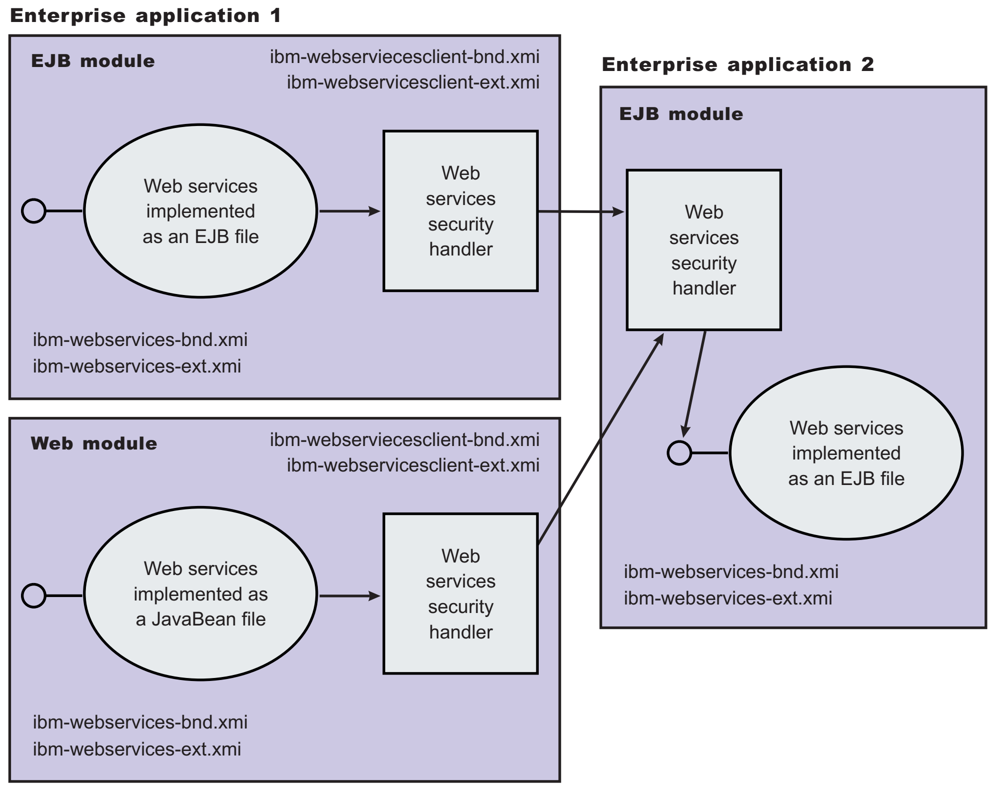



*Figure 10. IBM extension deployment descriptors and bindings*

The Web services security handler acts on the security constraints defined in the IBM extension deployment descriptor and enforces the security constraints accordingly. There are outbound and inbound configurations in both the client and server security constraints.

In a SOAP request, the following message points exist:
- Sender outbound
- Receiver inbound
- Receiver outbound
- Sender inbound

These message points correspond to the following four security constraints:
- Request sender (sender outbound)
- Request receiver (receiver inbound)
- Response sender (receiver outbound)
- Response receiver (sender inbound)

The security constraints of request sender and request receiver must match. Also, the security constraints of the response sender and response receiver must match. For example, if you specify integrity as a

constraint in the request receiver, then you must configure the request sender to have integrity applied to the SOAP message. Otherwise, the request is denied because the SOAP message does not include the integrity specified in the request constraint.

The four security constraints are shown in the following figure of Web services security constraints.

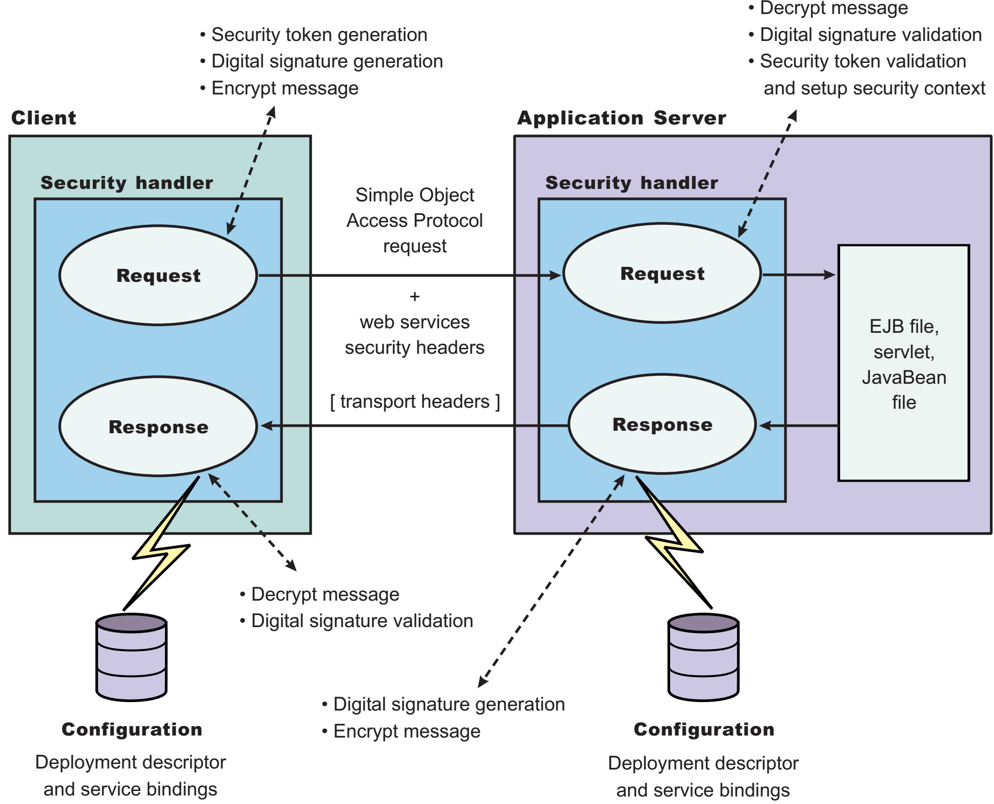



*Figure 11. Web services security constraints*

***Sample configuration:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

WebSphere Application Server provides the following sample key stores for sample configurations. These sample key stores are for testing and sample purposes only. Do not use them a in production environment.

- `{USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-sender.ks`
  - The keystore password is `client`
  - Trusted certificate with alias name, `soapca`
  - Personal certificate with alias name, `soaprequester` and key password `client` issued by intermediary certificate authority `Int CA2`, which is, in turn, issued by `soapca`
- `{USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-receiver.ks`
  - The keystore password is `server`

- Trusted certificate with alias name, `soapca`
- Personal certificate with alias name, `soapprovider` and key password `server`, issued by intermediary certificate authority `Int CA2`, which is, in turn, issued by `soapca`

- `{USER_INSTALL_ROOT}/etc/ws-security/samples/enc-sender.jceks`
  - The keystore password is `storepass`
  - Secret key `CN=Group1`, alias name `Group1`, and key password `keypass`
  - Public key `CN=Bob, O=IBM, C=US`, alias name `bob`, and key password `keypass`
  - Private key `CN=Alice, O=IBM, C=US`, alias name `alice`, and key password `keypass`
- `{USER_INSTALL_ROOT}/etc/ws-security/samples/enc-receiver.jceks`
  - The keystore password is `storepass`
  - Secret key `CN=Group1`, alias name `Group1`, and key password `keypass`
  - Private key `CN=Bob, O=IBM, C=US`, alias name `bob`, and key password `keypass`
  - Public key `CN=Alice, O=IBM, C=US`, alias name `alice`, and key password `keypass`
- `{USER_INSTALL_ROOT}/etc/ws-security/samples/intca2.cer`
  - The intermediary certificate authority is `Int CA2`.

**Default binding (cell and server level)**

WebSphere Application Server provides the following default binding information:

**Trust anchors**

Used to validate the trust of the signer certificate.

- `SampleClientTrustAnchor` is used by the response receiver to validate the signer certificate.
- `SampleServerTrustAnchor` is used by the request receiver to validate the signer certificate.

**Collection Certificate Store**

Used to validate the certificate path.

- `SampleCollectionCertStore` is used by the response receiver and the request receiver to validate the signer certificate path.

**Key Locators**

Used to locate the key for signature, encryption, and decryption.

- `SampleClientSignerKey` is used by the requesting sender to sign the SOAP message. The signing key name is `clientsignerkey`, which can be referenced in the signing information as the signing key name.
- `SampleServerSignerKey` is used by the responding sender to sign the SOAP message. The signing key name is `serversignerkey`, which can be referenced in the signing information as the signing key name.
- `SampleSenderEncryptionKeyLocator` is used by the sender to encrypt the SOAP message. It is configured to use the `{USER_INSTALL_ROOT}/etc/ws-security/samples/enc-sender.jceks` keystore and the `com.ibm.wsspi.wssecurity.config.KeyStoreKeyLocator` keystore key locator.
- `SampleReceiverEncryptionKeyLocator` is used by the receiver to decrypt the encrypted SOAP message. The implementation is configured to use the `{USER_INSTALL_ROOT}/etc/ws-security/samples/enc-receiver.jceks` keystore and the `com.ibm.wsspi.wssecurity.config.KeyStoreKeyLocator` keystore key locator. The implementation is configured for symmetric encryption (DES or TRIPLEDES). However, to use it for asymmetric encryption (RSA), you must add the private key `CN=Bob, O=IBM, C=US`, alias name `bob`, and key password `keypass`.
- `SampleResponseSenderEncryptionKeyLocator` is used by the response sender to encrypt the SOAP response message. It is configured to use the `{USER_INSTALL_ROOT}/etc/ws-security/samples/enc-receiver.jceks` keystore and the `com.ibm.wsspi.wssecurity.config.WSIdKeyStoreMapKeyLocator` key locator. This key locator

maps an authenticated identity (of the current thread) to a public key for encryption. By default, WebSphere Application Server is configured to map to public key `alice`, and you must change WebSphere Application Server to the appropriate user. The `SampleResponseSenderEncryptionKeyLocator` key locator also can set a default key for encryption. By default, this key locator is configured to use public key `alice`.

**Trusted ID Evaluator**

Used to establish trust before asserting to the identity in identity assertion.

`SampleTrustedIDEvaluator` is configured to use the `com.ibm.wsspi.wssecurity.id.TrustedIDEvaluatorImpl` implementation. The default implementation of `com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator` contains a list of trusted identities. The list is defined as properties with `trustedId_*` as the key and the value as the trusted identity. Define this information for the server level in the administration console by completing the following steps:

1. Click **Servers** > **Application Servers** > *server1*.
2. Under Additional Properties, click **Web Services: Default bindings for Web Services Security** > **Trusted ID Evaluators** > *SampleTrustedIDEvaluator*

**Login Mapping**

Used to authenticate the incoming security token in the Web services security SOAP header of a SOAP message.

- The BasicAuth authentication method is used to authenticate user name security token (user name and password).
- The signature authentication method is used to map a distinguished name (DN) into a WebSphere Application Server Java Authentication and Authorization Server (JAAS) Subject.
- The IDAssertion authentication method is used to map a trusted identity into a WebSphere Application Server JAAS Subject for identity assertion.
- The Lightweight Third Party Authentication (LTPA) authentication method is used to validate a LTPA security token.

The previous default bindings for trust anchors, collection certificate stores, and key locators are for testing or sample purpose only. Do not use them for production.

### A sample configuration

The following examples demonstrate what IBM deployment descriptor extensions and bindings can do. The unnecessary information was removed from the examples to improve clarity. Do not copy and paste these examples into your application deployment descriptors or bindings. These examples serve as reference only and are not representative of the recommended configuration.

Use the following tools to create or edit IBM deployment descriptor extensions and bindings:

- Use an assembly tool to create or edit the IBM deployment descriptor extensions.
- Use an assembly tool or the administrative console to create or edit the bindings file.

The following example illustrates a scenario that:

- Signs the SOAP body, time stamp, and security token.
- Encrypts the body content and user name token.
- Sends the user name token (basic authentication data).
- Generates the time stamp for the request.

For the response, the SOAP body and time stamp are signed, the body content is encrypted, and the SOAP message freshness is checked using the time stamp. The freshness of the message indicates whether the message complies with predefined time constraints.

The request sender and the request receiver are a pair. Similarly, the response sender and the response receiver are a pair.

**Tip:** It is recommended that you use the WebSphere Application Server variables for specifying the path to the key stores. In the administrative console, click **Environment > Manage WebSphere Variables**. These variables often help with platform differences such as file system naming conventions. In the following examples, `${USER_INSTALL_ROOT}` is used for specifying the path to the key stores.

**Client-side IBM deployment descriptor extension**

The client-side IBM deployment descriptor extension describes the following constraints:

Request Sender

- Signs the SOAP body, time stamp and security token
- Encrypts the body content and user name token
- Sends the basic authentication token (user name and password)
- Generates the time stamp to expire in three minutes

Response Receiver

- Verifies that the SOAP body and time stamp are signed
- Verifies that the SOAP body content is encrypted
- Verifies that the time stamp is present (also check for message freshness)

**Example 1: Sample client IBM deployment descriptor extension**

The `xmi:id` statements are removed for readability. These statements must be added for this example to work.

**Important:** In the following code sample, lines 2 through 4 were split into three lines due to the width of the printed page.

```
<?xml version="1.0" encoding="UTF-8"?>
<com.ibm.etools.webservice.wscext:WsClientExtension xmi:version="2.0"
  xmlns:xmi="http://www.omg.org/XMI" xmlns:com.ibm.etools.webservice.wscext=
  http://www.ibm.com/websphere/appserver/schemas/5.0.2/wscext.xmi">
  <serviceRefs serviceRefLink="service/myServ">
    <portQnameBindings portQnameLocalNameLink="Port1">
      <clientServiceConfig actorURI="myActorURI">
        <securityRequestSenderServiceConfig actor="myActorURI">
          <integrity>
            <references part="body"/>
            <references part="timestamp"/>
            <references part="securitytoken"/>
          </integrity>
          <confidentiality>
            <confidentialParts part="bodycontent"/>
            <confidentialParts part="usernametoken"/>
          </confidentiality>
          <loginConfig authMethod="BasicAuth"/>
          <addCreatedTimeStamp flag="true" expires="PT3M"/>
        </securityRequestSenderServiceConfig>
        <securityResponseReceiverServiceConfig>
          <requiredIntegrity>
            <references part="body"/>
            <references part="timestamp"/>
          </requiredIntegrity>
```

```
          <requiredConfidentiality>
            <confidentialParts part="bodycontent"/>
          </requiredConfidentiality>
          <addReceivedTimeStamp flag="true"/>
        </securityResponseReceiverServiceConfig>
      </clientServiceConfig>
    </portQnameBindings>
  </serviceRefs>
</com.ibm.etools.webservice.wscext:WsClientExtension>
```

### Client-side IBM extension bindings

Example 2 shows the client-side IBM extension binding for the security constraints described previously in the discussion on client-side IBM deployment descriptor extensions.

The signer key and encryption (decryption) key for the message can be obtained from the keystore key locator implementation (`com.ibm.wsspi.wssecurity.config.KeyStoreKeyLocator`). The signer key is used for encrypting the response. The sample is configured to use the Java Certification Path API to validate the certificate path of the signer of the digital signature. The user name token (basic authentication) data is collected from the standard in (stdin) prompts using one of the default Java Authentication and Authorization Service (JAAS) implementations :`javax.security.auth.callback.CallbackHandler` implementation (`com.ibm.wsspi.wssecurity.auth.callback.StdinPromptCallbackHandler`).

### Example 2: Sample client IBM extension binding

**Important:** In the following code sample, several lines were split into multiple lines due to the width of the printed page. See the close bracket for an indication of where each line of code ends.

```
<?xml version="1.0" encoding="UTF-8"?>
<com.ibm.etools.webservice.wscbnd:ClientBinding xmi:version="2.0"
  xmlns:xmi="http://www.omg.org/XMI"
  xmlns:com.ibm.etools.webservice.wscbnd=
  "http://www.ibm.com/websphere/appserver/schemas/5.0.2/wscbnd.xmi">
  <serviceRefs serviceRefLink="service/MyServ">
    <portQnameBindings portQnameLocalNameLink="Port1">
      <securityRequestSenderBindingConfig>
        <signingInfo>
          <signatureMethod algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
          <signingKey name="clientsignerkey" locatorRef="SampleClientSignerKey"/>
          <canonicalizationMethod algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
          <digestMethod algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        </signingInfo>
        <keyLocators name="SampleClientSignerKey" classname=
        "com.ibm.wsspi.wssecurity.config.KeyStoreKeyLocator">
          <keyStore storepass="{xor}PDM2OjEr" path=
          "${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-sender.ks" type="JKS"/>
          <keys alias="soaprequester" keypass="{xor}PDM2OjEr" name="clientsignerkey"/>
        </keyLocators>
        <encryptionInfo name="EncInfo1">
          <encryptionKey name="CN=Bob, O=IBM, C=US" locatorRef=
          "SampleSenderEncryptionKeyLocator"/>
          <encryptionMethod algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
          <keyEncryptionMethod algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
        </encryptionInfo>
        <keyLocators name="SampleSenderEncryptionKeyLocator" classname=
        "com.ibm.wsspi.wssecurity.config.KeyStoreKeyLocator">
          <keyStore storepass="{xor}LCswLTovPiws" path=
          "${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-sender.jceks" type="JCEKS"/>
          <keys alias="Group1" keypass="{xor}NDomLz4sLA==" name="CN=Group1"/>
        </keyLocators>
```

```
        <loginBinding authMethod="BasicAuth" callbackHandler=
        "com.ibm.wsspi.wssecurity.auth.callback.StdinPromptCallbackHandler"/>
      </securityRequestSenderBindingConfig>
      <securityResponseReceiverBindingConfig>
        <signingInfos>
          <signatureMethod algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
          <certPathSettings>
            <trustAnchorRef ref="SampleClientTrustAnchor"/>
            <certStoreRef ref="SampleCollectionCertStore"/>
          </certPathSettings>
          <canonicalizationMethod algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
          <digestMethod algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        </signingInfos>
        <trustAnchors name="SampleClientTrustAnchor">
          <keyStore storepass="{xor}PDM2OjEr" path=
          "${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-sender.ks" type="JKS"/>
        </trustAnchors>
        <certStoreList>
          <collectionCertStores provider="IBMCertPath" name="SampleCollectionCertStore">
            <x509Certificates path="${USER_INSTALL_ROOT}/etc/ws-security/samples/intca2.cer"/>
          </collectionCertStores>
        </certStoreList>
        <encryptionInfos name="EncInfo2">
          <encryptionKey locatorRef="SampleReceiverEncryptionKeyLocator"/>
          <encryptionMethod algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
          <keyEncryptionMethod algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
        </encryptionInfos>
        <keyLocators name="SampleReceiverEncryptionKeyLocator" classname=
        "com.ibm.wsspi.wssecurity.config.KeyStoreKeyLocator">
          <keyStore storepass="{xor}PDM2OjEr" path=
          "${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-sender.ks" type="JKS"/>
          <keys alias="soaprequester" keypass="{xor}PDM2OjEr" name="clientsignerkey"/>
        </keyLocators>
      </securityResponseReceiverBindingConfig>
    </portQnameBindings>
  </serviceRefs>
</com.ibm.etools.webservice.wscbnd:ClientBinding>
```

**Server-side IBM deployment descriptor extension**

The client-side IBM deployment descriptor extension describes the following constraints:

Request Receiver (`ibm-webservices-ext.xmi` and `ibm-webservices-bnd.xmi`)

- Verifies that the SOAP body, time stamp, and security token are signed.
- Verifies that the SOAP body content and user name token are encrypted.
- Verifies that the basic authentication token (user name and password) is in the Web services security SOAP header.
- Verifies that the time stamp is present (also check for message freshness). The freshness of the message indicates whether the message complies with predefined time constraints.

Response Sender (`ibm-webservices-ext.xmi` and `ibm-webservices-bnd.xmi`)

- Signs the SOAP body and time stamp
- Encrypts the SOAP body content
- Generates the time stamp to expire in 3 minutes

**Example 3: Sample server IBM deployment descriptor extension**

**Important:** In the following code sample, several lines were split into multiple lines due to the width of the printed page. See the close bracket for an indication of where each line of code ends.

```
<?xml version="1.0" encoding="UTF-8"?>
<com.ibm.etools.webservice.wsext:WsExtension xmi:version="2.0"
  xmlns:xmi="http://www.omg.org/XMI"
  xmlns:com.ibm.etools.webservice.wsext=
http://www.ibm.com/websphere/appserver/schemas/5.0.2/wsext.xmi">
  <wsDescExt wsDescNameLink="MyServ">
    <pcBinding pcNameLink="Port1">
      <serverServiceConfig actorURI="myActorURI">
        <securityRequestReceiverServiceConfig>
          <requiredIntegrity>
            <references part="body"/>
            <references part="timestamp"/>
            <references part="securitytoken"/>
          </requiredIntegrity>
          <requiredConfidentiality">
            <confidentialParts part="bodycontent"/>
            <confidentialParts part="usernametoken"/>
          </requiredConfidentiality>
          <loginConfig>
            <authMethods text="BasicAuth"/>
          </loginConfig>
          <addReceivedTimestamp flag="true"/>
        </securityRequestReceiverServiceConfig>
        <securityResponseSenderServiceConfig actor="myActorURI">
          <integrity>
            <references part="body"/>
            <references part="timestamp"/>
          </integrity>
          <confidentiality>
            <confidentialParts part="bodycontent"/>
          </confidentiality>
          <addCreatedTimestamp flag="true" expires="PT3M"/>
        </securityResponseSenderServiceConfig>
      </serverServiceConfig>
    </pcBinding>
  </wsDescExt>
</com.ibm.etools.webservice.wsext:WsExtension>
```

### Server-side IBM extension bindings

The following binding information reuses some of the default binding information defined either at the server level or the cell level, which depends upon the installation. For example, request receiver is referencing the `SampleCollectionCertStore` certification store and the `SampleServerTrustAnchor` trust store is defined in the default binding. However, the encryption information in the request receiver is referencing a `SampleReceiverEncryptionKeyLocator` key locator defined in the application-level binding (the same `ibm-webservices-bnd.xmi` file). The response sender is configured to use the signer key of the digital signature of the request to encrypt the response using one of the default key locator (`com.ibm.wsspi.wssecurity.config.CertInRequestKeyLocator`) implementations.

### Example 4: Sample server IBM extension binding

```
<?xml version="1.0" encoding="UTF-8"?>
<com.ibm.etools.webservice.wsbnd:WSBinding xmi:version="2.0"
  xmlns:xmi="http://www.omg.org/XMI"
  xmlns:com.ibm.etools.webservice.wsbnd=
  http://www.ibm.com/websphere/appserver/schemas/5.0.2/wsbnd.xmi">
  <wsdescBindings wsDescNameLink="MyServ">
    <pcBindings pcNameLink="Port1" scope="Session">
      <securityRequestReceiverBindingConfig>
        <signingInfos>
          <signatureMethod algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
          <certPathSettings>
```

```
        <trustAnchorRef ref="SampleServerTrustAnchor"/>
        <certStoreRef ref="SampleCollectionCertStore"/>
      </certPathSettings>
      <canonicalizationMethod algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
      <digestMethod algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
    </signingInfos>
    <encryptionInfos name="EncInfo1">
      <encryptionKey locatorRef="SampleReceiverEncryptionKeyLocator"/>
      <encryptionMethod algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
      <keyEncryptionMethod algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
    </encryptionInfos>
    <keyLocators name="SampleReceiverEncryptionKeyLocator" classname=
     "com.ibm.wsspi.wssecurity.config.KeyStoreKeyLocator">
      <keyStore storepass="{xor}LCswLTovPiws" path="${USER_INSTALL_ROOT}/
       etc/ws-security/samples/enc-receiver.jceks" type="JCEKS"/>
      <keys alias="Group1" keypass="{xor}NDomLz4sLA==" name="CN=Group1"/>
      <keys alias="bob" keypass="{xor}NDomLz4sLA==" name="CN=Bob, O=IBM, C=US"/>
    </keyLocators>
  </securityRequestReceiverBindingConfig>
  <securityResponseSenderBindingConfig>
    <signingInfo>
      <signatureMethod algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      <signingKey name="serversignerkey" locatorRef="SampleServerSignerKey"/>
      <canonicalizationMethod algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
      <digestMethod algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
    </signingInfo>
    <encryptionInfo  name="EncInfo2">
      <encryptionKey locatorRef="SignerKeyLocator"/>
      <encryptionMethod algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
      <keyEncryptionMethod algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
    </encryptionInfo>
    <keyLocators name="SignerKeyLocator" classname=
     "com.ibm.wsspi.wssecurity.config.CertInRequestKeyLocator"/>
  </securityResponseSenderBindingConfig>
 </pcBindings>
</wsdescBindings>
<routerModules transport="http" name="StockQuote.war"/>
</com.ibm.etools.webservice.wsbnd:WSBinding>
```

## Authentication method overview

The Web services security implementation for WebSphere Application Server supports the following authentication methods: BasicAuth, Lightweight Third Party Authentication (LTPA), digital signature, and identity assertion.

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

When the WebSphere Application Server is configured to use the BasicAuth authentication method, the sender attaches the LTPA token as a BinarySecurityToken from the current security context or from basic authentication data configuration in the binding file in the SOAP message header. The Web services security message receiver authenticates the sender by validating the user name and password against the configured user registry. With the LTPA method, the sender attaches the LTPA BinarySecurityToken it previously received in the SOAP message header. The receiver authenticates the sender by validating the LTPA token and the token expiration time. With the Digital Signature authentication method, the sender attaches a BinarySecurityToken from a X509 certificate to the Web services security message header along with a digital signature of the message body, time stamp, security token, or any combination of the three. The receiver authenticates the sender by verifying the validity of the X.509 certificate and the digital signature using the public key from the verified certificate.

The identity assertion authentication method is different from the other three authentication methods. This method establishes the security credential of the sender based on the trust relationship. You can use the identity assertion authentication method, for example, when an intermediary server must invoke a service from a downstream server on behalf of the client, but does not have the client authentication information. The intermediary server might establish a trust relationship with the downstream server and then assert the client identity to the same downstream server.

Web Services Security supports the following trust modes:

- BasicAuth
- Digital signature
- Presumed trust

When you use the BasicAuth and digital signature trust modes, the intermediary server passes its own authentication information to the downstream server for authentication. The presumed trust mode establishes a trust relationship using some external mechanism. For example, the intermediary server might pass SOAP messages through a Secure Socket Layers (SSL) connection with the downstream server and transport layer client certificate authentication.

The Web services security implementation for WebSphere Application Server validates the trust relationship by following this procedure:

1. The downstream server validates the authentication information of the intermediary server.
2. The downstream server verifies whether the authenticated intermediary server is authorized for identity assertion. For example, the intermediary server must be in the trust list for the downstream server.

The client identity might be represented by a name string, a distinguished name (DN), or an X.509 certificate. The client identity is attached in the Web services security message in a UsernameToken with just a user name, DN, or in a BinarySecurityToken of a certificate. The following table summarizes the type of security token that is required for each authentication method.

*Table 17. Authentication methods and their security tokens*

| Authentication method | Security token |
|---|---|
| BasicAuth | BasicAuth requires `<wsse:UsernameToken>` with `<wsse:Username>` and `<wsse:Password>`. |
| Signature | Signature requires `<ds:Signature>` and `<wsse:BinarySecurityToken>`. |
| IDAssertion | IDAssertion requires `<wsse:UsernameToken>` with `<wsse:Username>` or `<wsse:BinarySecurityToken>` with a X.509 certificate for client identity depending on `<idType>`. This method also requires other security tokens according to the `<trustMode>`:<br>• If the `<trustMode>` is BasicAuth, IDAssertion requires `<wsse:UsernameToken>` with `<wsse:Username>` and `<wsse:Password>`.<br>• If the `<trustMode>` is Signature, IDAssertion requires `<wsse:BinarySecurityToken>`. |
| LTPA | LTPA requires `<wsse:BinarySecurityToken>` with an LTPA token. |

A Web service can support multiple authentication methods simultaneously. The receiver side of the Web services deployment descriptor can specify all the authentication methods that are supported in the `ibm-webservices-ext.xmi` XML file. The Web services receiver-side, as shown in the following example, is configured to accept all the authentication methods described previously:

```
<loginConfig xmi:id="LoginConfig_1052760331326">
      <authMethods xmi:id="AuthMethod_1052760331326" text="BasicAuth"/>
      <authMethods xmi:id="AuthMethod_1052760331327" text="IDAssertion"/>
      <authMethods xmi:id="AuthMethod_1052760331336" text="Signature"/>
      <authMethods xmi:id="AuthMethod_1052760331337" text="LTPA"/>
</loginConfig>
<idAssertion xmi:id="IDAssertion_1052760331336" idType="Username" trustMode="Signature"/>
```

You can define only one authentication method in the sender-side Web services deployment descriptor. A Web service client can use any of the authentication methods that are supported by the particular Web services application. The following example illustrates an identity assertion authentication method configuration in the `ibm-webservicesclient-ext.xmi` deployment descriptor extension of the Web service client:

```
<loginConfig xmi:id="LoginConfig_1051555852697">
      <authMethods xmi:id="AuthMethod_1051555852698" text="IDAssertion"/>
</loginConfig>
<idAssertion xmi:id="IDAssertion_1051555852697" idType="Username" trustMode="Signature"/>
```

As shown in the previous example, the client identity type is `Username` and the trust mode is digital signature.

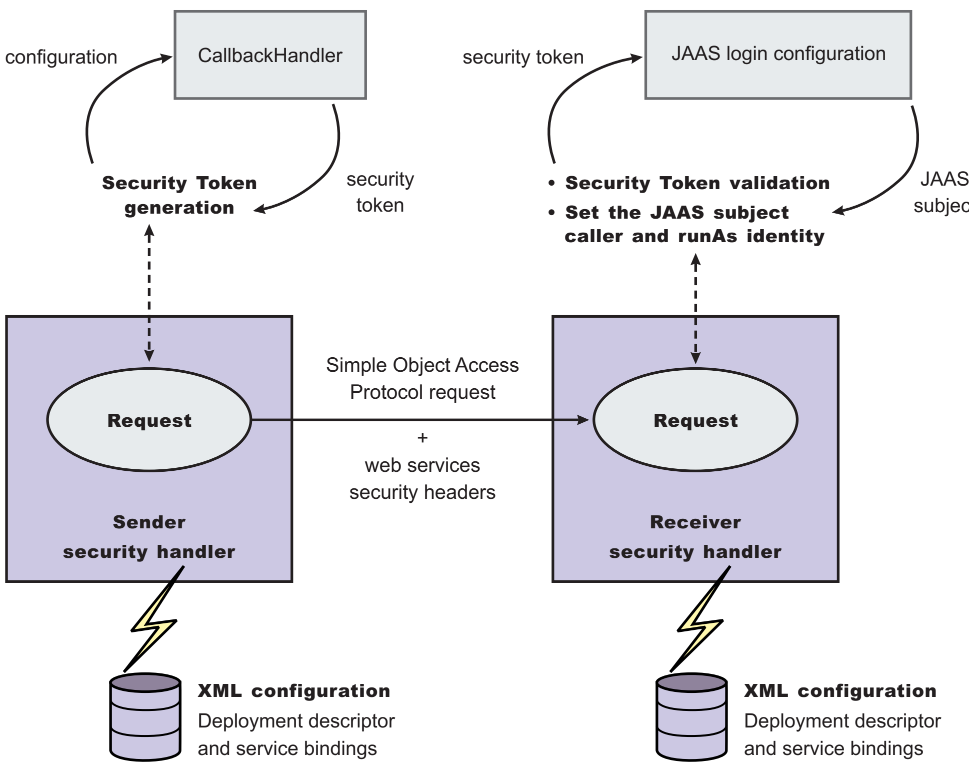



*Figure 12. Security token generation and validation*

The sender security handler invokes the `handle()` method of an implementation of the `javax.security.auth.callback.CallbackHandler` interface. The `javax.security.auth.callback.CallbackHandler` interface creates the security token and passes it back to the sender security handler. The sender security handler constructs the security token based on the authentication information in the callback array and inserts the security token into the Web services security message header.

The receiver security handler compares the token type in the message header with the expected token types configured in the deployment descriptor. If none of the expected token types are found in the Web services security header of the SOAP message, the request is rejected with a SOAP fault exception. Otherwise, the token type is used to map to a Java Authentication and Authorization Service (JAAS) login configuration for validating the token. If the authentication is successful, a JAAS Subject is created and associated with the running thread. Otherwise, the request is rejected with a SOAP fault exception.

## XML digital signature

XML-Signature Syntax and Processing (XML signature) is a specification that defines XML syntax and processing rules to sign and verify digital signatures for digital content. The specification was developed jointly by the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF).

XML signature does not introduce new cryptographic algorithms. WebSphere Application Server uses XML signature with existing algorithms such as RSA, HMAC, and SHA1. XML signature defines many methods for describing key information and enables the definition of a new method.

XML canonicalization (c14n) is often needed when you use XML signature. Information can be represented in various ways within serialized XML documents. For example, although their octet representations are different, the following examples are identical:

- `<person first="John" last="Smith"/>`
- `<person last="Smith" first="John"></person>`

C14n is a process used to canonicalize XML information. Select an appropriate c14n algorithm because the information that is canonicalized is dependent upon this algorithm. One of the major c14n algorithms, Exclusive XML Canonicalization, canonicalizes the character encoding scheme, attribute order, namespace declarations, and so on. The algorithm does not canonicalize white space outside tags, namespace prefixes, or data type representation.

### XML signature in the Web Services Security-Core specification

The Web Services Security-Core (WSS-Core) specification defines a standard way for Simple Object Access Protocol (SOAP) messages to incorporate an XML signature. You can use almost all of the XML signature features in WSS-Core except enveloped signature and enveloping signature. However, WSS-Core has some recommendations such as exclusive canonicalization for the c14n algorithm and some additional features such as `SecurityTokenReference` and `KeyIdentifier`. The `KeyIdentifier` is the value of the SubjectKeyIdentifier field within the X.509 certificate. For more information on the `KeyIdentifier`, see ″Reference to a Subject Key Identifier″ within the OASIS Web Services Security X.509 Certificate Token Profile documentation.

By including XML signature in SOAP messages, the following are realized:

**Message integrity**
A message receiver can confirm that attackers or accidents have not altered parts of the message after these parts are signed by a key.

**Authentication**
You can assume that a valid signature is *proof of possession*. A message with a digital certificate issued by a certificate authority and a signature in the message that is validated successfully by a public key in the certificate, is proof that the signer has the corresponding private key. The receiver can authenticate the signer by checking the trustworthiness of the certificate.

### XML signature in the current implementation

XML signature is supported in Web services security, however, an application programming interface (API) is not available. The current implementation has many hardcoded behaviors and has some user-operable configuration items. To configure the client for digital signature, see Configuring the client for response

digital signature verification: Verifying the message parts. To configure the server for digital signature, see Configuring the server for request digital signature verification: Verifying the message parts.

**Security considerations**

In a replay attack, an attacker taps the lines, receives a signed message, and then returns the message to the receiver. In this case, the receiver receives the same message twice and might process both of them if the signatures are valid. Processing both messages can cause damage to the receiver if the message is a claim for money. If you have the signed generation time stamp and the signed expiration time in a message replay, attacks might be reduced. However, this is not a complete solution. A message must have a nonce value to prevent these attacks and the receiver must reject a message that contains a processed nonce. The current implementation does not provide a standard way to generate and check nonces in messages. In WebSphere Application Server, Version 5.1, nonce is supported in username tokens only. The username token profile contains concrete nonce usage scenarios for username tokens. Applications handle nonces (such as serial numbers) and they need to be signed.

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

***Signing parameter configuration settings:***

Use this page to configure new signing parameters.

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

The specifications that are listed on this page for the signature method, digest method, and canonicalization method are located in the World Wide Web Consortium (W3C) document entitled, *XML Signature Syntax and Specification: W3C Recommendation 12 Feb 2002*.

To view this administrative console page, complete the following steps:
1. Click **Enterprise Applications >***application_name*.
2. Under Related items, click **EJB modules** or **Web Modules >** *URI_file_name*
3. **5.x application** Under Additional properties, you can access the signing information for the following bindings:
   a. For the Request sender binding, click **Web services: Client security bindings**. Under Request sender binding, click **Edit**. Under Additional properties, click **Signing information**.
   b. For the Response sender binding, click **Web services: Server security bindings**. Under Response sender binding, click **Edit**. Under Additional properties, click **Signing information**.
4. In the Request Sender Binding column, click **Edit > Signing Information**.

If the signing information is not available, select **None**.

If the signing information is available, select **Dedicated Signing Information** and specify the configuration in the following fields:

*Signature method:*

Specifies the algorithm Uniform Resource Identifiers (URI) of the signature method.

The following algorithms are supported:

- http://www.w3.org/2000/09/xmldsig#rsa-sha1
- http://www.w3.org/2000/09/xmldsig#dsa-sha1
- http://www.w3.org/2000/09/xmldsig#hmac-sha1

**You can also add custom algorithms.**

*Digest method:*

Specifies the algorithm URI of the digest method.

WebSphere Application Server supports the http://www.w3.org/2000/09/xmldsig#sha1 algorithm.

*Canonicalization method:*

Specifies the algorithm URI of the canonicalization method.

The following algorithms are supported:
- http://www.w3.org/2001/10/xml-exc-c14n#
- http://www.w3.org/2001/10/xml-exc-c14n#WithComments
- http://www.w3.org/TR/2001/REC-xml-c14n-20010315
- http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments

*Key name:*

Specifies the name of the key object found in the keystore file.

*Key locator reference:*

Specifies the name used to reference the key locator

You can configure these key locator reference options on the server level and the application level. The configurations that are listed in the field are a combination of the configurations on these two levels.

You can specify a key locator configuration for the following bindings on the following levels:

| Binding name | Cell level, server level, or application level | Path |
|---|---|---|
| N/A | Server level | 1. Click **Servers > Application servers >***server_name*.<br>2. Under Security, click **Web services: Default bindings for Web services security**.<br>3. Under Additional properties, click **Key locators**. |
| Request sender | Application level | 1. Click **Applications > Enterprise applications** > *application_name*.<br>2. Under Related items, click **EJB modules** > *URI_name*.<br>3. Click **Web services: Client security bindings**.<br>4. Under Request sender binding, click **Edit**.<br>5. Under Additional properties, click **Key locators**. |

| Binding name | Cell level, server level, or application level | Path |
| --- | --- | --- |
| Request receiver | Application level | 1. Click **Applications > Enterprise applications** > *application_name*. 2. Under Related items, click **EJB modules** > *URI_name*. 3. Click **Web services: Server security bindings**. 4. Under Request receiver binding, click **Edit**. 5. Under Additional properties, click **Key locators**. |
| Response sender | Application level | 1. Click **Applications > Enterprise applications** > *application_name*. 2. Under Related items, click **EJB modules** > *URI_name*. 3. Click **Web services: Server security bindings**. 4. Under Response sender binding, click **Edit**. 5. Under Additional properties, click **Key locators**. |
| Response receiver | Application level | 1. Click **Applications > Enterprise applications** > *application_name*. 2. Under Related items, click **EJB modules** > *URI_name*. 3. Click **Web services: Client security bindings**. 4. Under Response receiver binding, click **Edit**. 5. Under Additional properties, click **Key locators**. |

## Securing Web services for version 5.x applications using XML digital signature

### Important distinction between Version 5.x and Version 6 applications

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

WebSphere Application Server provides several different methods to secure your Web services; Extensible Markup Language (XML) digital signature is one of these methods. You might secure your Web services using any of the following methods:

- XML digital signature
- XML encryption
- Basicauth authentication
- Identity assertion authentication
- Signature authentication
- Pluggable token

XML digital signature provides both message integrity and authentication capabilities when it is used with SOAP messages. A message receiver can verify that attackers or accidents have not altered parts of the message after the message was signed by a key. If a message has a digital certificate issued by a

certificate authority (CA) and a signature in the message is validated successfully by a public key in the certificate, it is proof that the signer has the corresponding private key. To use XML digital signature to secure Web services, complete the following steps:

1. Define the security constraints or extensions. To configure the security constraints, you must use and assembly tool. For more information, see Assembly tools.
   a. Configure the client to digitally sign a message request. To configure the client, complete the following steps to specify which parts of the SOAP message to digitally sign and define the method used to digitally sign the message. The client in these steps is the request sender.
      1) Specify the message parts by following the steps found in Configuring the client for request signing: digitally signing message parts.
      2) Select the method used to digitally sign the request message. You can select the digital signature method by following the steps in Configuring the client for request signing: choosing the digital signature method.
   b. Configure the server to verify the digital signature that is used in the message request. To configure the server, you must specify which parts of the SOAP message, sent by the request sender, contain digitally signed information and which method was used to digitally sign the message. The settings chosen for the request receiver, or the server in this step, must match the settings chosen for the request sender in the previous step.
      1) Define the message parts by following the steps found in Configuring the server for request digital signature verification: verifying message parts.
      2) Select the same method used by the request sender to digitally sign the message. You can select the digital signature method by following the steps in Configuring the server for request digital signature verification: choosing the verification method
   c. Configure the server to digitally sign a message response. To configure the server, complete the following steps to specify which parts of the SOAP message to digitally sign and define the method used to digitally sign the message. The sender in these steps is the response sender.
      1) Specify which message parts to digitally sign by following the steps found in Configuring the server for response signing: digitally signing message parts.
      2) Select the method used to digitally sign the response message. You can select the digital signature method by following the steps in Configuring the server for response signing: choosing the digital signature method
   d. Configure the client to verify the digital signature that is used in the message response. To configure the client, you must specify which parts of the SOAP message sent by the response sender contain digitally signed information and which method was used to digitally sign the message. The settings chosen for the response receiver, or client in this step, must match the settings chosen for the response sender in the previous step.
      1) Define the message parts by following the steps found in Configuring the client for response digital signature verification: verifying message parts
      2) Select the same method used by the response sender to digitally sign the message. You can select the digital signature method by following the steps in Configuring the client for response digital signature verification: choosing the verification method
2. Define the client security bindings. To configure the client security bindings, complete the steps in either of the following topics:
   - Configuring the client security bindings using the Application Server Toolkit
   - Configuring the client security bindings using the administrative console
3. Define the server security bindings. To configure the server security bindings, complete the steps in either of the following topics:
   - Configuring the server security bindings using the Application Server Toolkit
   - Configuring the server security bindings using the administrative console

After completing these steps, you have secured your Web services using XML digital signature.

***Default configuration for WebSphere Application Server:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

In the WebSphere Application Server, each application server has a copy of the `ws-security.xml` file, which defines the default binding information for Web services security. The following list contains the defaults defined in the `ws-security.xml` file:

**Trust anchors**
Identifies the trusted root certificates for signature verification.

**Collection certificate stores**
Contains certificate revocation lists (CRLs) and nontrusted certificates for verification.

**Key locators**
Locates the keys for digital signature and encryption.

**Trusted ID evaluators**
Evaluates the trust of the received identity before identity assertion.

**Login mappings**
Contains the Java Authentication and Authorization Service (JAAS) configurations for AuthMethod token validation.

If the Web services security constraints specified in the deployment descriptors and the required bindings are not defined in the bindings file, the default constraints in the `ws-security.xml` file are used.

When you use the **addNode** command, the `ws-security.xml` file is added with the server configuration to the new cell. The following figure shows the activity when you use the **addNode** command.

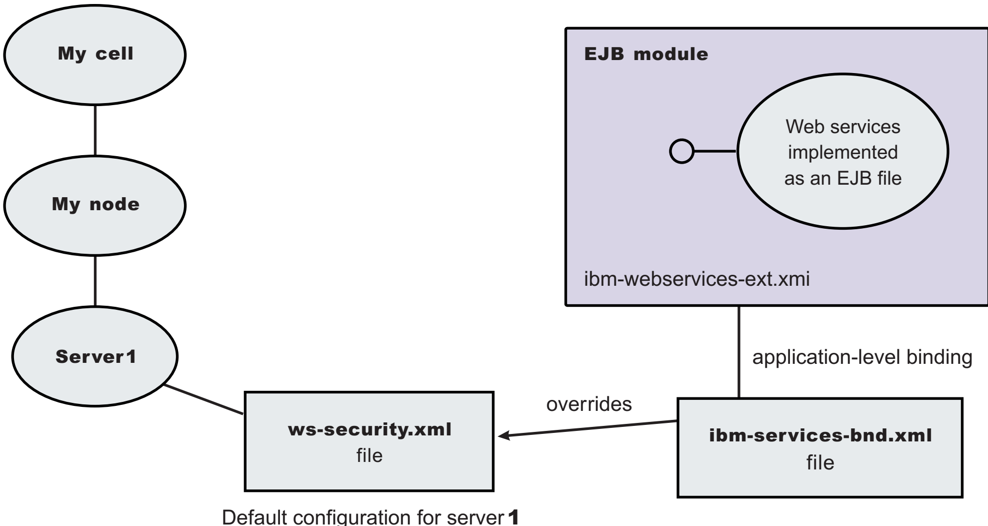



*Figure 13. Configuration when using the* **addNode** *command*

***Default binding:***

The default binding information is defined in the `ws-security.xml` file and can be administered by either the administrative console or by scripting.

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

Certain applications can share certain binding information. This information includes truststores, keystores, and authentication methods (token validation). WebSphere Application Server provides support for default binding information. Administrators can define binding information at:

- The server level

Applications can refer to this binding information.

You can define the following binding information in the `ws-security.xml` file:

### Trust anchors (truststore)

- *Trust anchors* contain key store configuration information that has the root-trusted certificates. Trust anchors are used for certificate path validation of the incoming X.509-formatted security tokens.
- The Trust Anchor Name is used in the binding file (`ibm-webservices-bnd.xmi` and `ibm-webservicesclient-bnd-xmi` when Web services is running as a client) to refer to the trust anchor defined in the default binding information. The trust anchor name must be unique in the trust anchor collection.

### Collection certificate store

- The *collection certificate store* specifies a list of untrusted, intermediate certificates and is used for certificate path validation of incoming X.509-formatted security tokens. The default provider is IBMCertPath.
- The Certificate Store Name is used in the binding file (`ibm-webservices-bnd.xmi` and `ibm-webservicesclient-bnd-xmi` when Web services is running as a client) to refer to the certificate store defined in the default binding information. The Certificate Store Name must be unique to the collection certificate store collection.

### Key locators

- *Key locators* specify implementation of the com.ibm.wsspi.wssecurity.config.KeyLocator interface. This interface is used to retrieve keys for signature or encryption. Customer implementations can extend the key locator interface to retrieve keys using other methods. WebSphere Application Server provides implementations to retrieve a key from the key store, map an authenticated identity to a key in the key store, or retrieve a key from the signer certificate (mapping and retrieving actions are used for encrypting the response).
- The Key Locator Name is used in the binding file (`ibm-webservices-bnd.xmi` and `ibm-webservicesclient-bnd-xmi` when Web services is running as a client) to refer to the key locator defined in the default binding information. The Key Locator Name must be unique to the key locators collection in the default binding information.

### Trusted ID evaluators

- *Trusted ID evaluators* are an implementation of the com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator interface. This interface is used to make sure the identity (ID)-asserting authority is trusted. Additionally, you can extend the trusted identity evaluator to validate the trust. WebSphere Application Server provides a default implementation for validating trust based on a predefined list of identities.
- The Trusted ID Evaluator Name is used in the binding file (`ibm-webservices-bnd.xmi`) to refer to the trusted identity evaluator defined in the default binding information. The Trusted ID Evaluator Name must be unique to the Trusted ID Evaluator collection.

**Login mappings**

- *Login mappings* define the mapping of the authentication method to the Java Authentication and Authorization Service (JAAS) login configuration. The mappings are used to authenticate the incoming security token embedded in the Web services security Simple Object Access Protocol (SOAP) message header. The JAAS login configuration is defined in the administrative console under **Security > JAAS Configuration > Application Logins**.
- WebSphere Application Server defines the following authentication methods:

  **BasicAuth**
  Authenticates user name and password.

  **Signature**
  Maps the subject distinguished name (DN) in the certificate to a WebSphere Application Server credential.

  **IDAssertion**
  Maps the identity to a WebSphere Application Server credential.

  **LTPA** Authenticates a Lightweight Third Party Authentication (LTPA) token.

  After identity authentication, the associated credential is used in the downstream call.
- This method can be extended to authenticate custom security tokens by providing a custom JAAS login configuration and by using the com.ibm.wsspi.wssecurity.auth.module.WSSecurityMappingModule to create the principal and credential required by WebSphere Application Server.
- If LoginConfig (`AuthMethod`) is defined in the IBM extension deployment descriptor (`ibm-webservices-ext.xmi`), but there are no login mapping bindings (`ibm-webservices-bnd.xmi`) defined for the `AuthMethod`, Web services security run time uses the login mapping defined in the default binding information.

**WebSphere Application Server**

In the WebSphere Application Server, each server has a copy of the `ws-security.xml` file (default binding information for Web services security). There is no cell-level copy of the `ws-security.xml` file, which is only available in the WebSphere Application Server Network Deployment installation. To navigate to the server-level default binding in the administrative console, complete the following steps:

1. Click **Servers > Application Servers >** *server1*.
2. Under Additional Properties, click **Web Services: Default bindings for Web Services Security**.

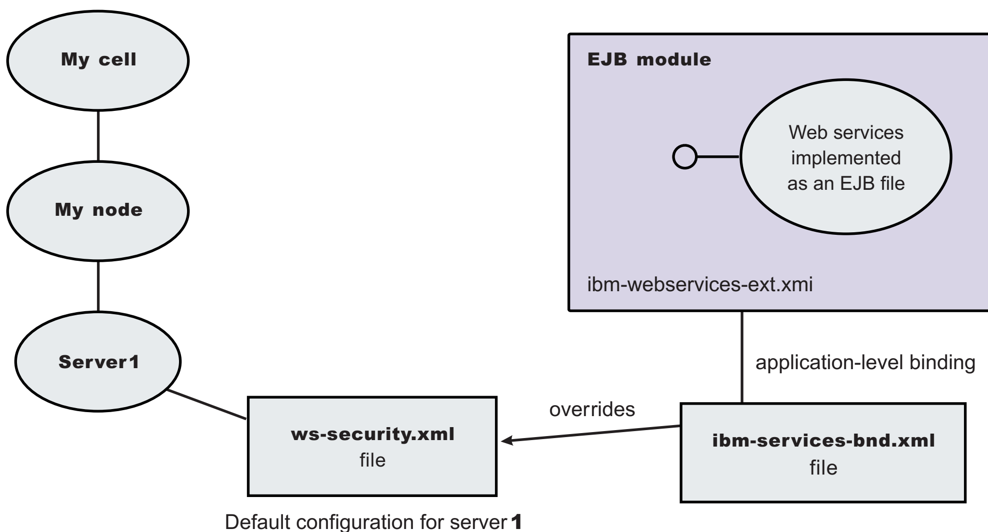



*Figure 14. Web services security application-level bindings and server-level default binding information*

Web services security run time uses the binding information in the application Enterprise JavaBeans (EJB) or Web module binding file (`ibm-webservices-bnd.xmi` or `ibm-webservicesclient-bnd.xmi` if Web services is acting as a client on the server) if the binding information is defined in the application-level binding file. For example, if key locator K1 is defined in both the application-level binding file and the default binding file (`ws-security.xml`), the K1 in the application-level binding file is used.

***Trust anchors:***

A *trust anchor* specifies key stores that contain trusted root certificates that validate the signer certificate.

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

These key stores are used by the request receiver (as defined in the `ibm-webservices-bnd.xmi` file) and the response receiver (as defined in the `ibm-webservicesclient-bnd.xmi` file when Web services is acting as client) to validate the signer certificate of the digital signature. The key stores are critical to the integrity of the digital signature validation. If they are tampered with, the result of the digital signature verification is doubtful and comprised. Therefore, it is recommended that you secure these key stores. The binding configuration specified for the request receiver in the `ibm-webservices-bnd.xmi` file must match the binding configuration for the response receiver in the `ibm-webservicesclient-bnd.xmi` file.

The trust anchor is defined as javax.security.cert.TrustAnchor in the Java CertPath application programming interface (API). The Java CertPath API uses the trust anchor and the certificate store to validate the incoming X.509 certificate that is embedded in the SOAP message.

The Web services security implementation in WebSphere Application Server supports this trust anchor. In WebSphere Application Server, the trust anchor is represented as a Java key store object. The type, path, and password of the key store are passed to the implementation through the administrative console or by scripting.

***Configuring trust anchors using an assembly tool:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

This document describes how to configure trust anchors or trust stores at the application level. It does not describe how to configure trust anchors at the server or cell level. Trust anchors defined at the application level have a higher precedence over trust anchors defined at the server or cell level. You can configure an application-level trust anchor using an assembly tool or the administrative console. This document describes how to configure the application-level trust anchor using an assembly tool. For more information on creating and configuring trust anchors at the server or cell level, see either “Configuring the server security bindings using an assembly tool” on page 875 or “Configuring the server security bindings using the administrative console” on page 877.

A trust anchor specifies key stores that contain trusted root certificates, which validate the signer certificate. These key stores are used by the request receiver (as defined in the `ibm-webservices-bnd.xmi` file) and the response receiver (as defined in the `application-client.xml` file when Web services is acting as client) to validate the signer certificate of the digital signature. The key stores are critical to the integrity of the digital signature validation. If they are tampered with, the result of the digital signature verification is doubtful and comprised. Therefore, it is recommended that you secure these key stores. The binding configuration specified for the request receiver in the `ibm-webservices-bnd.xmi` file must match the binding configuration for the response receiver in the `application-client.xml` file.

Complete the following steps to configure trust anchors using an assembly tool.

1. Configure an assembly tool to work with a Java 2 Platform, Enterprise Edition (J2EE) enterprise application. For more information, see Assembling applications
2. Create a Web services-enabled J2EE enterprise application. If you have not created a Web services-enabled J2EE enterprise application, see Developing Web services applications. Also, see either “Configuring the server security bindings using an assembly tool” on page 875 or “Configuring the server security bindings using the administrative console” on page 877 for an introduction on how to manage Web services security binding information on the server.
3. Configure the client-side response receiver, which is defined in the `ibm-webservicesclient-bnd.xmi` bindings extensions file.
   a. Use an assembly tool to import your J2EE application.
   b. Click **Windows > Open Perspective > Other > J2EE**.
   c. Click **Application Client projects >***application_name* **> appClientModule > META-INF**
   d. Right-click the `application-client.xml` file, select **Open with > Deployment Descriptor Editor**, and click the WS Binding tab. The Client Deployment Descriptor is displayed.
   e. Locate the Port qualified name binding section and either select an existing entry or click **Add**, to add a new port binding. The Web services client port binding editor displays for the selected port.
   f. Locate the Trust anchor section and click **Add**. The Trust anchor dialog box is displayed.
      1) Enter a unique name within the port binding for the **Trust anchor name**.

         The name is used to reference the trust anchor that is defined.
      2) Enter the key store password, path, and key store type.

         The supported key store types are Java Cryptography Extension (JCE) and JCEKS.

      Click **Edit** to edit the selected trust anchor.

      Click **Remove** to remove the selected trust anchor.

      When you start the application, the configuration is validated in the run time while the binding information is loading.
   g. Save the changes.

4. Configure the server-side request receiver, which is defined in the `ibm-webservices-bnd.xmi` bindings extensions file.
   a. Click **Windows > Open perspective > J2EE**.
   b. Select the Web services enabled EJB or Web module.
   c. In the Package Explorer window, click the `META-INF` directory for an EJB module or the `WEB-INF` directory for a Web module.
   d. Right-click the `webservices.xml` file, select **Open with > Web services editor**, and click the bindings tab. The Web services bindings editor is displayed.
   e. Locate the Web service description bindings section and either select an existing entry or click **Add** to add a new Web services descriptor.
   f. Click **Binding configurations**. The Web services binding configurations editor is displayed for the selected Web services descriptor.
   g. Locate the Trust anchor section and click **Add**. The Trust anchor dialog box is displayed.
      1) Enter a unique name within the binding for the **Trust anchor name**.

         This unique name is used to reference the trust anchor defined.
      2) Enter the key store password, path, and key store type. The supported key store types are JCE and JCEKS.

      Click **Edit** to edit the selected trust anchor.

      Click **Remove** to remove the selected trust anchor.

      When you start the application, the configuration is validated in the run time while the binding information is loading.
   h. Save the changes.

This procedure defines trust anchors that can be used by the request receiver or the response receiver (if the Web services is acting as client) to verify the signer certificate.

The request receiver or the response receiver (if the Web service is acting as a client) uses the defined trust anchor to verify the signer certificate. The trust anchor is referenced using the trust anchor name.

To complete the signing information configuration process for request receiver, complete the following tasks:
1. "Configuring the server for request digital signature verification: Verifying the message parts" on page 860
2. "Configuring the server for request digital signature verification: choosing the verification method" on page 862

To complete the process for the response receiver, if the Web services is acting as a client, complete the following tasks:
1. "Configuring the client for response digital signature verification: verifying the message parts" on page 867
2. "Configuring the client for response digital signature verification: choosing the verification method" on page 869

***Configuring trust anchors using the administrative console:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

This document describes how to configure trust anchors or trust stores at the application level. It does not describe how to configure trust anchors at the server or cell level. Trust anchors defined at the application

level have a higher precedence over trust anchors defined at the server or cell level. For more information on creating and configuring trust anchors at the server or cell level, see either “Configuring the server security bindings using an assembly tool” on page 875 or “Configuring the server security bindings using the administrative console” on page 877.

You can configure an application-level trust anchor using an assembly tool or the administrative console. This document describes how to configure the application-level trust anchor using the administrative console.

A trust anchor specifies key stores that contain trusted root certificates, which validate the signer certificate. These key stores are used by the request receiver (as defined in the `ibm-webservices-bnd.xmi` file) and the response receiver (as defined in the `ibm-webservicesclient-bnd.xmi` file when Web services is acting as client) to validate the signer certificate of the digital signature. The keystores are critical to the integrity of the digital signature validation. If they are tampered with, the result of the digital signature verification is doubtful and comprised. Therefore, it is recommended that you secure these keystores. The binding configuration specified for the request receiver in the `ibm-webservices-bnd.xmi` file must match the binding configuration for the response receiver in the `ibm-webservicesclient-bnd.xmi` file.

The following steps are for the client-side response receiver, which is defined in the `ibm-webservicesclient-bnd.xmi` file and the server-side request receiver, which is defined in the `ibm-webservices-bnd.xmi` file.

1. Configure an assembly tool to work with a Java 2 Platform, Enterprise Edition (J2EE) enterprise application. For more information, see Assembling applications
2. Create a Web services-enabled J2EE enterprise application. If you have not created a Web services-enabled J2EE enterprise application, see Developing Web services applications. Also, see either “Configuring the server security bindings using an assembly tool” on page 875 or “Configuring the server security bindings using the administrative console” on page 877 for an introduction on how to manage Web services security binding information on the server.
3. Click **Applications > Enterprise applications >** *enterprise_application*.
4. Under Related items, click either **EJB Modules** or **Web Modules** and then click the Web services-enabled module in the **URI** field.
5. Under Additional properties, click **Web services: client security bindings** to edit the response receiver binding information, if Web services is acting as a client.
   a. Under Response receiver binding, click **Edit**.
   b. Under Additional properties, click **Trust anchors**.
   c. Click **New** to create a new trust anchor.
   d. Enter a unique name within the request receiver binding for the Trust anchor name field. The name is used to reference the trust anchor that is defined.
   e. Enter the key store password, path, and key store type.
   f. Click the trust anchor name link to edit the selected trust anchor.
   g. Click **Remove** to remove the selected trust anchor or anchors.

      When you start the application, the configuration is validated in the run time while the binding information is loading.
6. Return to the Web services-enabled module panel accessed in step 2.
7. Under Additional properties, click **Web services: server security bindings** to edit the request receiver binding information.
   a. Under Request receiver binding, click **Edit**.
   b. Under Additional properties, click **Trust anchors**.
   c. Click **New** to create a new trust anchor

      Enter a unique name within the request receiver binding for the Trust anchor name field. The name is used to reference the trust anchor that is defined.

Enter the key store password, path, and key store type.

Click the trust anchor name link to edit the selected trust anchor.

Click **Remove** to remove the selected trust anchor or anchors.

When you start the application, the configuration is validated in the run time while the binding information is loading.

8. Save the changes.

This procedure defines trust anchors that can be used by the request receiver or the response receiver (if the Web services is acting as client) to verify the signer certificate.

The request receiver or the response receiver (if the Web service is acting as a client) uses the defined trust anchor to verify the signer certificate. The trust anchor is referenced using the trust anchor name.

To complete the signing information configuration process for request receiver, complete the following tasks:

1. "Configuring the server for request digital signature verification: Verifying the message parts" on page 860
2. "Configuring the server for request digital signature verification: choosing the verification method" on page 862

To complete the process for the response receiver, if the Web services is acting as client, complete the following tasks:

1. "Configuring the client for response digital signature verification: verifying the message parts" on page 867
2. "Configuring the client for response digital signature verification: choosing the verification method" on page 869

***Collection certificate store:***

A *collection certificate store* is a collection of nonroot, certificate authority (CA) certificates and certificate revocation lists (CRLs). This collection of CA certificates and CRLs is used to check the signature of a digitally signed SOAP message.

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

The collection certificate stores are used when processing a received SOAP message. This collection is configured in the securityRequestReceiverBindingConfig section of the binding file for servers and in the securityResponseReceiverBindingConfig section of the binding file for clients.

A collection certificate store is one kind of certificate store. A certificate store is defined as javax.security.cert.CertStore in the Java CertPath application programming interface (API). The Java CertPath API defines the following types of certificate stores:

**Collection certificate store**
: A collection certificate store accepts the certificates and CRLs as Java collection objects.

**Lightweight Directory Access Protocol certificate store**
: The Lightweight Directory Access Protocol (LDAP) certificate store accepts certificates and CRLs as LDAP entries.

The CertPath API uses the certificate store and the trust anchor to validate the incoming X.509 certificate that is embedded in the SOAP message.

The Web services security implementation in the WebSphere Application Server supports the collection certificate store. Each certificate and CRL is passed as an encoded file. This configuration is done using either the administrative console or by scripting.

***Configuring the client-side collection certificate store using an assembly tool:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

A collection certificate store is a collection of nonroot, certificate authority (CA) certificates and certificate revocation lists (CRLs). This collection of CA certificates and CRLs are used to check the signature of a digitally signed Simple Object Access Protocol (SOAP) message.

You can configure the collection certificate either by using an assembly tool or the WebSphere Application Server administrative console. Complete the following steps to configure the client-side collection certificate store using the assembly tool.

1. Launch an assembly tool. For more information on the assembly tools, see Assembly tools
2. Click **Windows > Open perspective > Other > J2EE**.
3. Click **Application Client projects >***application_name* **> appClientModule > META-INF**
4. Right-click the `application-client.xml` file, select **Open with > Deployment Descriptor Editor**, and click the WS Binding tab. The Client Deployment Descriptor is displayed.
5. Click the Port binding tab in deployment descriptor editor within the assembly tool. The Web services client port binding window is displayed.
6. Select one of the Port qualified name binding entries.
7. Expand the **Security response receiver binding configuration > certificate store list > Collection certificate store** section.
8. Click **Add** to create a new collection certificate store, click **Edit** to edit an existing certificate store, or click **Remove** to delete an existing certificate store.
9. Enter a name in the Name field. This name is referenced in the Certificate store reference field in the Signing info dialog box.
10. Leave the Provider field as `IBMCertPath`.
11. Click **Add** to enter the path to your certificate store. For example, the path might be: `${USER_INSTALL_ROOT}/etc/ws-security/samples/intca2.cer`. If you have additional certificate store paths, click **Add** to add the paths.
12. Click **OK** when you finish adding paths.

***Configuring the client-side collection certificate store using the administrative console:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

A collection certificate store is a collection of non-root, certificate authority (CA) certificates and certificate revocation lists (CRLs). This collection of CA certificates and CRLs are used to check the signature of a digitally signed Simple Object Access Protocol (SOAP) message.

You can configure the collection certificate either by using the assembly tools or the WebSphere Application Server administrative console. Complete the following steps to configure the client-side collection certificate store using the administrative console.

1. Connect to the WebSphere Application Server administrative console. You can connect to the administrative console by typing http://localhost:9060/ibm/console in your Web browser unless you have changed the port number.
2. Click **Applications > Enterprise applications >** *application_name*.
3. Under Related items, click either **Web modules** or **EJB modules** depending on the type of module you are securing.
4. Click the name of the module you are securing.
5. Under Additional properties, click either **Web services: client security bindings** to add the collection certificate store to the client security bindings. If you do not see any entries, return to the assembly tool and configure the security extensions for either the client or the server.

   To configure the security extensions for the client, see the following topics:
   - "Configuring the client for response digital signature verification: verifying the message parts" on page 867
   - "Configuring the client for response digital signature verification: choosing the verification method" on page 869
6. Under Response receiver binding, click **Edit** to edit the client security bindings.
7. Click **Collection certificate store**.
8. Click a Certificate store name to edit an existing certificate store or click **New** to add a new certificate store name.
9. Enter a name in the Certificate store name field. The name entered in this field is a name that is referenced in the Certificate store field on the Signing information configuration page.
10. Leave the Certificate store provider field value as `IBMCertPath`.
11. Click **Apply**.
12. Under Additional properties, click **X.509 certificates > New**.
13. Enter the path to your certificate store. For example, the path might be: `${USER_INSTALL_ROOT}/etc/ws-security/samples/intca2.cer`. If you have any additional certificate store paths to enter, click **New** and add the path names.
14. Click **OK**.

***Configuring the server-side collection certificate store using an assembly tool:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

A collection certificate store is a collection of nonroot, certificate authority (CA) certificates and certificate revocation lists (CRLs). This collection of CA certificates and CRLs are used to check the signature of a digitally signed SOAP message.

You can configure the collection certificate either by using an assembly tool or the WebSphere Application Server administrative console. Complete the following steps to configure the server-side collection certificate store using an assembly tool.

1. Start an assembly tool. For more information on the assembly tools, see Assembly tools.
2. Click **Windows > Open perspective > Other > J2EE**.
3. Click **EJB projects >***application_name* **> ejbModule > META-INF**
4. Right-click the `webservices.xml` file, select **Open with > Web Services Editor**.
5. Click the Binding configurations tab in the Web services editor within the assembly tool. The Web Service Binding Configuration window is displayed.
6. Select one of the Web service description binding entries under the Port Component Binding section.

7. Expand the **Request receiver binding configuration details > Certificate store list > Collection certificate store** section.
8. Click **Add** to create a new collection certificate store, click **Edit** to edit an existing certificate store, or click **Remove** to delete an existing certification store.
9. Enter a name in the Name field. This name is referenced in the **Certificate store reference** field in the Signing info dialog.
10. Leave the Provider field as IBMCertPath.
11. Click **Add** to enter the path to your certificate store. For example, the path might be: `${USER_INSTALL_ROOT]/etc/ws-security/samples/intca2.cer`. If you have additional certificate store paths, click **Add** to add the paths.
12. Click **OK** when you finish adding paths.

***Configuring the server-side collection certificate store using the administrative console:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

A *collection certificate store* is a collection of nonroot, certificate authority (CA) certificates and certificate revocation lists (CRLs). This collection of CA certificates and CRLs is used to check the signature of a digitally signed SOAP message.

You can configure the collection certificate either by using an assembly tool or the WebSphere Application Server administrative console. Complete the following steps to configure the server-side collection certificate store using the administrative console.

1. Connect to the WebSphere Application Server administrative console. You can connect to the administrative console by typing http://localhost:9060/ibm/console in your Web browser unless you have changed the port number.
2. Click **Applications > Enterprise applications >** *application_name*.
3. Under Related items, click either **Web modules** or **EJB modules** depending on the type of module you are securing.
4. Click the name of the module you are securing.
5. Under Additional properties, click **Web services: server security bindings** to add the collection certificate store to the server security bindings. If you do not see any entries, return to the assembly tool and configure the security extensions for the server.

   To configure the security extensions for the server, see the following topics:
   - “Configuring the server for request digital signature verification: Verifying the message parts” on page 860
   - “Configuring the server for request digital signature verification: choosing the verification method” on page 862
6. Click **Edit** under Request Receiver Binding to edit the server security bindings.
7. Click **Collection certificate store**.
8. Click a Certificate store name to edit an existing certificate store or click **New** to add a new certificate store name.
9. Enter a name in the Certificate store name field. The name entered in this field is a name that is referenced in the Certificate store field on the Signing information configuration page.
10. Leave the Certificate store provider field as `IBMCertPath`.
11. Click **Apply**.
12. Under Additional Properties, click **X.509 Certificates > New**.

13. Enter the path to your certificate store. For example, the path might be: `${USER_INSTALL_ROOT}/etc/ws-security/samples/intca2.cer`. If you have any additional certificate store paths to enter, click **New** and add the path names.
14. Click **OK**.

***Configuring default collection certificate stores at the server level in the WebSphere Application Server administrative console:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

A collection certificate store is a collection of nonroot, certificate authority (CA) certificates and certificate revocation lists (CRLs). This collection of CA certificates and CRLs are used to check the signature of a digitally signed SOAP message. A certificate store typically refers to a certificate store located in the file system. The location of the certificate store can vary from machine to machine, so you might configure a default collection certificate store for a specific machine and reference it from within the signing information. The signing information is found within the binding configurations of any application installed on the machine. This suggestion enables you to define a single collection certificate store for all of the applications that need to use the same certificates. You also can specify the default binding information at the cell level.

Complete the following steps to configure the default collection certificate store at the server level using the WebSphere Application Server administrative console:

1. Connect to the administrative console. You can access the administrative console by typing `http://localhost:9060/ibm/console` in your Web browser unless you have changed the port number.
2. Click **Servers > Application servers >***server_name*.
3. Under Security, click **Web Services: Default bindings for Web Services Security**.
4. Under Additional properties, click **Collection certificate store**.
5. Enter a name in the Certificate store name field. This name is referenced in the Certificate store field on the Signing information configuration page.
6. Leave the Certificate store provider field value as `IBMCertPath`.
7. Click **Apply**.
8. Under Additional properties, click **X.509 certificates > New**.
9. Enter the path to your certificate store. For example, the path might be: `${USER_INSTALL_ROOT}/etc/ws-security/samples/intca2.cer`.

   If you have any additional certificate store paths to enter, click **New** and add the path names.
10. Click **OK**.

***Key locator:***

A *key locator* (com.ibm.wsspi.wssecurity.config.KeyLocator) is an abstraction of the mechanism that retrieves the key for digital signature and encryption.

You can use any of the following infrastructure from which to retrieve the keys depending upon the implementation:

- Java keystore file
- Database
- Lightweight Third Party Authentication (LTPA) server

Key locators search the key using some type of a clue. The following types of clues are supported:

- A string label of the key, which is explicitly passed through the application programming interface (API). The relationships between each key and its name (string label) is maintained inside the key locator.
- The execution context of the key locator; explicit information is not passed to the key locator. A key locator determines the appropriate key according to the execution context.

Current versions of key locators do not support the retrieval of verification keys because current Web services security implementations do not support the secret key-based signature. Because the key locators support the public key-based signature only, the key for verification is embedded in the X.509 certificate as a `<BinarySecurityToken>` element in the incoming message.

For example, key locators can obtain the identity of the caller from the context and can retrieve the public key of the caller for response encryption.

**Usage scenarios**

This section describes the usage scenarios for key locators.

**Signing**

The name of the signing key is specified in the Web services security configuration. This value is passed to the key locator and the actual key is returned. The corresponding X.509 certificate also can be returned.

**Verification**

As described previously, key locators are not used in signature verification.

**Encryption**

The name of the encryption key is specified in the Web services security configuration. This value is passed to the key locator and the actual key is returned.

**Decryption**

The Web services security specification recommends using the key identifier instead of the key name. However, while the algorithm for computing the identifier for the public keys is defined in Internet Engineering Task Force (IETF) Request for Comment (RFC) 3280, there is no agreed upon algorithm for the secret keys. Therefore, the current implementation of Web services security uses the identifier only when public key-based encryption is performed. Otherwise, the ordinal key name is used.

When you use public key-based encryption, the value of the key identifier is embedded in the incoming encrypted message. Then, the Web services security implementation searches for all the keys managed by the key locator and decrypts the message using the key whose identifier value matches the one in the message.

When you use secret key-based encryption, the value of the key name is embedded in the incoming encrypted message. The Web services security implementation asks the key locator for the key with the name that matches the name in the message and decrypts the message using the key.

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

***Keys:***

Keys are used for XML signature and encryption.

There are two predominant kinds of keys used in the current Web services security implementation:

- Public key - such as Rivest Shamir Adleman (RSA) encryption and Digital Signature Algorithm (DSA) encryption
- Secret key - such as Data Encryption Standard (DES) encryption

In public key-based signature, a message is signed using the sender private key and is verified using the sender public key. In public key-based encryption, a message is encrypted using the receiver public key and is decrypted using the receiver private key. In secret key-based signature and encryption, the same key is used by both parties.

While the current implementation of Web services security can support both kinds of keys, there are a few items to note:

- Secret key-based signature is not supported.
- The format of the message differs slightly between public key-based encryption and secret key-based encryption.

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

***Web services security service provider programming interfaces:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

Several Service Provider Interfaces (SPIs) are provided to extend the capability of the Web services security run time. The following list contains the SPIs that are available for WebSphere Application Server:

- `com.ibm.wsspi.wssecurity.config.KeyLocator` is an abstract for obtaining the keys for digital signature and encryption. The following list contains the default implementations:
  - `com.ibm.wsspi.wssecurity.config.KeyStoreKeyLocator`

    Implements the Java key store.
  - `com.ibm.wsspi.wssecurity.config.WSIdKeyStoreMapKeyLocator`

    Provides a mapping of the authenticated identity to a key for encryption. Or, the implementation uses the default key that is specified. This implementation is typically used in the response sender configuration.
  - `com.ibm.wsspi.wssecurity.config.CertInRequestKeyLocator`

    Provides the capability of using the signer key for encryption in the response message. This implementation is typically used in the response sender configuration.
- `com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator` is an interface that is used to evaluate the trust for identity assertion. The default implementation is `com.ibm.wsspi.wssecurity.id.TrustedIDEvaluatorImpl`, which enables you to define a list of trusted identities.
- The Java Authentication and Authorization Service (JAAS) CallbackHandler application programming interfaces (APIs) are used for token generation by the request sender. This interface can be extended to generate a custom token that can be inserted in the Web services security header. The following list contains the default implementations that are provided by WebSphere Application Server:
  - `com.ibm.wsspi.wssecurity.auth.callback.GUIPromptCallbackHandler`

    Presents a login prompt to gather the basic authentication data. Use this implementation in the client environment only.

- `com.ibm.wsspi.wssecurity.auth.callback.StdinPromptCallbackHandler`

  Collects the basic authentication data in the standard in (stdin) prompt. Use this implementation in the client environment only.
- `com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler`

  Reads the basic authentication data from the application binding file. This implementation might be used on the server side to generate a user name token.
- `com.ibm.wsspi.wssecurity.auth.callback.LTPATokenCallbackHandler`

  Generates a Lightweight Third Party Authentication (LTPA) token in the Web services security header as a binary security token. If basic authentication data is defined in the application binding file, it is used to perform a login, to extract the LTPA token from the WebSphere credentials, and to insert the token in the Web services security header. Otherwise, it will extract the LTPA security token from the invocation credentials (run as identity) and insert the token in the Web services security header.

The JAAS LoginModule API is used for token validation on the request receiver side of the message. You can implement a custom LoginModule API to perform validation of the custom token on the request receiver of the message. After the token is verified and validated, the token is set as the caller and then run as the identity in the WebSphere Application Server run time. The identity is used for authorization checks by the containers before a Java 2 Platform, Enterprise Edition (J2EE) resource is invoked. The following list presents the are the default AuthMethod configurations provided by WebSphere Application Server:

**BasicAuth**
: Validates a user name token.

**Signature**
: Maps the distinguished name (DN) of a verified certificate to a Java Authentication and Authorization Service (JAAS) subject.

**IDAssertion**
: Maps a trusted identity to a JAAS subject.

**LTPA**
: Validates an LTPA token that is received in the message and creates a JAAS subject.

***Configuring key locators using an assembly tool:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

This task provides instructions on how to configure key locators using an assembly tool. You can configure key locators in various locations within the assembly tool. This task provides instructions on how to configure key locators at any of these locations because the concept is the same.

1. Start an assembly tool. For more information on the assembly tools, see Assembly tools.
2. Click **Windows > Open perspective > Other > J2EE**.
3. Click **Application Client projects >***application_name* **> appClientModule > META-INF**
4. Right-click the `application-client.xml` file, select **Open with > Deployment Descriptor Editor**, and click the WS Binding tab. The Client Deployment Descriptor is displayed.
5. Click the WS Binding tab in deployment descriptor editor within the assembly tool or the Binding configurations tab in the Web services editor within the assembly tool.
6. Expand one of the **Binding configuration** sections.
7. Expand the **Key locators** section.
8. Click **Add** to create a new key locator, click **Edit** to edit an existing key locator, or click **Remove** to delete an existing key locator.

9. Enter a key locator name. The name entered for the **Key locator name** is used to refer to the key locator from the Encryption information and Signing Information sections.
10. Enter a key locator class. The key locator class is the implementation of the `KeyLocator` interface. When using default implementations, select a class from the menu.
11. Determine whether to click **Use key store**. Select this option when you use the default implementations as they use key stores. If you click **Use key store**, complete the following steps:
    a. Enter a value in the key store storepass field. The key store storepass is the password used to access the key store.
    b. Enter a path name in the key store path field. The key store path is the location on the file system where the key store resides. Make sure that the location can be found wherever you deploy the application.
    c. Enter a type value in the key store type field. The valid types to enter are JKS and JCEKS. JKS is used when you are not using Java Cryptography Extensions (JCE). JCEKS is used when you are using JCE. Although the JCEKS type is more secure, it might decrease performance.
    d. Click **Add** to create an entry for a key in the key store.
        1) Enter a value in the Alias field.
           The key alias is a reference to this particular key from the Signing Information section.
        2) Enter a value in the Key pass field.
           The key pass is the password associated with the certificate which is created using the Development Kit, Java Technology Edition `keytool.exe` file.
        3) Enter a value in the Key name field.
           The key name refers to the alias of the certificate as found in the key store.
12. Click **Add** to create a custom property. The property can be used by custom key locator implementations. For example, you can use properties with the `WSIdKeyStoreMapKeyLocator` default implementation. The key locator implementation has the following property names:
    - *id_*, which maps to a credential user ID.
    - *mappedName_* , which maps to the key alias to use for this user name.
    - *default*, which maps to a key alias to use when a credential does not have an associated *id_* entry.

    A typical set of properties for this key locator might be: `id_1=user1`, `mappedName_1=key1`, `id_2=user2`, `mappedName_2=key2`, `default=key3`. If `user1` or `user2` authenticates, then the associated `key1` or `key2` is used, respectively. However, if none of the user properties authenticate or the user is not `user1` or `user2`, then `key3` is used.
    a. Enter a name in the Name field. The name entered is the property name.
    b. Enter a value in the Value field. This value entered is the property value.

***Configuring key locators using the administrative console:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

This task provides instructions on how to configure key locators using the WebSphere Application Server administrative console. You can configure binding information in the administrative console. You must use an assembly tool to configure extensions. The following steps are used to configure a key locator in the administrative console for a specific application:

1. Connect to the administrative console by typing `http://localhost:9060/ibm/console` in your Web browser unless you have changed the port number.
2. Click **Applications > Enterprise Applications >** *application_name*.
3. Under Related Items, click either **Web Modules** or **EJB Modules**, depending on the type of module you are securing.

4. Click the name of the module you are securing.
5. Under Additional Properties, click either **Web services: Client security bindings** or **Web services: Server security bindings**, depending on whether you are adding the key locator to the client security bindings or to the server security bindings. If you do not see any entries, return to the assembly tool and configure the security extensions.
6. Edit the Request Sender Binding, Response Receiver Binding, Request Receiver Binding, or Response Sender Binding.
   - If you are editing your client security bindings, click **Edit** for either the Request Sender Binding or the Response Receiver Binding.
   - If you are editing your server security bindings, click **Edit** for either the Request Receiver Binding or the Response Sender Binding.
7. Click **Key Locators**.
8. Click **New** to configure a new key locator, select the box next to a key locator name and click **Delete** to delete a key locator, or click the name of a key locator to edit its configuration. If you are configuring a new key locator or editing an existing one, complete the following steps:
   a. Specify a name for the key locator in the **Key Locator Name** field.
   b. Specify a name for the key locator class implementation in the **Key Locator Classname** field. WebSphere Application Server has the following default key locator class implementations:

      **com.ibm.wsspi.wssecurity.config.WSIdKeyStoreMapKeyLocator**
      This class is used by the response sender to map an authenticated identity to a key. If encryption is used, this class is used to locate a key to encrypt the response message. The com.ibm.wsspi.wssecurity.config.WSIdKeyStoreMapKeyLocator class has the capability to map an authenticated identity from the invocation credential of the current thread to a key that is used to encrypt the message. If an authenticated identity is present on the current thread, the class maps the ID to the mapped name. For example, user1 is mapped to mappedName_1. Otherwise, `name="default"`. When a matching key is not found, the authenticated identity is mapped to the default key specified in the binding file.

      **com.ibm.wsspi.wssecurity.config.KeyStoreKeyLocator**
      This class is used by the response receiver, the request sender, and the request receiver to map a name to an alias. Encryption uses this class to obtain a key to encrypt a message and digital signature uses this class to obtain a key to sign a message. The com.ibm.wsspi.wssecurity.config.KeyStoreKeyLocator class maps a logical name to a key alias in the key store file. For example, key #105115176771 maps to `CN=Alice, O=IBM, C=US`.
   c. Specify the password used to access the key store password in the **Key Store Password** field. This field is optional because the key locator does not use a key store.
   d. Specify the path name used to access the key store in the **Key Store Path** field. This field is optional because the key locator does not use a key store. Use `${USER_INSTALL_ROOT}` because this path expands to the WebSphere Application Server path on your machine.
   e. Select a keystore type from the **Key Store Type** field. This field is optional because the key locator does not use a key store. Use the **JKS** option if you are not using Java Cryptography Extensions (JCE) and use **JCEKS** if you are using JCE.

***Trusted ID evaluator:***

Trusted ID evaluator (com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator) is an abstraction of the mechanism that evaluates whether the given ID name is trusted.

Depending upon the implementation, you can use various types of infrastructure to store a list of the trusted IDs, such as:
- Plain text file
- Database

- Lightweight Directory Access Protocol (LDAP) server

The trusted ID evaluator is typically used by the eventual receiver in a multi-hop environment. The Web services security implementation invokes the trusted ID evaluator and passes the identity name of the intermediary as a parameter. If the identity is evaluated and deemed trustworthy, the procedure continues. Otherwise, an exception is thrown and the procedure is stopped.

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

***Login mappings:***

Login mappings, found in the `ibm-webservices-bnd.xmi` Extended Markup Language (XML) file, contains a mapping configuration. This mapping configuration defines how the Web services security handler maps the token `<ValueType>` element that is contained within the security token extracted from the message header, to the corresponding authentication method. The token `<ValueType>` element is contained within the security token extracted from a SOAP message header.

The sender-side Web services security handler generates and attaches security tokens based on the <AuthMethods> element that is specified in the deployment descriptor. For example, if the authentication method is BasicAuth, the sender-side security handler generates and attaches UsernameToken (with both user name and password) to the SOAP message header. The Web services security run time uses the Java Authentication and Authorization Service (JAAS) `javax.security.auth.callback.CallbackHandler` interface as a security provider to generate security tokens on the client side (or when Web services is acting as client).

The sender security handler invokes the `handle()` method of a `javax.security.auth.callback.CallbackHandler` interface implementation. This implementation creates the security token and passes the token back to the sender security handler. The senders security handler constructs the security token based on the authentication information in the callback array. The security handler then inserts the security token into the Web Services Security message header.

The CallbackHandler interface implementation that you use to generate the required security token is defined in the <loginBinding> element in the `ibm-webservicesclient-bnd.xmi` Web services security binding file. For example,

```
<loginBinding xmi:id="LoginBinding_1052760331526" authMethod="BasicAuth"
      callbackHandler="com.ibm.wsspi.wssecurity.auth.callback.StdinPromptCallbackHandler"/>
```

The `<loginBinding>` element associates the `com.ibm.wsspi.wssecurity.auth.callback.StdinPromptCallbackHandler` interface with the BasicAuth authentication method. WebSphere Application Server provides the following set of CallbackHandler interface implementations you can use to create various security token types:

**`com.ibm.wsspi.wssecurity.auth.callback.GUIPromptCallbackHandler`**
: If there is no basic authentication data defined in the login binding information (this information is not the same as the HTTP basic authentication information), the previous token type prompts for user name and password through a login panel. The implementation uses the basic authentication data defined in the login binding. Use this CallbackHandler with the BasicAuth authentication method. Do not use this CallbackHandler implementation on the server because it prompts you for login binding information.

**`com.ibm.wsspi.wssecurity.auth.callback.StdinPromptCallbackHandler`**
: If basic authentication data is not defined in the login binding (this information is not the same as the HTTP basic authentication information), the implementation prompts for the user name and password using standard in (stdin). The implementation uses the basic authentication data defined

in the login binding. Use this CallbackHandler implementation with the BasicAuth authentication method. Do not use this CallbackHandler implementation on the server because it prompts you for login binding information.

**com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler**
This CallbackHandler implementation does not prompt. Rather, it uses the basic authentication data defined in the login binding (this information is not the same as the HTTP basic authentication information). This CallbackHandler implementation is meant for use with the BasicAuth authentication method. You must define the basic authentication data in the login binding information for this CallbackHandler implementation. You can use this implementation when Web services is running as a client and needs to send basic authentication (<wsse:UsernameToken>) to the downstream call.

**com.ibm.wsspi.wssecurity.auth.callback.LTPATokenCallbackHandler**
The CallbackHandler generates Lightweight Third Party Authentication (LTPA) tokens from the run as JAAS Subject (invocation Subject) of the current WebSphere Application Server security context. However, if basic authentication data is defined in the login binding information (not the HTTP basic authentication information), the implementation uses the basic authentication data and LTPA token generated. The **Token Type URI** and **Token Type Local Name** values must be defined in the login binding information for this CallbackHandler implementation. The token value type is used to validate the token to the request sender and request receiver binding configuration. The Web services security run time inserts the LTPA token as a binary security token (<wsse:BinarySecurityToken>) into the message SOAP header. The value type is mandatory. (See LTPA for more information). Use this CallbackHandler implementation with the LTPA authentication method.

Figure 1 shows the sender security handler in the request sender message process.

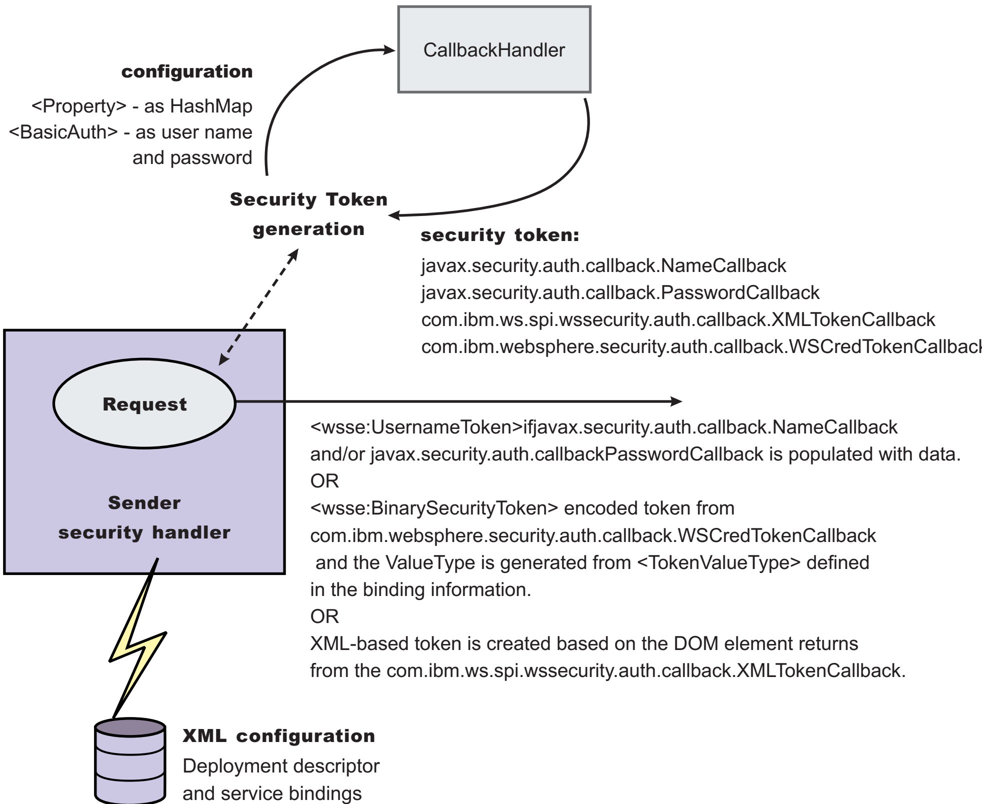



*Figure 15. Request sender SOAP message process*

You can configure the receiver-side security server to support multiple authentication methods and multiple types of security tokens. The following steps describe the request sender SOAP message process:

1. After receiving a message, the receiver Web services security handler compares the token type (in the message header) with the expected token types configured in the deployment descriptor.
2. The Web services security handler extracts the security token form the message header and maps the token `<ValueType>` element to the corresponding authentication method. The mapping configuration is defined in the `<loginMappings>` element in the `ibm-webservices-bnd.xmi` XML file. For example:

```
<loginMappings xmi:id="LoginMapping_1051977980074" authMethod="LTPA"
      configName="WSLogin">
     <callbackHandlerFactory xmi:id="CallbackHandlerFactory_1051977980081"
     classname="com.ibm.wsspi.wssecurity.auth.callback.WSCallbackHandlerFactoryImpl"/>
      <tokenValueType xmi:id="TokenValueType_1051977980081"
      uri="http://www.ibm.com/websphere/appserver/tokentype/5.0.2" localName="LTPA"/>
</loginMappings>
```

   The com.ibm.wsspi.wssecurity.auth.callback.CallbackHandlerFactory interface is a factory for javax.security.auth.callback.CallbackHandler.
3. The Web services security run time initiates the factory implementation class and passes the authentication information from Web services security header to the factory class through the set methods.
4. The Web services security run time invokes the `newCallbackHandler()` method to obtain the `javax.security.auth.CallbackHandler` object, which generates the required security token.
5. When the security handler receives an LTPA `BinarySecurityToken`, it uses the `WSLogin` JAAS login configuration and the `newCallbackHandler()` method to validate the security token. If none of the expected token types are found in the SOAP message Web services security header, the request is rejected with an SOAP fault. Otherwise, the token type is used to map to a JAAS login configuration

for token validation. If authentication is successful, a JAAS Subject is created and associated with the running thread. Otherwise, the request is rejected with a SOAP fault.

The following table shows the authentication methods and the JAAS login configurations.

| Authentication method | JAAS login configuration |
| --- | --- |
| BasicAuth | `WSLogin` |
| Signature | `system.wssecurity.Signature` |
| LTPA | `WSLogin` |
| IDAssertion | `system.wssecurity.IDAssertion` |

Figure 2 shows the receiver security handler in the request receiver message process.

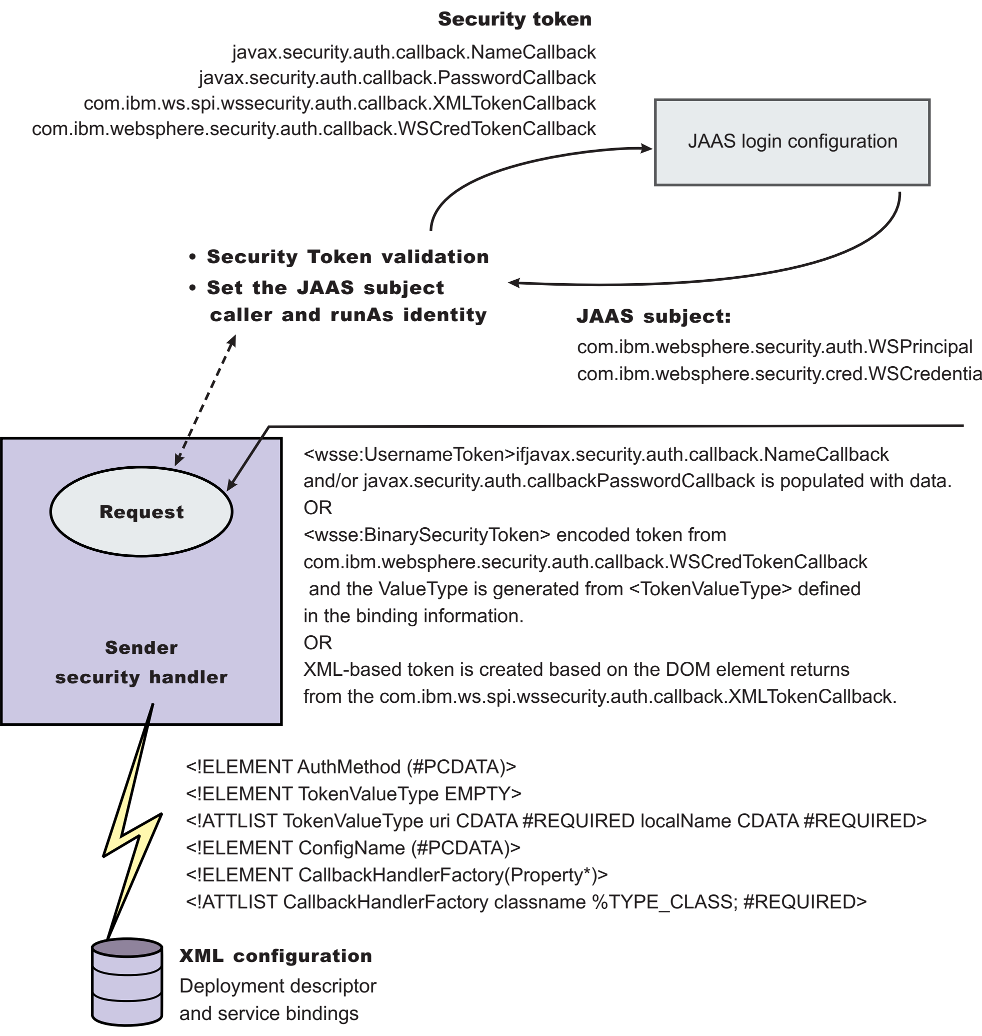



*Figure 16. Request receiver SOAP message process*

The default `<LoginMapping>` is defined in the following files:

- Server-level `ws-security.xml` file

If nothing is defined in the binding file information, the `ws-security.xml` default is used. However, an administrator can override the default by defining a new `<LoginMapping>` element in the binding file.

6. The client reads the default binding information in the `${install_dir}/properties/ws-security.xml` file.
7. The server run-time component loads the following files if they exist:
   - Server-level `ws-security.xml` file

   On a base application server, the server run time component only loads the server-level `ws-security.xml` file. The server-side `ws-security.xml` file and the application Web services security binding information are managed using the administrative console and the WSADMIN command. You can specify the binding information during application deployment either through the administrative console or through the WSADMIN command. The Web services security policy is defined in the deployment descriptor extension (`ibm-webservicesclient-ext.xmi`) and the bindings are stored in the IBM binding extension (`ibm-webservicesclient-bnd.xmi`). However, the `${install_dir}/properties/ws-security.xml` file defines the default binding value for the client. If the binding information is not specified in the binding file, the run time reads the binding information from the default `${install_dir}/properties/ws-security.xml` file.

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

*Login mappings collection:*

Use this page to view a list of configurations for validating security tokens within incoming messages. Login mappings map an authentication method to a Java Authentication and Authorization Service (JAAS) login configuration to validate the security token. Four authentication methods are predefined in the WebSphere Application Server: BasicAuth, Signature, IDAssertion, and Lightweight Third Party Authentication (LTPA).

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

To view this administrative console page for the server level, complete the following steps:
1. Click **Servers > Application servers >***server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Login mappings**.
4. Click either **New** to create a new login mapping configuration or click the name of an existing configuration.

To view this administrative console page for the application level, complete the following steps:
1. Click **Applications > Enterprise applications** > *application_name*.
2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.
3. Under Additional properties, click **Web services: Server security bindings**.
4. Click **Edit** under Request receiver binding.
5. Click **Login mappings**.

If you click **Update runtime**, the Web services security run time is updated with the default binding information, which is contained in the `ws-security.xml` file that was previously saved. After you specify the authentication method, the Java Authentication and Authorization Service (JAAS) configuration name, and the Callback Handler Factory class name on this panel, you must complete the following steps:
1. Click **Save** in the messages section at the top of the administrative console.

2. Click **Update runtime**. When you click **Update runtime**, the configuration changes made to the other Web services also are updated in the Web services security run time.

**Important:** If the login mapping configuration is not found on the application level, the Web services run time searches for the login mapping configuration on the server level. If the configuration is not found on the server level, the Web services run time searches the cell.

*Authentication method:*

Specifies the authentication method used for validating the security tokens.

The following authentication methods are available:

**BasicAuth**
The basic authentication method includes both a user name and a password in the security token. The information in the token is authenticated by the receiving server and is used to create a credential.

**Signature**
The signature authentication method sends an X.509 certificate as a security token. For Lightweight Directory Access Protocol (LDAP) registries, the distinguished name (DN) is mapped to a credential, which is based on the LDAP certificate filter settings. For local OS registries, the first attribute of the certificate, usually the common name (CN) is mapped directly to a user name in the registry.

**IDAssertion**
The identity assertion method maps a trusted identity (ID) to a WebSphere Application Server credential. This authentication method only includes a user name in the security token. An additional token is included in the message for trust purposes. When the additional token is trusted, the IDAssertion token user name is mapped to a credential.

**LTPA** Lightweight Third Party Authentication (LTPA) validates an LTPA token.

*JAAS configuration name:*

Specifies the name of the Java Authentication and Authorization Service (JAAS) configuration.

*Callback handler factory class name:*

Specifies the name of the factory for the CallbackHandler class.

*Login mapping configuration settings:*

Use this page to specify the Java Authentication and Authorization Service (JAAS) login configuration settings that are used to validate security tokens within incoming messages.

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

To view this administrative console page for the server level, complete the following steps:

1. Click **Servers > Application servers >***server_name*.
2. Under Security, click **Web services: Default bindings for Web services security**.
3. Under Additional properties, click **Login mappings**.
4. Click either **New** to create a new login mapping configuration or click the name of an existing configuration.

To use this administrative console page for the application level, complete the following steps:

1. Click **Applications > Enterprise applications** > *application_name*.
2. Under Related items, click **EJB modules** or **Web modules** > *URI_name*.
3. Under Additional properties, click **Web services: Server security bindings**.
4. Click **Edit** under Request receiver binding.
5. Click **Login mappings**.
6. Click either **New** to create a new login mapping configuration or click the name of an existing configuration.

**Important:** If the login mapping configuration is not found on the application level, the Web services run time searches for the login mapping configuration on the server level. If the configuration is not found on the server level, the Web services run time searches the cell.

*Authentication method:*

Specifies the method of authentication.

You can use any string, but the string must match the element in the service-level configuration. The following words are reserved and have special meanings:

**BasicAuth**
Uses both a user name and a password.

**IDAssertion**
Uses only a user name, but requires that additional trust is established on the receiving server using a TrustedIDEvaluator mechanism.

**Signature**
Uses the distinguished name (DN) of the signer.

**LTPA** Validates a token.

*JAAS configuration name:*

Specifies the name of the Java Authentication and Authorization Service (JAAS) configuration.

You can use the following predefined system login configurations:

**system.wssecurity.IDAssertion**
Enables a version 5.x application to use identity assertion to map a user name to a WebSphere Application Server credential principal.

**system.wssecurity.Signature**
Enables a version 5.x application to map a distinguished name (DN) in a signed certificate to a WebSphere Application Server credential principal.

**system.LTPA_WEB**
Processes login requests that are used by the Web container such as servlets and JavaServer Pages (JSP) files.

**system.WEB_INBOUND**
Handles logins for Web application requests, which include servlets and JavaServer Pages. This login configuration is used by WebSphere Application Server Version 5.1.1.

**system.RMI_INBOUND**
Handles logins for inbound Remote Method Invocation (RMI) requests. This login configuration is used by WebSphere Application Server Version 5.1.1.

**system.DEFAULT**
: Handles the logins for inbound requests made by internal authentications and most of the other protocols except Web applications and RMI requests. This login configuration is used by WebSphere Application Server Version 5.1.1.

**system.RMI_OUTBOUND**
: Processes RMI requests that are sent outbound to another server when either the com.ibm.CSI.rmiOutboundLoginEnabled or the com.ibm.CSIOutboundPropagationEnabled properties are true. These properties are set in the CSIv2 authentication panel. To access the panel, click **Security > Authentication protocol > CSIv2 Outbound authentication**. To set the com.ibm.CSI.rmiOutboundLoginEnabled property, select **Custom outbound mapping**. To set the com.ibm.CSIOutboundPropagationEnabled property, select **Security attribute propagation**.

**system.wssecurity.X509BST**
: Verifies an X.509 binary security token (BST) by checking the validity of the certificate and the certificate path.

**system.wssecurity.PKCS7**
: Verifies an X.509 certificate with a certificate revocation list in a PKCS7 object.

**system.wssecurity.PkiPath**
: Verifies an X.509 certificate with a public key infrastructure (PKI) path.

**system.wssecurity.UsernameToken**
: Verifies basic authentication (user name and password).

These system login configurations are defined on the System logins panel, which is accessible by completing the following steps:
1. Click **Security > Global security**.
2. Under Authentication, click **JAAS Configuration > System logins**.

**Attention:** The predefined system login configurations are listed on the System logins configuration panel without the system prefix. For example, the system.wssecurity.UsernameToken configuration listed in the JAAS configuration name option corresponds to the wssecurity.UsernameToken configuration that is on the System logins configuration panel.

You can use the following predefined application login configurations:

**ClientContainer**
: Specifies the login configuration that is used by the client container application, which uses the CallbackHandler API that is defined in the deployment descriptor of the client container.

**WSLogin**
: Specifies whether all applications can use the WSLogin configuration to perform authentication for the WebSphere Application Server security run time.

**DefaultPrincipalMapping**
: Specifies the login configuration used by Java 2 Connectors (J2C) to map users to principals that are defined in the J2C authentication data entries.

These application login configurations are defined on the Application logins panel, which is accessible by completing the following steps:
1. Click **Security > Global security**.
2. Under Authentication, click **JAAS Configuration > Application logins**.

Do not remove these predefined system or application login configurations. Within these configurations, you can add module class names and specify the order in which WebSphere Application Server loads each module.

*Callback handler factory class name:*

Specifies the name of the factory for the CallbackHandler class.

You must implement the com.ibm.wsspi.wssecurity.auth.callback.CallbackHandlerFactory class in this field.

*Token type URI:*

Specifies the namespace Uniform Resource Identifiers (URI), which denotes the type of security token that is accepted.

If binary security tokens are accepted, the value denotes the ValueType attribute in the element. The ValueType element identifies the type of security token and its namespace. If Extensible Markup Language (XML) tokens are accepted, the value denotes the top-level element name of the XML token.

If the reserved words are specified previously in the Authentication method field, this field is ignored.

| | |
|---|---|
| **Data type:** | Unicode characters except for non-ASCII characters, but including the number sign (#), the percent sign (%), and the square brackets ([ ]). |

*Token type local name:*

Specifies the local name of the security token type, for example, X509v3.

If binary security tokens are accepted, the value denotes the ValueType attribute in the element. The ValueType attribute identifies the type of security token and its namespace. If Extensible Markup Language (XML) tokens are accepted, the value denotes the top-level element name of the XML token.

If the reserved words are specified previously in the Authentication method field, this field is ignored.

*Nonce maximum age:*

Specifies the time, in seconds, before the nonce timestamp expires. Nonce is a randomly generated value.

You must specify a minimum of 300 seconds for the Nonce maximum age field. However, the maximum value cannot exceed the number of seconds specified in the Nonce cache timeout field for either the server level or the cell level.

You can specify the Nonce maximum age value for the server level by completing the following steps:

1. Click **Servers > Application Servers >***server_name*.
2. Under Additional Properties, click **Web Services: Default bindings for Web Services Security**.

**Important:** The Nonce maximum age field on this panel is optional and only valid if the BasicAuth authentication method is specified. If you specify another authentication method and attempt to specify values for this field, the following error message displays and you must remove the specified value: `Nonce is not supported for authentication methods other than BasicAuth`.

If you specify the BasicAuth method, but do not specify values for the Nonce maximum age field, the Web services security run time searches for a Nonce maximum age value on the server level. If a value is not found on the server level, the run time searches the cell level. If a value is not found on either the server level or the cell level, the default is 300 seconds.

| | |
|---|---|
| **Default** | 300 seconds |

| | |
|---|---|
| **Range** | 300 to Nonce cache timeout seconds |

*Nonce clock skew:*

Specifies the clock skew value, in seconds, to consider when WebSphere Application Server checks the freshness of the message. Nonce is a randomly generated value.

You can specify the **Nonce clock skew** value for the server level by completing the following steps:
1. Click **Servers > Application Servers >***server_name*.
2. Under Additional Properties, click **Web Services: Default bindings for Web Services Security**.

You must specify a minimum of zero (0) seconds for the Nonce Clock Skew field. However, the maximum value cannot exceed the number of seconds that is specified in the Nonce maximum age field on this Login mappings panel.

**Important:** The Nonce clock skew field on this panel is optional and only valid if the BasicAuth authentication method is specified. If you specify another authentication method and attempt to specify values for this field, the following error message displays and you must remove the specified value: `Nonce is not supported for authentication methods other than BasicAuth`.

**Note:** If you specify BasicAuth, but do not specify values for the Nonce clock skew field, WebSphere Application Server searches for a Nonce clock skew value on the server level. If a value is not found on the server level, the run time searches the cell level. If a value is not found on either the server level or the cell level, the default is zero (0) seconds.

| | |
|---|---|
| **Default** | 0 seconds |
| **Range** | 0 to Nonce Maximum Age seconds |

***Configuring the client for request signing: digitally signing message parts:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

Prior to completing these steps, read either of the following topics to become familiar with the Security Extensions tab and the Port Binding tab in the Web Services Client Editor within an assembly tool such as the Application Server Toolkit or Rational Web Developer.
- “Configuring the client security bindings using an assembly tool” on page 871
- “Configuring the security bindings on a server acting as a client using the administrative console” on page 873

These two tabs are used to configure the Web services security extensions and the Web services security bindings, respectively.

Complete the following steps to specify which message parts to digitally sign when configuring the client for request signing:
1. Launch an assembly tool. For more information on the assembly tools, see Assembly tools
2. Click **Windows > Open perspective > Other > J2EE**.
3. Click **Application Client projects >***application_name* **> appClientModule > META-INF**

4. Right-click the `application-client.xml` file, select **Open with > Deployment Descriptor Editor**, and click the WS Extension tab. The Client Deployment Descriptor is displayed.
5. Expand **Request sender configuration > Integrity**. Integrity refers to digital signature while confidentiality refers to encryption. Integrity decreases the risk of data modification while the data is transmitted across the Internet. For more information on digitally signing SOAP messages, see "XML digital signature" on page 570.
6. Indicate which parts of the message to sign by clicking **Add** and selecting **body**, **timestamp**, or **SecurityToken**. The following list contains descriptions of the message parts

   **Body** The body is the user data portion of the message.

   **Timestamp**
   The time stamp determines if the message is valid based on the time that the message is sent and then received. If **timestamp** is selected, proceed to the next step and select **Add created time stamp** to add a time stamp to a message.

   **Securitytoken**
   The security token authenticates the client. If this option is selected, the message is signed.

   You can choose to digitally sign the message using a time stamp if **Add created time stamp** is selected and configured. You can digitally sign the message using a security token if a login configuration authentication method is selected.
7. **Optional:** Expand the **Add created time stamp** section and select this option if you want a time stamp added to the message. You can specify an expiration time for the time stamp, which helps defend against replay attacks.The lexical representation for duration is the [ISO 8601] extended format *PnYnMnDTnHnMnS*, where:
   - *nY* represents the number of years
   - *nM* represents the number of months
   - *nD* represents the number of days
   - *T* is the date and time separator
   - *nH* represents the number of hours
   - *nM* represents the number of minutes
   - *nS* represents the number of seconds. The number of seconds can include decimal digits to arbitrary precision.

   For example, to indicate a duration of 1 year, 2 months, 3 days, 10 hours, and 30 minutes, the format is: `P1Y2M3DT10H30M`. Typically, you configure a message time stamp for about 10 to 30 minutes, for example, 10 minutes is represented as: `P0Y0M0DT0H10M0S`. The *P* character precedes time and date values.

**Important:** If you configure the client and server signing information correctly, but receive a `Soap body not signed` error when executing the client, you might need to configure the actor. You can configure the actor in the following locations on the client in the Web Services Client Editor within the assembly tool (such as the Application Server Toolkit or Rational Web Developer):

- Click **Security extensions > Client service configuration details** and indicate the actor information in the **Actor URI** field.
- Click **Security extensions > Request sender configuration > Details** and indicate the actor information in the **Actor** field.

You must configure the same actor strings for the Web service on the server, which processes the request and sends the response back. Configure the actor in the following locations in the Web Services Editor within the WebSphere Application Server Toolkit:

- Click **Security extensions > Server service configuration**.
- Click **Security extensions > Response sender service configuration details > Details** and indicate the actor information in the **Actor** field.

The actor information on both the client and server must refer to the same exact string. When the actor fields on the client and server match, the request or response is acted upon instead of being forwarded downstream. The **actor** fields might be different when you have Web services acting as a gateway to other Web services. However, in all other cases, make sure that the actor information matches on the client and server. When Web services are acting as a gateway and they do not have the same actor configured as the request passing through the gateway, Web services do not process the message from a client. Instead, these Web services send the request downstream. The downstream process that contains the correct actor string processes the request. The same situation occurs for the response. Therefore, it is important that you verify that the appropriate client and server actor fields are synchronized.

After you specify which message parts to digitally sign when the client sends a message to a server.

Once you have specified which message parts to digitally sign, you must specify which method is used to digitally sign the message. See "Configuring the client for request signing: choosing the digital signature method" for more information.

***Configuring the client for request signing: choosing the digital signature method:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

Prior to completing these steps, read either of the following topics to become familiar with the Security extensions tab and the Port binding tab in the Web services client editor within an assembly tool such as the Application Server Toolkit or Rational Web Developer:
- "Configuring the client security bindings using an assembly tool" on page 871
- "Configuring the security bindings on a server acting as a client using the administrative console" on page 873

These two tabs are used to configure the Web services security extensions and the Web services security bindings, respectively. You must specify which parts of the message sent by the client must be digitally signed. See "Configuring the client for request signing: digitally signing message parts" on page 857 for more information.

Complete the following steps to specify which message parts to digitally sign when configuring the client for request signing:
1. Launch an assembly tool. For more information on the assembly tools, see Assembly tools.
2. Click **Windows > Open perspective > > Other > J2EE**.
3. Click **Application Client projects >***application_name* **> appClientModule > META-INF**
4. Right-click the `application-client.xml` file, select **Open with > Deployment Descriptor Editor**, and click the WS Binding tab. The Client Deployment Descriptor is displayed.
5. Expand **Security request sender binding configuration > Signing information**.
6. Select **Edit** to view the signing information and select a digital signature method from the Signature method algorithm field. The following table describes the purpose of this information. Some of these definitions are based on the XML-Signature specification, which is located at the following Web site http://www.w3.org/TR/xmldsig-core.

| Name | Purpose |
| --- | --- |
| **Canonicalization method algorithm** | Canonicalizes the `<SignedInfo>` element before the information is digested as part of the signature operation. |

| Name | Purpose |
| --- | --- |
| **Digest method algorithm** | Applies to the data after transforms are applied, if specified, to yield the `<DigestValue>` element. Signing the `<DigestValue>` element binds the resource content to the signer key. The algorithm selected for the client request sender configuration must match the algorithm selected in the server request receiver configuration. |
| **Signature method algorithm** | Converts the canonicalized `<SignedInfo>` element into the `<SignatureValue>` element. The algorithm selected for the client request sender configuration must match the algorithm selected in the server request receiver configuration. |
| **Signing key name** | Represents the key entry associated with the signing key locator. The key entry refers to an alias of the key, which is found in the key store and is used to sign the request. |
| **Signing key locator** | Represents a reference to a key locator implementation class that locates the correct hey store where the alias and the certificate exist. |

**Important:** If you configure the client and server signing information correctly, but receive a `Soap body not signed` error when executing the client, you might need to configure the actor. You can configure the actor in the following locations on the client in the Web services client editor within an assembly tool:

- Click **Security extensions > Client service configuration details** and indicate the actor information in the **Actor URI** field.
- Click **Security extensions > Request sender configuration > Details** and indicate the actor information in the **Actor** field.

You must configure the same actor strings for the Web service on the server, which processes the request and sends the response back. Configure the actor in the following locations in the Web services editor within an assembly tool:

- Click **Security extensions > Server service configuration**.
- Click **Security extensions > Response sender service configuration details > Details** and indicate the actor information in the **Actor** field.

The actor information on both the client and server must refer to the same exact string. When the actor fields on the client and server match, the request or response is acted upon instead of being forwarded downstream. The **actor** fields might be different when you have Web services acting as a gateway to other Web services. However, in all other cases, make sure that the actor information matches on the client and server. When Web services are acting as a gateway and they do not have the same actor configured as the request passing through the gateway, Web services do not process the message from a client. Instead, these Web services send the request downstream. The downstream process that contains the correct actor string processes the request. The same situation occurs for the response. Therefore, it is important that you verify that the appropriate client and server actor fields are synchronized.

You have specified which method is used to digitally sign a message when the client sends a message to a server.

After you configure the client to digitally sign the message, you must configure the server to verify the digital signature. See "Configuring the server for request digital signature verification: Verifying the message parts" for more information.

***Configuring the server for request digital signature verification: Verifying the message parts:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

Prior to completing these steps, read either of the following topics to become familiar with the Extensions tab and the Binding Configurations tab in the Web services editor within the assembly tool such as the Application Server Toolkit or Rational Web Developer:

- “Configuring the server security bindings using an assembly tool” on page 875
- “Configuring the server security bindings using the administrative console” on page 877

You can use these two tabs to configure the Web services security extensions and the Web services security bindings, respectively. Also, you must specify which parts of the message sent by the client must be digitally signed. See “Configuring the client for request signing: digitally signing message parts” on page 857 to determine which message parts are digitally signed. The message parts specified for the client request sender must match the message parts specified for the server request receiver.

Complete the following steps to configure the server for request digital signature verification. The steps describe how to modify the extensions to indicate which parts of the request to verify.

1. Launch an assembly tool. For more information on the assembly tools, see Assembly tools
2. Click **Windows > Open perspective > Other >J2EE**.
3. Click **EJB Projects >***application_name* **> ejbModule > META-INF**.
4. Right-click the `webservices.xml` file, and click **Open with > Web services editor**.
5. Click the Extensions tab in the Web services editor.
6. Expand the **Request receiver service configuration details > Required integrity** section. Required integrity refers to the parts of the message that require digital signature verification. The purpose of digital signature verification is to make sure that the message parts have not been modified while transmitting across the Internet.
7. Indicate parts of the message to verify by clicking **Add**, and selecting one of the following three parts: `body`, `timestamp`, or `SecurityToken`. You can determine which parts of the message to verify by looking at the Web service request sender configuration in the client application. To view the Web service request sender configuration information in the Web services client editor, click the Security extensions tab and expand **Request sender configuration > Integrity**. The following includes a list and description of the message parts:

   **Body** This is the user data portion of the message.

   **Timestamp**
   The `timestamp` determines if the message is valid based on the time that the message is sent and then received. If `timestamp` option is selected, proceed to the next step to `Add Created Time Stamp` to the message.

   **Securitytoken**
   The security token authenticates the client. If the `SecurityToken` is selected, the message is signed.
8. **Optional:** Expand the **Add received time stamp** section. The Add Received Time Stamp value indicates to validate the Add Created Time Stamp option configured by the client. You must select this option if you selected the Add Created Time Stamp on the client. The time stamp ensures message integrity by indicating the timeliness of the request. This option helps defend against replay attacks.

**Important:** If you configure the client and server signing information correctly, but receive a `Soap body not signed` error when executing the client, you might need to configure the actor. You can configure the actor in the following locations on the client in the Web Services Client Editor within the Application Server Toolkit:

- Click **Security extensions > Client service configuration details** and indicate the actor information in the **Actor URI** field.
- Click **Security extensions > Request sender configuration > Details** and indicate the actor information in the **Actor** field.

You must configure the same actor strings for the Web service on the server, which processes the request and sends the response back. Configure the actor in the following locations in the Web Services Editor within the Application Server Toolkit:

- Click **Security extensions > Server service configuration**.
- Click **Security extensions > Response sender service configuration details > Details** and indicate the actor information in the **Actor** field.

The actor information on both the client and server must refer to the same exact string. When the actor fields on the client and server match, the request or response is acted upon instead of being forwarded downstream. The **actor** fields might be different when you have Web services acting as a gateway to other Web services. However, in all other cases, make sure that the actor information matches on the client and server. When Web services are acting as a gateway and they do not have the same actor configured as the request passing through the gateway, Web services do not process the message from a client. Instead, these Web services send the request downstream. The downstream process that contains the correct actor string processes the request. The same situation occurs for the response. Therefore, it is important that you verify that the appropriate client and server actor fields are synchronized.

You have specified which message parts are digitally signed and must be verified by the server when the client sends a message to a server.

After you specify which message parts contain a digital signature that must be verified by the server, you must configure the server to recognize the digital signature method used to digitally sign the message. See “Configuring the server for request digital signature verification: choosing the verification method” for more information.

***Configuring the server for request digital signature verification: choosing the verification method:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

Prior to completing these steps, read either of the following topics to become familiar with the Extensions tab and the Binding Configurations tab in the Web Services Editor within the assembly tool such as the Application Server Toolkit or Rational Web Developer:

- “Configuring the server security bindings using an assembly tool” on page 875
- “Configuring the server security bindings using the administrative console” on page 877

You can use these two tabs to configure the Web services security extensions and Web services security bindings, respectively. You must specify which message parts contain digital signature information that must be verified by the server. See “Configuring the server for request digital signature verification: Verifying the message parts” on page 860. The message parts specified for the client request sender must match the message parts specified for the server request receiver. Likewise, the digital signature method chosen for the client must match the digital signature method used by the server.

Complete the following steps to configure the server for request digital signature verification. The steps describe how to modify the extensions to indicate which digital signature method the server will use during verification.

1. Launch an assembly tool. For more information on the assembly tools, see Assembly tools

2. Click **Windows > Open perspective > Other > J2EE**.
3. Click **EJB Projects >***application_name* **> ejbModule > META-INF**.
4. Right-click the `webservices.xml` file, and click **Open with > Web services editor**.
5. Click the Binding Configurations tab.
6. Expand the **Security request receiver binding configuration details > Signing information** section.
7. Click **Edit** to edit the signing information. The signing information dialog is displayed, select or enter the following information:
   - Canonicalization method algorithm
   - Digest method algorithm
   - Signature method algorithm
   - Use certificate path reference
   - Trust anchor reference
   - Certificate store reference
   - Trust any certificate

   For more conceptual information on digitally signing Simple Object Access Protocol (SOAP) messages, see XML digital signature. The following table describes the purpose for each of these selections. Some of the following definitions are based on the XML-Signature specification, which is located at the following Web address: http://www.w3.org/TR/xmldsig-core.

| Name | Purpose |
|---|---|
| **Canonicalization method algorithm** | Canonicalizes the `<SignedInfo>` element before it is digested as part of the signature operation. The algorithm selected for the server request receiver configuration must match the algorithm selected in the client request sender configuration. |
| **Digest method algorithm** | Applies to the data after transforms are applied, if specified, to yield the `<DigestValue>` element. The signing of the `<DigestValue>` element binds resource content to the signer key. The algorithm selected for the server request receiver configuration must match the algorithm selected in the client request sender configuration. |
| **Signature method algorithm** | Converts the canonicalized `<SignedInfo>` element into the `<SignatureValue>` element. The algorithm selected for the server request receiver configuration must match the algorithm selected in the client request sender configuration. |
| **Use certificate path reference** or **Trust any certificate** | Validates a certificate or signature sent with a message. When a message is signed, the public key used to sign it is sent with the message. This public key or certificate might not be validated at the receiving end. By selecting **User certificate path reference**, you must configure a trust anchor reference and a certificate store reference to validate the certificate sent with the message. By selecting **Trust any certificate**, the signature is validated by the certificate sent with the message without the certificate itself being validated. |
| **Use certificate path reference: Trust anchor reference** | Refers to a key store that contains trusted, self-signed certificates and certificate authority (CA) certificates. These certificates are trusted certificates that you can use with any applications in your deployment. |

| Name | Purpose |
| --- | --- |
| **Use certificate path reference: Certificate store reference** | Contains a collection of X.509 certificates. These certificates are not trusted for all applications in your deployment, but might be used as an intermediary to validate certificates for an application. See |

**Important:** If you configure the client and server signing information correctly, but receive a `Soap body not signed` error when executing the client, you might need to configure the actor. You can configure the actor in the following locations on the client in the Web Services Client Editor within the Application Server Toolkit:

- Click **Security extensions > Client service configuration details** and indicate the actor information in the Actor URI field.
- Click **Security extensions > Request sender configuration > Details** and indicate the actor information in the Actor field.

You must configure the same actor strings for the Web service on the server, which processes the request and sends the response back. Configure the actor in the following locations in the Web Services Editor within the WebSphere Application Server Toolkit:

- Click **Security extensions > Server service configuration**.
- Click **Security extensions > Response sender service configuration details > Details** and indicate the actor information in the Actor field.

The actor information on both the client and server must refer to the same exact string. When the actor fields on the client and server match, the request or response is acted upon instead of being forwarded downstream. The **actor** fields might be different when you have Web services acting as a gateway to other Web services. However, in all other cases, make sure that the actor information matches on the client and server. When Web services are acting as a gateway and they do not have the same actor configured as the request passing through the gateway, Web services do not process the message from a client. Instead, these Web services send the request downstream. The downstream process that contains the correct actor string processes the request. The same situation occurs for the response. Therefore, it is important that you verify that the appropriate client and server actor fields are synchronized.

You have specified the method that the server uses to verify the digital signature in the message parts.

After you configure the client for request signing and the server for request digital signature verification, you must configure the server and the client to handle the response. Next, specify the response signing for the server. See "Configuring the server for response signing: digitally signing message parts" for more information.

***Configuring the server for response signing: digitally signing message parts:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

Prior to completing these steps, read either of the following topics to become familiar with the Extensions tab and the Binding configurations tab in the Web services editor within an assembly tool such as the Application Server Toolkit or Rational Web Developer:

- "Configuring the server security bindings using an assembly tool" on page 875
- "Configuring the server security bindings using the administrative console" on page 877

These two tabs are used to configure the Web services security extensions and the Web services security bindings, respectively.

Complete the following steps to specify which message parts to digitally sign when configuring the server for response signing:

1. Launch an assembly tool. For more information on the assembly tools, see Assembly tools
2. Click **Windows > Open perspective > Other > J2EE**.
3. Click **EJB Projects >***application _name* **>ejbModule > META-INF**
4. Right-click the `webservices.xml` file and click **Open with > Web services editor**.
5. Click the Extensions tab, which is located at the bottom of the Web Services Editor within the assembly tool.
6. Expand **Response sender service configuration details > Integrity**. Integrity refers to digital signature while confidentiality refers to encryption. Integrity decreases the risk of data modification while the data is transmitted across the Internet. For more information on digitally signing SOAP messages, see XML digital signature.
7. Indicate the parts of the message to sign by clicking **Add**, and selecting **Body**, **Timestamp**, or **Securitytoken**.

   The following list contains descriptions of the message parts:

   **Body** The body is the user data portion of the message.

   **Timestamp**
   The time stamp determines if the message is valid based on the time that the message is sent and then received. If **timestamp** if selected, proceed to the next step and click **Add Created Time Stamp**, which indicates that the time stamp is added to the message.

   **Securitytoken**
   The security token is used for authentication. If this option is selected, the authentication information is added to the message.
8. **Optional:** Expand the **Add created time stamp** section. Select this option if you want a time stamp added to the message. You can specify an expiration time for the time stamp, which helps defend against replay attacks. The lexical representation for duration is the ISO 8601 extended format, *PnYnMnDTnHnMnS*, where:
   - *nY* represents the number of years.
   - *nM* represents the number of months.
   - *nD* represents the number of days.
   - *T* is the date and time separator.
   - *nH* represents the number of hours.
   - *nM* represents the number of minutes.
   - *nS* represents the number of seconds. The number of seconds can include decimal digits to arbitrary precision.

   For example, to indicate a duration of 1 year, 2 months, 3 days, 10 hours, and 30 minutes, the format is: `P1Y2M3DT10H30M`. Typically, you configure a message time stamp for about 10 to 30 minutes. 10 minutes is represented as: `P0Y0M0DT0H10M0S`. The *P* character precedes time and date values.

**Important:** If you configure the client and server signing information correctly, but receive a `Soap body not signed` error when executing the client, you might need to configure the actor. You can configure the actor in the following locations on the client in the Web Services Client Editor within the assembly tool:

- Click **Security extensions > Client service configuration details** and indicate the actor information in the **Actor URI** field.
- Click **Security extensions > Request sender configuration > Details** and indicate the actor information in the **Actor** field.

You must configure the same actor strings for the Web service on the server, which processes the request and sends the response back. Configure the actor in the following locations in the Web Services Editor within the WebSphere Application Server Toolkit:

- Click **Security extensions > Server service configuration**.
- Click **Security extensions > Response sender service configuration details > Details** and indicate the actor information in the **Actor** field.

The actor information on both the client and server must refer to the same exact string. When the actor fields on the client and server match, the request or response is acted upon instead of being forwarded downstream. The **actor** fields might be different when you have Web services acting as a gateway to other Web services. However, in all other cases, make sure that the actor information matches on the client and server. When Web services are acting as a gateway and they do not have the same actor configured as the request passing through the gateway, Web services do not process the message from a client. Instead, these Web services send the request downstream. The downstream process that contains the correct actor string processes the request. The same situation occurs for the response. Therefore, it is important that you verify that the appropriate client and server actor fields are synchronized.

You have specified which message parts to digitally sign when the server sends a response to the client.

After you specifying which message parts to digitally sign, you must specify which method is used to digitally sign the message. See "Configuring the server for response signing: choosing the digital signature method" for more information.

***Configuring the server for response signing: choosing the digital signature method:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

Prior to completing these steps, read either of the following topics to become familiar with the Extensions tab and the Binding configurations tab in the Web services editor within the assembly tools such as the Application Server Toolkit or Rational Web Developer:

- "Configuring the server security bindings using an assembly tool" on page 875
- "Configuring the server security bindings using the administrative console" on page 877

These two tabs are used to configure the Web services security extensions and the Web services security bindings, respectively.

Complete the following steps to specify which digital signature method to use when configuring the server for response signing:

1. Launch an assembly tool. For more information on the assembly tools, see Assembly tools.
2. Click **Windows > Open perspective > Other > J2EE**.
3. Click **EJB Projects** >*application _name* >**ejbModule > META-INF**
4. Right-click the `webservices.xml` file and click **Open with > Web services editor**.
5. Click the Binding Configurations tab.
6. Expand **Response sender binding configuration details > Signing information**.
7. Click **Edit** to choose a signing method. The signing info dialog is displayed and either select or enter the following information:
   - **Canonicalization method algorithm**
   - **Digest method algorithm**
   - **Signature method algorithm**

- **Signing key name**
- **Signing key locator**

The following table describes the purpose of this information. Some of these definitions are based on the XML-Signature specification, which is located at the following address: http://www.w3.org/TR/xmldsig-core.

| Name | Purpose |
|---|---|
| **Canonicalization method algorithm** | Canonicalizes the `<SignedInfo>` element before the information is digested as part of the signature operation. Use the same algorithm on the client response receiver. The algorithm selected for the server response sender configuration must match the algorithm selected in the client response receiver configuration. |
| **Digest method algorithm** | Applies to the data after transforms are applied, if specified, to yield the <DigestValue> element. Signing the <DigestValue> binds resource content to the signer key. The algorithm selected for the server response sender configuration must match the algorithm selected in the client response receiver configuration. |
| **Signature method algorithm** | Converts the canonicalized `<SignedInfo>` element into the `<SignatureValue>` element. The algorithm selected for the server response sender configuration must match the algorithm selected in the client response receiver configuration. |
| **Signing key name** | Represents the key entry associated with the signing key locator. The key entry refers to an alias of the key, which is found in the key store and is used to sign the request. |
| **Signing key locator** | Represents a reference to a key locator implementation class that locates the correct key store where the alias and certificate exists. For more information on configuring key locators, see any of the following files:<br>• “Configuring key locators using an assembly tool” on page 845<br>• “Configuring key locators using the administrative console” on page 846 |

You have specified which method is used to digitally sign a message when the server sends a message to a client.

After you configure the server to digitally sign the response message, you must configure the client to verify the digital signature contained in the response message. See “Configuring the client for response digital signature verification: verifying the message parts” for more information.

***Configuring the client for response digital signature verification: verifying the message parts:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

Prior to completing these steps, read either of the following topics to becomes familiar with the WS Extension tab and the WS Binding tab in the Client Deployment Descriptor within the assembly tool such as the Application Server Toolkit or Rational Web Developer:

- “Configuring the client security bindings using an assembly tool” on page 871

- "Configuring the security bindings on a server acting as a client using the administrative console" on page 873

You can use these two tabs to configure the Web services security extensions and the Web services security bindings, respectively.

Complete the following steps to configure the client for response digital signature verification. The steps describe how to modify the extensions to indicate which parts of the response to verify.

1. Launch an assembly tool. For more information on the assembly tools, see Assembly tools.
2. Click **Windows > Open perspective > Other > J2EE**.
3. Click **Application Client projects >***application_name* **> appClientModule > META-INF**.
4. Right-click the `application-client.xml` file and click **Open With > Deployment descriptor editor**.
5. Click the WS extension tab.
6. Expand the **Response receiver configuration > Required integrity** section. Required integrity refers to parts that require digital signature verification. Digital signature verification decreases the risk that the message parts have been modified while the message is transmitted across the Internet.
7. Indicate the parts of the message that must be verified. You can determine which parts of the message to verify by looking at the Web service response sender configuration. Click **Add** and select one of the following parts:

   **Body** The body is the user data portion of the message.

   **Timestamp**
   The time stamp determines if the message is valid based on the time that the message is sent and then received. If the time stamp option is selected, proceed to the next step to add a received time stamp to the message.

   **Securitytoken**
   The security token authenticates the client. If Securitytoken option is selected, the message is signed.
8. **Optional:** Expand the **Add received time stamp** section. Select **Add received time stamp** to add the received time stamp to the message.

**Important:** If you configure the client and server signing information correctly, but receive a `Soap body not signed` error when executing the client, you might need to configure the actor. You can configure the actor in the following locations on the client in the Web services client editor within an assembly tool:

- Click **Security extensions > Client service configuration details** and indicate the actor information in the Actor URI field.
- Click **Security extensions > Request sender configuration > Details** and indicate the actor information in the Actor field.

You must configure the same actor strings for the Web service on the server, which processes the request and sends the response back. Configure the actor in the following locations in the Web services editor within an assembly tool:

- Click **Security extensions > Server service configuration**.
- Click **Security extensions > Response sender service configuration details > Details** and indicate the actor information in the Actor field.

The actor information on both the client and server must refer to the same exact string. When the actor fields on the client and server match, the request or response is acted upon instead of being forwarded downstream. The actor fields might be different when you have Web services acting as a gateway to other Web services. However, in all other cases, make sure that the actor information matches on the client and server. When Web services are acting as

a gateway and they do not have the same actor configured as the request passing through the gateway, Web services do not process the message from a client. Instead, these Web services send the request downstream. The downstream process that contains the correct actor string processes the request. The same situation occurs for the response. Therefore, it is important that you verify that the appropriate client and server actor fields are synchronized.

You have specified which message parts are digitally signed and must be verified by the client when the server sends a response message to the client.

After you specify which message parts contain a digital signature that must be verified by the client, you must configure the client to recognize the digital signature method used to digitally sign the message. See "Configuring the client for response digital signature verification: choosing the verification method" for more information.

***Configuring the client for response digital signature verification: choosing the verification method:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

Prior to completing these steps, read either of the following topics to become familiar with the WS extension tab and the WS binding tab in the Web services editor within an assembly tool such as the Application Server Toolkit or Rational Web Developer:

- "Configuring the server security bindings using an assembly tool" on page 875
- "Configuring the server security bindings using the administrative console" on page 877

You can use these two tabs to configure the Web services security extensions and Web services security bindings, respectively. Also, you must specify which message parts contain digital signature information that must be verified by the client. See "Configuring the client for response digital signature verification: verifying the message parts" on page 867 to specify which message parts are digitally signed by the server and must be verified by the client. The message parts specified for the server response sender must match the message parts specified for the client response receiver. Likewise, the digital signature method chosen for the server must match the digital signature method used by the client.

Complete the following steps to configure the client for response digital signature verification. The steps describe how to modify the extensions to indicate which digital signature method the client will use during verification.

1. Launch an assembly tool. For more information on the assembly tools, see Assembly tools.
2. Click **Windows > Open perspective > Other > J2EE**.
3. Click **Application Client Projects >***application_name* **> appClientModule > META-INF**.
4. Right-click the `application-client.xml` file, select **Open with > Deployment descriptor editor**.
5. Click the WS Binding tab.
6. Expand the **Security response receiver binding configuration > Signing information** section.
7. Click **Edit** to select a digital signature method. The signing info dialog displays and either select or enter the following information:
   - **Canonicalization method algorithm**
   - **Digest method algorithm**
   - **Signature method algorithm**
   - **Signing key name**
   - **Signing key locator**

For more conceptual information on digitally signing SOAP messages, see XML digital signature. The following table describes the purpose for each of these selections. Some of the following definitions are based on the XML-Signature specification, which can be found at: http://www.w3.org/TR/xmldsig-core.

| Name | Purpose |
|---|---|
| **Canonicalization method algorithm** | The canonicalization method algorithm is used to canonicalize the `<SignedInfo>` element before it is digested as part of the signature operation. |
| **Digest method algorithm** | The digest method algorithm is the algorithm applied to the data after transforms are applied, if specified, to yield the `<DigestValue>`. The signing of the `<DigestValue>` binds resource content to the signer key. The algorithm selected for the client response receiver configuration must match the algorithm selected in the server response sender configuration. |
| **Signature method algorithm** | The signature method is the algorithm that is used to convert the canonicalized `<SignedInfo>` element into the `<SignatureValue>` element. The algorithm selected for the client response receiver configuration must match the algorithm selected in the server response sender configuration. |
| **Use certificate path reference** or **Trust any certificate** | When a message is signed, the public key used to sign it is transmitted with the message. To validate this public key at the receiving end, configure a certificate path reference. By selecting **User certificate path reference,** you must configure a trust anchor reference and certificate store reference to validate the certificate sent with the message. By selecting **trust any certificate**, the signature is validated by the certificate sent with the message without the certificate itself being validated. |
| **Use certificate path reference: Trust anchor reference** | A trust anchor is a configuration that refers to a keystore that contains trusted, self-signed certificates and certificate authority (CA) certificates. These certificates are trusted certificates that you can use with any applications in your deployment. |
| **Use certificate path reference: Certificate store reference** | A certificate store is a configuration that has a collection of X.509 certificates. These certificates are not trusted for all applications in your deployment, but might be used as an intermediary to validate certificates for an application. |

**Important:** If you configure the client and server signing information correctly, but receive a `Soap body not signed` error when executing the client, you might need to configure the actor. You can configure the actor in the following locations on the client in the Web services client editor within an assembly tool:

- Click **Security extensions > Client service configuration details** and indicate the actor information in the Actor URI field.
- Click **Security extensions > Request sender configuration > Details** and indicate the actor information in the Actor field.

You must configure the same actor strings for the Web service on the server, which processes the request and sends the response back. Configure the actor in the following locations in the Web services editor within an assembly tool:

- Click **Security extensions > Server service configuration**.
- Click **Security extensions > Response sender service configuration details > Details** and indicate the actor information in the Actor field.

The actor information on both the client and server must refer to the same exact string. When the actor fields on the client and server match, the request or response is acted upon instead of being forwarded downstream. The actor fields might be different when you have Web services acting as a gateway to other Web services. However, in all other cases, make sure that the actor information matches on the client and server. When Web services are acting as a gateway and they do not have the same actor configured as the request passing through the gateway, Web services do not process the message from a client. Instead, these Web services send the request downstream. The downstream process that contains the correct actor string processes the request. The same situation occurs for the response. Therefore, it is important that you verify that the appropriate client and server actor fields are synchronized.

You have specified which method the client uses to verify the digital signature in the message parts.

After you configure the server for response signing and the client for request digital signature verification, verify that you have configured the client and the server to handle the message request.

***Configuring the client security bindings using an assembly tool:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

When configuring a client for Web services security, the bindings describe how to run the security specifications found in the extensions. Use the Web services client editor within an assembly tool to include the binding information in the client enterprise archive (EAR) file.

You can configure the client-side bindings from a pure client accessing a Web service or from a Web service accessing a downstream Web service. This document focuses on the pure client situation. However, the concepts, and in most cases the steps, also apply when a Web service is configured to communicate downstream to another Web service that has client bindings. Complete the following steps to edit the security bindings on a pure client (or server acting as a client) using an assembly tool:

1. Import the Web services client EAR file into an assembly tool. When you edit the client bindings on a server acting as a client, the same basic steps apply. Refer to the Assembly tools documentation for additional information.
2. Click **Windows > Open perspective > Other > J2EE**.
3. Click **Application Client Projects >***application_name* **> appClientModule > META-INF**.
4. Right-click the `application-client.xml` file, select **Open with > Deployment descriptor editor**. The Client Deployment Descriptor is displayed.
5. Click the WS Extension tab.
6. On the WS extension tab, select the Port QName Bindings that you want to configure. The Web services security extensions are configured for outbound requests and inbound responses. You need to configure the following information for Web services security extensions. These topics are discussed in more detail in other sections of the documentation.

   **Request sender configuration details**

   **Details**
   "Configuring the client for request signing: digitally signing message parts" on page 857

   **Integrity**
   "Configuring the client for request signing: digitally signing message parts" on page 857

   **Confidentiality**
   "Configuring the client for request encryption: Encrypting the message parts" on page 886

   **Login Config**

**BasicAuth**
"Configuring the client for basic authentication: specifying the method" on page 900

**IDAssertion**
"Configuring the client for identity assertion: specifying the method" on page 908

**Signature**
"Configuring the client for signature authentication: specifying the method" on page 914

**LTPA** "Configuring the client for LTPA token authentication: specifying LTPA token authentication" on page 931

**ID assertion**
"Configuring the client for identity assertion: specifying the method" on page 908

**Add created time stamp**
"Configuring the client for request signing: digitally signing message parts" on page 857

**Response receiver configuration details**

**Required integrity**
"Configuring the client for response digital signature verification: verifying the message parts" on page 867

**Required confidentiality**
"Configuring the client for response decryption: decrypting the message parts" on page 897

**Add received time stamp**
"Configuring the client for response digital signature verification: verifying the message parts" on page 867

7. On the WS binding tab, select the Port Qualified Name Binding that you want to configure. The Web services security bindings are configured for outbound requests and inbound responses. You need to configure the following information for Web services security bindings. These topics are discussed in more details in other sections of the documentation.

**Security request sender binding configuration**

**Signing information**
"Configuring the client for request signing: choosing the digital signature method" on page 859

**Encryption information**
"Configuring the client for request encryption: choosing the encryption method" on page 887

**Key locators**
"Configuring key locators using an assembly tool" on page 845

**Login binding**

**Basic auth**
"Configuring the client for basic authentication: collecting the authentication information" on page 902

**ID assertion**
"Configuring the client for identity assertion: collecting the authentication method" on page 909

**Signature**
"Configuring the client for signature authentication: collecting the authentication information" on page 915

**LTPA** "Configuring the client for LTPA token authentication: collecting the authentication method information" on page 932

**Security response receiver binding configuration**

**Signing information**
: "Configuring the client for response digital signature verification: choosing the verification method" on page 869

**Encryption information**
: "Configuring the client for response decryption: choosing a decryption method" on page 898

**Trust anchor**
: "Configuring trust anchors using an assembly tool" on page 834

**Certificate store list**
: "Configuring the client-side collection certificate store using an assembly tool" on page 839

**Key locators**
: "Configuring key locators using an assembly tool" on page 845

**Important:** When configuring the security request sender binding configuration, you must synchronize the information used to perform the specified security with the security request receiver binding configuration, which is configured in the server EAR file. These two configurations must be synchronized in all respects because there is no negotiation during run time to determine the requirements of the server.

For example, when configuring the encryption information in the security request sender binding Configuration, you must use the public key from the server for encryption. Therefore, the key locator that you choose must contain the public key from the server configuration. The server must contain the private key to decrypt the message. This example illustrates the important relationship between the client and server configuration. Additionally, when configuring the security response receiver binding configuration, the server must send the response using security information known by this client security response receiver binding configuration.

The following table shows the related configurations between the client and the server. The client request sender and the server request receiver are relative configurations that must be synchronized with each other. The server response sender and the client response receiver are related configurations that must be synchronized with each other. Note that the related configurations are end points for any request or response. One end point must communicate its actions with the other end point because run time requirements are not negotiated.

*Table 18. Related configurations*

| Client configuration | Server configuration |
| --- | --- |
| Request sender | Request receiver |
| Response receiver | Response sender |

***Configuring the security bindings on a server acting as a client using the administrative console:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

When configuring a client for Web services security, the bindings describe how to run the security specifications found in the extensions. Use the Web services client editor within an assembly tool to include the binding information in the client enterprise archive (EAR) file.

You can configure the client-side bindings from a pure client accessing a Web service or from a Web service accessing a downstream Web service. Complete the following steps to find the location in which to

edit the client bindings from a Web service that is running on the server. When a Web service communicates with another Web service, you must configure client bindings to access the downstream Web service.

1. Deploy the Web service using the WebSphere Application Server administrative console by clicking **Applications > Install New Application**. You can access the administrative console by typing `http://localhost:9060/ibm/console` in your Web browser unless you have changed the port number. For more information on installing an application, see Installing a new application
2. Click **Applications > Enterprise applications >** *application_name*.
3. Under Related Items, click either **Web modules** or **EJB modules**, depending upon which type of service is the client to the downstream service.
   - For Web modules, click the Web archive (WAR) file that you configured as the client.
   - For Enterprise JavaBeans (EJB) modules, click the Java archive (JAR) file that you configured as the client.
4. Click the name of the WAR or JAR file.
5. Under Additional Properties, click **Web Services: Client security bindings**. A table displays with the following columns:
   - Component Name
   - Port
   - Web Service
   - Request Sender Binding
   - Request Receiver Binding
   - HTTP Basic Authentication
   - HTTP SSL Configuration

   For Web services security, you must edit the request sender binding and response receiver binding configurations. You can use the defaults for some of the information at the server level. Default bindings are convenient because you can configure commonly reused elements such as key locators once and then reference their aliases in the application bindings.
6. View the default bindings for the server using the administrative console by clicking **Servers > Application server >** *server_name*. Under Additional Properties, click **Web Services: Default bindings for Web services security**. You can configure the following sections. These topics are discussed in more detail in other sections of the documentation.
   - Request sender binding
     - “Signing parameter configuration settings” on page 827
     - “Encryption information configuration settings” on page 734
     - “Key locator configuration settings” on page 695
     - “Login bindings configuration settings” on page 883
   - Response receiver binding
     - “Signing information configuration settings” on page 716
     - “Encryption information configuration settings” on page 729
     - “Trust anchor configuration settings” on page 661
     - “Collection certificate store configuration settings” on page 666
     - “Key locator configuration settings” on page 695

**Important:** When configuring the security request sender binding configuration, you must synchronize the information used to perform the specified security with the security request receiver binding configuration, which is configured in the server EAR file. These two configurations must be synchronized in all respects because there is no negotiation during run time to determine the requirements of the server. For example, when configuring the encryption information in the security request sender binding configuration, you must use the public key from the server for

encryption. Therefore, the key locator that you choose must contain the public key from the server configuration. The server must contain the private key to decrypt the message. This example illustrates the important relationship between the client and server configuration. Additionally, when configuring the security response receiver binding configuration, the server must send the response using security information known by this client security response receiver binding configuration.

The following table shows the related configurations between the client and the server. The client request sender and the server request receiver are relative configurations that must be synchronized with each other. The server response sender and the client response receiver are related configurations that must be synchronized with each other. Note that related configurations are end points for any request or response. One end point must communicate its actions with the other end point because run time requirements are not required.

*Table 19. Related configurations*

| Client configuration | Server configuration |
| --- | --- |
| Request sender | Request receiver |
| Response receiver | Response sender |

***Configuring the server security bindings using an assembly tool:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

Create an Enterprise JavaBean (EJB) file Java archive (JAR) file or a Web archive (WAR) file containing the security binding file (`ibm-webservices-bnd.xmi`) and the security extension file (`ibm-webservices-ext.xmi`). If this archive is acting as a client to a downstream service, you also need the client-side binding file (`ibm-webservicesclient-bnd.xmi`) and the client-side extension file (`ibm-webservicesclient-ext.xmi`). These files are generated using the WSDL2Java command. You can edit these files using the Web services editor in the Assembly tools.

When configuring server-side security for Web services security, the security extensions configuration specifies what security is performed, the security bindings configuration indicates how to perform what is specified in the security extensions configuration. You can use the defaults for some elements at the cell and server levels in the bindings configuration, including key locators, trust anchors, the collection certificate store, trusted ID evaluators, and login mappings and reference these elements from the WAR and JAR binding configurations.

Prior to importing the Web services enterprise archive (EAR) file into the assembly tool, make sure that you have already run the `wsdl2java` command on your Web service to enable your J2EE application. You must import the Web services enterprise archive (EAR) file into the assembly tool.

Open the Web services editor in an assembly tool to begin editing the server security extensions and bindings. The following steps can locate the server security extensions and bindings. Other tasks specify how to configure each section of the extensions and bindings in more detail.

1. Launch an assembly tool. For more information on the assembly tools, see Assembly tools.
2. Click **Windows** > **Open perspective** > **Other** > **J2EE**.
3. Configure the server for inbound requests and outbound responses security configuration. To configure the server for inbound requests and outbound responses, complete the following steps:
   a. Click **EJB Projects** >*application_name* **> ejbModule > META-INF**.

b. Right-click the `webservices.xml` file and click **Open with > Web services editor**. The `webservices.xml` file represents the server-side (inbound) Web services configuration. The `webservicesclient.xml` file represents the client-side (outbound) Web services configuration.

4. In the Web services editor (for the `webservices.xml` file and inbound requests and outbound responses Web services configuration), there are several tabs at the bottom of the editor including Web Services, Port Components, Handlers, Security Extensions, Bindings, and Binding Configurations. The security extensions are edited using the Security Extensions tab. The security bindings are edited using the Security Bindings tab.

a. Click the WS Extensions tab and select the port component binding to edit. The Web services security extensions are configured for inbound requests and outbound responses. You need to configure the following information for Web services security extensions. These topics are discussed in more detail in other topics in the documentation.

**Request receiver service configuration details**

**Required integrity**
"Configuring the server for request digital signature verification: Verifying the message parts" on page 860

**Required confidentiality**
"Configuring the server for request decryption: decrypting the message parts" on page 890

**Login config**

**Basic auth**
"Configuring the server to handle BasicAuth authentication information" on page 904

**ID assertion**
"Configuring the server to handle identity assertion authentication" on page 910

**Signature**
"Configuring the server to support signature authentication" on page 917

**LTPA** "Configuring the server to handle LTPA token authentication information" on page 933

**Add received time stamp**
"Configuring the server for request digital signature verification: Verifying the message parts" on page 860

**Response sender service configuration details**

**Details**
"Configuring the server for response signing: digitally signing message parts" on page 864

**Integrity**
"Configuring the server for response signing: digitally signing message parts" on page 864

**Confidentiality**
"Configuring the server for response encryption: encrypting the message parts" on page 894

**Add created time stamp-**
"Configuring the server for response signing: digitally signing message parts" on page 864

b. Click the Binding Configurations tab and select the port component binding to edit. The Web services security bindings are configured for inbound requests and outbound responses. You need to configure the following information for Web services security bindings. These topics are discussed in more details in other topics in the documentation.

**Response receiver binding configuration details**

**Signing Information**
: "Configuring the server for request digital signature verification: choosing the verification method" on page 862

**Encryption Information**
: "Configuring the server for request decryption: choosing the decryption method" on page 891

**Trust Anchor**
: "Configuring trust anchors using an assembly tool" on page 834

**Certificate Store List**
: "Configuring the server-side collection certificate store using an assembly tool" on page 840

**Key Locators**
: "Configuring key locators using an assembly tool" on page 845

**Login Mapping**

**Basic auth**
: "Configuring the server to validate BasicAuth authentication information" on page 905

**ID assertion**
: "Configuring the server to validate identity assertion authentication information" on page 911

**Signature**
: "Configuring the server to validate signature authentication information" on page 918

**LTPA** "Configuring the server to validate LTPA token authentication information" on page 933

**Trusted ID evaluator**

**Trusted ID evaluator reference**

**Response sender binding configuration details**

**Signing information**
: "Configuring the server for response signing: choosing the digital signature method" on page 866

**Encryption information**
: "Configuring the server for response encryption: choosing the encryption method" on page 894

**Key locators**
: "Configuring key locators using an assembly tool" on page 845

Configure the client for outbound requests and inbound responses security configuration by right-clicking the `webservicesclient.xml` file and clicking **Open With > Deployment descriptor editor**. For more information, see "Configuring the client security bindings using an assembly tool" on page 871.

***Configuring the server security bindings using the administrative console:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

Create an Enterprise JavaBean (EJB) file Java archive (JAR) file or Web archive (WAR) file containing the security binding file (ibm-webservices-bnd.xmi) and the security extension file (ibm-webservices-ext.xmi). If this archive is acting as a client to a downstream service, you also need the client-side binding file (ibm-webservicesclient-bnd.xmi) and the client-side extension file (ibm-webservicesclient-ext.xmi). These files are generated using the WSDL2Java command command. You can edit these files using the Web Services Editor in the Assembly tools.

When configuring server-side security for Web services security, the security extensions configuration specifies what security is to be performed while the security bindings configuration indicates how to perform what is specified in the security extensions configuration. You can use the defaults for some elements at the cell and server levels in the bindings configuration, including key locators, trust anchors, the collection certificate store, trusted ID evaluators, and login mappings and reference them from the WAR and JAR binding configurations.

The following steps describe how to edit bindings for a Web service after these bindings are deployed on a server. When one Web service communicates with another Web service, you also must configure the client bindings to access the downstream Web service.

1. Deploy the Web service using the WebSphere Application Server administrative console. The Administrative Console is accessible by typing http://localhost:*9060*/ibm/console in a Web browser. After you log into the administration console, click **Applications > Install new application** to deploy the Web service. For more information, see Installing application files with the console.
2. After you deploy the Web service, click **Applications > Enterprise applications >** *application _name*.
3. Under Related Items, click either **Web modules** or **EJB modules** depending on which service you want to configure.
   a. If you select **Web modules**, click the WAR file that you want to edit.
   b. If you select **EJB modules**, click the JAR file that you want to edit.
4. After you select a WAR or JAR file, under Additional properties, click **Web services: client security bindings** for outbound requests and inbound responses. Click **Web services: server security bindings** for inbound requests and outbound responses.
5. If you click **Web services: server security bindings**, the following sections can be configured. These topics are discussed in more detail in other sections of the documentation.
   - Request receiver binding
     - Signing information
     - Encryption information
     - Trust anchors
     - Collection certificate store
     - Key locator
     - Trusted ID evaluator
     - Login mappings
   - Response sender binding
     - Signing parameters
     - Encryption information
     - Key locator

## XML encryption

XML encryption is a specification developed by World Wide Web (WWW) Consortium (W3C) in 2002 that contains the steps to encrypt data, the steps to decrypt encrypted data, the XML syntax to represent encrypted data, the information used to decrypt the data, and a list of encryption algorithms such as triple DES, AES, and RSA.

You can apply XML encryption to an XML element, XML element content, and arbitrary data, including an XML document. For example, suppose that you need to encrypt the CreditCard element shown in the example 1.

**Example 1: Sample XML document**

```
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <CreditCard Limit='5,000' Currency='USD'>
    <Number>4019 2445 0277 5567</Number>
    <Issuer>Example Bank</Issuer>
    <Expiration>04/02</Expiration>
  </CreditCard>
</PaymentInfo>
```

**Example 2: XML document with a common secret key**

Example 2 shows the XML document after encryption. The EncryptedData element represents the encrypted CreditCard element. The EncryptionMethod element describes the applied encryption algorithm, which is triple DES in this example. The KeyInfo element contains the information to retrieve a decryption key, which is a KeyName element in this example. The CipherValue element contains the ciphertext obtained by serializing and encrypting the CreditCard element.

```
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <EncryptedData Type='http://www.w3.org/2001/04/xmlenc#Element'
     xmlns='http://www.w3.org/2001/04/xmlenc#'>
    <EncryptionMethod
       Algorithm='http://www.w3.org/2001/04/xmlenc#tripledes-cbc'/>
       <KeyInfo xmlns='http://www.w3.org/2000/09/xmldsig#'>
         <KeyName>John Smith</KeyName>
       </KeyInfo>
       <CipherData>
         <CipherValue>ydUNqHkMrD...</CipherValue>
       </CipherData>
  </EncryptedData>
</PaymentInfo>
```

**Example 3: XML document encrypted with the public key of the recipient**

In example 2, it is assumed that both the sender and recipient have a common secret key. If the recipient has a public and private key pair, which is most likely the case, the CreditCard element can be encrypted as shown in example 3. The EncryptedData element is the same as the EncryptedData element found in Example 2. However, the KeyInfo element contains an EncryptedKey .

```
<PaymentInfo xmlns='http://example.org/paymentv2'>
  <Name>John Smith</Name>
  <EncryptedData Type='http://www.w3.org/2001/04/xmlenc#Element'
    xmlns='http://www.w3.org/2001/04/xmlenc#'>
    <EncryptionMethod
      Algorithm='http://www.w3.org/2001/04/xmlenc#tripledes-cbc'/>
    <KeyInfo xmlns='http://www.w3.org/2000/09/xmldsig#'>
      <EncryptedKey xmlns='http://www.w3.org/2001/04/xmlenc#'>
        <EncryptionMethod
          Algorithm='http://www.w3.org/2001/04/xmlenc#rsa-1_5'/>
        <KeyInfo xmlns='http://www.w3.org/2000/09/xmldsig#'>
          <KeyName>Sally Doe</KeyName>
        </KeyInfo>
        <CipherData>
          <CipherValue>yMTEyOTA1M...</CipherValue>
        </CipherData>
      </EncryptedKey>
    </KeyInfo>
    <CipherData>
```

```
      <CipherValue>ydUNqHkMrD...</CipherValue>
    </CipherData>
  </EncryptedData>
</PaymentInfo>
```

### XML Encryption in the WSS-Core

WSS-Core specification is under development by Organization for the Advancement of Structured Information Standards (OASIS). The specification describes enhancements to Simple Object Access Protocol (SOAP) messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. The message confidentiality is realized by encryption based on XML Encryption.

The WSS-Core specification supports encryption of any combination of body blocks, header blocks, their sub-structures, and attachments of a SOAP message. The specification also requires that when you encrypt parts of a SOAP message, you mprepend a reference from the security header block to the encrypted parts of the message. The reference can be a clue for a recipient to identify which encrypted parts of the message to decrypt.

The XML syntax of the reference varies according to what information is encrypted and how it is encrypted. For example, suppose that the CreditCard element in example 4 is encrypted with either a common secret key or the public key of the recipient.

#### Example 4: Sample SOAP message

```
<SOAP-ENV:Envelope
  SOAP-ENV:encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
  xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'>
  <SOAP-ENV:Body>
    <PaymentInfo xmlns='http://example.org/paymentv2'>
      <Name>John Smith</Name>
      <CreditCard Limit='5,000' Currency='USD'>
        <Number>4019 2445 0277 5567</Number>
        <Issuer>Example Bank</Issuer>
        <Expiration>04/02</Expiration>
      </CreditCard>
    </PaymentInfo>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The resulting SOAP messages are shown in Examples 5 and 6. In these example, the ReferenceList and EncryptedKey elements are used as references, respectively.

#### Example 5: SOAP message encrypted with a common secret key

```
<SOAP-ENV:Envelope
  SOAP-ENV:encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
  xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'>
  <SOAP-ENV:Header>
    <Security SOAP-ENV:mustUnderstand='1'
      xmlns='http://schemas.xmlsoap.org/ws/2003/06/secext'>
      <ReferenceList xmlns='http://www.w3.org/2001/04/xmlenc#'>
        <DataReference URI='#ed1'/>
      </ReferenceList>
    </Security>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <PaymentInfo xmlns='http://example.org/paymentv2'>
      <Name>John Smith</Name>
      <EncryptedData Id='ed1'
        Type='http://www.w3.org/2001/04/xmlenc#Element'
        xmlns='http://www.w3.org/2001/04/xmlenc#'>
        <EncryptionMethod
```

```
          Algorithm='http://www.w3.org/2001/04/xmlenc#tripledes-cbc'/>
        <KeyInfo xmlns='http://www.w3.org/2000/09/xmldsig#'>
          <KeyName>John Smith</KeyName>
        </KeyInfo>
        <CipherData>
          <CipherValue>ydUNqHkMrD...</CipherValue>
        </CipherData>
      </EncryptedData>
    </PaymentInfo>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

**Example 6: SOAP message encrypted with the public key of the recipient**

```
<SOAP-ENV:Envelope
  SOAP-ENV:encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
  xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'>
  <SOAP-ENV:Header>
    <Security SOAP-ENV:mustUnderstand='1'
      xmlns='http://schemas.xmlsoap.org/ws/2003/06/secext'>
      <EncryptedKey xmlns='http://www.w3.org/2001/04/xmlenc#'>
        <EncryptionMethod
          Algorithm='http://www.w3.org/2001/04/xmlenc#rsa-1_5'/>
        <KeyInfo xmlns='http://www.w3.org/2000/09/xmldsig#'>
          <KeyName>Sally Doe</KeyName>
        </KeyInfo>
        <CipherData>
          <CipherValue>yMTEyOTA1M...</CipherValue>
        </CipherData>
        <ReferenceList>
          <DataReference URI='#ed1'/>
        </ReferenceList>
      </EncryptedKey>
    </Security>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <PaymentInfo xmlns='http://example.org/paymentv2'>
      <Name>John Smith</Name>
      <EncryptedData Id='ed1'
        Type='http://www.w3.org/2001/04/xmlenc#Element'
        xmlns='http://www.w3.org/2001/04/xmlenc#'>
        <EncryptionMethod
          Algorithm='http://www.w3.org/2001/04/xmlenc#tripledes-cbc'/>
        <CipherData>
          <CipherValue>ydUNqHkMrD...</CipherValue>
        </CipherData>
      </EncryptedData>
    </PaymentInfo>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

### Relationship to digital signature

The WSS-Core specification also provides message integrity, which is realized by a digital signature based on the XML-Signature specification.

A combination of encryption and digital signature over common data introduces cryptographic vulnerabilities.

### Important distinction between Version 5.x and Version 6 applications

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

## Securing Web services for version 5.x applications using XML encryption

### Important distinction between Version 5.x and Version 6 applications

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

WebSphere Application Server provides several different methods to secure your Web services. Extensible Markup Language (XML) encryption is one of these methods. You can secure your Web services using any of the following methods:

- XML digital signature
- XML encryption
- Basicauth authentication
- Identity assertion authentication
- Signature authentication
- Pluggable token

XML encryption enables you to encrypt an XML element, the content of an XML element, or arbitrary data such as an XML document. Like XML digital signature, a message is sent by the client as the request sender to the server as the request receiver. The response is sent by the server as the response sender to the client as the request receiver. Unlike XML digital signature, which verifies the authenticity of the sender, XML encryption scrambles the message content using a key, which can be unscrambled by a receiver that possesses the same key. You can use XML encryption in conjunction with XML digital signature to scramble the content while verifying the authenticity of the message sender.

To use XML encryption to secure Web services, you must use an assembly tool. For more information, see Assembly tools

To securing Web services for version 5.x applications using XML encryption, complete the following steps:

1. Specify the encryption settings for the request sender. The message parts and the encryption method settings chosen for the request sender on the client must match the message parts and the method settings chosen for the request receiver on the server. To specify the encryption settings for the request sender:
   a. “Configuring the client for request encryption: Encrypting the message parts” on page 886.
   b. “Configuring the client for request encryption: choosing the encryption method” on page 887.
2. Specify the encryption settings for the request receiver. The decryption settings chosen for the request receiver must match the encryption settings chosen for the request sender.

   To specify the decryption settings for the request receiver:
   a. “Configuring the server for request decryption: decrypting the message parts” on page 890.
   b. “Configuring the server for request decryption: choosing the decryption method” on page 891.
3. Specify the encryption settings for the response sender. The message parts and the encryption method settings chosen for the response sender on the server must match the message parts and the method settings chosen for the response receiver on the client. To specify the encryption settings for the response sender:
   a. “Configuring the server for response encryption: encrypting the message parts” on page 894.
   b. “Configuring the server for response encryption: choosing the encryption method” on page 894.
4. Specify the encryption settings for the response receiver.

   **Remember:** The decryption settings chosen for the response receiver must match the encryption settings chosen for the response sender.

   To specify the decryption settings for the response receiver, complete the following steps:

a. “Configuring the client for response decryption: decrypting the message parts” on page 897.
b. “Configuring the client for response decryption: choosing a decryption method” on page 898.

After completing these steps, you have secured your Web services using XML encryption.

***Login bindings configuration settings:***

Use this page to configure the encryption and decryption parameters.

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

The pluggable token uses the Java Authentication and Authorization Service (JAAS) CallBackHandler (javax.security.auth.callback.CallBackHandler) interface to generate the token that is inserted into the message. The following list describes the CallBack support implementations:

**com.ibm.wsspi.wssecurity.auth.callback.BinaryTokenCallback**
This implementation is used for generating binary tokens inserted as `<wsse:BinarySecurityToken/@ValueType>` in the message.

**javax.security.auth.callback.NameCallback and javax.security.auth.callback.PasswordCallback**
This implementation is used for generating user name tokens inserted as `<wsse:UsernameToken>` in the message.

**com.ibm.wsspi.wssecurity.auth.callback.XMLTokenSenderCallback**
This implementation is used to generate Extensible Markup Language (XML) tokens and is inserted as the `<SAML: Assertion>` element in the message.

**com.ibm.wsspi.wssecurity.auth.callback.PropertyCallback**
This implementation is used to obtain properties that are specified in the binding file.

To view this administrative console page, complete the following steps:

1. Click **Applications > Enterprise Applications >***application_name*.
2. Under Related Items, click **EJB modules** or **Web modules >***URI_file_name* **> Web Services: Client security bindings**.
3. Under Request Sender Bindings, click **Edit**.
4. Under Additional properties, click **Login binding**.

If the encryption information is not available, select **None**.

If the encryption information is available, select **Dedicated login binding** and specify the configuration in the following fields:

*Authentication method:*

Specifies the unique name for the authentication method.

You can uses any string to name the authentication method. However, the string must match the element in the server-level configuration. The following words are reserved by WebSphere Application Server:

**BasicAuth**
This method uses both a user name and a password.

**IDAssertion**
This method uses a user name, but it requires that additional trust is established by the receiving server using a trusted ID evaluator mechanism.

**Signature**
This method uses the distinguished name (DN) of the signer.

**LTPA** This method validates the token.

*Callback handler:*

Specifies the name of the callback handler. The callback handler must implement the javax.security.auth.callback.CallbackHandler interface.

*Basic authentication user ID:*

Specifies the user name for basic authentication. With the basic authentication method, you can define a user name and a password in the binding file.

*Basic authentication password:*

Specifies the password for basic authentication.

*Token type URI:*

Specifies the namespace Uniform Resource Identifiers (URI), which denotes the type of security token that is accepted.

The value of this field if is impacted by the following conditions:
- If binary security tokens are accepted, the value denotes the ValueType attribute in the element. The ValueType element identifies the type of security token and its namespace.
- If Extensible Markup Language (XML) tokens are accepted, the value denotes the top-level element name of the XML token.
- The Token type URI field is ignored if the reserved words, which are listed in the description of the Authentication method field, are specified.

This information is inserted as `<wsse:BinarySecurityToken>/ValueType` for the <SAML: Assertion> XML token.

*Token type local name:*

Specifies the local name of the security token type. For example, X509v3.

The value of this field if is impacted by the following conditions:
- If binary security tokens are accepted, the value denotes the ValueType attribute in the element. The ValueType element identifies the type of security token and its namespace.
- If Extensible Markup Language (XML) tokens are accepted, the value denotes the top-level element name of the XML token.
- The Token type URI field is ignored if the reserved words, which are listed in the description of the Authentication method field, are specified.

This information is inserted as `<wsse:BinarySecurityToken>/ValueType` for the <SAML: Assertion> XML token.

***Request sender:***

The security handler on the request sender side of the SOAP message enforces the security constraints, located in the `ibm-webservicesclient-ext.xmi` file, and bindings, located in the `ibm-webservicesclient-bnd.xmi file`. These constraints and bindings apply both to J2EE application clients or when Web

services is acting as a client. The security handler acts on the security constraints before sending the SOAP message. For example, the security handler might digitally sign the message, encrypt the message, create a time stamp, or insert a security token.

The security handler on the request sender side of the Simple Object Access Protocol (SOAP) message enforces the security constraints, located in the `ibm-webservicesclient-ext.xmi` file, and the bindings, located in the `ibm-webservicesclient-bnd.xmi` file. These constraints and bindings apply both to J2EE application clients or when Web services is acting as a client. The security handler acts on the security constraints before sending the SOAP message. Request sender security constraints must match the security constraint requirements defined in the request receiver. For example, the security handler might digitally sign the message, encrypt the message, create a time stamp, or insert a security token. You can specify the following security requirements for the request sender and apply them to the SOAP message:

**Integrity (digital signature)**
You can select multiple parts of a message to sign digitally. The following list contains the integrity options:
- Body
- Time stamp
- Security token

**Confidentiality (encryption)**
You can select multiple parts of a message to encrypt. The following list contains the confidentiality options:
- Body content
- Username token

**Security token**
You can insert only one token into the message. The following list contains the security token options:
- Basic authentication, which requires both a user name and a password
- Identity assertion, which requires a user name only
- X.509 binary security token
- Lightweight Third Party Authentication (LTPA) binary security token
- Custom token , which is pluggable and supports custom-defined tokens in the SOAP message

**Timestamp**
You can have a time stamp to indicate the timeliness of the message.
- Timestamp

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

*Request sender binding collection:*

Use this page to specify the binding configuration to send request messages for Web services security.

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

To view this administrative console page, complete the following steps:
1. Click **Applications > Enterprise applications >***application_name*.

2. Under Related items, click **EJB modules** or **Web modules** > *URI_file_name*
3. Under Additional properties, click **Web services: Client security bindings**.
4. Under Request sender binding, click **Edit**.

*Web services security namespace:* Specifies the namespace that is used by Web services security to send a request. However, this field configures the namespace value only and does not enforce the semantics of the specification related to the namespace. Web services security uses the processing semantic only in draft 13 of the OASIS specification. The following schemas are available:

- http://schemas.xmlsoap.org/ws/2003/06/secext
- http://schemas.xmlsoap.org/ws/2002/07/secext
- http://schemas.xmlsoap.org/ws/2002/04/secext
- None

The namespace used by the response sender is based on the namespace of the incoming message in the request receiver.

*Signing information:*

Specifies the configuration for the signing parameters. Signing information is used to sign and validate parts of the message including the body and time stamp.

You can also use these parameters for X.509 validation when the Authentication method is `IDAssertion` and the ID Type is `X509Certificate`, in the server-level configuration. In such cases, you must fill in the Certificate Path fields only.

*Encryption information:*

Specifies the configuration for the encrypting and decrypting parameters. Encryption information is used for encrypting and decrypting various parts of a message, including the body and user name token.

*Key locators:*

Specifies a list of key locator objects that retrieve the keys for digital signature and encryption from a keystore file or a repository. The key locator maps a name or a logical name to an alias or maps an authenticated identity to a key. This logical name is used to locate a key in a key locator implementation.

*Login mappings:*

Specifies a list of configurations for validating tokens within incoming messages.

Login mappings map the authentication method to the Java Authentication and Authorization Service (JAAS) configuration.

To configure JAAS, complete the following steps:

1. Click **Security > Global security**.
2. Under Authentication, click **JAAS Configuration**.

***Configuring the client for request encryption: Encrypting the message parts:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

Prior to completing these steps, read either of the following topics to familiarize yourself with the WS Extensions tab and the WS Binding tab in the Client Deployment Descriptor Editor within an assembly tool:

- "Configuring the client security bindings using an assembly tool" on page 871
- "Configuring the security bindings on a server acting as a client using the administrative console" on page 873

These two tabs are used to configure the Web services security extensions and Web services security bindings, respectively.

Complete the following steps to specify which message parts to encrypt when configuring the client for request encryption:

1. Launch an assembly tool. For more information on the assembly tools, see Assembly tools.
2. Open the J2EE perspective by clicking **Window > Open perspective > Other > J2EE**.
3. Click **Application Client Projects >***application_name* **> appClientModule > META-INF**.
4. Right-click the `application-client.xml` file, select **Open with > Deployment descriptor editor**.
5. Click the WS extensions tab, which is located at the bottom of Client Deployment Descriptor Editor within the assembly tool.
6. Expand **Request sender configuration > Confidentiality**. Confidentiality refers to encryption while integrity refers to digital signing. Confidentiality reduces the risk of someone understanding the message flowing across the Internet. With confidentiality specifications, the message is encrypted before it is sent and decrypted when it is received at the correct target. For more information on encrypting , see XML encryption.
7. Select the parts of the message that you want to encrypt by clicking **Add**. You can select one of the following parts:

   **Bodycontent**
   User data portion of the message

   **Usernametoken**
   Basic authentication information, if selected

After you specify which message parts to encrypt, you must specify which method to use to encrypt the request message. See "Configuring the client for request encryption: choosing the encryption method" for more information.

***Configuring the client for request encryption: choosing the encryption method:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

Prior to completing these steps, read either of the following topics to familiarize yourself with the WS Extensions tab and the WS Binding tab in the Client Deployment Descriptor editor within an assembly tool:

- "Configuring the client security bindings using an assembly tool" on page 871
- "Configuring the security bindings on a server acting as a client using the administrative console" on page 873

These two tabs are used to configure the Web services security extensions and Web services security bindings, respectively.

Complete the following steps to specify which encryption method to use when configuring the client for request encryption:

1. Launch an assembly tool. For more information on the assembly tools, see Assembly tools.

2. Open the J2EE perspective by clicking **Window > Open perspective > Other > J2EE**.
3. Click **Windows > Open perspective > Other > J2EE**.
4. Click **Application Client Projects >***application_name* **> appClientModule > META-INF**.
5. Right-click the `application-client.xml` file, select **Open with > Deployment descriptor editor**.
6. Click the WS binding tab, which is located at the bottom of the Client Deployment Descriptor editor within the assembly tool.
7. Expand **Security request sender binding configuration > Encryption information**.
8. Select an encryption option and click **Edit** to view the encryption information or click **Add** to add another option. The following table describes the purpose of this information. Some of these definitions are based on the XML-Encryption specification, which is located at the following Web address: http://www.w3.org/TR/xmlenc-core

   **Encryption name**
   : Refers to the name of the encryption information entry.

   **Data encryption method algorithm**
   : Encrypts and decrypts data in fixed size, multiple octet blocks.

   **Key encryption method algorithm**
   : Represents public key encryption algorithms that are specified for encrypting and decrypting keys.

   **Encryption key name**
   : Represents a Subject (Owner field of the certificate) from a public key certificate found by the encryption key locator, which is used by the key encryption method algorithm to encrypt the private key. The private key is used to encrypt the data.

     The key chosen must be a public key of the target. Encryption must be done using the public key and decryption must be done by the target using the private key (the personal certificate of the target).

   **Encryption key locator**
   : Represents a reference to a key locator implementation class that locates the correct key store where the alias and the certificate exist. For more information on configuring key locators, see "Configuring key locators using an assembly tool" on page 845 and "Configuring key locators using the administrative console" on page 846.

For more information, see "Configuring key locators using an assembly tool" on page 845 and "Configuring key locators using the administrative console" on page 846.

You must specify which parts of the request message to encrypt. See "Configuring the client for request encryption: Encrypting the message parts" on page 886 if you have not previously specified this information.

#### *Request receiver:* Important distinction between Version 5.x and Version 6 applications

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

The security handler on the request receiver side of the Simple Object Access Protocol (SOAP) message enforces the security specifications defined in the IBM extension deployment descriptor (`ibm-webservices-ext.xmi`) and bindings (`ibm-webservices-bnd.xmi`). The request receiver defines the security requirement of the SOAP message. The security constraint for request sender must match the security requirement of the request receiver for the server to accept the request. If the incoming SOAP message does not meet all the security requirements defined, then the request is rejected with the appropriate fault code returned to the sender. For security tokens, the token is validated using Java Authentication and Authorization Service (JAAS) login configuration and authenticated identity is set as the identity for the downstream invocation.

For example, if there is a security requirement to have the SOAP body digitally signed by Joe Smith and if the SOAP body of the incoming SOAP message is not signed by Joe Smith, then the request is rejected.

You can define the following security requirements for the request receiver:

**Required integrity (digital signature)**
: You can select multiple parts of a message to sign digitally. The following list contains the integrity options:
  - Body
  - Time stamp
  - Security token

**Required confidentiality (encryption)**
: You can select multiple parts of a message to encrypt. The following list contains the confidentiality options:
  - Body content
  - Token

  You can have multiple security tokens. The following list contains the security token options:
  - Basic authentication, which requires both a user name and a password
  - Identity assertion, which requires a user name only
  - X.509 binary security token
  - Lightweight Third Party Authentication (LTPA) binary security token
  - Custom token, which is pluggable and supports custom-defined tokens validated by the JAAS login configuration

**Received time stamp**
: You can have a time stamp for checking the timeliness of the message.
  - Time stamp

*Request receiver binding collection:*

Use this page to specify the binding configuration to receive request messages for Web services security.

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

To view this administrative console page, complete the following steps:
1. Click **Applications > Enterprise applications >***application_name*.
2. Under Related items, click **EJB modules** or **Web modules** > *URI_file_name*
3. Under Additional properties, click **Web services: Server security bindings**.
4. Under Request receiver binding, click **Edit**.

*Signing information:*

Specifies the configuration for the signing parameters. Signing information is used to sign and validate parts of a message including the body, the timestamp, and the user name token.

You also can use these parameters for X.509 certificate validation when the authentication method is `IDAssertion` and the ID Type is `X509Certificate` in the server-level configuration. In such cases, you must fill in the Certificate Path fields only.

*Encryption information:*

Specifies the configuration for the encrypting and decrypting parameters. This configuration is used to encrypt and decrypt parts of the message that include the body and the user name token.

*Trust anchors:*

Specifies a list of keystore objects that contain the trusted root certificates that are issued by a certificate authority (CA).

The certificate authority authenticates a user and issues a certificate. The CertPath API uses the certificate to validate the certificate chain of incoming, X.509-formatted security tokens or trusted, self-signed certificates.

*Collection certificate store:*

Specifies a list of the untrusted, intermediate certificate files.

The collection certificate store contains a chain of untrusted, intermediate certificates.The CertPath API attempts to validate these certificates, which are based on the trust anchor.

*Key locators:*

Specifies a list of key locator objects that retrieve the keys for digital signature and encryption from a keystore file or a repository. The key locator maps a name or a logical name to an alias or maps an authenticated identity to a key. This logical name is used to locate a key in a key locator implementation.

*Trusted ID evaluators:*

Specifies a list of trusted ID evaluators that determine whether to trust the identity-asserting authority or message sender.

The trusted ID evaluators are used to authenticate additional identities from one server to another server. For example, a client sends the identity of user A to server 1 for authentication. Server 1 calls downstream to server 2, asserts the identity of user A, and includes the user name and password of server 1. Server 2 attempts to establish trust with server 1 by authenticating its user name and password and checking the trust based on the TrustedIDEvaluator implementation. If the authentication process and the trust check are successful, server 2 trusts that server 1 authenticated user A and a credential is created for user A on server 2 to invoke the request.

*Login mappings:*

Specifies a list of configurations for validating tokens within incoming messages.

Login mappings map the authentication method to the Java Authentication and Authorization Service (JAAS) configuration.

To configure JAAS, complete the following steps:

1. Click **Security > Global security**.
2. Under Authentication, click **JAAS Configuration**.

***Configuring the server for request decryption: decrypting the message parts:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

Complete this task to specify which parts of the request message must be decrypted by the server. You must know which parts of the request message the client encrypts because the server must decrypt the same message parts.

Prior to completing these steps, read either of the following topics to become familiar with the WS Extensions tab and the WS Binding configurations tab:

- "Configuring the server security bindings using an assembly tool" on page 875
- "Configuring the server security bindings using the administrative console" on page 877

These two tabs are used to configure the Web services security extensions and Web services security bindings, respectively.

Complete the following steps to configure the request receiver extensions:

1. Launch an assembly tool. For more information on the assembly tools, see Assembly tools.
2. Open the J2EE perspective by clicking **Window > Open perspective > Other > J2EE**.
3. Click **EJB Projects >***application_name* **ejbModule > META_INF**.
4. Right-click the `webservices.xml` file, and click **Open with > Web services editor**.
5. Click the Extensions tab, which is located at the bottom of the Web services editor within the assembly tool.
6. Expand the **Request receiver service configuration details > Required confidentiality** section.
7. Select the parts of the message to decrypt. The message parts selected for the request decryption on the server must match the message parts selected for the message encryption on the client. Click **Add** and select either of the following message parts:

   **bodycontent**
   : The user data section of the message.

   **usernametoken**
   : This token is the basic authentication information.

After you specify which parts of the request message to decrypt, you must specify the method to use decrypt the message. See "Configuring the server for request decryption: choosing the decryption method" for more information.

***Configuring the server for request decryption: choosing the decryption method:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

Prior to completing these steps, read either of the following topics to become familiar with the WS Extensions tab and the WS Bindings tab:

- "Configuring the server security bindings using an assembly tool" on page 875
- "Configuring the server security bindings using the administrative console" on page 877

These two tabs are used to configure the Web services security extensions and Web services security bindings, respectively.

Complete this task to specify which decryption method is used by the server to decrypt the request message. You must know which decryption method the client uses because the server must use the same method.

1. Launch an assembly tool. For more information on the assembly tools, see Assembly tools.
2. Open the J2EE perspective by clicking **Window > Open perspective > Other > J2EE**.
3. Click **EJB Projects >***application_name* **ejbModule > META_INF**.
4. Right-click the `webservices.xml` file, select **Open with > Web services editor**.
5. Click the Binding Configurations tab, which is located at the bottom of the Web services editor within the assembly tool.
6. Expand the **Request receiver binding configuration details > Encryption information** section.
7. Click **Edit** to view the encryption information. The following table describes the purpose for each of these selections. Some definitions are taken from the XML-Encryption specification , which is located at the following Web address: http://www.w3.org/TR/xmlenc-core

   **Encryption name**
      Represents the name of this encryption information entry; an alias for the entry.

   **Data encryption method algorithm**
      Encrypts and decrypts data in fixed size, multiple octet blocks. This algorithm must be the same as the algorithm selected in the client request sender configuration.

   **Key encryption method algorithm**
      Represents algorithms specified for encrypting and decrypting keys. This algorithm must be the same as the algorithm selected in the client request sender configuration.

   **Encryption key name**
      Represents a Subject from a personal certificate, which is typically a distinguished name (DN) that is found by the encryption key locator. The subject is used by the key encryption method algorithm to decrypt the secret key, and the secret key is used to decrypt the data.

      The key chosen must be a private key in the key store configured by the key locator. The key requires the same Subject used by the client to encrypt the data. Encryption must be done using the public key and decryption by using the private key (personal certificate). To ensure that the client encrypts the data with the correct public or private key, you must extract the public key from the server key store and add it to the key store specified in the encryption configuration information for the client request sender.

      For example, the personal certificate of a server is `CN=Bob, O=IBM, C=US`. Therefore the server contains the public and private key pair. The client sending the request should encrypt the data using the public key for `CN=Bob, O=IBM, C=US`. The server decrypts the data using the private key for `CN=Bob, O=IBM, C=US`.

   **Encryption key locator**
      Represents a reference to a key locator implementation class that finds the correct keystore where the alias and the certificate exist. For more information on configuring key locators, go to the following sections: "Configuring key locators using an assembly tool" on page 845 and "Configuring key locators using the administrative console" on page 846.

It is important to note that for decryption, the encryption key name chosen must refer to a personal certificate that can be located by the key locator of the server referenced in the encryption information. Enter the Subject of the personal certificate here, which is typically a Distinguished Name (DN). The Subject uses the default key locator to find the key. If a custom key locator is written, the encryption key name can be anything used by the key locator to find the correct encryption key. The encryption key locator references the implementation class that finds the correct key store where this alias and certificate exist. Refer to "Configuring key locators using an assembly tool" on page 845 and "Configuring key locators using the administrative console" on page 846 for more information.

You must specify which parts of the request message to decrypt. See “Configuring the server for request decryption: decrypting the message parts” on page 890 if you have not previously specified this information.

***Response sender:*** **Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

The response sender defines the security requirements of the Simple Object Access Protocol (SOAP) response message. The security handler acts on the security constraints defined for the response in the IBM extension deployment descriptors, located in the `ibm-webservices-ext.xmi` file and the bindings, located in the `ibm-webservices-bnd.xmi` file. The security handler signs, encrypts, or generates the time stamp for the SOAP response message before the response is send to the caller.

**Integrity constraints (digital signature)**
You can select which parts of the message are digitally signed.

- Body
- Time stamp

**Confidentiality (encryption)**
You can encrypt the body content of the message.

**Time stamp**
You can have a time stamp for checking the timeliness of the message.

The security constraints that apply to the SOAP response message must match the security requirements defined in the response receiver. Otherwise, the response is rejected by the response receiver (caller).

*Response sender binding collection:*

Use this page to specify the binding configuration for sender response messages for Web services security.

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

To view this administrative console page, complete the following steps:

1. Click **Applications > Enterprise applications >***application_name*.
2. Under Related items, click **EJB modules** or **Web modules** > *URI_file_name*
3. Under Additional properties, click **Web services: Server security bindings**.
4. Under Response sender binding, click **Edit**.

*Signing information:*

Specifies the configuration for the signing parameters.

You also can use these parameters for X.509 certificate validation when the authentication method is IDAssertion and the ID Type is X509Certificate in the server-level configuration. In such cases, you must fill-in the Certificate Path fields only.

*Encryption information:*

Specifies the configuration for the encryption and decryption parameters.

*Key locators:*

Specifies a list of key locator objects that retrieve the keys for a digital signature and encryption from a keystore file or a repository. The key locator maps a name or logical name to an alias or maps an authenticated identity to a key. This logical name is used to locate a key in a key locator implementation.

***Configuring the server for response encryption: encrypting the message parts:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

Prior to completing these steps, read either of the following topics to become familiar with the WS Extensions tab and the WS Bindings tab in the Web services editor within an assembly tool:
- "Configuring the server security bindings using an assembly tool" on page 875
- "Configuring the server security bindings using the administrative console" on page 877

These two tabs are used to configure the Web services security extensions and the Web services security bindings, respectively.

Complete the following steps to specify which parts of the response message to encrypt when configuring the server for response encryption:
1. Launch an assembly tool. For more information on the assembly tools, see Assembly tools.
2. Open the J2EE perspective by clicking **Window > Open perspective > Other > J2EE**.
3. Click **EJB Projects** >*application_name* **ejbModule > META_INF**.
4. Right-click the `webservices.xml` file, select **Open with > Web services editor**.
5. Click the Extensions tab, which is located at the bottom of the Web Services Editor within the assembly tool.
6. Expand **Response sender service configuration details > Confidentiality**. Confidentiality refers to encryption while integrity refers to digital signing. Confidentiality reduces the risk of someone understanding the message flowing across the Internet. With confidentiality specifications, the response is encrypted before it is sent and decrypted when it is received at the correct target. For more information on encrypting, see "XML encryption" on page 878.
7. Select the parts of the response that you want to encrypt by clicking **Add** and selecting **Bodytoken** or **Usernametoken**. The following information describes the message parts:

   **`Bodycontent`**
   : User data portion of the message.

   **`Usernametoken`**
   : Basic authentication information, if selected.

     A user name token does not appear in the response so you do not need to select this option for the response. If you select this option, make sure that you also select it for the client response receiver. If you do not select this option, make sure that you do not select it for the client response receiver.

After you specify which message parts to encrypt, you must specify which method to use message encryption. See "Configuring the server for response encryption: choosing the encryption method" for more information.

***Configuring the server for response encryption: choosing the encryption method:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

Prior to completing these steps, read either of the following topics to become familiar with the Extensions tab and the Binding configurations tab in the Web services editor within an assembly tool:

- "Configuring the server security bindings using an assembly tool" on page 875
- "Configuring the server security bindings using the administrative console" on page 877

These two tabs are used to configure the Web services security extensions and Web services security bindings, respectively.

Complete the following steps to specify which method the server uses to encrypt the response message:

1. Launch an assembly tool. For more information on the assembly tools, see Assembly tools.
2. Click **Windows > Open perspective > Other > J2EE**.
3. Click **EJB Projects >***application_name* **ejbModule > META_INF**.
4. Right-click the `webservices.xml` file, and click **Open with > Web services editor**.
5. Click the Binding Configurations tab, which is located at the bottom of the Web Services Editor within the assembly tool.
6. Expand **Response sender binding configuration details > Encryption information**.
7. Click **Edit** to view the encryption information. The following table describes the purpose of this information. Some of these definitions are based on the XML-Encryption specification, which is located at the following Web address: http://www.w3.org/TR/xmlenc-core

   **Encryption name**
   Refers to the name of the encryption information entry.

   **Data encryption method algorithm**
   Encrypts and decrypts data in fixed size, multiple octet blocks. The algorithm selected for the server response sender configuration must match the algorithm selected in the client response receiver configuration.

   **Key encryption method algorithm**
   Represents public key encryption algorithms that are specified for encrypting and decrypting keys. The algorithm selected for the server response sender configuration must match the algorithm selected in the client response receiver configuration.

   **Encryption key name**
   Represents a Subject from a public key certificate typically distinguished name (DN) that is found by the encryption key locator and used by the key encryption method algorithm to encrypt the private key. The private key is used to encrypt the data.

   The key name chosen in the server response sender encryption information must be the public key of the key configured in the client response receiver encryption information. Encryption by the response sender must be done using the public key and decryption must be done by the response receiver using the associated private key (the personal certificate of the response receiver).

   **Encryption key locator**
   The encryption key locator represents a reference to a key locator implementation class that finds the correct key store where the alias and the certificate exist. For more information on configuring key locators, see "Configuring key locators using an assembly tool" on page 845 and "Configuring key locators using the administrative console" on page 846.

The encryption key name chosen must refer to a public key of the response receiver. For the encryption key name, use the Subject of the public key certificate, typically a Distinguished Name (DN). The name chosen is used by the default key locator to find the key. If you write a custom key locator , the encryption key name might be anything used by the key locator to find the correct encryption key (a public key). The

encryption key locator references the implementation class that finds the correct key store where the alias and certificate exist. For more information, see "Configuring key locators using an assembly tool" on page 845 and "Configuring key locators using the administrative console" on page 846.

You must specify which parts of the response message to encrypt. See "Configuring the server for response encryption: encrypting the message parts" on page 894 if you have not previously specified this information.

***Response receiver:*** **Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

The response receiver defines the security requirements of the response received from a request to a Web service. The security constraints for response sender must match the security requirements of the response receiver. If the constraints do not match, the response is not accepted by the caller or the sender. The security handler enforces the security constraints based on the security requirements defined in the IBM extension deployment descriptor, located in the `ibm-webservicesclient-ext.xmi` file and in the bindings, located in the `ibm-webservicessclient-bnd.xmi` file.

For example, the security requirement might have the response Simple Object Access Protocol (SOAP) body encrypted. If the SOAP body of the SOAP message is not encrypted, the response is rejected and the appropriate fault code is communicated back to the caller of the Web services.

You can specify the following security requirements for a response receiver:

**Required integrity (digital signature)**
: You can select which parts of a message are digitally signed. The following list contains the integrity options:
  - Body
  - Time stamp

**Required confidentiality (encryption)**
: You can encrypt the body content of the message.

**Received time stamp**
: You can have a time stamp for checking the timeliness of the message.

*Response receiver binding collection:*

Use this page to specify the binding configuration for receiver response messages for Web services security.

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

To view this administrative console page, complete the following steps:

1. Click **Applications > Enterprise applications >***application_name*.
2. Under Related items, click **EJB modules** or **Web modules** > *URI_file_name* > **Web Services: Client security bindings**.
3. Under Response receiver binding, click **Edit**.

*Signing information:*

Specifies the configuration for the signing parameters. Signing information is used to sign and to validate parts of the message including the body and the timestamp.

You can also use these parameters for X.509 validation when the authentication method is `IDAssertion` and the ID type is `X509Certificate`, in the server-level configuration. In such cases, you must fill in the certificate path fields only.

*Encryption information:*

Specifies the configuration for the encryption and decryption parameters.

Encryption information is used for encrypting and decrypting various parts of a message, including the body and the user name token.

*Trust anchors:*

Specifies a list of keystore objects that contain the trusted root certificates that are self-signed or issued by a certificate authority.

The certificate authority authenticates a user and issues a certificate. After the certificate is issued, the keystore objects, which contain these certificates, use the certificate for certificate path or certificate chain validation of incoming X.509-formatted security tokens.

*Collection certificate store:*

Specifies a list of the untrusted, intermediate certificate files.

The collection certificate store contains a chain of untrusted, intermediate certificates. The CertPath API attempts to validate these certificates, which are based on the trust anchor.

*Key locators:*

Specifies a list of key locator objects that retrieve the keys for a digital signature and encryption from a keystore file or a repository.

The key locator maps a name or a logical name to an alias or maps an authenticated identity to a key. This logical name is used to locate a key in a key locator implementation.

***Configuring the client for response decryption: decrypting the message parts:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

Prior to completing these steps, read either of the following topics to become familiar with the WS Extensions tab and the WS Binding tab in the Client Deployment Descriptor Editor within an assembly tool:

- “Configuring the client security bindings using an assembly tool” on page 871
- “Configuring the security bindings on a server acting as a client using the administrative console” on page 873

These two tabs are used to configure the Web services security extensions and the Web services security bindings, respectively.

Complete the following steps to specify which response message parts to decrypt when configuring the client for response decryption. The server response encryption and client response decryption configurations must match.

1. Launch an assembly tool. For more information on the assembly tools, see Assembly tools.
2. Open the J2EE perspective by clicking **Window > Open perspective > Other > J2EE**.
3. Click **Application Client Projects >***application_name* **> appClientModule > META-INF**.
4. Right-click the `application-client.xml` file, select **Open with > Deployment descriptor editor**.
5. Click the WS Extensions tab, which is located at the bottom of the deployment descriptor editor within the assembly tool.
6. Expand the **Response receiver configuration > Required confidentiality** section.
7. Select the parts of the message that you must decrypt by clicking **Add** and selecting either **Bodycontent** or **Usernametoken**. The following information describes these message parts:

   **Bodycontent**
      The user data portion of the message.

   **Usernametoken**
      The basic authentication information, if selected.

   The information selected in this step is encrypted by the server in the response sender.

   **Important:** A username token is typically not sent in the response. Thus, you usually do not need to select username token.

After you specify which message parts to decrypt, you must specify which method to use when decrypting the response message. See "Configuring the client for response decryption: choosing a decryption method" for more information.

***Configuring the client for response decryption: choosing a decryption method:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

Prior to completing these steps, read either of the following topics to become familiar with the WS Extensions tab and the WS Bindings tab in the Client Deployment Descriptor Editor within an assembly tool such as the Application Server Toolkit or Rational Web Developer:

- "Configuring the client security bindings using an assembly tool" on page 871
- "Configuring the security bindings on a server acting as a client using the administrative console" on page 873

These two tabs are used to configure the Web services security extensions and Web services security bindings, respectively.

Complete the following steps to specify which decryption method to use when the client decrypts the response message. The server response encryption and client response decryption configurations must match.

1. Launch an assembly tool. For more information on the assembly tools, see Assembly tools.
2. Click **Windows > Open perspective > Other > J2EE**.
3. Click **Application Client Projects >***application_name* **> appClientModule > META-INF**.
4. Right-click the `application-client.xml` file, select **Open with > Deployment descriptor editor**.
5. Click the WS Binding tab, which is located at the bottom of the deployment descriptor editor within the assembly tool.

6. Expand the **Security response receiver binding configuration > Encryption information** section. For more information on encrypting and decrypting Simple Object Access Protocol (SOAP) messages, see “XML encryption” on page 878.
7. Click **Edit** to view the encryption information. The following table describes the purpose for this information. Some of these definitions are based on the XML-Encryption specification, which is located at the following Web address: http://www.w3.org/TR/xmlenc-core

   **Encryption name**
   Refers to the alias used for the encryption information entry.

   **Data encryption method algorithm**
   Encrypts and decrypts data in fixed size, multiple octet blocks.

   **Key encryption method algorithm**
   Represents public key encryption algorithms specified for encrypting and decrypting keys.

   **Encryption key name**
   Represents a Subject from a personal certificate, which is typically a distinguished name (DN) that is found by the encryption key locator. The Subject is used by the key encryption method algorithm to decrypt the secret key. The secret key is used to decrypt the data.

   **Important:** The key chosen must be a private key of the client. Encryption must be done using the public key and decryption must be done by the private key (personal certificate). For example, the personal certificate of the client is: `CN=Alice, O=IBM, C=US`. Therefore, the client contains the public and private key pair. The target server that sends the response encrypts the secret key using the public key for `CN=Alice, O=IBM, C=US`. The client decrypts the secret key using the private key for CN=Alice, O=IBM, C=US

   .

   **Encryption key locator**
   The encryption key locator represents a reference to a key locator implementation class that finds the correct key store where the alias and the certificate exist. For more information on configuring key locators, see “Configuring key locators using an assembly tool” on page 845 and “Configuring key locators using the administrative console” on page 846.

For decryption, the encryption key name chosen must refer to a personal certificate that can be located by the client key locator. The Subject (owner field of the certificate) of the personal certificate should be entered in the Encryption key name, this is typically a Distinguished Name (DN). The default key locator uses the Encryption key name to find the key within the keystore. If you write a custom key locator, the encryption key name can be anything used by the key locator to find the correct encryption key. The encryption key locator references the implementation class that locates the correct key store where this alias and certificate exists. For more information, see “Configuring key locators using an assembly tool” on page 845 and “Configuring key locators using the administrative console” on page 846.

You must specify which parts of the request message to decrypt. See the topic“Configuring the client for response decryption: decrypting the message parts” on page 897 if you have not previously specified this information.

## Securing Web services for version 5.x applications using basicauth authentication

### Important distinction between Version 5.x and Version 6 applications

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

WebSphere Application Server provides several different methods to secure your Web services. Extensible Markup Language (XML) digital signature is one of these methods. You might also secure your Web services using any of the following methods:

- XML digital signature
- XML encryption
- Basicauth authentication
- Identity assertion authentication
- Signature authentication
- Pluggable token

With the basicauth authentication method, the request sender generates a basicauth security token using a callback handler. The request receiver retrieves the basicauth security token from the SOAP message and validates it using a Java Authentication and Authorization Service (JAAS) login module. Trust is established using user name and password validation. To use basicauth authentication to secure Web services, complete the following tasks:

1. Secure the client for basicauth authentication.
   a. Configure the client for basicauth authentication: Specifying the method
   b. Configure the client for basicauth authentication: Collecting the authentication information
2. Secure the server for basicauth authentication.
   a. Configure the server to handle basicauth authentication
   b. Configure the server to validate basicauth authentication information

After completing these steps, you have secured your Web services using basicauth authentication.

***Configuring the client for basic authentication: specifying the method:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

BasicAuth refers to the user ID and password of a valid user in the registry of the target server. BasicAuth information can be collected in many ways including through an administrative console prompt, a standard in (Stdin) prompt, or specified in the bindings, which prevents user interaction. For more information on BasicAuth authentication, see: "BasicAuth authentication method" on page 901.

**Attention:** WebSphere Application Server supports nonce (randomly generated token) with BasicAuth authentication. For more information, see Nonce.

Complete the following steps to specify BasicAuth as the authentication method:

1. Launch an assembly tool. For more information on the assembly tools, see Assembly tools.
2. Click **Windows > Open perspective > Other > J2EE**.
3. Click **Application Client Projects** >*application_name* **> appClientModule > META-INF**.
4. Right-click the `application-client.xml` file, select **Open with > Deployment descriptor editor**.
5. Click the WS Extensions tab, which is located at the bottom of the deployment descriptor editor within the assembly tool.
6. Expand the **Request sender configuration > Login configuration** section. The only valid login configuration choices for a pure client are BasicAuth and Signature.
7. Select **BasicAuth** to authenticate the client using a user ID and a password. This user ID and password must be specified in the target user registry. The other choice, Signature, attempts to authenticate the client using the certificate used to digitally sign the message.

For more information on getting started with the Web services client editor within the assembly tool, see either of the following topics:

- “Configuring the client security bindings using an assembly tool” on page 871
- “Configuring the security bindings on a server acting as a client using the administrative console” on page 873

After you specify the BasicAuth authentication method, you must specify how to collect the authentication information. See “Configuring the client for basic authentication: collecting the authentication information” on page 902.

*BasicAuth authentication method:*

When you use the BasicAuth authentication method, the security token that is generated is a <wsse:UsernameToken> element with <wsse:Username> and <wsse:Password> elements.

WebSphere Application Server supports text passwords but not password digest because passwords are not stored and cannot be retrieved from the server. On the request sender side, a callback handler is invoked to generate the security token. On the request receiver side, a Java Authentication and Authorization Service (JAAS) login module is used to validate the security token. These two operations, token generation and token validation, are described in the following sections.

**BasicAuth token generation**

The request sender generates a BasicAuth security token using a callback handler. The security token returned by the callback handler is inserted in the Simple Object Access Protocol (SOAP) message. The callback handler that is used is specified in the <LoginBinding> element of the bindings file, `ibm-webservicesclient-bnd.xmi` . The following callback handler implementations are provided with WebSphere Application Server and can be used with the BasicAuth authentication method:

- com.ibm.wsspi.wssecurity.auth.callback.GUIPromptCallbackHandler
- com.ibm.wsspi.wssecurity.auth.callback.StdinPromptCallbackHandler
- com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler

You can add your own callback handlers that implement the `javax.security.auth.callback.CallbackHandler` method.

**BasicAuth token validation**

The request receiver retrieves the BasicAuth security token from the SOAP message and validates it using a JAAS login module. The <wsse:Username> and <wsse:Password> elements in the security token are used to perform the validation. If the validation is successful, the login module returns a JAAS Subject. This Subject is set as the identity of the running thread. If the validation fails, the request is rejected with a SOAP fault exception.

The JAAS login configuration is specified in the <LoginMapping> element of the bindings file. Default bindings are specified in the `ws-security.xml` file. However, you can override these bindings using the application-specific `ibm-webservices-bnd.xmi` file. The configuration information consists of a CallbackHandlerFactory and a ConfigName value. The CallbackHandlerFactory option specifies the name of a class that is used for creating the JAAS CallbackHandler object. WebSphere Application Server provides the com.ibm.wsspi.wssecurity.auth.callback.WSCallbackHandlerFactoryImpl CallbackHandlerFactory implementation. The ConfigName value specifies a JAAS configuration name entry. WebSphere Application Server searches the `security.xml` file for a matching configuration name entry. If a match is not found, it searches the `wsjaas.conf` file for a match. WebSphere Application Server provides the WSLogin default configuration entry, which is suitable for the BasicAuth authentication method.

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

***Configuring the client for basic authentication: collecting the authentication information:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

BasicAuth refers to the user ID and the password of a valid user in the registry of the target server. Collection of BasicAuth information can occur in many ways including through a user interface prompt, a standard in (Stdin) prompt, or specified in the bindings, which prevents user interaction. For more information on BasicAuth authentication, see "BasicAuth authentication method" on page 901.

Complete this task to specify the authentication information needed for BasicAuth authentication:

1. Launch an assembly tool. For more information on the assembly tools, see Assembly tools.
2. Click **Windows > Open perspective > Other > J2EE**.
3. Click **Application Client Projects >***application_name* **> appClientModule > META-INF**.
4. Right-click the `application-client.xml` file, select **Open with > Deployment descriptor editor**.
5. Click the WS Binding tab, which is located at the bottom of deployment descriptor editor within the assembly tool such as the Application Server Toolkit or Rational Web Developer.
6. Expand the **Security request sender binding configuration > Login binding** section.
7. Click **Edit** or **Enable** to view the login binding information. The login binding information displays and enter the following information:

   **Authentication method**
   Specifies the type of authentication. Select **BasicAuth** to use basic authentication.

   **Token value type URI and Token value type local name**
   When you select BasicAuth, you cannot edit the token value type URI and the local name values. Specifies values for custom authentication types. For BasicAuth authentication, leave these values blan.

   **Callback handler**
   Specifies the Java Authentication and Authorization Server (JAAS) callback handler implementation for collecting the BasicAuth information. You can use the following default implementations for the callback handler:

   **com.ibm.wsspi.wssecurity.auth.callback.StdinPromptCallbackHandler**
   This implementation is used for non-user interface console prompts.

   **com.ibm.wsspi.wssecurity.auth.callback.GUIPromptCallbackHandler**
   This implementation is used for user interface panel prompts.

   **com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler**
   This implementation is used when you plan to always enter the user ID and password in the BasicAuth user ID and password section that follows.

   **Basic Authentication user ID and Basic Authentication password**
   Specifies values for the BasicAuth user ID and password, regardless of the default callback handler indicated previously, these user ID and password values are used to authenticate to the server for the Web services security authentication. If you leave these values blank, use either the GUIPromptCallbackHandler or the StdinPromptCallbackHandler implementation, but only on a pure client. Always fill-in these values for any Web service that acts as a client to another Web service that you want to specify for BasicAuth for authentication downstream. If you want the client identity of the originator to flow downstream, configure the Web service client to use either ID assertion or Lightweight Third Party Authentication (LTPA).

**Property**
: Specifies properties with name and value pairs for custom callback handlers to use. For BasicAuth authentication, you do not need to enter any information. To enter a new property, click **Add** and enter the new property and value.

**Other basic authentication entries**: There is a basic authentication entry in the Port Qualified Name Binding Details section. This entry is used for HTTP transport authentication, which might be required if the router servlet is protected.

Information specified in the Web services security basic authentication section overrides the basic authentication information specified in the Port Qualified Name Binding Details section for authorizing the Web service.

For a server that acts as a client, do not specify a user interface or non-user interface prompt callback handler. To configure BasicAuth authentication from one Web service to a downstream Web service, select the `com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHander` implementation and explicitly specify the BasicAuth user ID and password. If you want the client identity of the originator to flow downstream, configure the Web service client to use ID assertion.

To use the BasicAuth authentication method, you must specify the method in the Login configuration section of the assembly tool . See “Configuring the client for basic authentication: specifying the method” on page 900 if you have not previously specified this information.

*Identity assertion authentication method:*

When using the identity assertion (IDAssertion) authentication method, the security token generated is a <wsse:UsernameToken> element that contains a <wsse:Username> element.

On the request sender side, a callback handler is invoked to generate the security token. On the request receiver side, the security token is validated. These two operations, token generation and token validation operations, are described in the following sections.

**Identity assertion token validation**:

The request receiver retrieves the IDAssertion security token from the SOAP message and validates it using a Java Authentication and Authorization Service (JAAS) login module. With identity assertion, special processing is required to establish trust before asserting the identity as the established identity of the running thread. This special processing is defined by the <IDAssertion> element in the deployment descriptor file, `ibm-webservices-ext.xmi`. If all the validation checks are successful, the asserted identity is set as the identity of the running thread of. If the validation fails, the request is rejected with a SOAP fault exception.

The JAAS login configuration is specified in the<LoginMapping> element of the bindings file. Default bindings are specified in the `ws-security.xml` file. However, you can override these bindings using the application specific `ibm-webservices-bnd.xmi` file. The configuration information consists of CallbackHandlerFactory and a ConfigName. CallbackHandlerFactory specifies the name of a class that is used for creating the JAAS CallbackHandler object. WebSphere Application Server provides the `com.ibm.wsspi.wssecurity.auth.callback.WSCallbackHandlerFactoryImpl` CallbackHandlerFactory implementation. ConfigName specifies a JAAS configuration name entry.

WebSphere Application Server searches the `security.xml` file for a matching configuration name entry. If a match is not found it searches the `wsjaas.conf` file. WebSphere Application Server provides the system.wssecurity.IDAssertion default configuration entry, which is suitable for the identity assertion authentication method.

The <IDAssertion> element in the `ibm-webservices-ext.xmi` deployment descriptor file specifies the special processing required when using the identity assertion authentication method. The <IDAssertion> element is composed of two sub-elements: <IDType> and <TrustMode>.

The <IDType> element specifies the method for asserting the identity. The supported values for asserting the identity are:

- Username
- Distinguished name (DN)
- X.509 certificate

When <IDType> is *username*, a username token (for example, Bob) is provided. This user name is mapped to a user in the user registry and is the asserted identity after successful trust validation. When the <IDType> value is *DN*, a user name token containing a distinguished name is provided (for example, `cn=Bob Smith, o=ibm, c=us`). This DN is mapped to a user in the user registry and this user is the asserted identity after successful trust validation. When the <IDType> is *X509Certificate*, a binary security token containing an X509 certificate is provided and the SubjectDN value from the certificate (for example, `cn=Bob Smith, o=ibm, c=us`) is extracted. This SubjectDN value is mapped to a user in the user registry and this user is the asserted identity after successful trust validation.

The `<TrustMode>` element specifies how the trust authority, or asserting authority, provides trust information. The supported values are:

- Signature
- BasicAuth
- No value specified

When the <TrustMode> value is `Signature` the signature is validated. Then, the signer (for example, `cn=IBM Authority, o=ibm, c=us`) is mapped to an identity in the user registry (for example, IBMAuthority). To ensure that the asserting authority is trusted, the mapped identity (for example, IBMAuthority) is validated against a list of trusted identities. When the <TrustMode> element is `BasicAuth`, there is a user name token with a user name and password, which is the user name and password of the asserting authority.

The user name and password are validated. If they are successfully validated, that user name (for example, IBMAuthority) is validated against a list of trusted identities. If a value is not specified for `<TrustMode>`, trust is presumed and additional trust validation is not performed. This type of identity assertion is called *presumed trust mode*. Use the presumed trust mode only in an environment where the trust is established using some other mechanism.

If all the validations described previously succeed, the asserted identity (for example, Bob) is set as the identity of the running thread. If any of the validations fail, the request is rejected with a SOAP fault exception.

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

***Configuring the server to handle BasicAuth authentication information:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

BasicAuth refers to the user ID and the password of a valid user in the registry of the target server. After a request is received that contains basic authentication information, the server needs to log in to form a credential. The credential is used for authorization. If the user ID and the password supplied are not valid, an exception is thrown and the request ends without invoking the resource. For more information on BasicAuth authentication, see "BasicAuth authentication method" on page 901.

Complete the following steps to configure the server to handle BasicAuth authentication information:
1. Launch an assembly tool. For more information on the assembly tools, see Assembly tools.
2. Open the J2EE perspective by clicking **Window > Open perspective > Other > J2EE**.
3. Click **EJB Projects >***application_name* **> ejbModule > META-INF**.
4. Right-click the `webservices.xml` file, and click **Open with > Web services editor**.
5. Click the Extensions tab, which is located at the bottom of the Web services editor within an assembly tool such as the Application Server Toolkit or Rational Web Developer.
6. Expand the **Request receiver service configuration details > Login configuration** section. You can select the following options:
   - BasicAuth
   - Signature
   - ID assertion
   - Lightweight Third Party Authentication (LTPA)

   .
7. Select **BasicAuth** to authenticate the client with a user ID and a password. The client must specify a valid user ID and password in the server user registry.

   You can select multiple login configurations, which means that different types of security information might be received at the server. The order in which the login configurations are added decides the order in which they are processed when a request is received. Problems can occur if you have multiple login configurations added that have security tokens in common. For example, ID assertion contains a BasicAuth token. For ID assertion to work properly, list ID assertion ahead of BasicAuth in the processing list or the BasicAuth processing overrides the IDAssertion processing.

After you specify how the server handles BasicAuth authentication information, you must specify how the server validates the authentication information. See "Configuring the server to validate BasicAuth authentication information" for more information.

***Configuring the server to validate BasicAuth authentication information:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

BasicAuth refers to the user ID and the password of a valid user in the registry of the target server. Once a request is received that contains basic authentication information, the server needs to log in to form a credential. The credential is used for authorization. If the user ID and the password supplied is invalid, an exception is thrown and the request ends without invoking the resource. For more information on BasicAuth authentication, see "BasicAuth authentication method" on page 901.

Complete the following steps to specify how the server validates the BasicAuth authentication information:
1. Launch an assembly tool. For more information on the assembly tools, see Assembly tools.
2. Open the J2EE perspective by clicking **Window > Open perspective > Other > J2EE**.
3. Click **EJB Projects >***application_name* **> ejbModule > META-INF**.
4. Right-click the `webservices.xml` file, and click **Open with > Web services editor**.

5. Click the Binding Configurations tab, which is located at the bottom of the Web services editor within an assembly tool such as the Application Server Toolkit or Rational Web Developer.
6. Expand the **Request receiver binding configuration details > Login mapping** section.
7. Click **Edit** to view the login mapping information or click **Add** to add new login mapping information. The login mapping dialog is displayed. Select or enter the following information:

   **Authentication method**
   Specifies the type of authentication that occurs. Select **BasicAuth** to use basic authentication.

   **Configuration name**
   Specifies the Java Authentication and Authorization Service (JAAS) login configuration name. For the BasicAuth authentication method, enter `WSLogin` for the JAAS login Configuration name.

   **Use token valid type**
   Determines if you want to specify a custom token type. For the default authentication method selections, you do not need to specify this option.

   **Token value type URI and Token value type URI local name**
   When you select BasicAuth, you cannot edit the token value type URI and local name values. Specifies custom authentication types. For BasicAuth authentication leave these fields blank.

   **Callback handler factory class name**
   Creates a JAAS CallbackHandler implementation that understands the following callbacks:
   - javax.security.auth.callback.NameCallback
   - javax.security.auth.callback.PasswordCallback
   - com.ibm.wsspi.wssecurity.auth.callback.BinaryTokenCallback
   - com.ibm.wsspi.wssecurity.auth.callback.XMLTokenReceiverCallback
   - com.ibm.wsspi.wssecurity.auth.callback.PropertyCallback

   **Callback handler factory property name and Callback handler factory property value**
   Specifies callback handler properties for custom callback handler factory implementations. You do not need to specify any properties for the default callback handler factory implementation. For BasicAuth, you do not need to enter any property values.

   **Login mapping property name and Login mapping property value**
   Specifies properties for a custom login mapping. For the default implementations including BasicAuth, leave these fields blank.

You must specify how the server handles the BasicAuth authentication method. See “Configuring the server to handle BasicAuth authentication information” on page 904 if you have not previously specified this information.

## Identity assertion

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

Identity assertion is a method for expressing the identity of the sender (for example, user name) in a SOAP message. When identity assertion is used as an authentication method, the authentication decision is performed based only on the name of the identity, and on other information such as passwords and certificates.

**ID type**

The Web Services Security implementation in WebSphere Application Server can handle these identity types:

**User name**
: Denotes the user name, such as the one in the local operating system (for example, ″alice″). This name is embedded in the <Username> element within the <UsernameToken> element.

**DN**
: Denotes the distinguished name (DN) for the user, such as ″`CN=alice, O=IBM, C=US`″. This name is embedded in the <Username> element within the <UsernameToken> element.

**X.509 certificate**
: Represents the identity of the user as an X.509 certificate instead of a string name. This certificate is embedded in the `<BinarySecurityToken>` element.

**Managing trust**

The intermediary host in the SOAP message itinerary can assert claimed identity of the initial sender. Two methods (called *trust mode*) are supported for this assertion:

**Basic authentication**
: The intermediary adds its user name and password pair to the message.

**Signature**
: The intermediary digitally signs the <UsernameToken> element of the initial sender.

**Note:** This trust mode does not support the X.509 certificate ID type.

**Typical scenario**

ID assertion is typically used in the multihop environment where the SOAP message passes through one or more intermediary hosts. The intermediary host authenticates the initial sender. The following scenario describes the process:

1. The initial sender sends a SOAP message to the intermediary host with some embedded authentication information. This authentication information might be a user name and a password pair with an Lightweight Third Party Authentication (LTPA) token.
2. The intermediary host authenticates the initial sender according to the embedded authentication information.
3. The intermediary host removes the authentication information from the SOAP message and replaces it with the <UsernameToken> element, which contains a user name.
4. The intermediary host asserts the trust according to the trust mode.
5. The intermediary host sends the updated SOAP message to the ultimate receiver.
6. The ultimate receiver checks the trust against the intermediary host information according to the configured trust mode. Also, the trusted ID evaluator is invoked.
7. If trust is established by the final receiver, the receiver invokes the Web service under the authorization of the user name (that is, the initial sender) in the SOAP message.

## Securing Web services for version 5.x applications using identity assertion authentication

### Important distinction between Version 5.x and Version 6 applications

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

WebSphere Application Server provides several different methods to secure your Web services. Extensible Markup Language (XML) digital signature is one of these methods. You might also secure your Web services using any of the following methods:

- XML digital signature
- XML encryption
- Basicauth authentication

- Identity assertion authentication
- Signature authentication
- Pluggable token

With the identity assertion authentication method, the security token generates a `<wsee:Username Token>` element that contains a `<wsse:Username>` element. On the request sender side, a callback handler is invoked to generate the security token. On the request receiver side, the security token is validated. Unlike basicauth authentication, trust is established through the use of a security token rather than through user name and password validation. To use identity assertion authentication to secure Web services, complete the following tasks:

1. Secure the client for identity assertion authentication.
   a. "Configuring the client for identity assertion: specifying the method"
   b. "Configuring the client for identity assertion: collecting the authentication method" on page 909
2. Secure the server for identity assertion authentication.
   a. "Configuring the server to handle identity assertion authentication" on page 910
   b. "Configuring the server to validate identity assertion authentication information" on page 911

After completing these steps, you have secured your Web services using identity assertion authentication.

***Configuring the client for identity assertion: specifying the method:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

This task is used to configure identity assertion authentication. The purpose of identity assertion is to assert the authenticated identity of the originating client from a Web service to a downstream Web service. Do not attempt to configure identity assertion from a pure client. Identity assertion works only when you configure on the client-side of a Web service acting as a client to a downstream Web service.

In order for the downstream Web service to accept the identity of the originating client (just the user name), you must supply a special trusted BasicAuth credential that the downstream Web service trusts and can authenticate successfully. You must specify the user ID of the special BasicAuth credential in a trusted ID evaluator on the downstream Web service configuration. For more information on trusted ID evaluators, see "Trusted ID evaluator" on page 847.

Complete the following steps to specify identity assertion as the authentication method:

1. Launch an assembly tool. For more information on the assembly tools, see Assembly tools.
2. Click **Windows > Open perspective > Other > J2EE**.
3. Click **Application Client Projects >***application_name* **> appClientModule > META-INF**.
4. Right-click the `application-client.xml` file, select **Open with > Deployment descriptor editor**.
5. Click the WS Extension tab, which is located at the bottom of the deployment descriptor editor within the assembly tool.
6. Expand the **Request sender configuration > Login configuration** section.
7. Select **IDAssertion** as the authentication method. For more conceptual information on identity assertion authentication, see "Identity assertion" on page 906.
8. Expand the **IDAssertion** section.
9. For the ID Type, select **Username**. This value works with all registry types and originating authentication methods.
10. For the Trust Mode, select either **BasicAuth** or **Signature**.

- By selecting **BasicAuth**, you must include basic authentication information (user ID and password), which the downstream Web service has specified in the trusted ID evaluator as a trusted user ID. See "Configuring the client for signature authentication: collecting the authentication information" on page 915 to specify the user ID and password information.
- By selecting **Signature** the certificate configured in the signature information section used to sign the data also is that is used as the trusted subject. The Signature is used to create a credential and user ID, which the certificate mapped to the downstream registry, is used in the trusted ID evaluator as a trusted user ID.

See "Configuring the client security bindings using an assembly tool" on page 871 for more information on the Web services client editor within the assembly tool.

After you specify identity assertion as the authentication method used by the client, you must specify how to collect the authentication information. See "Configuring the client for identity assertion: collecting the authentication method" for more information.

***Configuring the client for identity assertion: collecting the authentication method:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

This task is used to configure identity assertion authentication. The purpose of identity assertion is to assert the authenticated identity of the originating client from a Web service to a downstream Web service. Do not attempt to configure identity assertion from a pure client. Identity assertion works only when you configure on the client-side of a Web service acting as a client to a downstream Web service.

In order for the downstream Web service to accept the identity of the originating client (just the user name), you must supply a special trusted BasicAuth credential that the downstream Web service trusts and can authenticate successfully. You must specify the user ID of the special BasicAuth credential in a trusted ID evaluator on the downstream Web service configuration. For more information on trusted ID evaluators, see "Trusted ID evaluator" on page 847.

Complete the following steps to specify how the client collects the authentication information:

1. Launch an assembly tool. For more information on the assembly tools, see Assembly tools.
2. Click **Windows > Open perspective > Other > J2EE**.
3. Click **Application Client Projects >***application_name* **> appClientModule > META-INF**.
4. Right-click the `application-client.xml` file, select **Open with > Deployment descriptor editor**.
5. Click the WS Binding tab, which is located at the bottom of the Deployment Descriptor Editor within an assembly tool.
6. Expand the **Security request sender binding configuration > Login binding** section.
7. Click **Edit** to view the login binding information and select **IDAssertion**. The login binding dialog is displayed. Select or enter the following information:

   **Authentication method**
   : The authentication method specifies the type of authentication that occurs. Select **IDAssertion** to use identity assertion.

   **Token value type URI and Token value type Local name**
   : When you select IDAssertion, you cannot edit the token value type Universal Resource Identifier (URI) and the local name. Specifies custom authentication types. For IDAssertion authentication, leave these values blank.

   **Callback handler**
   : Specifies the Java Authentication and Authorization Service (JAAS) callback handler

implementation for collecting the BasicAuth information. Specify the `com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler` implementation for IDAssertion.

**Basic authentication User ID and Basic authentication Password**
In this field the trust mode entered in the extensions is BasicAuth. Specifies the trusted user ID and password in these fields. The user ID specified must be an ID that is trusted by the downstream Web service. The Web service trusts the user ID if it is entered as a trusted ID in a trusted ID evaluator in the downstream Web service bindings. If the trust mode entered in the extensions is `Signature`, you do not need to specify any information in this field.

**Property name and Property value**
Specifies properties with name and value pairs, for use by custom callback handlers. For IDAssertion, you do not need to specify any information in this field.

To use the identity assertion authentication method, you must specify the method in the Security extensions section of an assembly tool. See "Configuring the client for identity assertion: specifying the method" on page 908 if you have not previously specified this information.

***Configuring the server to handle identity assertion authentication:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

Use this task to configure identity assertion authentication. The purpose of identity assertion is to assert the authenticated identity of the originating client from a Web service to a downstream Web service. Do not attempt to configure identity assertion from a pure client.

For the downstream Web service to accept the identity of the originating client (user name only), you must supply a special trusted BasicAuth credential that the downstream Web service trusts and can authenticate successfully. You must specify the user ID of the special BasicAuth credential in a trusted ID evaluator on the downstream Web service configuration. For more information on trusted ID evaluators, see "Trusted ID evaluator" on page 847. The server side passes the special BasicAuth credential into the trusted ID evaluator, which returns `true` or `false` that this ID is trusted. Once it is trusted, the user name of the client is mapped to the credential, which is used for authorization.

Complete the following steps to configure the server to handle identity assertion authentication information:

1. Launch an assembly tool. For more information on the assembly tools, see Assembly tools.
2. Open the J2EE perspective by clicking **Window > Open perspective > Other > J2EE**.
3. Click **EJB Projects >***application_name* **> ejbModule > META-INF**.
4. Right-click the `webservices.xml` file, and click **Open with > Web services editor**.
5. Click the Extensions tab, which is located at the bottom of the Web services editor within the assembly tool.
6. Expand the **Request receiver service configuration details > Login configuration** section. The options you can select are:
   - **BasicAuth**
   - **Signature**
   - **ID assertion**
   - **Lightweight Third Party Authenticatnon (LTPA)**

7. Select **IDAssertion** to authenticate the client using the identity assertion data provided. The user ID of the client must be in the target user registry configured in WebSphere Application Server global security. You can select global security in the administrative console by clicking **Security > Global security**.

   You can select multiple login configurations, which means that different types of security information can be received at the server. The order in which the login configurations are added determines the processing order when a request is received. Problems can occur if you have multiple login configurations added that have common security tokens. For example, ID assertion contains a BasicAuth token, which is the trusted token. For ID assertion to work properly, you must list ID assertion ahead of BasicAuth in the list or BasicAuth processing overrides ID assertion processing.
8. Expand the **IDAssertion** section and select both the **ID Type** and the **Trust Mode**.
   a. For ID Type, the options are:
      - Username
      - Distinguished name (DN)
      - X509certificate

      These choices are just preferences and are not guaranteed. Most of the time the Username option is used. You must choose the same ID Type as the client.
   b. For Trust Mode, the options are:
      - BasicAuth
      - Signature

      The Trust Mode refers to the information sent by the client as the trusted ID.
      1) If you select **BasicAuth**, the client sends basic authentication data (user ID and password). This basicauth data is authenticated to the configured user registry. When the authentication occurs successfully, the user ID must be part of the trusted ID evaluator trust list.
      2) If you select **Signature**, the client signing certificate is sent. This certificate must be mappable to the configured user registry. For **Local OS**, the common name (CN) of the distinguished name (DN) is mapped to a user ID in the registry. For **Lightweight Directory Access Protocol (LDAP)**, the DN is mapped to the registry for the ExactDN mode. If it is in the CertificateFilter mode, attributes are mapped accordingly. In addition, the user name from the credential generated must be in the Trusted ID Evaluator trust list.

For more information on getting started with the Web Services Editor within an assembly tool, see “Configuring the server security bindings using an assembly tool” on page 875.

After you specify how the server handles identity assertion authentication information, you must specify how the server validates the authentication information. See “Configuring the server to validate identity assertion authentication information” for more information.

***Configuring the server to validate identity assertion authentication information:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

Use this task to configure identity assertion authentication. The purpose of identity assertion is to assert the authenticated identity of the originating client from a Web service to a downstream Web service. Do not attempt to configure identity assertion from a pure client.

For the downstream Web service to accept the identity of the originating client (user name only), you must supply a special trusted BasicAuth credential that the downstream Web service trusts and can authenticate successfully. You must specify the user ID of the special BasicAuth credential in a trusted ID evaluator on

the downstream Web service configuration. For more information on trusted ID evaluators, see "Trusted ID evaluator" on page 847. The server side passes the special BasicAuth credential into the trusted ID evaluator, which returns a `true` or `false` response that this ID is trusted. Once it is trusted, the user name of the client is mapped to the credential, which is used for authorization.

Complete the following steps to validate the identity assertion authentication information:

1. Launch an assembly tool. For more information on the assembly tools, see Assembly tools.
2. Open the J2EE perspective by clicking **Window > Open perspective > Other > J2EE**.
3. Click **EJB Projects >***application_name* **> ejbModule > META-INF**.
4. Right-click the `webservices.xml` file, and click **Open with > Web services editor**.
5. Click the Binding Configurations tab, which is located at the bottom of the Web services editor within the assembly tool.
6. Expand the **Request receiver binding configuration details > Login mapping** section.
7. Click **Edit** to view the login mapping information. Click **Add** to add new login mapping information. The login mapping dialog is displayed. Select or enter the following information:

   **Authentication method**
   Specifies the type of authentication that occurs. Select **IDAssertion** to use basic authentication.

   **Configuration name**
   Specifies the Java Authentication and Authorization Service (JAAS) login configuration name. For the IDAssertion authentication method, enter `system.wssecurity.IDAssertion` for the Java Authentication and Authorization Service (JAAS) login configuration name.

   **Use token value type**
   Determines if you want to specify a custom token type. For the default authentication method selections, you do not need to specify this option.

   **Token value type URI and Token value type local name**
   When you select ID assertion, you cannot edit the token value type URI and local name values. Specifies custom authentication types. For the ID assertion authentication method, leave these values blank.

   **Callback handler factory class name**
   Ccreates a JAAS CallbackHandler implementation that understands the following callbacks:

   - javax.security.auth.callback.NameCallback
   - javax.security.auth.callback.PasswordCallback
   - com.ibm.wsspi.wssecurity.auth.callback.BinaryTokenCallback
   - com.ibm.wsspi.wssecurity.auth.callback.XMLTokenReceiverCallback
   - com.ibm.wsspi.wssecurity.auth.callback.PropertyCallback

   For any of the default authentication methods (BasicAuth, IDAssertion, and Signature), use the callback handler factory default implementation. Enter the following class name for any of the default Authentication methods including IDAssertion:

   ```
   com.ibm.wsspi.wssecurity.auth.callback.WSCallbackHandlerFactoryImpl
   ```

   This implementation creates the correct callback handler for the default implementations.

   **Callback handler factory property name and Callback handler factory property value**
   Specifies callback handler properties for custom callback handler factory implementations. The default callback handler factory implemetation does not need any specified properties. For ID assertion, leave these values blank.

**Login mapping property name and Login mapping property value**
: Specifies properties for a custom login mapping. For the default implementations including IDAssertion, leave these values blank.

8. Expand the **Trusted ID evaluator** section.
9. Click **Edit** to see a dialog that displays all the trusted ID evaluator information. The following table describes the purpose of this information.

**Class name**
: Refers to the implementation of the trusted ID evaluator that you want to use. Enter the default implementation as `com.ibm.wsspi.wssecurity.id.TrustedIDEvaluatorImpl`. If you want to implement your own trusted ID evaluator, you must implement the com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator interface.

**Property name**
: Represents the name of this configuration. Enter `BasicIDEvaluator`.

**Property value**
: Defines the name and value pairs that can be used by the trusted ID evaluator implementation. For the default implementation, the trusted list is defined here. When a request comes in and the trusted ID is verified, the user ID, as it appears in the user registry, must be listed in this property. Specify the property as a name and value pair where the name is *trustedId_n*. *n* is an integer starting from 0 and the value is the user ID associated with that name. An example list with the trusted names include two properties.

For example: `trustedId_0 = user1`, `trustedId_1 = user2`. The previous example means that both user1 and user2 are trusted. `user1` and `user2` must be listed in the configured user registry.

10. Expand the **Trusted ID evaluator reference** section.
11. Click **Enable** to add a new entry. The text you enter for the **Trusted ID evaluator reference** must be the same as the name entered previously in the **Trusted ID evaluator**. Make sure that the name matches exactly because the information is case sensitive. If an entry is already specified, you can change it by clicking **Edit**.

You must specify how the server handles the identity assertion authentication method. See "Configuring the server to handle identity assertion authentication" on page 910 if you have not previously specified this information.

## Securing Web services for version 5.x applications using signature authentication

### Important distinction between Version 5.x and Version 6 applications

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

WebSphere Application Server provides several different methods to secure your Web services. Extensible Markup Language (XML) digital signature is one of these methods. You might secure your Web services using any of the following methods:

- XML digital signature
- XML encryption
- Basicauth authentication
- Identity assertion authentication
- Signature authentication
- Pluggable token

With the signature authentication method, the request sender generates a signature security token using a callback handler. The security token returned by the callback handler is inserted in the Simple Object

Access Protocol (SOAP) message. The request receiver retrieves the Signature security token from the SOAP message and validates it using a Java Authentication and Authorization Service (JAAS) login module. To use signature authentication to secure Web services, complete the following tasks:

1. Secure the client for signature authentication.
   a. "Configuring the client for signature authentication: specifying the method."
   b. "Configuring the client for signature authentication: collecting the authentication information" on page 915.
2. Secure the server for signature authentication.
   a. "Configuring the server to support signature authentication" on page 917.
   b. "Configuring the server to validate signature authentication information" on page 918.

After completing these steps, you have secured your Web services using signature authentication.

***Configuring the client for signature authentication: specifying the method:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

This task is used to configure signature authentication. A signature refers to the use of an X.509 certificate to login on the target server. For more information on signature authentication, see "Signature authentication method."

Complete the following steps to specify signature as the authentication method:

1. Launch an assembly tool. For more information on the assembly tools, see Assembly tools.
2. Click **Windows > Open perspective > Other > J2EE**.
3. Click **Application Client Projects >***application_name* **> appClientModule > META-INF**.
4. Right-click the `application-client.xml` file, select **Open with > Deployment descriptor editor**.
5. Click the WS Extension tab, which is located at the bottom of the deployment descriptor editor within the assembly tool.
6. Expand the **Request sender configuration > Login configuration** section. The following login configuration options are valid for a managed client and Web services acting as a client are:

   **BasicAuth**
   Use this option for a managed client.

   **Signature**
   Use this option for a managed client.

   **IDAssertion**
   Use this option for Web services acting as a client.
7. Select **Signature** to authenticate the client using the certificate used to digitally sign the request.

For more information on getting started with the Web services client editor within the assembly tool, see "Configuring the client security bindings using an assembly tool" on page 871.

After you specify signature as the authentication method, you must specify how to collect the authentication information. See "Configuring the client for signature authentication: collecting the authentication information" on page 915 for more information.

*Signature authentication method:*

When using the signature authentication method, the security token is generated with a <ds:Signature> and a <wsse:BinarySecurityToken> element.

On the request sender side, a callback handler is invoked to generate the security token. On the request receiver side, a Java Authentication and Authorization Service (JAAS) login module is used to validate the security token. These two operations, token generation and token validation, are described in the following sections.

**Signature token generation**

The request sender generates a Signature security token using a callback handler. The security token returned by the callback handler is inserted in the SOAP message. The callback handler is specified in the <LoginBinding> element of the bindings file, `ibm-webservicesclient-bnd.xmi`. WebSphere Application Server provides the following callback handler implementation that can be used with the Signature authentication method:
com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler

You can add your own callback handlers that implement the javax.security.auth.callback.CallbackHandler implementation.

**Security token validation**

The request receiver retrieves the Signature security token from the Simple Object Access Protocol (SOAP) message and validates it using a JAAS login module. The <ds:Signature> and <wsse:BinarySecurityToken> elements in the security token are used to perform the validation. If the validation is successful, the login module returns a Java Authentication and Authorization Service (JAAS) Subject. This Subject then is set as the identity of the running thread. If the validation fails, the request is rejected with a SOAP fault exception.

The JAAS login configuration is specified in the `<LoginMapping>` element of the bindings file. Default bindings are specified in the `ws-security.xml` file. However, you can override these bindings using the application-specific `ibm-webservices-bnd.xmi` file. The configuration information consists of a CallbackHandlerFactory and a ConfigName. The CallbackHandlerFactory specifies the name of a class that is used for creating the JAAS CallbackHandler object. WebSphere Application Server provides the com.ibm.wsspi.wssecurity.auth.callback.WSCallbackHandlerFactoryImp CallbackHandlerFactory implementation. The ConfigName specifies a JAAS configuration name entry. WebSphere Application Server searches in the `security.xml` file for a matching configuration name entry. If a match is not found, it searches the `wsjaas.conf` file. WebSphere Application Server provides the system.wssecurity.Signature default configuration entry, which is suitable for the signature authentication method.

**Remember:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

***Configuring the client for signature authentication: collecting the authentication information:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

This task is used to configure signature authentication. A signature refers to the use of an X.509 certificate to login on the target server. For more information on signature authentication, see “Signature authentication method” on page 914.

Complete the following steps to specify how the client collects the authentication information for signature authentication:

1. Launch an assembly tool. For more information on the assembly tools, see Assembly tools.

2. Click **Windows > Open perspective > Other > J2EE**.
3. Click **Application Client Projects >***application_name* **> appClientModule > META-INF**.
4. Right-click the `application-client.xml` file, select **Open with > Deployment descriptor editor**.
5. Click the WS Binding tab, which is located at the bottom of the deployment descriptor editor within the assembly tool.
6. Expand the **Security request sender binding configuration > Signing information** and click **Edit** to modify the signing key name and signing key locator. To create new signing information, click **Enable**. The certificate that is sent to log in at the server is the one configured in the Signing Information section. Review the section on “Key locator” on page 842 to understand how the signing key name maps to a key within the key locator entry.

   The following list describes the purpose of this information. Some of these definitions are based on the XML-Signature specification, which is located at the following Web address: http://www.w3.org/TR/xmldsig-core

   **Canonicalization method algorithm**
   Canonicalizes the SignedInfo element before it is digested as part of the signature operation.

   **Digest mehod algorithm**
   Represents the algorithm that is applied to the data after transforms are applied, if specified, to yield the <DigestValue> element. The signing of the DigestValue element binds the resource content to the signer key. The algorithm selected for the client request sender configuration must match the algorithm selected in the server request receiver configuration.

   **Signature method algorithm**
   Represents the algorithm that is used to convert the canonicalized <SignedInfo> value into the <SignatureValue> value. The algorithm selected for the client request sender configuration must match the algorithm selected in the server request receiver configuration.

   **Signing key name**
   Represents the key entry associated with the signing key locator. The key entry refers to an alias of the key, which is used to sign the request.

   **Signing key locator**
   TRepresents a reference to a key locator implementation. For more information on configuring key locators, see “Key locator” on page 842.
7. Expand the **Security request sender binding configuration > Login binding** section.
8. Click **Edit** to view the login binding information. Then, select or enter the following information:

   **Authentication method**
   Specifies the type of authentication that occurs. Select **Signature** to use signature authentication.

   **Token value type URI and Token value type URI local name**
   When you select **Signature**, you cannot edit the Token value type URI and Local name values. Specifies custom authentication types. For signature authentication, leave these fields blank.

   **Callback handler**
   Specifies the Java Authentication and Authorization Server (JAAS) callback handler implementation for collecting signature information. Enter the following callback handler for signature authentication:
   `com.ibm.wsspi.wssecurity.auth.callback.NonPromptCallbackHandler`

   This callback handler is used because the signature method does not require user interaction.

   **Basic authentication user ID and Basic authentication password**
   Leave the BasicAuth fields blank when Signature authentication is used.

**Property name and property value**
This field enables you to enter properties and name and value pairs for use by custom callback handlers. For signature authentication, you do not need to enter any information.

**Other customization entries**: There is a basic authentication entry in the Port Qualified Name Binding Details section. This entry is used for HTTP transport authentication, which might be required if the router servlet is protected.

Information specified in the Web services security signature authentication section overrides the basic authentication information specified in the Port Qualified Name Binding Details section for authorizing the Web service.

To use the signature authentication method, you must specify the authentication method in the Login config section of an assembly tool. See “Configuring the client for signature authentication: specifying the method” on page 914 if you have not previously specified this information.

***Configuring the server to support signature authentication:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

Use this task to configure signature authentication at the server. Signature authentication refers to an X.509 certificate sent by the client to the server. The certificate is used to authenticate to the user registry configured at the server. After a request is received by the server that contains the certificate, the server needs to log in to form a credential. The credential is used for authorization. If the certificate supplied cannot be mapped to an entry in the user registry, an exception is thrown and the request ends without invoking the resource. For more information on signature authentication, see “Signature authentication method” on page 914.

1. Launch an assembly tool. For more information on the assembly tools, see Assembly tools.
2. Open the J2EE perspective by clicking **Window > Open perspective > Other > J2EE**.
3. Click **EJB Projects >***application_name* **> ejbModule > META-INF**.
4. Right-click the `webservices.xml` file, and click **Open with > Web services editor**.
5. Click the Extensions tab, which is located at the bottom of the Web Services Editor within the assembly tool.
6. Expand the **Request receiver service configuration details > Login configuration** section. You can select from the following options:
   - BasicAuth
   - Signature
   - ID assertion
   - Lightweight Third Party Authentication (LTPA)
7. Select **Signature** to authenticate the client using an X509 certificate. The certificate that is sent from the client is the certificate that issued for signing the message. You must be able to map this certificate to the configured user registry. For Local operating system (OS) registries, the common name (cn) of the distinguished name (DN) is mapped to a user ID in the registry. For Lightweight Directory Access Protocol (LDAP), you can configure multiple mapping modes:
   - EXACT_DN is the default mode that directly maps the DN of the certificate to an entry in the LDAP server.
   - CERTIFICATE_FILTER is the mode that provides the LDAP advanced configuration with a place to specify a filter that maps specific attributes of the certificate to specific attributes of the LDAP server.

For more information on getting started with the Web services editor within the assembly tool, see “Configuring the server security bindings using an assembly tool” on page 875.

After you specify how the server handles signature authentication information, you must specify how the server validates the authentication information. See “Configuring the server to validate signature authentication information” for more information.

***Configuring the server to validate signature authentication information:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

Use this task to configure signature authentication at the server. Signature authentication refers to an X.509 certificate sent by the client to the server. The certificate is used to authenticate to the user registry configured at the server. Once a request is received by the server that contains the certificate, the server needs to log in to form a credential. The credential is used for authorization. If the certificate supplied cannot be mapped to an entry in the user registry, an exception is thrown and the request ends without invoking the resource. For more information on signature authentication, see “Signature authentication method” on page 914.

Complete the following steps to configure the server to validate signature authentication:

1. Launch an assembly tool. For more information on the assembly tools, see Assembly tools.
2. Open the J2EE perspective by clicking **Window > Open perspective > Other > J2EE**.
3. Click **EJB Projects >***application_name* **> ejbModule > META-INF**.
4. Right-click the `webservices.xml` file, and click **Open with > Web services editor**.
5. Click the Binding Configurations tab, which is located at the bottom of the Web services editor within the assembly tool.
6. Expand the **Request receiver binding configuration details > Login mapping** section.
7. Click **Edit** to view the login mapping information or click **Add** to add new login mapping information. The login mapping dialog is displayed and you select (or enter) the following information:

   **Authentication method**
   Specifies the type of authentication. Select **Signature** to use signature authentication.

   **Configuration name**
   Specifies the Java Authentication and Authorization Service (JAAS) login configuration name. For the signature authentication method, enter `system.wssecurity.Signature` for the JAAS login configuration name. This specification logs in with the com.ibm.wsspi.wssecurity.auth.module.SignatureLoginModule JAAS login module.

   **Use token value type**
   Determines if you want to specify a custom token type. For the default authentication method selections, you can leave this field blank.

   **URI and local name**
   When you select Signature method, you cannot edit the token value type URI and local name values. Specifies custom authentication types. For signature authentication, you can leave this field blank.

   **Callback handler factory class name**
   Creates a JAAS CallbackHandler implementation that understands the following callback handlers:

   - javax.security.auth.callback.NameCallback
   - javax.security.auth.callback.PasswordCallback

- com.ibm.wsspi.wssecurity.auth.callback.BinaryTokenCallback
- com.ibm.wsspi.wssecurity.auth.callback.XMLTokenReceiverCallback
- com.ibm.wsspi.wssecurity.auth.callback.PropertyCallback

For any of the default authentication methods (BasicAuth, IDAssertion, and Signature), use the callback handler factory default implementation. Enter the following class name for any of the default authentication methods including signature:

```
com.ibm.wsspi.wssecurity.auth.callback.WSCallbackHandlerFactoryImpl
```

This implementation creates the correct callback handler for the default implementations.

**Callback handler factory property name and callback handler factory property value**
Specifies callback handler properties for custom callback handler factory implementations. You do not need to specify any properties for the default callback handler factory implementation. For signature, you can leave this field blank.

**Login mapping property name and login mapping property value**
Specifies properties for a custom login mapping to use. For the default implementations including signature, you can leave this field blank.

Specify how the server handles the signature authentication method. See "Configuring the server to support signature authentication" on page 917 if you have not previously specified this information.

## Token type overview

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

The proposed Web services security draft defined two types of security tokens:

- Username token
- Binary security token

A user name token consists of a user name and, optionally, password information. You can include a user name token directly in the <Security> header within the message. Binary tokens, such as X.509 certificates, Kerberos tickets, Lightweight Third Party Authentication (LTPA) tokens, or other non-XML formats, require a special encoding for inclusion. The Web services security specification describes how to encode binary security tokens such as X.509 certificates and Kerberos tickets, and it also describes how to include opaque encrypted keys. The specification also includes extensibility mechanisms that you can use to further describe the characteristics of the credentials that are included with a message.

WebSphere Application Server, Version 5.0.2 supports user name tokens, which include both user name and password for basic authentication and user name, which is used for identity assertion. The WebSphere Application Server, Version 5.0.2 binary security token implementation supports both X.509 certificates and LTPA binary security. You extend the implementation to generate other types of tokens. However, Kerberos tickets are not supported in WebSphere Application Server, Version 5.0.2. Each type of token is processed by a corresponding token generation and validation module. The binary token generation and validation modules are pluggable that is based on the Java Authentication and Authorization Service (JAAS) framework. For example, an arbitrary XML-based token format is supported using the JAAS pluggable framework. WebSphere Application Server, Version 5.0.2 does not support an XML-based token that is used in the SecurityTokenReference.

You can define the types of tokens that the message can accept in the deployment descriptor extension file, ibm.webservices-ext.xmi. A message receiver might support one or more types of security tokens. The following example shows that the receiver supports four types of security tokens:

**Important:** In the following code sample, several lines were split into multiple lines due to the width of the printed page. See the close bracket for an indication of where each line of code ends.

```
?xml version="1.0" encoding="UTF-8"?>
<com.ibm.etools.webservice.wsext:WsExtension xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:com.ibm.etools.webservice.wsext=
"http://www.ibm.com/websphere/appserver/schemas/5.0.2/wsext.xmi"
xmi:id="WsExtension_1052760331306" routerModuleName="StockQuote.war">
  <wsDescExt xmi:id="WsDescExt_1052760331306" wsDescNameLink="StockQuoteFetcher">
    <pcBinding xmi:id="PcBinding_1052760331326" pcNameLink="urn:xmltoday-delayed-quotes"
      scope="Session">
       <serverServiceConfig
         xmi:id="ServerServiceConfig_1052760331326"actorURI="myActorURI">
          <securityRequestReceiverServiceConfig
           xmi:id="SecurityRequestReceiverServiceConfig_1052760331326">
            <loginConfig xmi:id="LoginConfig_1052760331326">
              <authMethods xmi:id="AuthMethod_1052760331326" text="BasicAuth"/>
              <authMethods xmi:id="AuthMethod_1052760331327" text="IDAssertion"/>
              <authMethods xmi:id="AuthMethod_1052760331336" text="Signature"/>
              <authMethods xmi:id="AuthMethod_1052760331337" text="LTPA"/>
           </loginConfig>
<idAssertion xmi:id="IDAssertion_1052760331336" idType="Username" trustMode="Signature"/>
```

The message sender might choose one of the token types that are supported by the receiver when sending a message. You can define the type of token to be used by the sending side in the client descriptor extension file, `ibm-webservicesclient-ext.xmi`. The following example shows that the sender chooses to send a UsernameToken to the receiver:

**Important:** In the following code sample, several lines were split into multiple lines due to the width of the printed page. See the close bracket for an indication of where each line of code ends.

```
<?xml version="1.0" encoding="UTF-8"?>
<com.ibm.etools.webservice.wscext:WsClientExtension xmi:version="2.0"
mlns:xmi="http://www.omg.org/XMI"
xmlns:com.ibm.etools.webservice.wscext=
"http://www.ibm.com/websphere/appserver/schemas/5.0.2/wscext.xmi"
xmi:id="WsClientExtension_1052760331496">
<ServiceRefs xmi:id="ServiceRef_1052760331506" serviceRefLink="service/StockQuoteService">
    <portQnameBindings xmi:id="PortQnameBinding_1052760331506"
portQnameLocalNameLink="StockQuote">
      <clientServiceConfig xmi:id="ClientServiceConfig_1052760331506"
actorURI="myActorURI">
          <securityRequestSenderServiceConfig
xmi:id="SecurityRequestSenderServiceConfig_1052760331506" actor="myActorURI">
          <loginConfig xmi:id="LoginConfig_1052760331506" authMethod="BasicAuth"/>
```

***Username token element:***

You can use the UsernameToken element to propagate a user name and, optionally, password information. Also, you can use this token type to carry basic authentication information. Both a user name and a password are used to authenticate the message. A UsernameToken containing the user name is used in identity assertion, which establishes the identity of the user based on the trust relationship.

The following example shows the syntax of the UsernameToken element:

```
<UsernameToken Id="...">
    <Username>...</Username>
    <Password Type="...">...</Password>
</UsernameToken>
```

The Web services security specification defines the following password types:

**wsse:PasswordText (default)**
: This type is the actual password for the user name.

**wsse:PasswordDigest**
: The type is the digest of the password for the user name. The value is a base64-encoded SHA1 hash value of the UTF8-encoded password.

WebSphere Application Server supports the default PasswordText type. However, it does not support password digest because most user registry security policies do not expose the password to the application software.

The following example illustrates the use of the <UsernameToken> element:

```
<S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
            xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext">
    <S:Header>
            ...
        <wsse:Security>
            <wsse:UsernameToken>
                <wsse:Username>Joe</wsse:Username>
                <wsse:Password>ILoveJava</wsse:Password>
            </wsse:UsernameToken>
        </wsse:Security>
    </S:Header>
</S:Envelope>
```

**Remember:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

*Nonce, a randomly generated token:*

*Nonce* is a randomly generated, cryptographic token used to prevent the theft of user name tokens used with SOAP messages. Nonce is used with the basicauth authentication method.

Without nonce, when a user name token is passed from one machine to another machine using a non-secure transport, such as HTTP, the token might be intercepted and used in a replay attack. The same key might be reused when the username token is transmitted between the client and the server, which leaves it vulnerable to attack. The user name token can be stolen even if you use XML digital signature and XML encryption.

To help eliminate these replay attacks, the `<wsse:Nonce>` and `<wsu:Created>` elements are generated within the `<wsee: usernameToken>` element and used to validate the message. The request receiver or response receiver checks the freshness of the message to verify the difference between when the message is created and the current time falls within a specified time period. Also, WebSphere Application Server verifies that the receiver has not processed the token within the specified time period. These two features are used to lessen the chance that a user name token is used for a replay attack.

*Configuring nonce for the application level:*

**Important:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

*Nonce* is a randomly generated, cryptographic token used to thwart the highjacking of username tokens used with Simple Object Access Protocol (SOAP) messages. Nonce is used in conjunction with the BasicAuth authentication method.

This task provides instructions on how to configure nonce for the application level using the WebSphere Application Server administrative console.

You can configure nonce at the application level and cell level.

However, you must consider the order of precedence:

1. Application level
2. Server level

If you configure nonce on the application level and the server level, the values specified for the application level take precedence over the values specified for the server level.

1. Connect to the administrative console by typing `http://localhost:9060/ibm/console` in your Web browser unless you have changed the port number.
2. Click **Applications > Enterprise applications >***application_name*.
3. Under Related Items, click **Web module** or **EJB module >** *URI_name*.
4. Under Additional properties, click **Web services: Server security bindings**.
5. Click **Edit** under Request receiver binding
6. Under Additional properties, click **Login mappings > New**.
7. Specify (optional) a value, in seconds, for the **Nonce maximum age** field. This panel is optional and only valid if the BasicAuth authentication method is specified. If you specify another authentication method and attempt to specify values for this field, the following error message displays and you must remove the specified value:

   ```
   Nonce is not supported for authentication methods other than
   BasicAuth.
   ```

   If you specify BasicAuth, but do not specify values for the Nonce maximum age field, the Web services security run time searches for a Nonce Maximum Age value on the server level. If a value is not found on the server level, the run time searches the cell level. If a value is not found on either the server level or the cell level, the default is 300 seconds.

   The value specified for the Nonce Maximum Age field indicates how long the nonce is valid. You must specify a minimum of 300 seconds, but the value cannot exceed the number of seconds specified for the Nonce Cache Timeout field for either the server level

   You can specify the Nonce Cache Timeout value for the server level by completing the following steps:

   a. Click **Servers > Application servers >***server_name*.
   b. Under Security, click **Web Services: Default bindings for Web services security**.
8. Specify (optional) a value, in seconds, for the Nonce clock skew field. The value specified for the Nonce Clock Skew field specifies the amount of time, in seconds, to consider when the message receiver checks the timeliness of the value. This panel is optional and only valid if the BasicAuth authentication method is specified. If you specify another authentication method and attempt to specify values for this field, the following error message displays and you must remove the specified value:

   ```
   Nonce is not supported for authentication methods other than
   BasicAuth.
   ```

   If you specify BasicAuth, but do not specify values for the Nonce clock skew field, the Web services security run time searches for a Nonce clock skew value on the server level. If a value is not found on the server level, the run time searches the cell level. If a value is not found on either the server level or the cell level, the default is 0 seconds.

   Consider the following information when you set this value:

   - Difference in time between the message sender and the message receiver if the clocks are not synchronized.
   - Time needed to encrypt and transmit the message.
   - Time needed to get through network congestion.
9. Restart the server.

*Configuring nonce for the server level:*

**Important:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

*Nonce* is a randomly generated, cryptographic token used to prevent the theft of username tokens used with Simple Object Access Protocol (SOAP) messages. Nonce is used in conjunction with the BasicAuth authentication method.

This task provides instructions on how to configure nonce for the server level using the WebSphere Application Server administrative console.

However, you must consider the order of precedence:
1. Application level
2. Server level

If you configure nonce on the application level and the server level, the values specified for the application level take precedence over the values specified for the server level.

In a WebSphere Application Server or WebSphere Application Server Express environment, you must specify values for the Nonce cache timeout, Nonce maximum age, and Nonce clock skew fields on the server level to use nonce effectively.

Complete the following steps to configure nonce on the server level:
1. Connect to the administrative console by typing `http://localhost:9060/ibm/console` in your Web browser unless you have changed the port number.
2. Click **Servers > Application servers >***server_level*.
3. Under Security, click **Web Services: Default bindings for Web services security**.
4. Specify a value, in seconds, for the Nonce cache timeout field. The value specified for the Nonce cache timeout field indicates how long the nonce remains cached before it is expunged. You must specify a minimum of 300 seconds. However, if you do not specify a value, the default is 600 seconds. This field is required for the server level.
5. Specify (optional) a value, in seconds, for the Nonce maximum age field.

   The value specified for the Nonce Maximum Age field indicates how long the nonce is valid. You must specify a minimum of 300 seconds, but the value cannot exceed the number of seconds specified for the Nonce cache timeout field on the server level.
6. Specify a value, in seconds, for the Nonce clock skew field. The value specified for the Nonce clock skew field specifies the amount of time, in seconds, to consider when the message receiver checks the timeliness of the value. Consider the following information when you set this value:
   - Difference in time between the message sender and the message receiver if the clocks are not synchronized.
   - Time needed to encrypt and transmit the message.
   - Time needed to get through network congestion.

   You must specify at least 0 seconds for the Nonce clock skew field. However, the maximum value cannot exceed the number of seconds specified in the Nonce maximum age field on the server level. If you do not specify a value, the default is 0 seconds.
7. Restart the server. If you change the Nonce cache timeout value and do not restart the server, the change is not recognized by the server.

***Binary security token:***

The ValueType attribute identifies the type of the security token, for example, an LTPA token. The EncodingType indicates how the security token is encoded, for example, Base64Binary. The BinarySecurityToken element defines a security token that is binary encoded. The encoding is specified using the EncodingType attribute. The value type and space are specified using the ValueType attribute. The Web services security implementation for WebSphere Application Server, Version 5.0.2 supports both LTPA and X.509 certificate binary security tokens.

A binary security token has the following attributes that are used for interpretation:

- Value type
- Encoding type

The following example depicts an LTPA binary security token in a Web services security message header:

```
<wsse:BinarySecurityToken  xmlns:ns7902342339871340177=
                              ″http://www.ibm.com/websphere/appserver/tokentype/5.0.2″
      EncodingType=″wsse:Base64Binary″
      ValueType=″ns7902342339871340177:LTPA″>
              MIZ6LGPt2CzXBQfio9wZTo1VotWov0NW3Za6lU5K7Li78DSnIK6iHj3hxXgrUn6p4wZI
              8Xg26havepvmSJ8XxiACMihTJuh1t3ufsrjbFQJOqh5VcRvI+AKEaNmnEgEV65jUYAC9
              C/iwBBWk5U/6DIk7LfXcTT0ZPAd+3D3nCS0f+6tnqMou8EG9mtMeTKccz/pJVTZjaRSo
              msu0sewsOKfl/WPsjW0bR/2g3NaVvBy18VlTFBpUbGFVGgzHRjBKAGo+ctkl80nlVLIk
              TUjt/XdYvEpOr6QoddGi4okjDGPyyoDxcvKZnReXww5UsoqlpfXwN4KG9as=
</wsse:BinarySecurityToken></wsse:Security></soapenv:Header>
```

As shown in the example, the token is Base64Binary encoded.

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

***XML token:***

XML tokens are offered in two formats, Security Assertion Markup Language (SAML) and Extensible rights Markup Language (XrML).

XML-based security tokens are growing in popularity. Two well-known formats are:

- Security Assertion Markup Language (SAML)
- Extensible rights Markup Language (XrML)

Using textensibility of the <wsse:Security> header in XML-based security tokens, you can directly insert these security tokens into the header.

SAML assertions are attached to Web services security messages using Web services by placing assertion elements inside the <wsse:Security> header. The following example illustrates a Web services security message with a SAML assertion token.

```
<S:Envelope xmlns:S="...">&
      <wsse:Security xmlns:wsse="...">
          <saml:Assertion
                    MajorVersion="1"
                    MinorVersion="0"
                    AssertionID="SecurityToken-ef375268"
                       Issuer="elliotw1"
                       IssueInstant="2002-07-23T11:32:05.6228146-07:00"
                     xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion">
                     ...
           </saml:Assertion>
```

```
      </wsse:Security>
 </S:Header>
 <S:Body>
 ...
 </S:Body>
</S:Envelope>
```

For more information on SAML and XrML, see Web services: Resources for learning.

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

## Security token

A security token represents a set of claims made by a client that might include a name, password, identity, key, certificate, group, privilege, and so on.

Web services security provides a general-purpose mechanism to associate security tokens with messages for single message authentication. A specific type of security token is not required by Web services security. Web services security is designed to be extensible and support multiple security token formats to accommodate a variety of authentication mechanisms. For example, a client might provide proof of identity and proof of a particular business certification.

A security token is embedded in the SOAP message within the SOAP header. The security token within the SOAP header is propagated from the message sender to the intended message receiver. On the receiving side, the WebSphere Application Server security handler authenticates the security token and sets up the caller identity on the running thread.

**Remember:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

## Securing Web services for version 5.x applications using a pluggable token

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

WebSphere Application Server provides several different methods to secure your Web services; a pluggable token is one of these methods. You might secure your Web services using any of the following methods:

- XML digital signature
- XML encryption
- Basicauth authentication
- Identity assertion authentication
- Signature authentication
- Pluggable token

Complete the following steps to secure your Web services using a pluggable token:

1. Generate a security token using the Java Authentication and Authorization Service (JAAS) CallbackHandler interface. The Web services security run time uses the JAAS CallbackHandler interface as a plug-in to generate security tokens on the client side or when Web services is acting as a client.

2. Configure your pluggable token. To use pluggable tokens to secure your Web services, you must configure both the client request sender and the server request receiver. You can configure your pluggable tokens using either the WebSphere Application Server administrative console or the WebSphere Application Server Toolkit. For more information, see the following topics:
   - "Configuring pluggable tokens using an assembly tool"
   - "Configuring pluggable tokens using the administrative console" on page 928

***Configuring pluggable tokens using an assembly tool:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

This document describes how to configure a pluggable token in the request sender (`ibm-webservicesclient-ext.xmi` and `ibm-webservicesclient-bnd.xmi` file) and request receiver (`ibm-webservices-ext.xmi` and `ibm-webservices-bnd.xmi` file).

The pluggable token is required for the request sender and request receiver because they are a pair. The request sender and the request receiver must match for the receiver to accept a request.

Prior to completing these steps, it is assumed that you have already created a Web services-enabled Java 2 Platform, Enterprise Edition (J2EE) with a Web Services for J2EE (JSR 109) enterprise application. If not, see Developing Web services applications to create Web services-enabled J2EE with a JSR 109 enterprise application. See either of the following topics for an introduction of how to manage Web services security binding information for the server:
- "Configuring the server security bindings using an assembly tool" on page 875
- "Configuring the server security bindings using the administrative console" on page 877

You must specify the security constraints in the `ibm-webservicesclient-ext.xmi` and the `ibm-webservices-ext.xmi` files for the required tokens using an assembly tool such as the Application Server Toolkit or Rational Web Developer.

Complete the following steps to configure the request sender using the `ibm-webservicesclient-ext.xmi` and `ibm-webservicesclient-bnd.xmi` files:
1. Launch an assembly tool. For more information on the assembly tools, see Assembly tools.
2. Click **Windows > Open perspective > Other > J2EE**.
3. Click **Application Client Projects >***application_name* **> appClientModule > META-INF**.
4. Right-click the `application-client.xml` file, select **Open with > Deployment descriptor editor**.
5. Click the WS Extension tab. The Web service client security extensions editor is displayed.
   a. Under Service References, select an existing service reference or click **Add** to create a new reference.
   b. Under Port Qname Bindings, select an existing port qualified name for the selected service reference or click **Add** to create a new port name binding.
   c. Under Request Sender Configuration: Login Configuration, select an exiting authentication method or type in a new one in the editable list box (Lightweight Third Party Authorization (LTPA) is a supported token generation when Web services is acting as client).
   d. Click **File > Save** to save the changes.
6. Click the Web services client binding tab. The Web services client binding editor is displayed.
   a. Under Port qualified name binding, select an existing entry or click **Add** to add a new port name binding. The Web services client binding editor displays for the selected port.
   b. Under Login binding, click **Edit** or **Enable**. The Login Binding dialog box is displayed.

1) In the Authentication Method field, enter the authentication method. The authentication method that you enter in this field must match the authentication method defined on the Security Extension tab for the same Web service port. This field is mandatory.
2) (Optional) Enter the token value type information in the URI and Local name fields. These fields are ignored for the BasicAuth, Signature, and IDAssertion authentication methods, but required for other authentication methods. The token value type information is inserted into the <wsse:BinarySecurityToken>@ValueType element for binary security token and is used as the namespace for the XML-based token.
3) Enter an implementation of the Java Authentication and Authorization Service (JAAS) javax.security.auth.callback.CallbackHandler interface. This field is mandatory.
4) Enter the basic authentication information in the User ID and Password fields. The basic authentication information is passed to the construct of the CallbackHandler implementation. The use of the basic authentication information depends on the implementation of CallbackHandler.
5) In the Property field, add name and value pairs. These pairs are passed to the construct of the CallbackHandler implementation as java.util.Map values.
6) Click **OK**.

Click **Disable** under Login binding on the Web services client port binding tab to remove the authentication method login binding.

c. Click **File > Save** to save the changes.

7. In the Package Explorer window, right-click the `webservices.xml` file and click **Open with > Web services editor**. The Web Services window displays.
   a. Click the Security extensions tab. The Web service security extensions editor is displayed.
      1) Under Web Service Description Extension, select an existing service reference or click **Add** to create a new extension.
      2) Under Port Component Binding, select an existing port qualified name for the selected service reference or click **Add** to create a new one.
      3) Under Request Receiver Service Configuration Details: Login Configuration, select an exiting authentication method or click **Add** and enter a new method in the Add AuthMethod field that displays. You can select multiple authentication methods for the request receiver. The security token of the incoming message is authenticated against the authentication methods in the order that they are specified in the list. Click **Remove** to remove the selected authentication method or methods.
   b. Click **File > Save** to save the changes.
   c. Click the Bindings tab. The Web services bindings editor is displayed.
      1) Under Web service description bindings, select an existing entry or click **Add** to add a new Web services descriptor.
      2) Click the Binding configurations tab. The Web services binding configurations editor is displayed for the selected Web services descriptor.
      3) Under Request receiver binding configuration details: login mapping, click **Add** to create a new login mapping or click **Edit** to edit the selected login mapping. The Login mapping dialog is displayed.
         a) In the Authentication method field, enter the authentication method. The information entered in this field must match the authentication method defined on the Security Extensions tab for the same Web service port. This field is mandatory.
         b) In the Configuration name field, enter a JAAS login configuration name. You must define the JAAS login configuration name in the WebSphere Application Server administrative console under **Security > Global security**. Under Authentication, click **JAAS configuration > Application logins**. This is a mandatory field. For more information, see “Configuring application logins for Java Authentication and Authorization Service” on page 242.

c) (Optional) Select **Use Token value type** and enter the token value type information in the URI and Local name fields. This information is optional for BasicAuth, Signature and IDAssertion authentication methods, but required for any other authentication method. The token value type is used to validate the <wsse:BinarySecurityToken>@ValueType element for binary security tokens and to validate the namespace of the XML-based token.
d) Under Callback Handler Factory, enter an implementation of the com.ibm.wsspi.wssecurity.auth.callback.CallbackHandlerFactory interface in the Class name field. This field is mandatory.
e) Under Callback Handler Factory property, click **Add** and enter the name and value pairs for the Callback Handler Factory Property. These name and value pairs are passed as java.util.Map to the com.ibm.wsspi.wssecurity.auth.callback.CallbackHandlerFactory.init() method. The use of these name and value pairs is determined by the CallbackHandlerFactory implementation.
f) Under Login Mapping Property, click **Add** and enter the name and value pairs for the Login mapping property. These name and value pairs are available to the JAAS Login Modules through thecom.ibm.wsspi.wssecurity.auth.callback.PropertyCallback JAAS Callback interface. Click **Remove** to delete the selected login mapping.
g) Click **OK**.

d. Click **File > Save** to save the changes.

The previous steps define how to configure the request sender to create security tokens in the Simple Object Access Protocol (SOAP) message and to configure the request receiver to validate the security tokens found in the incoming SOAP message. WebSphere Application Server supports pluggable security tokens.

You can use the authentication method defined in the login bindings and login mappings to generate security tokens in the request sender and validate security tokens in the request receiver.

After you configure pluggable tokens, you must configure both the client and the server to support pluggable tokens. See the following topics to configure the client and the server:
- Configuring the client for LTPA token authentication: Specifying LTPA token authentication
- Configuring the client for LTPA token authentication: Collecting the authentication information
- Configuring the server to handle LTPA token authentication
- Configuring the server to validate LTPA token authentication information

***Configuring pluggable tokens using the administrative console:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

Prior to completing these steps, it is assumed that you have already created a Web services-enabled Java 2 Platform, Enterprise Edition (J2EE) with a Web Services for J2EE (JSR 109) enterprise application. If not, see Developing Web services applications to create Web services-enabled J2EE with a JSR 109 enterprise application. See either of the following topics for an introduction of how to manage Web services security binding information for the server:
- “Configuring the server security bindings using an assembly tool” on page 875
- “Configuring the server security bindings using the administrative console” on page 877

This document describes how to configure a pluggable token in the request sender (`ibm-webservicesclient-ext.xmi` and `ibm-webservicesclient-bnd.xmi` file) and request receiver (`ibm-webservices-ext.xmi` and `ibm-webservices-bnd.xmi` file).

**Important:** The pluggable token is required for the request sender and request receiver as they are a pair. The request sender and the request receiver must match for a request to be accepted by the receiver.

Prior to completing these steps, it is assumed that you deployed a Web services-enabled enterprise application to the WebSphere Application Server.

Use the following steps to configure the client-side request sender (`ibm-webservicesclient-bnd.xmi` file) or server-side request receiver (`ibm-webservices-bnd.xmi` file) using the WebSphere Application Server Administrative Console.

1. Click **Applications > Enterprise applications >** *enterprise_application*.
2. Under Related items, click either **EJB modules** or **Web modules >** *URI*. The *URI* is the Web services-enabled module
   a. Under Additional properties, click **Web services: client security bindings** to edit the response sender binding information, if Web services is acting as client.
      1) Under Response sender binding, click **Edit**.
      2) Under Additional Properties, click **Login binding**.
      3) Select **Dedicated login binding** to define a new login binding.
         a) Enter the authentication method, this must match the authentication method defined in IBM extension deployment descriptor. The authentication method must be unique in the binding file.
         b) Enter an implementation of the JAAS `javax.security.auth.callback.CallbackHandler` interface.
         c) Enter the basic authentication information (User ID and Password) and the basic authentication information is passed to the construct of the CallbackHandler implementation. The usage of the basic authentication information is up to the implementation of the CallbackHandler.
         d) Enter the token value type, it is optional for BasicAuth, Signature and IDAssertion authentication methods but required for any other authentication method. The token value type is inserted into the `<wsse:BinarySecurityToken>@ValueType` for binary security token and used as the namespace of the XML based token.
         e) Click **Properties**. Define the property with name and value pairs. These pairs are passed to the construct of the CallbackHandler implementation as `java.util.Map`.

      Select **None** to deselect the login binding.
   b. Under Additional Properties, click **Web services: server security bindings** to edit the request receiver binding information.
      1) Under Request Receiver Binding, click **Edit**.
      2) Under Additional Properties, click **Login mappings**.
      3) Click **New** to create new login mapping.
         a) Enter the authentication method, this must match the authentication method defined in the IBM extension deployment descriptor. The authentication method must be unique in the login mapping collection of the binding file.
         b) Enter a JAAS Login Configuration name. The JAAS Login Configuration must be defined under **Security > Global security**. Under Authentication, click **JAAS Configuration > Application Logins**. For more information, see “Configuring application logins for Java Authentication and Authorization Service” on page 242.
         c) Enter an implementation of the `com.ibm.wsspi.wssecurity.auth.callback.CallbackHandlerFactory` interface. This is a mandatory field.
         d) Enter the token value type, it is optional for BasicAuth, Signature and IDAssertion authentication methods but required for any other authentication method. The token value

type is used to validate against the `<wsse:BinarySecurityToken>@ValueType` for binary security token and against the namespace of the XML based token.

e) Enter the name and value pairs for the ″Login Mapping Property″ by clicking **Properties** . These name and value pairs are available to the JAAS Login Module or Modules by `com.ibm.wsspi.wssecurity.auth.callback.PropertyCallback` JAAS Callback. **Note**: This is true when editing existing login mappings but not when creating new login mappings.

f) Enter the name and value pairs for the ″Callback Handler Factory Property″, these name and value pairs is passed as `java.util.Map` to the `com.ibm.wsspi.wssecurity.auth.callback.CallbackHandlerFactory.init()` method. The usage of these name and value pairs is up to the CallbackHandlerFactory implementation.

c. Click authentication method link to edit the selected login mapping.

d. Click **Remove** to remove the selected login mapping or mappings.

3. Click **Save** .

The previous steps define how to configure the request sender to create security tokens in the Simple Object Access Protocol (SOAP) message and the request receiver to validate the security tokens found in the incoming SOAP message. WebSphere Application Server supports pluggable security tokens.

You can use the authentication method defined in the login bindings and login mappings to generate security tokens in the request sender and validate security tokens in the request receiver.

Once you have configured pluggable tokens, you must configure both the client and the server to support pluggable tokens. See the following topics to configure the client and the server:


- "Configuring the client for LTPA token authentication: specifying LTPA token authentication" on page 931
- "Configuring the client for LTPA token authentication: collecting the authentication method information" on page 932
- "Configuring the server to handle LTPA token authentication information" on page 933
- "Configuring the server to validate LTPA token authentication information" on page 933


***Pluggable token support:*** **Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

You can extend the WebSphere Application Server login mapping mechanism to handle new types of authentication tokens. WebSphere Application Server provides a pluggable framework to generate security tokens on the sender-side of the message and to validate the security token on the receiver-side of the message. The framework is based on the Java Authentication and Authorization Service (JAAS) Application Programming Interfaces (APIs). Pluggable security token support provides plug-in points to support customer security token types including token generation, token validation, and client identity mapping to a WebSphere Application Server identity that is used by the Java 2 platform, Enterprise Edition (J2EE) authorization engine. Moreover, the pluggable token generation and validation framework supports XML-based tokens to be inserted into the Web service message header and validated on the receiver-side validation.

Use the javax.security.auth.callback.CallbackHandler implementation to create a new type of security token following these guidelines:

- Use a constructor that takes a user name (a string or null, if not defined), a password (a char[] or null, if not defined) and java.util.Map (empty, if properties are not defined).
- Use handle() methods that can process the following implementations:
  - javax.security.auth.callback.NameCallback
  - javax.security.auth.callback.PasswordCallback
  - com.ibm.wsspi.wssecurity.auth.callback.XMLTokenCallback

– com.ibm.websphere.security.auth.callback.WSCredTokenCallbackImpl

If:

1. Either the javax.security.auth.callback.NameCallback or the javax.security.auth.callback.PasswordCallback implementation is populated with data, then a <wsse:UsernameToken> element is created.
2. com.ibm.websphere.security.auth.callback.WSCredTokenCallbackImpl is populated, the <wsse:BinarySecurityToken> element is created from the com.ibm.websphere.security.auth.callback.WSCredTokenCallbackImpl implementation.
3. com.ibm.wsspi.wssecurity.auth.callback.XMLTokenCallback is populated, a XML-based token is created based on the Document Object Model (DOM) element that is returned from the XMLTokenCallback.

Encode the token byte by using the security handler and not by using the javax.security.auth.callback.CallbackHandler implementation.

You can implement the com.ibm.wsspi.wssecurity.auth.callback.CallbackHandlerFactory interface, which is a factory for instantiating the javax.security.auth.callback.CallbackHandler implementation. For your own implementation, you must provide the javax.security.auth.callback.CallbackHandler interface. The Web service security run time instantiates the factory implementation class and passes the authentication information from the Web services message header to the factory class through the setter methods. The Web services security run time then invokes the newCallbackHandler() method of the factory implementation class to obtain an instance of the javax.security.auth.CallbackHandler object. The object is passed to the JAAS login configuration.

The following is an example the definition of the CallbackHandlerFactory interface:

```
public interface com.ibm.wsspi.wssecurity.auth.callback.CallbackHandlerFactory {
       public void setUsername(String username);
       public void setRealm(String realm);
       public void setPassword(String password);
       public void setHashMap(Map properties);
       public void setTokenByte(byte[] token);
       public void setXMLToken(Element xmlToken);
       public CallbackHandler newCallbackHandler();
```

***Configuring the client for LTPA token authentication: specifying LTPA token authentication:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

Use this task to configure Lightweight Third-Party Authentication (LTPA) token authentication. Only configure the client for LTPA token authentication if the authentication mechanism configured in WebSphere Application Server is LTPA. When a client authenticates to a WebSphere Application Server, the credential created contains an LTPA token. When a Web service calls a downstream Web service, you can configure the first Web service to send the LTPA token from the originating client. Do not attempt to configure LTPA from a pure client. LTPA works only when you configure the client-side of a Web service acting as a client to a downstream Web service. For the downstream Web service to validate the LTPA token, the LTPA keys on both servers must be the same.

Complete the following steps to specify LTPA token as the authentication method:

1. Launch an assembly tool. For more information on the assembly tools, see Assembly tools.
2. Click **Windows > Open perspective > Other > J2EE**.
3. Click **Application Client Projects >***application_name* **> appClientModule > META-INF**.
4. Right-click the `application-client.xml` file, select **Open with > Deployment descriptor editor**.

5. Click the Extensions tab, which is located at the bottom of the deployment descriptor editor within the assembly tool.
6. Expand the **Request sender configuration > Login configuration** section.
7. Select **LTPA** as the authentication method. For more conceptual information on LTPA authentication, see “Lightweight Third Party Authentication” on page 935.

After you specify LTPA token as the authentication method, you must specify how to collect the LTPA token information. See “Configuring the client for LTPA token authentication: collecting the authentication method information” for more information.

***Configuring the client for LTPA token authentication: collecting the authentication method information:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

Use this task to configure Lightweight Third-Party Authentication (LTPA) token authentication. Do not configure the client for LTPA token authentication unless the authentication mechanism configured in WebSphere Application Server is LTPA. When a client authenticates to a WebSphere Application Server, the credential created contains an LTPA token. When a Web service calls a downstream Web service, you can configure the first Web service to send the LTPA token from the originating client. Do not attempt to configure LTPA from a pure client. LTPA works only when you configure the client-side of a Web service acting as a client to a downstream Web service. In order for the downstream Web service to validate the LTPA token, the LTPA keys on both servers must be the same.

Complete the following steps to specify how to collect the LTPA token authentication information:
1. Launch an assembly tool. For more information on the assembly tools, see Assembly tools.
2. Click **Windows > Open perspective > Other > J2EE**.
3. Click **Application Client Projects >***application_name* **> appClientModule > META-INF**.
4. Right-click the `application-client.xml` file, select **Open with > Deployment descriptor editor**.
5. Click the WS Bindings tab, which is located at the bottom of the deployment descriptor editor within the assembly tool.
6. Expand the **Security request sender binding configuration > Login binding** section.
7. Click **Edit** to view the login binding information and select **LTPA**. If LTPA is not already there, enter it as an option. The login binding dialog is displayed. Select or enter the following information:

   **Authentication method**
   Specifies the type of authentication that occurs. Select **LTPA** to use identity assertion.

   **Token value type URI and token value type local name**
   When you select **LTPA**, you must edit the **token value type URI** and the **local name** fields. Specifies values for custom authentication types, which are authentication methods not mentioned in the specification. For the token value type URI field, enter the following string: `http://www.ibm.com/websphere/appserver/tokentype/5.0.2`. For the local name field, enter the following string: `LTPA`.

   **Callback handler**
   Specifies the Java Authentication and Authorization Service (JAAS) callback handler implementation for collecting the LTPA information. Specify the com.ibm.wsspi.wssecurity.auth.callback.LTPATokenCallbackHandler implementation for LTPA.

   **Basic authentication user ID and basic authentication password**
   For LTPA, you can leave these fields empty. However, when you omit this information, the

LTPA CallbackHandler implementation attempts to obtain the LTPA token from the invocation (RunAs) credential. If an invocation (RunAs) credential does not exist, then the LTPA token is not propagated.

**Property name and property value**
For LTPA, you can leave these fields empty.

See "Configuring the client for LTPA token authentication: specifying LTPA token authentication" on page 931 if you have not previously specified this information.

***Configuring the server to handle LTPA token authentication information:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

This task is used to configure Lightweight Third-Party Authentication (LTPA). LTPA is a type of authentication mechanism in WebSphere Application Server security that defines a particular token format. The purpose of the LTPA token authentication is to flow the LTPA token from the first Web service, which authenticated the originating client, to the downstream Web service. Do not attempt to configure LTPA from a pure client. Once the downstream Web service receives the LTPA token, it validates the token to verify that the token has not been modified and has not expired. For validation to be successful, the LTPA keys used by both the sending and receiving servers must be the same.

Complete the following steps to specify that LTPA is authentication method. The authentication method indicated in these steps must match the authentication method specified for the client.
1. Launch an assembly tool. For more information on the assembly tools, see Assembly tools.
2. Open the J2EE perspective by clicking **Window > Open perspective > Other > J2EE**.
3. Click **EJB Projects >***application_name* **> ejbModule > META-INF**.
4. Right-click the `webservices.xml` file, and click **Open with > Web services editor**.
5. Click the Extensions tab, which is located at the bottom of the Web services editor within the assembly tool.
6. Expand the **Request receiver service configuration details > Login configuration** section. You can select from the following options:
   - BasicAuth
   - Signature
   - ID assertion
   - LTPA
7. Select **LTPA** to authenticate the client using the LTPA token received from the request.

After you specify the authentication method, you must specify the information that the server must validate. See "Configuring the server to validate LTPA token authentication information" for more information.

***Configuring the server to validate LTPA token authentication information:***

**Important distinction between Version 5.x and Version 6 applications**

**Note:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

This task is used to configure Lightweight Third-Party Authentication (LTPA). LTPA is a type of authentication mechanism in WebSphere Application Server security that defines a particular token format. The purpose of the LTPA token authentication is to flow the LTPA token from the first Web service, which

authenticated the originating client, to the downstream Web service. Do not attempt to configure LTPA from a pure client. Once the downstream Web service receives the LTPA token, it validates the token to verify that the token has not been modified and has not expired. For validation to be successful, the LTPA keys used by both the sending and receiving servers must be the same.

Complete the following steps to specify how the server must validate the LTPA token authentication information:

1. Launch an assembly tool. For more information on the assembly tools, see Assembly tools.
2. Open the J2EE perspective by clicking **Window > Open perspective > Other > J2EE**.
3. Click **EJB Projects >***application_name* **> ejbModule > META-INF**.
4. Right-click the `webservices.xml` file, and click **Open with > Web services editor**.
5. Click the Binding Configurations tab, which is located at the bottom of the Web services editor within the assembly tool.
6. Expand the **Request receiver binding configuration details > Login mapping** section.
7. Click **Edit** to view the login mapping information. The login mapping information is displayed. Select or enter the following information:

   **Authentication method**
   : Specifies the type of authentication that occurs. Select **LTPA** to use LTPA token authentication.

   **Configuration name**
   : Specifies the Java Authentication and Authorization Service (JAAS) login configuration name. For the LTPA authentication method, enter `WSLogin` for the JAAS login configuration name. This configuration understands how to validate an LTPA token.

   **Use token value type**
   : Determines if you want to specify a custom token type. For LTPA authentication, you must select this option because LTPA is considered a custom type. LTPA is not in the Web Services Security Specification.

   **Token value type URI and local name**
   : Specifies custom authentication types. If you select **Use Token value type** you must enter data into the Token value Type URI and local name fields. For the token value type URI field, enter the following string: `http://www.ibm.com/websphere/appserver/tokentype/5.0.2`. For the local name, enter the following string: `LTPA`

   **Callback handler factory class name**
   : Creates a JAAS CallbackHandler implementation that understands the following callback handlers:

     - javax.security.auth.callback.NameCallback
     - javax.security.auth.callback.PasswordCallback
     - com.ibm.wsspi.wssecurity.auth.callback.BinaryTokenCallback
     - com.ibm.wsspi.wssecurity.auth.callback.XMLTokenReceiverCallback
     - com.ibm.wsspi.wssecurity.auth.callback.PropertyCallback

     For any of the default authentication methods (BasicAuth, IDAssertion, Signature, and LTPA), use the callback handler factory default implementation. Enter the following class name for any of the default authentication methods including LTPA:
     `com.ibm.wsspi.wssecurity.auth.callback.WSCallbackHandlerFactoryImpl`

     This implementation creates the correct callback handler for the default implementations.

   **Callback handler factory property**
   : Specifies callback handler properties for custom callback handler factory implementations. Default callback handler factory implementation does not any property specifications. For LTPA, you can leave this field blank.

**Login mapping property**
Specifies properties for a custom login mapping. For default implementations including LTPA, you can leave this field blank.

See "Configuring the server to handle LTPA token authentication information" on page 933 if you have not previously specified this information.

*Lightweight Third Party Authentication:*

When you use the lightweight third party authentication (LTPA) method, the <wsse:BinarySecurityToken> security token is generated. On the request sender side, the security token is generated by invoking a callback handler. On the request receiver side, the security token is validated by a Java Authentication and Authorization Service (JAAS) login module.

The following information describes token generation and token validation operations.

**LTPA token generation**
The request sender uses a callback handler to generate an LTPA security token. The callback handler returns a security token that is inserted in the Simple Object Access Protocol (SOAP) message. Specify the appropriate callback handler in the <LoginBinding> element of the bindings file (`ibm-webservicesclient-bnd.xmi`). The following callback handler implementation can be used with the LTPA authentication method:

- com.ibm.wsspi.wssecurity.auth.callback.LTPATokenCallbackHandler

You can add your own callback handlers that implement the `javax.security.auth.callback.CallbackHandler` property.

When using the LTPA authentication method (or any authentication method other than BasicAuth, Signature or IDAssertion), the TokenValueType attribute of the <LoginBinding> element in the bindings file (`ibm-webservicesclient-bnd.xmi`) must be specified. The values to use for the LTPA TokenValueType attribute are:

- `uri="http://www.ibm.com/websphere/appserver/tokentype/5.0.2"`
- `localName="LTPA"`

**LTPA token validation**
The request receiver retrieves the LTPA security token from the SOAP message and validates the message using a JAAS login module. The <wsse:BinarySecurityToken> security token is used to perform the validation. If the validation is successful, the login module returns a JAAS Subject. Subsequently, this Subject is set as the identity of the running thread. If the validation fails, the request is rejected with a SOAP fault.

The appropriate JAAS login configuration to use is specified in the bindings file <LoginMapping> element. Default bindings specified in the `ws-security.xml` file, but these can be overridden using the application-specific `ibm-webservices-bnd.xmi` file. The configuration information consists of a CallbackHandlerFactory, a ConfigName and a TokenValueType attribute. The CallbackHandlerFactory specifies the name of a class to use to create the JAAS CallbackHandler object. A CallbackHandlerFactory implementation is provided (com.ibm.wsspi.wssecurity.auth.callback.WSCallbackHandlerFactoryImpl). The ConfigName attribute specifies a JAAS configuration name entry. The Web services security run time first searches the `security.xml` file for a matching entry and if a matching entry is not found, the run time searches the `wsjaas.conf` file. A default configuration entry suitable for the LTPA authentication method is provided (`WSLogin`). An appropriate TokenValueType element is located in the LTPA LoginMapping section of the default `ws-security.xml` file.

**Remember:** The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6. The information does not apply to version 6 applications.

## Tuning Web services security

The Java Cryptography Extension (JCE) is integrated into the software development kit (SDK) version 1.4.x and is no longer an optional package. However, due to export and import regulations, the default Java Cryptography Extension (JCE) jurisdiction policy file shipped with the SDK enables you to use strong, but limited, cryptography only. To enforce this default policy, WebSphere Application Server uses a JCE jurisdiction policy file that introduces a significant performance impact. The default JCE jurisdiction policy has a significant performance impact on the cryptographic functions supported by Web services security. If you have Web services applications that use transport level security for XML encryption or digital signatures, you might encounter performance degradation over previous releases of WebSphere Application Server. However, IBM and Sun Microsystems provide versions of these jurisdiction policy files that do not have restrictions on cryptographic strengths. If you are permitted by your governmental import and export regulations, download one of these jurisdiction policy files. After downloading one of these files, the performance of JCE and Web Services security might improve substantially.

For WebSphere Application Server platforms using IBM Developer Kit, Java Technology Edition Version 1.4.2, including the AIX, Linux, and Windows platforms, you can obtain unlimited jurisdiction policy files by completing the following steps:

1. Go to the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html
2. Click **Java 1.4.2 material**
3. Click **IBM SDK Policy files**.
4. Select **Unrestricted JCE Policy files for SDK 1.4.2**
5. Enter your user ID and password or register with IBM to download the policy files. The policy files are downloaded onto your machine.

For WebSphere Application Server platforms using the Sun-based Java Development Kit (JDK) Version 1.4.2, including the Solaris environments and the HP-UX platform, you can obtain unlimited jurisdiction policy files by completing the following steps:

1. Go to the following Web site: http://java.sun.com/j2se/1.4.2/download.html
2. Click **Archive area**.
3. Locate the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 1.4.2 information and click **Download**. The policy files are downloaded onto your machine.

After following either of these sets of steps, two Java Archive (JAR) files are placed in the JVM `jre/lib/security/` directory.

# Configuring UDDI Security Roles

Each interface to the UDDI Registry (either through Version 1 and 2 SOAP, Version 3 SOAP, EJB or the GUI) is supplied with two roles:

**Publish role(s) and custody transfer**
: mapped to AllAuthenticatedUsers. By default, this is configured to use SSL (that is HTTPS), but this only applies when WebSphere security is enabled.

**Inquiry role**
: mapped to Everyone. By default, this is configured to use HTTP (that is not SSL).

In addition UDDI Version 3 SOAP has an additional role for version 3 SOAP Custody Transfer, which is mapped in the same way as the Publish Role.

The security role mappings can be altered by users through the Administration Console.

Authentication uses the standard WebSphere facilities and there is no separate registration function for the Registry. If WebSphere security is enabled, you will need to supply your WebSphere userid and password for Publish functions (unless you have changed the supplied Publish role).

You will need to set up WebSphere security configuration to be used by UDDI. It is expected that, for development use, security will be disabled and security will be enabled for production environments.

The V3 SOAP services also support the use of the V3 Security API set get_authToken and discard_authToken, but its use is optional, and defined by a combination of UDDI Policy and Security Roles.

- If security is enabled, and the service's Role is set to AllAuthenticated Users, WebSphere security takes priority over UDDI authentication, but if the Publish (or Custody Transfer) Role is mapped to Everyone and policy is set to require Authorization for publish (or Custody Transfer) (see WebSphere Administration console (UDDI => UDDI Nodes => UDDI NodeID => Policy Groups => APIs => General Properties)) then an authToken is required, and the user and password supplied in get_authToken will be checked by WebSphere.
- If security is disabled, Role Mapping does not apply, and the use of the V3 Security API set is defined by policy. If the 'Use authInfo if provided' option is checked (see WebSphere Administrative console (UDDI => UDDI Nodes => General Properties)) the userid supplied in get_authToken is used (but the password will not be checked). If 'Use authInfo if provided' is not checked the default user, set to UNAUTHENTICATED by default (but configurable, see WebSphere Administrative console (UDDI => UDDI Nodes => General Properties => Default user ID)) is used.

The V1/2 SOAP interface also supports the UDDI API for get_authToken and discard_authToken API but use of this is optional.

- If security is disabled and get_authToken is not called, the default user, UNAUTHENTICATED, is used.
- If security is disabled and get_authToken is called, the specified userid is used (but the password is not checked).
- If WebSphere security is enabled, it takes priority over UDDI authentication, but if the Publish role is mapped to Everyone, get_authToken must be used and the userid and password will be checked by WebSphere.

The Security Roles provided with the UDDI Registry are as follows:

- GUI_Publish_User
- GUI_Inquiry_User
- SOAP_Publish_User
- SOAP_Inquiry_user
- EJB_Inquiry_Role
- EJB_Publish_Role
- V3 SOAP_Inquiry_User_Role
- V3 SOAP_Publish_User_Role
- V3 SOAP_CustodyTransfer_User_Role
- V3 SOAP_Security_User_Role

## Security API for the UDDI V3 Registry

The security API is a part of the Version 2 Publishing API, but in Version 3 it has its own API set. The Version 3 security API includes the following API calls:

**discard_authToken**
: Used to inform a node that a previously obtained authentication token is no longer required and should be considered invalid if used after this message is received. The token is to be discarded and the session is effectively ended.

**get_authToken**
Used to request an authentication token in the form of an authInfo element from a UDDI node.

For full details of the syntax of the above queries, refer to the API specification at `http://www.uddi.org/pubs/uddi_v3.htm`.

# Data access resources

## Security of lookups with component managed authentication

External Java clients (stand alone clients or servers from other cells) with Java Naming and Directory Interface (JNDI) access can look up a Java 2 Connector (J2C) resource such as a data source or Java Message Service (JMS) queue. However, they are not permitted to take advantage of the component managed *authentication alias* defined on the resource. This alias is a default value used when the *user* and *password* are not supplied on the getConnection() call. Therefore, if an external client needs to get a connection, it must assume responsibility for the authentication by passing it through arguments on the getConnection() call.

Any client running in the WebSphere Application Server process (such as a Servlet or an enterprise bean) within the same cell that can look up a resource in the JNDI namespace can obtain connections without explicitly providing authentication data on the getConnection() call. In this case, if the component's res-auth setting is **Application**, authentication is taken from the component-managed authentication alias defined on the connection factory. With res-auth set to **Container**, authentication is taken from the login configuration defined on the component's resource-reference. It is important to note that J2C authentication alias is per **cell**. An enterprise bean or Servlet in one application server cannot look up a resource in another server process which is in a different cell, because the alias would not be resolved.

# Messaging resources

## Configuring authorization security for a Version 5 default messaging provider

Use this task to configure authorization security for the default messaging provider on a WebSphere Application Server version 5 node in a deployment manager cell.

To configure authorization security for the version 5 default messaging provider complete the following steps.

**Note:** Security for the version 5 default messaging provider is enabled when you enable global security for WebSphere Application Server on the version 5 node. For more information about enabling global security, see Managing secured applications.

1. Configure authorization settings to access JMS resources owned by the embedded WebSphere JMS provider. Authorization to access JMS resources owned by the embedded WebSphere JMS provider is controlled by settings in the `*WAS_install_root*\config\cells*your_cell_name*\integral-jms-authorizations.xml` file.

   The settings grant or deny authenticated userids access to internal JMS provider resources (queues or topics). As supplied, the integral-jms-authorisations.xml file grants the following permissions:
   - Read and write permissions to all queues.
   - Pub, sub, and persist to all topics.

   To configure authorization settings, edit the integral-jms-authorisations.xml file according to the information in this topic and in that file. Please note the file is in Unicode, which requires a binary FTP to the host from a workstation.

2. Edit the queue-admin-userids element to create a list of userids with administrative access to all queues. Administrative access is needed to create queues and perform other administrative activities on queues. For example, consider the following queue-admin-userids section:

```
<queue-admin-userids>
    <userid>adminid1</userid>
    <userid>adminid2</userid>
</queue-admin-userids>
```

   In this example the userids adminid1 and adminid2 are defined to have administrative access to all queues.

3. Edit the queue-default-permissions element to define the default queue access permissions. These permissions are used for queues for which you do not define specific permissions (in queue sections). If this section is not specified, then access permissions exist only for those queues for which you have specifically created queue elements.

   For example, consider the following queue-default-permissions element:

```
<queue-default-permissions>
  <permission>write</permission>
</queue-default-permissions>
```

   In this example the default access permission for all queues is **write**. This can be overridden for a specific queue by creating a queue element that sets its access permission to **read**.

4. If you want to define specific access permissions for a queue, create a queue element, then define the following elements:

   For example, consider the following queue element:

```
<queue>
  <name>q1</name>
  <public>
  </public>
  <authorize>
    <userid>useridr</userid>
    <permission>read</permission>
  </authorize>
  <authorize>
    <userid>useridw</userid>
    <permission>write</permission>
  </authorize>
  <authorize>
    <userid>useridrw</userid>
    <permission>read</permission>
    <permission>write</permission>
  </authorize>
</queue>
```

   In this example for the queue q1, the userid useridr has read permission, the userid useridw has write permission, the userid useridrw has both read and write permissions, and all other userids have no access permissions (<public></public>).

5. Edit topic elements to define the access permissions for publish/subscribe topic destinations.

   For topics, you can grant and deny access permissions. Full permission inheritance is supported on topics. If you do not define specific access permissions for a userid on a specific topic then permissions are inherited first from the public permissions on that topic then from the parent topic. The inheritance of access permissions continues until the root topic from which the root permissions are assumed.

   a. If you want to define default access permissions for the root topic, edit a topic element with an empty name element. If you omit such a topic section, topics have no default topic permissions other than those defined by specific topic elements. For example, consider the following topic element for the root topic:

```
<topic>
   <name></name>
   <public>
     <permission>+pub</permission>
   </public>
 </topic>
```

In this example, the default access permission for all topics is set to publish. This can be overridden by other topic elements for specific topic names.

b. If you want to define access permissions for a specific topic, create a topic element with the name for the topic then define the access permissions in the public and authorize elements of the topic element. For example, consider the following topic section:

```
<topic>
  <name>a/b/c</name>
  <public>
    <permission>+sub</permission>
  </public>
  <authorize>
    <userid>useridpub</userid>
    <permission>+pub</permission>
  </authorize>
</topic>
```

In this example, the subscribe permission is granted to anyone accessing any topic whose name starts with `a/b/c`. Also, the userid useridpub is granted publish permission for any topic whose name starts with `a/b/c`.

6. Save the integral-jms-authorizations.xml file.

If the dynamic update setting is selected, changes to the integral-jms-authorizations.xml file become active when the changed file is saved, so there is no need to stop and restarted the JMS server. If the dynamic update setting is not selected, you need to stop and restart the JMS server to make changes active.

## Authorization settings for Version 5 default JMS resources

Use the integral-jms-authorisations.xml file to view or change the authorization settings for JMS resources owned by the default messaging provider on WebSphere Application Server version 5 nodes.

Authorization to access default JMS resources owned by the default messaging provider on WebSphere Application Server nodes is controlled by the following settings in the *was_install*`\config\cells\`*your_cell_name*`\integral-jms-authorisations.xml` file.

This structure of the settings in integral-jms-authorisations.xml is shown in the following example. Descriptions of these settings are provided after the example. To configure authorization settings, follow the instructions provided in Configuring authorization security for the Version 5 JMS providers

```
<integral-jms-authorizations>

  <dynamic-update>true</dynamic-update>

  <queue-admin-userids>
    <userid>adminid1</userid>
    <userid>adminid2</userid>
  </queue-admin-userids>

  <queue-default-permissions>
    <permission>write</permission>
  </queue-default-permissions>

  <queue>
    <name>q1</name>
    <public>
    </public>
    <authorize>
      <userid>useridr</userid>
```

```
      <permission>read</permission>
    </authorize>
    <authorize>
      <userid>useridw</userid>
      <permission>write</permission>
    </authorize>
  </queue>

  <queue>
    <name>q2</name>
    <public>
      <permission>write</permission>
    </public>
    <authorize>
      <userid>useridr</userid>
      <permission>read</permission>
    </authorize>
  </queue>

  <topic>
    <name></name>
    <public>
      <permission>+pub</permission>
    </public>
  </topic>

  <topic>
    <name>a/b/c</name>
    <public>
      <permission>+sub</permission>
    </public>
    <authorize>
      <userid>useridpub</userid>
      <permission>+pub</permission>
    </authorize>
  </topic>

</integral-jms-authorizations>
```

***dynamic-update:*** Controls whether or not the JMS Server checks dynamically for updates to this file.

**true** (Default) Enables dynamic update support.

**false** Disables dynamic update checking and improves authorization performance.

***queue-admin-userids:*** This element lists those userids with administrative access to all Version 5 default queue destinations. Administrative access is needed to create queues and perform other administrative activities on queues. You define each userid within a separate userid sub element:

**<userid>***adminid***</userid>**

Where *adminid* is a user ID that can be authenticated by IBM WebSphere Application Server.

***queue-default-permissions:*** This element defines the default queue access permissions that are assumed if no permissions are specified for a specific queue name. These permissions are used for queues for which you do not define specific permissions (in queue elements). If this element is not specified, then no access permissions exist unless explicitly authorized for individual queues.

You define the default permission within a separate permission sub element:

**<permission>***read-write***</permission>**

Where *read-write* is one of the following keywords:

**read** By default, userids have read access to Version 5 default queue destinations.

**write** By default, userids have write access to Version 5 default queue destinations.

***queue:*** This element contains the following authorization settings for a single queue destination:

**name** The name of the queue.

**public** The default public access permissions for the queue. This is used only for those userids that have

no specific authorize element. If you leave this element empty, or do not define it at all, only those userids with authorize elements can access the queue.

You define each default permission within a separate permission element.

**authorize**
The access permissions for a specific userid. Within each authorize element, you define the following elements:

**userid** The userid that you want to assign a specific access permission.

**permission**
An access permission for the associated userid.

You define each permission within a separate permission element. Each permission element can contain the keyword read or write to define the access permission.

For example, consider the following queue element:

```
<queue>
  <name>q1</name>
  <public>
  </public>
  <authorize>
    <userid>useridr</userid>
    <permission>read</permission>
  </authorize>
  <authorize>
    <userid>useridw</userid>
    <permission>write</permission>
  </authorize>
  <authorize>
    <userid>useridrw</userid>
    <permission>read</permission>
    <permission>write</permission>
  </authorize>
</queue>
```

***topic:*** This element contains the following authorization settings for a single topic destination:

Each topic element has the following sub elements:

**name** The name of the topic, without wildcards or other substitution characters.

**public** The default public access permissions for the topic. This is used only for those userids that have no specific authorize element. If you leave this element empty, or do not define it at all, only those userids with authorize elements can access the topic.

You define each default permission within a separate permission element.

**authorize**
The access permissions for a specific userid. Within each authorize element, you define the following elements:

**userid** The userid that you want to assign a specific access permission.

**permission**
An access permission for the associated userid.

You define each permission within a separate permission element. Each permission element can contain one of the following keywords to define the access permission:

**+pub** Grant publish permission

**+sub** Grant subscribe permission

**+persist**
Grant persist permission

**-pub** Deny publish permission

**-sub** Deny subscribe permission

**-persist**
Deny persist permission

## Securing WebSphere MQ messaging directories and log files

Use this task to restrict access to the /var/mqm directories and log files needed for WebSphere MQ as a JMS provider.

You need to set the permissions described in this topic, to reduce the risk of severe security exposures.

**Note:** The /var file system is used to store all the security logging information for the system, and is used to store the temporary files for email and printing. Therefore, it is critical that you maintain free space in /var for these operations and prevent unauthorized access to the file system. If you do not create a separate file system for messaging data, and /var fills up, all security logging will be stopped on the system until some free space is available in /var. Also, email and printing will no longer be possible until some free space is available in /var.

This procedure involves steps that you complete at different stages of installing and using IBM WebSphere Application Server, as described below. The steps are also described at appropriate points in other tasks, but are collected here for completeness.

**This procedure applies only to the ordinary UNIX file system.** If your site uses access-control lists, secure the files by using that mechanism. Any site-specific requirements can affect the desired owner, group and corresponding privileges. For example, on AIX, complete the following steps:

1. Before installing WebSphere MQ, create and mount a journalized file system called /var/mqm. Use a partition strategy with a separate volume for the messaging data. This means that other system activity is not affected if a large amount of messaging work builds up in /var/mqm.
2. Install WebSphere MQ as a messaging provider.

   As part of this stage, the installation program creates the /var/mqm/errors and /var/mqm/qmgrs/@SYSTEM/errors directories used to hold messaging logging files.
3. Restrict access to the /var/mqm/errors directory and the logging files, by using the following commands:

   ```
   chmod 3777 /var/mqm/errors
   chown mqm:mqm /var/mqm/errors

   touch /var/mqm/errors/AMQERR01.LOG
   chown mqm:mqm /var/mqm/errors/AMQERR01.LOG
   chmod 666 /var/mqm/errors/AMQERR01.LOG

   touch /var/mqm/errors/AMQERR02.LOG
   chown mqm:mqm /var/mqm/errors/AMQERR02.LOG
   chmod 666 /var/mqm/errors/AMQERR02.LOG

   touch /var/mqm/errors/AMQERR03.LOG
   chown mqm:mqm /var/mqm/errors/AMQERR03.LOG
   chmod 666 /var/mqm/errors/AMQERR03.LOG
   ```
4. Restrict access to the /var/mqm/qmgrs/@SYSTEM/errors directory and the logging files, by using the following commands:

   ```
   chmod 3777 /var/mqm/qmgrs/@SYSTEM/errors
   chown mqm:mqm /var/mqm/qmgrs/@SYSTEM/errors

   touch /var/mqm/qmgrs/@SYSTEM/errors/AMQERR01.LOG
   chown mqm:mqm /var/mqm/qmgrs/@SYSTEM/errors/AMQERR01.LOG
   chmod 666 /var/mqm/qmgrs/@SYSTEM/errors/AMQERR01.LOG

   touch /var/mqm/qmgrs/@SYSTEM/errors/AMQERR02.LOG
   chown mqm:mqm /var/mqm/qmgrs/@SYSTEM/errors/AMQERR02.LOG
   chmod 666 /var/mqm/qmgrs/@SYSTEM/errors/AMQERR02.LOG

   touch /var/mqm/qmgrs/@SYSTEM/errors/AMQERR03.LOG
   chown mqm:mqm /var/mqm/qmgrs/@SYSTEM/errors/AMQERR03.LOG
   chmod 666 /var/mqm/qmgrs/@SYSTEM/errors/AMQERR03.LOG
   ```

5. For each application server that uses JMS resources provided by WebSphere MQ, restrict access to the server's /var/mqm/qmgrs/*long_server_name*/errors directory and its messaging logging files. You should restrict access to the server's directory and logging files, as soon after creating the application server as possible.

   To restrict access to the server's directory and logging files, use the following commands:

   ```
   chmod 3775 /var/mqm/qmgrs/long_server_name/errors
   chown mqm:mqm /var/mqm/qmgrs/long_server_name/errors

   touch /var/mqm/qmgrs/long_server_name/errors/AMQERR01.LOG
   chown mqm:mqm /var/mqm/qmgrs/long_server_name/errors/AMQERR01.LOG
   chmod 666 /var/mqm/qmgrs/long_server_name/errors/AMQERR01.LOG

   touch /var/mqm/qmgrs/long_server_name/errors/AMQERR02.LOG
   chown mqm:mqm /var/mqm/qmgrs/long_server_name/errors/AMQERR02.LOG
   chmod 666 /var/mqm/qmgrs/long_server_name/errors/AMQERR02.LOG

   touch /var/mqm/qmgrs/long_server_name/errors/AMQERR03.LOG
   chown mqm:mqm /var/mqm/qmgrs/long_server_name/errors/AMQERR03.LOG
   chmod 666 /var/mqm/qmgrs/long_server_name/errors/AMQERR03.LOG
   ```

   Where *long_server_name* is the long name assigned to the server, in the following form: `WAS_nodename_server_name`. For example, if you created an application server called `server1` to run on the node called `appnode1`, the long server name would be: `WAS_appnode1_server1`.

This task has restricted access to the /var/mqm directories and log files needed for WebSphere MQ as a JMS provider, such that only the user ID mqm or members of the mqm user group have write access.

## Configuring security for message-driven beans that use listener ports

Use this task to configure resource security and security permissions for EJB 2.0 message-driven beans deployed to use listener ports.

Messages arriving at a listener port have no client credentials associated with them. The messages are anonymous.

To call secure enterprise beans from a message-driven bean, the message-driven bean needs to be configured with a RunAs Identity deployment descriptor. Security depends on the role specified by the RunAs Identity for the message-driven bean as an EJB component.

For more information about EJB security, see EJB component security. For more information about configuring security for your application, see Assembling secured applications.

JMS connections used by message-driven beans can benefit from the added security of using J2C container-managed authentication. To enable the use of J2C container authentication aliases and mapping, define a J2C container-managed alias on the JMS connection factory definition that the message-driven bean is using to listen upon (defined by the **Connection factory JNDI name** property of the listener port). If defined, the listener uses the container-managed authentication alias for its JMSConnection security credentials instead of any application-managed alias. To set the container-managed alias, use the administrative console to complete the following steps:

1. To display the listener port settings, click **Servers** → **Application Servers** → *application_server* → **[Communications] Messaging** → **Message Listener Service** → **Listener Ports** → *listener_port*
2. To get the name of the JMS connection factory, look at the **Connection factory JNDI name** property.
3. Display the JMS connection factory properties. For example, to display the properties of a WebSphere queue connection factory provided by the default messaging provider, click **Resources** → **JMS Providers** → **Default Messaging Provider** → → **[Content pane] WebSphere Queue Connection Factories** → *connection_factory*
4. Set the **Authentication alias** property.
5. Click **OK**

## Configuring security for EJB 2.1 message-driven beans

Use this task to configure resource security and security permissions for EJB 2.1 message-driven beans.

Messages handled by message-driven beans have no client credentials associated with them. The messages are anonymous.

To call secure enterprise beans from a message-driven bean, the message-driven bean needs to be configured with a RunAs Identity deployment descriptor. Security depends on the role specified by the RunAs Identity for the message-driven bean as an EJB component.

For more information about EJB security, see EJB component security. For more information about configuring security for your application, see Assembling secured applications.

Connections used by message-driven beans can benefit from the added security of using J2C container-managed authentication. To enable the use of J2C container authentication aliases and mapping, define an authentication alias on the J2C activation specification that the message-driven bean is configured with. If defined, the message-driven bean uses the authentication alias for its JMSConnection security credentials instead of any application-managed alias.

To set the authentication alias, you can use the administrative console to complete one the following steps. This task description assumes that you have already created an activation specification. If you want to create a new activation specification, see the related tasks.

- For a message-driven bean listening on a JMS destination of the default messaging provider, set the authentication alias on a JMS activation specification.
  1. To display the JMS activation specification settings, click **Resources** → **JMS Providers** → **Default messaging** → **[Activation Specifications] JMS activation specification**
  2. If you have already created a JMS activation specification, click its name in the list displayed. Otherwise, click **New** to create a new JMS activation specification.
  3. Set the **Authentication alias** property.
  4. Click **OK**
  5. Save your changes to the master configuration.
- For a message-driven bean listening on a destination (or endpoint) of another JCA provider, set the authentication alias on a J2C activation specification.
  1. To display the J2C activation specification settings, click **Resources** → **Resource Adapters** → **adapter_name** → **J2C Activation specifications** → **activation specification_name**
  2. Set the **Authentication alias** property.
  3. Click **OK**
  4. Save your changes to the master configuration.

# Mail, URLs, and other J2EE resources

## JavaMail security permissions best practices

In many of its activities, the JavaMail API needs to access certain configuration files. The JavaMail and JavaBeans Activation Framework binary packages themselves already contain the necessary configuration files. However, the JavaMail API allows the user to define user-specific and installation-specific configuration files to meet special requirements.

The two locations where such configuration files can exist are `<user.home>`and `<java.home>/lib`. For example, if the JavaMail API needs to access a file named `mailcap` when sending a message, it first tries to access `<user.home>/.mailcap`. If that attempt fails, either due to lack of security permission or a nonexistent file, the API continues to try`<java.home> /lib/mailcap`. If that attempts also fails, it tries META-INF/mailcap in the class path, which actually leads to the configuration files contained in the

mail.jar and activation.jar files. WebSphere Application Server uses the general-purpose JavaMail configuration files contained in the mail.jar and activation.jar files and does not put any mail configuration files in <user.home>and <java.home>/lib. File read permission for both the mail.jar and activation.jar files is granted to all applications to ensure proper functioning of the JavaMail API, as shown in the following segment of the app.policy file:

```
grant codeBase "file:${application}" {
 // The following are required by Java mail
permission java.io.FilePermission "${was.install.root}${/}java${/}jre${/}lib${/}ext${/}mail.jar", "read";
permission java.io.FilePermission "${was.install.root}${/}java${/}jre${/}lib${/}ext${/}activation.jar", "read";
};
```

JavaMail code attempts to access configuration files at <user.home>and <java.home>/lib causing AccessControlExceptions to be thrown, since there is no file read permission granted for those two locations by default. This activity does not affect the proper functioning of the JavaMail API, but you might see a large amount of JavaMail-related security exceptions reported in the system log, which might swamp harmful errors that you are looking for. If this situation is a problem, consider adding two more permission lines to the permission block above. This should eliminate most, if not all, JavaMail-related harmless security exceptions from the log file. The application permission block in the app.policy file now looks like:

```
 grant codeBase "file:${application}" {
// The following are required by Java mail
permission java.io.FilePermission "${was.install.root}${/}java${/}jre${/}lib${/}ext${/}mail.jar", "read";
permission java.io.FilePermission "${was.install.root}${/}java${/}jre${/}lib${/}ext${/}activation.jar", "read";
permission java.io.FilePermission "${user.home}${/}.mailcap", "read";
permission java.io.FilePermission "${java.home}${/}lib${/}mailcap", "read";
};
```

## Learn about WebSphere programming extensions

Use this section as a starting point to investigate the WebSphere programming model extensions for enhancing your application development and deployment.

See Learn about WebSphere applications: Overview and new features for an introduction to each WebSphere extension.

| | | | | |
|---|---|---|---|---|
| ActivitySessions | How do I?... | Overview | | Samples |
| Application profiling | How do I?... | Overview | | Samples |
| Asynchronous beans | How do I?... | Overview | | Samples |
| Dynamic caching | How do I?... | Overview | | |
| Dynamic query | How do I?... | Overview | | Samples |
| Internationalization | How do I?... | Overview | | Samples |
| Object pools | How do I?... | Overview | | |
| Scheduler | How do I?... | Overview | | Samples |
| Startup beans | How do I?... | Overview | | |
| Work areas | How do I?... | Overview | | |

## Scheduler

### Securing scheduler tasks

Scheduled tasks are protected using application isolation and administrative roles. If a task is created using a J2EE server application, only applications with the same name can access those tasks. Tasks created with a WASScheduler MBean using the AdminClient interface or scripting are not part of a J2EE

application and have access to all tasks regardless of the application with which they were created. Tasks created with a WASScheduler MBean are only accessible from the WASScheduler MBean API and are not accessible from the Scheduler API.

If the Use Administration Roles attribute is enabled on a scheduler and global security is enabled on the Application Server, all Scheduler API methods and WASScheduler MBean API operations enforce access based on the WebSphere Administration Roles. If either of these attributes are disabled, then all API methods are fully accessible by all users.

1. Configure global security.
2. Manage schedulers.

**Related concepts**

"Global security" on page 141
Global security applies to all applications running in the environment and determines whether security is used at all, the type of registry against which authentication takes place, and other values, many of which act as defaults.

Scheduler task user authorization

**Related tasks**

"Configuring global security" on page 142

Managing schedulers

**Related reference**

"Admin roles" on page 122

Scheduler interface

# Chapter 15. Tuning security configurations

Performance issues typically involve trade-offs between function and speed. Usually, the more function and the more processing involved, the slower the performance. Consider what type of security is necessary and what you can disable in your environment. For example, if your application servers are running in a Virtual Private Network (VPN), consider whether you must disable Single Sockets Layer (SSL). If you have a lot of users, can they be mapped to groups and then associated to your J2EE roles? These questions are things to consider when designing your security infrastructure.

Complete the following steps for general security tuning:

1. Consider disabling Java 2 Security Manager if you know exactly what code is put onto your server and you do not need to protect process resources. Remember that in doing so, you put your local resources at some risk.
2. Disable security for the specific application server that does not require resource protection because some application servers do not have protected resources. If the application server needs to go downstream with credentials, however, this action might not be feasible.
3. Consider propagating new security settings to all nodes before restarting the deployment manager and node agents to change the new security policy.

   Configuration changes are generally propagated using configuration synchronization. If auto-synchronization is enabled, you can wait for the automatic synchronization interval to pass, or you can force synchronization before the synchronization interval expires. If you are using manual synchronization, you must synchronize all nodes.

   If the cell is in a configuration state (the security policy is mixed with nodes that have security enabled and disabled) you can use the syncNode utility to synchronize the nodes where the new settings are not propagated.

   Refer to the article, “Enabling global security” on page 144 in the WebSphere Application Server Network Deployment package for more detailed information about enabling security in a distributed environment.
4. Consider increasing the cache and token time-out if you feel your environment is secure enough. By doing so, you have to re-authenticate less often. This action supports subsequent requests to reuse the credentials that already are created. The downside of increasing the token time-out is the exposure of having a token hacked and providing the hacker more time to hack into the system before the token expires. You can use security cache properties to determine the initial size of the primary and secondary hashtable caches, which affect the frequency of rehashing and the distribution of the hash algorithms.

   See the article “Security cache properties” on page 951 for a list of these properties.
5. Consider changing your administrative connector from Simple Object Access Protocol (SOAP) to Remote Method Invocation (RMI) because RMI uses stateful connections while SOAP is completely stateless. Run a benchmark to determine if the performance is improved in your environment.
6. Use the `wsadmin` script to complete the access IDs for all the users and or groups to speed up the application startup. Complete this action if applications contain many users, or groups, or if applications are stopped and started frequently.
7. Consider tuning the Object Request Broker (ORB) because it is a factor in enterprise bean performance with or without security enabled. Refer to the article, ORB tuning guidelines.

## Tuning CSIv2

1. Consider using SSL client certificates instead of a user ID and password to authenticate Java clients. Since you are already making the SSL connection, using mutual authentication adds little overhead while removing the service context containing the user ID and password completely.

© Copyright IBM Corp. 2004

2. If you send a large amount of data that is not very security sensitive, reduce the strength of your ciphers. The more data you have to bulk encrypt and the stronger the cipher, the longer this action takes. If the data is not sensitive, do not waste your processing with 128-bit ciphers.
3. Consider putting just an asterisk (*) in the trusted server ID list (meaning trust all servers) when you use Identity Assertion for downstream delegation. Use SSL mutual authentication between servers to provide this trust. Adding this extra step in the SSL handshake performs better than having to fully authenticate the upstream server and check the trusted list. When an asterisk is used, we simply trust the identity token. The SSL connection trusts the server by way of client certificate authentication.
4. Ensure that stateful sessions are enabled for Common Secure Interoperability Version 2 (CSIv2). This is the default, but only requires authentication on the first request and any subsequent token expirations.
5. If you are only communicating with WebSphere Application Server Version 5 servers, make the Active Authentication Protocol CSI, instead of CSI and SAS. This action removes an interceptor invocation for every request on both the client and server sides.
6. For a pure Java client, you can disable the creation of server sockets used for Object Request Broker (ORB) callbacks. Do this only if you are communicating with servers running WebSphere Application Server, Version 5 or later.
   a. In the `sas.client.props` file, add `com.ibm.CSI.claimTransportAssocSSLTLSRequired=false` and `com.ibm.CSI.claimTransportAssocSSLTLSSupported=false`.
   b. Set the active protocol to `csiv2` instead of `both` in the `sas.client.props` file. The protocol property changes to `com.ibm.CSI.protocol=csiv2`.

## Tuning LDAP authentication

1. Select the **Ignore Case** check box in the WebSphere Application Server LDAP User Registry configuration, when case-sensitivity is not important.
2. Select **Reuse Connections** in the WebSphere Application Server LDAP User Registry configuration.
3. Check to see which caches your LDAP server has and take advantage of them. This action is best with LDAP servers that do not change frequently.
4. Choose the directory type of either IBM_Directory_Server or SecureWay, if you are using an IBM Directory Server. The IBM Directory Server yields improved performance because it is programmed to use the new group membership attributes to improve group membership searches. However, it is required that authorization is case insensitive to use IBM Directory Server.
5. Choose either iPlanet Directory Server (also known as Sun ONE) or Netscape as the directory if you are an iPlanet Directory user. Using the iPlanet Directory Server directory increases performance in group membership lookup. However, only use **Role** for group mechanisms.

## Tuning Web authentication

1. Consider increasing the cache and token time-out if you feel your environment is secure enough. The Web authentication information is stored in these caches and as long as the authentication information is in the cache, the login module is not invoked to authenticate the user. This supports subsequent requests to reuse the credentials already created. The downside of increasing the token time-out is the exposure of having a token stolen and providing the thief more time to hack into the system before the token expires.

   See the article "Security cache properties" on page 951 for a list of these properties.
2. Consider enabling single singon (SSO). SSO is only available when you select **LTPA** as the authentication mechanism in the **Global Security** panel. When you select SSO, a single authentication to one application server is enough to make requests to multiple application servers in the same SSO domain. There are some situations where SSO is not desirable and should not be used in those situations.
3. Consider disabling or enabling the **Web Inbound Security Attribute Propagation** option on the SSO panel if the function is not required. In some cases, having the function enabled improves

performance. This improvement is most likely for higher volume cases where a considerable number of user registry calls reduces performance. In other cases, having the feature disabled improves performance. This improvement is most likely when the user registry calls do not take considerable resources.

## Tuning authorization

1. Consider mapping your users to groups in the user registry. Then associate the groups with your J2EE roles. This association greatly improves performance as the number of users increases.
2. Judiciously assign method-permissions for enterprise beans. For example, you can use an asterisk (*) to indicate all methods in the method-name element. When all the methods in enterprise beans require the same permission, use an asterisk (*) for the method-name to indicate all methods. This indication reduces the size of deployment descriptors and reduces the memory required to load the deployment descriptor. It also reduces the search time during method-permission match for the enterprise beans method.
3. Judiciously assign security-constraints for servlets. For example, you can use the URL pattern `*.jsp` to apply the same authentication data constraints to indicate all JSP files. For a given URL, the exact match in the deployment descriptor takes precedence over the longest path match. Use the extension match (`*.jsp`, `*.do`, `*.html`) if there is no exact match and longest path match for a given URL in the security constraints.

There is always a trade off between performance, feature and security. Security typically adds more processing time to your requests, but for a good reason. Not all security features are required in your environment. When you decide to tune security, you should create a benchmark before making any change to ensure the change is improving performance.

In a large scale deployment, performance is very important. Running benchmark measurements with different combinations of features can help you to determine the best performance versus the benefit configuration for your environment. Continue to run benchmarks if anything changes in your environment, to help determine the impact of these changes.

## Security cache properties

The following administrative console custom properties determine the initial size of the primary and secondary hash table caches, which affect the frequency of rehashing and the distribution of the hash algorithms. To get to the custom properties click **Servers > Application Servers > server-name > Process Definition > Control | Servant > Java Virtual Machine > Custom Properties**. The larger the number of available hash values, the less likely a hash collision occurs, retrieval time might be slower. If several entries compose a hash table cache, creating the table in a larger capacity supports more efficient hash entries than letting automatic rehashing determine the growth of the table. Rehashing causes every entry to move each time.

The following explains why you would use these particular hash tables.

**com.ibm.websphere.security.util.tokenCacheSize**

This cache stores LTPA credentials in the cache using the LTPA token as a lookup value. When using an LTPA token to log in, the LTPA credential is created at the security server for the first time. This cache prevents the need to go to the security server on subsequent logins using an LTPA token.

**com.ibm.websphere.security.util.LTPAValidationCacheSize**

Given the credential token for login, this cache returns the concrete LTPA credential object, without the need to revalidate at the security server. If the token has expired, revalidation is required.

## Secure Sockets Layer performance tips

The following are two types of Secure Sockets Layer (SSL) performance:
- Handshake
- Bulk encryption and decryption

When an SSL connection is established, an SSL handshake occurs. After a connection is made, SSL performs bulk encryption and decryption for each read-write. The performance cost of an SSL handshake is much larger than that of bulk encryption and decryption.

To enhance SSL performance, decrease the number of individual SSL connections and handshakes.

Decreasing the number of connections increases performance for secure communication through SSL connections, as well as non-secure communication through simple TCP/IP connections. One way to decrease individual SSL connections is to use a browser that supports HTTP 1.1. Decreasing individual SSL connections can be impossible if you cannot upgrade to HTTP 1.1.

Another common approach is to decrease the number of connections (both TCP/IP and SSL) between two WebSphere Application Server components. The following guidelines help to verify the HTTP transport of the application server is configured so that the Web server plug-in does not repeatedly reopen new connections to the application server:

- Verify that the maximum number of keep alives are, at minimum, as large as the maximum number of requests per thread of the Web server (or maximum number of processes for IBM HTTP Server on UNIX). Make sure that the Web server plug-in is capable of obtaining a keep alive connection for every possible concurrent connection to the application server. Otherwise, the application server closes the connection after a single request is processed. Also, the maximum number of threads in the Web container thread pool should be larger than the maximum number of keep alives, to prevent the keep alive connections from consuming the Web container threads.

  **Note:** HTTP Transports have been deprecated. For instructions on how to set a maximum keep alive value for channel based configurations, see HTTP transport channel settings.
- Increase the maximum number of requests per keep alive connection. The default value is 100, which means the application server closes the connection from the plug-in after 100 requests. The plug-in then has to open a new connection. The purpose of this parameter is to prevent denial of service attacks when connecting to the application server and preventing continuous send requests to tie up threads in the application server.
- Use a hardware accelerator if the system performs several SSL handshakes.

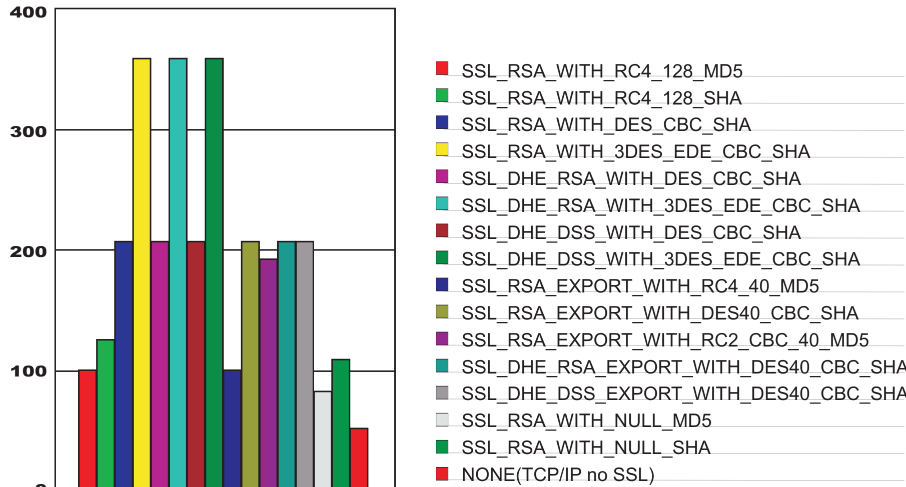
  Hardware accelerators currently supported by WebSphere Application Server only increase the SSL handshake performance, not the bulk encryption and decryption. An accelerator typically only benefits the Web server because Web server connections are short-lived. All other SSL connections in WebSphere Application Server are long-lived.
- Use an alternative cipher suite with better performance.

  The performance of a cipher suite is different with software and hardware. Just because a cipher suite performs better in software does not mean a cipher suite will perform better with hardware. Some algorithms are typically inefficient in hardware (for example, DES and 3DES), however, specialized hardware can provide efficient implementations of these same algorithms.

  The performance of bulk encryption and decryption is affected by the cipher suite used for an individual SSL connection. The following chart displays the performance of each cipher suite. The test software calculating the data was Java Secure Socket Extension (JSSE) for both the client and server software, which used no crypto hardware support. The test did not include the time to establish a connection, but only the time to transmit data through an established connection. Therefore, the data reveals the relative SSL performance of various cipher suites for long running connections.

  Before establishing a connection, the client enables a single cipher suite for each test case. After the connection is established, the client times how long it takes to write an integer to the server and for the server to write the specified number of bytes back to the client. Varying the amount of data had

negligible effects on the relative performance of the cipher suites.




An analysis of the above data reveals the following:

- Bulk encryption performance is only affected by what follows the WITH in the cipher suite name. This is expected since the portion before the WITH identifies the algorithm used only during the SSL handshake.
- MD5 and SHA are the two hash algorithms used to provide data integrity. MD5 is 25% faster than SHA, however, SHA is more secure than MD5.
- DES and RC2 are slower than RC4.Triple DES is the most secure, but the performance cost is high when using only software.
- The cipher suite providing the best performance while still providing privacy is SSL_RSA_WITH_RC4_128_MD5. Even though SSL_RSA_EXPORT_WITH_RC4_40_MD5 is cryptographically weaker than RSA_WITH_RC4_128_MD5, the performance for bulk encryption is the same. Therefore, as long as the SSL connection is a long-running connection, the difference in the performance of high and medium security levels is negligible. It is recommended that a security level of high be used, instead of medium, for all components participating in communication only among WebSphere Application Server products. Make sure that the connections are long running connections.

## Tuning security

Enabling security decreases performance. The following tuning parameters give you considerations for increasing performance.

- Disable security on any application servers that does not need security. There are two ways to disable security. Use either of the following two procedures and then restart the application server:

  **Using the administrative console**

  1. Click **Servers > Application servers >** *server_name*.
  2. Under Security, click **Server Security.**. Under Additional properties, click **Server-level security**.
  3. Disable the **Enable global security** option.

  **Using the command line**

  1. Type `wsadmin.sh -conntype NONE`
  2. When the system command prompt redisplays, type `securityoff`

- Fine-tune the **Cache timeout** value on the Global security panel as described in the “Global security settings” on page 145 article. Click **Security > Global security** to access the Global security panel.
- Configure the security cache properties as described in the “Security cache properties” on page 951 article.
- Enable the **SSL session tracking mechanism** option as described in Session management settings.
- Improve the performance of Web services security by downloading a Java Cryptography Extension (JCE) unlimited jurisdiction policy file that does not have restrictions on cryptography strength as described in the “Tuning Web services security” on page 936 article.
- Read the “Secure Sockets Layer performance tips” on page 952 and Chapter 15, “Tuning security configurations,” on page 949 articles for more information.

# Chapter 16. Troubleshooting security configurations

Refer to Security components troubleshooting tips for instructions on how to troubleshoot errors related to security.

The following topics explain how to troubleshoot specific problems related to configuring and enabling security configurations:
- Errors when configuring or enabling security
- Errors or access problems after enabling security
- Errors trying to enable or configure Secure Socket Layer (SSL) encrypted access
- Errors after enabling Secure Sockets Layer (SSL) or SSL-related error messages

## Errors when trying to configure or enable security

What kind of error are you seeing?
- “"LTPA password not set. validation failed" message displayed as error in the Administrative Console after saving global security settings ”
- “"Validation failed for user userid. Please try again..." displayed in the Administrative Console after saving global security settings ”
- “The setupClient.bat or setupClient.sh file is not working correctly” on page 956
- “Java HotSpot Server VM warning: Unexpected Signal 11 occurred under user-defined signal handler 0x7895710a message occurs in the native_stdout.log file when enabling security on the HP-UX11i platform” on page 956
- “WebSphere Application Server Version 6 is not working correctly with Enterprise Workload Manager (EWLM)” on page 956
- If you have successfully configured security (made changes, saved the configuration, and enabled security with no errors), but are now having problems accessing Web resources or the administrative console, refer to Errors or access problems after enabling security.

For general tips on diagnosing and resolving security-related problems, see the topic Troubleshooting the security component.

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

### "LTPA password not set. validation failed" message displayed as error in the Administrative Console after saving global security settings

This error can be caused if, when configuring WebSphere Application Server security, ″LTPA″ is selected as the authentication mechanism, and the LTPA password field is not set. To resolve this problem:
- Select Security **Authentication Mechanism > LTPA** in the console left-hand navigation pane.
- Complete the password and confirm password fields.
- Click **OK**.
- Try setting Global Security again.

### "Validation failed for user userid. Please try again..." displayed in the Administrative Console after saving global security settings

This typically indicates that a setting in the User Registry configuration is not valid:
- If the user registry is LocalOS, it is likely that either the server user ID and password are invalid or the server user ID does not have ″Act As Part of the Operating System″ (for NT) or root authority (for UNIX). The server user ID needs this authority for authentication using the LocalOS user registry.
- If the user registry is Lightweight Directory Access Protocol (LDAP):
  - Any of the settings that enable WebSphere Application Server to communicate with LDAP might be invalid, such as the LDAP server’s user ID, password, host, port, or LDAP filter. When you select

© Copyright IBM Corp. 2004

**Apply** or **OK** on the Global Security panel, a validation routine connects to the registry just as it would during runtime when security is enabled. This is done in order to verify any configuration problems immediately, instead of waiting until the server restarts.

  – Verify whether your LDAP server requires the Bind Distinguished Name (DN) to find the user in the LDAP directory. If the bind distinguished name is required, you must specify a DN instead of a short name. You can specify the bind distinguished name by clicking **Security > User Registries > LDAP** in the administrative console. For example, you might add `cn=root`.
  – Sometimes the LDAP server might be down during configuration. The best way to check this is to issue a command line search using a utility such as ldapsearch to search for the server ID. This way you can determine if the server is running and if the server ID is a valid entry in the LDAP. The ldapsearch utility is installed during an LDAP or Lotus Notes installation.
- If the user registry is Custom, double check that your implementation is in the classpath. Also, check to see if your implementation is authenticating properly.
- Regardless of registry type, check the User Registries configuration panels to see if you can find a configuration error:
  – Go back to the User Registries configuration panels and retype the password for the server ID.
- See if there is an obvious configuration error. Double check the attributes specified.

**The setupClient.bat or setupClient.sh file is not working correctly**

The `setupClient.bat` file on Windows platforms and the `setupClient.sh` file on UNIX platforms incorrectly specify the location of the SOAP security properties file.

In the `setupClient.bat` file, the correct location should be:

```
set CLIENTSOAP=-Dcom.ibm.SOAP.ConfigURL=file:%WAS_HOME%/properties/soap.client.props
```

In the `setupClient.sh` file, the CLIENTSOAP variable should be:

```
CLIENTSOAP=-Dcom.ibm.SOAP.ConfigURL=file:$WAS_HOME/properties/soap.client.props
```

In the `setupClient.bat` and `setupClient.sh` files, complete the following steps:
1. Remove the leading / after file:.
2. Change `sas` to `soap`.

**Java HotSpot Server VM warning: Unexpected Signal 11 occurred under user-defined signal handler 0x7895710a message occurs in the `native_stdout.log` file when enabling security on the HP-UX11i platform**

After you enable security on HP-UX 11i platforms, the following error in the `native_stdout.log` file occurs, along with a core dump and WebSphere Application Server does not start:

```
Java HotSpot(TM) Server VM warning:
Unexpected Signal 11 occurred under user-defined signal handler 0x7895710a
```

To work around this error, apply the fixes recommended by HP for Java at the following URL: http://www.hp.com/products1/unix/java/infolibrary/patches.html.

**WebSphere Application Server Version 6 is not working correctly with Enterprise Workload Manager (EWLM)**

To use WebSphere Application Server Version 6 with Enterprise Workload Manager (EWLM), you must manually update the WebSphere Application Serve server.policy files. For example:

```
grant codeBase "file:/<EWLM_Install_Home>/classes/ARM/arm4.jar" {
 permission java.security.AllPermission;
};
```

Otherwise, you might encounter a Java 2 security exception for violating the Java 2 security permission.

Refer to "Configuring server.policy files" on page 491 for more information on configuring server.policy files.

For current information available from IBM Support on known problems and their resolution, see the IBM Support page.

IBM Support has documents that can save you time gathering information needed to resolve this problem. Before opening a PMR, see the IBM Support page.

## Access problems after enabling security

What kind of error are you seeing?
- I cannot access all or part of the administrative console or use the wsadmin tool after enabling security
- I cannot access a Web page after enabling security
- Authentication error accessing a Web page
- Authorization error accessing a Web page
- The client cannot access an enterprise bean after enabling security
- The client never gets prompted when accessing a secured enterprise bean
- I cannot stop an application server, node manager, or node after enabling security
- AccessControlException is reported in SystemOut.log
- After enabling single signon, I cannot log on to the administrative console

For general tips on diagnosing and resolving security-related problems, see the topic Troubleshooting the security component.

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, see Obtaining help from IBM.

**I cannot access all or part of the administrative console or use the wsadmin tool after enabling security**
- If you cannot access the administrative console, or view and update certain objects, look in the SystemOut log of the application server which hosts the administrative console page for a related error message.
- You might not have authorized your ID for administrative tasks. This problem is indicated by errors such as:
  - `[8/2/02 10:36:49:722 CDT] 4365c0d9 RoleBasedAuth A CWSCJ0305A: Role based authorization check failed for security name MyServer/myUserId, accessId MyServer/S-1-5-21-882015564-4266526380-2569651501-1005 while invoking method getProcessType on resource Server and module Server.`
  - Exception message: `"CWWMN0022E: Access denied for the getProcessType operation on Server MBean"`
  - `When running the command: wsadmin -username j2ee -password j2ee: CWWAX7246E: Cannot establish "SOAP" connection to host "BIRKT20" because of an authentication failure. Please ensure that user and password are correct on the command line or in a properties file.`

  To grant an ID administrative authority, from the administrative console, click **System Administration > Console Users** and validate that the ID is a member. If it is not, add the ID with at least monitor access privileges, for read-only access.
- Check that the *enable_trusted_application* flag is set to true. To check the *enable_trusted_application* flag, from the Administrative Console, click **Security > Global Security > Custom Properties > Enable Trusted Application** and verify that it is set to true.

**I cannot access a Web page after enabling security**

When secured resources are not accessible, probable causes include:

- Authentication errors - WebSphere Application Server security cannot identify the ID of the person or process. Symptoms of authentication errors include:

  On a Netscape browser:
  - `Authorization failed. Retry?` message displays after an attempt to log in.
  - Accepts any number of attempts to retry login and displays `Error 401` message when Cancel is clicked to stop retry.
  - A typical browser message displays: `Error 401: Basic realm='Default Realm'`.

  On an Internet Explorer browser:
  - Login prompt displays again after an attempt to log in.
  - Allows three attempts to retry login.
  - Displays `Error 401` message after three unsuccessful retries.
- Authorization errors - The security function has identified the requesting person or process as not authorized to access the secured resource. Symptoms of authorization errors include:
  - Netscape browser: ″Error 403: AuthorizationFailed″ message is displayed.
  - Internet Explorer:
    - ″You are not authorized to view this page″ message is displayed.
    - ″HTTP 403 Forbidden″ error is also displayed.
- SSL errors - WebSphere Application Server security uses Secure Socket Layer (SSL) technology internally to secure and encrypt its own communication, and incorrect configuration of the internal SSL settings can cause problems. Also you might have enabled SSL encryption for your own Web application or enterprise bean client traffic which, if configured incorrectly, can cause problems regardless of whether WebSphere Application Server security is enabled.
  - SSL related problems are often indicated by error messages which contain a statement such as: `ERROR: Could not get the initial context or unable to look up the starting context.Exiting.` followed by `javax.net.ssl.SSLHandshakeException`

**The client cannot access an enterprise bean after enabling security**

If client access to an enterprise bean fails after security is enabled:
- Review the steps for securing and granting access to resources.
- Browse the server JVM logs for errors relating to enterprise bean access and security. Look up any errors in the message table.

  Errors similar to `Authorization failed for /UNAUTHENTICATED while invoking` *`resource`* `securityName:/UNAUTHENTICATED;accessId:UNAUTHENTICATED not granted any of the required roles` *`roles`* indicate that:
  - An unprotected servlet or JavaSever Page (JSP) file accessed a protected enterprise bean. When an unprotected servlet is accessed, the user is not prompted to log in and the servlet runs as UNAUTHENTICATED. When the servlet makes a call to an enterprise bean that is protected the servlet fails.

    To resolve this problem, secure the servlet that is accessing the protected enterprise bean. Make sure the *runAs* property for the servlet is set to an ID that can access the enterprise bean.
  - An unauthenticated Java client program is accessing an enterprise bean resource that is protected. This situation can happen if the file read by the `sas.client.props` properties file used by the client program does not have the `securityEnabled` flag set to **true**.

    To resolve this problem, make sure that the `sas.client.props` file on the client side has its securityEnabled flag set to **true**.

  Errors similar to **Authorization failed for *valid_user* while invoking *resource* securityName:/username;accessId:xxxxxx not granted any of the required roles *roles*** indicate that a client attempted to access a secured enterprise bean resource, and the supplied user ID is not assigned the required roles for that enterprise bean.
  - Check that the required roles for the enterprise bean resource are accessed. View the required roles for the enterprise bean resource in the deployment descriptor of the Web resource.

– Check the authorization table and make sure that the user or the group that the user belongs to is assigned one of the required roles. You can view the authorization table for the application that contains the enterprise bean resource using the administrative console.

If `org.omg.CORBA.NO_PERMISSION` exceptions occur when programmatically logging on to access a secured enterprise bean, an authentication exception has occurred on the server. Typically the CORBA exception is triggered by an underlying `com.ibm.WebSphereSecurity.AuthenticationFailedException`. To determine the actual cause of the authentication exception, examine the full trace stack:

1. Begin by viewing the text following `WSSecurityContext.acceptSecContext(), reason:` in the exception. Typically, this text describes the failure without further analysis.
2. If this action does not describe the problem, look up the CORBA minor code. The codes are listed in the article titled Troubleshooting the security components reference.

   For example, the following exception indicates a CORBA minor code of 49424300. The explanation of this error in the CORBA minor code table reads:

   ```
   authentication failed error.
   ```

   In this case the user ID or password supplied by the client program is probably invalid:

   ```
   org.omg.CORBA.NO_PERMISSION: Caught WSSecurityContextException in WSSecurityContext.acceptSecContext(), reason: Major
   ```

A `CORBA INITIALIZE` exception with `CWWSA1477W: SECURITY CLIENT/SERVER CONFIGURATION MISMATCH` error embedded, is received by client program from the server.

This error indicates that the security configuration for the server differs from the client in some fundamental way. The full exception message lists the specific mismatches. For example, the following exception lists three errors:

```
Exception received: org.omg.CORBA.INITIALIZE:
 CWWSA1477W: SECURITY CLIENT/SERVER CONFIG MISMATCH:
The client security configuration (sas.client.props or outbound settings in GUI) does not support the server security con
ERROR 1: CWWSA0607E: The client requires SSL Confidentiality but the server does not support it.
ERROR 2: CWWSA0610E: The server requires SSL Integrity but the client does not support it.
ERROR 3: CWWSA0612E: The client requires client (e.g., userid/password or token), but the server does not support it.
     minor code: 0
     completed: No
at com.ibm.ISecurityLocalObjectBaseL13Impl.SecurityConnectionInterceptor.getConnectionKey(SecurityConnectionInterceptor.j
```

In general, resolving the problem requires a change to the security configuration of either the client or the server. To determine which configuration setting is involved, look at the text following the `CWWSA` error message. For more detailed explanations and instructions, look in the message reference, by selecting the **Reference** view of the information center navigation and expanding **Messages** in the navigation tree.

In these particular cases:

- In ERROR 1, the client is requiring SSL confidentiality but the server does not support SSL confidentiality. Resolve this mismatch in one of two ways. Either update the server to support SSL confidentiality or update the client so that it no longer requires it.
- In ERROR 2, the server requires SSL integrity but the client does not support SSL integrity. Resolve this mismatch in one of two ways. Either update the server to support SSL integrity or update the client so that it no longer requires it.
- In ERROR 3, the client requires client authentication through a user id and password, but the server does not support this type of client authentication. Either the client or the server needs to change the configuration. To change the client configuration, modify the `SAS.CLIENT.PROPS` file for a pure client or change the outbound configuration for the server in the Security GUI. To change the configuration for the target server, modify the inbound configuration in the Security GUI.

Similarly, an exception like `org.omg.CORBA.INITIALIZE: JSAS0477W: SECURITY CLIENT/SERVER CONFIG MISMATCH:` appearing on the server trying to service a client request indicates a security configuration mismatch between client and server. The steps for resolving the problem are the same as for the JSAS1477W exceptions previously described.

### Client program never gets prompted when accessing secured enterprise bean

Even though it appears security is enabled and an enterprise bean is secured, it can happen that the client executes the remote method without prompting. If the remote method is protected, an authorization failure results. Otherwise, execute the method as an unauthenticated user.

Possible reasons for this problem include:
- The server with which you are communicating might not have security enabled. Check with the WebSphere Application Server administrator to ensure that the server security is enabled. Access the global security settings from within the Security section of the administrative console.
- The client does not have security enabled in the sas.client.props file. Edit the sas.client.props file to ensure the property **com.ibm.CORBA.securityEnabled** is set to true.
- The client does not have a ConfigURL specified. Verify that the property **com.ibm.CORBA.ConfigURL** is specified on the command line of the Java client, using the -D parameter.
- The specified ConfigURL has an invalid URL syntax, or the `sas.client.props` that is pointed to cannot be found. Verify that the **com.ibm.CORBA.ConfigURL** property is valid, for example, similar to the C:/WebSphere/AppServer/properties/sas.client.props file on Windows systems. Check the Java documentation for a description of URL formatting rules. Also, validate that the file exists at the specified path.
- The client configuration does not support message layer client authentication (user ID and password). Verify that the `sas.client.props` file has one of the following properties set to true:
  - com.ibm.CSI.performClientAuthenticationSupported=true
  - com.ibm.CSI.performClientAuthenticationRequired=true
- The server configuration does not support message layer client authentication (user ID and password). Check with the WebSphere Application Server administrator to verify that user ID and password authentication is specified for the inbound configuration of the server within the System Administration section of the administrative console administration tool.

### Cannot stop an application server, node manager, or node after enabling security

If you use command line utilities to stop WebSphere Application Server processes, apply additional parameters after enabling security to provide authentication and authorization information.

Use the ./stopServer.sh -help command to display the parameters to use.

Use the following command options after enabling security:
- `./stopServer.sh hostname -username` *name* `-password` *password*
- `./stopNode.sh -username` *name* `-password` *password*
- `./stopManager.sh -username` *name* `-password` *password*

### After enabling single signon, I cannot log on to the administrative console

This problem occurs when single signon (SSO) is enabled, and you attempt to access the administrative console using the short name of the server, for example `http://myserver:9060/ibm/console`. The server accepts your user ID and password, but returns you to the log on page instead of the administrative console.

To correct this problem, use the fully qualified host name of the server, for example `http://myserver.mynetwork.mycompany.com:9060/ibm/console`.

## Errors after enabling security

What kind of error are you seeing?
- Authentication error accessing a Web page
- Authorization error accessing a Web page
- Error Message: `CWSCJ0314E: Current Java 2 security policy reported a potential violation`

- `CWMSG0508E: The JMS Server security service was unable to authenticate user ID:` error displayed in SystemOut.log when starting an application server
- Error Message: `CWSCJ0237E: One or more vital LTPAServerObject configuration attributes are null or not available` after enabling security and starting the application server
- An AccessControlException is reported in the SystemOut.log
- Error Message: `CWSCJ0336E: Authentication failed for user {0} because of the following exception {1}`

For general tips on diagnosing and resolving security-related problems, see the topic Troubleshooting the security component.

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, see Obtaining help from IBM.

### Authentication error accessing a Web page

Possible causes for authentication errors include:

- **Incorrect user name or passwords.** Check the user name and password and make sure they are correct.
- **Security configuration error : User registry type is not set correctly.** Check the user registry property in global security settings in the administrative console. Verify that it is the intended user registry.
- **Internal program error.** If the client application is a Java standalone program, this program might not gather or send credential information correctly.

If the user registry configuration, user ID, and password appear correct, use the WebSphere Application Server trace to determine the cause of the problem. To enable security trace, use the com.ibm.ws.security.*=all=enabled trace specification.

### Authorization error accessing a Web page

If a user who should have access to a resource does not, there is probably a missing configuration step. Review the steps for securing and granting access to resources.

Specifically:
- Check required roles for the accessed Web resource.
- Check the authorization table to make sure that the user, or the groups to which the user belongs, is assigned to one of the required roles.
- View required roles for the Web resource in the deployment descriptor of the Web resource.
- View the authorization table for the application that contains the Web resource, using the administrative console.
- Test with a user who is granted the required roles, to see if the user can access the problem resources.
- If the problem user is required to have one or more of the required roles, use the administrative console to assign that user to required roles. Then stop and restart the application.

If the user is granted required roles, but still fails to access the secured resources, enable security trace, using com.ibm.ws.security.*=all=enabled as the trace specification. Collect trace information for further resolution.

### Error Message: `CWSCJ0314E: Current Java 2 security policy reported a potential violation on server`

If you find errors on your server similar to:

```
Error Message: CWSCJ0314E: Current Java 2 Security policy reported a potential violation of Java 2 Security Permission. I
{0}Permission\:{1}Code\:{2}{3}Stack Trace\:{4}Code Base Location\:{5}
```

The Java security manager `checkPermission()` method has reported an exception, `SecurityException`.

**The reported exception might be critical to the secure system.** Turn on security trace to determine the potential code that might have violated the security policy. Once the violating code is determined, verify if the attempted operation is permitted with respect to Java 2 Security, by examining all applicable Java 2 security policy files and the application code.

A more detailed report is enabled by either configuring RAS trace into debug mode, or specifying a Java property.

- Check the trace enabling section for instructions on how to configure RAS trace into debug mode, or
- Specify the following property in the **Application Servers >** ***server name*** **> ProcessDefinition > Java Virtual Machine** panel from the administrative console in the **Generic JVM arguments** panel:
  - Add the run-time flag **java.security.debug**
  - Valid values:

    **access**
    Print all debug information including: required permission, code, stack, and code base location.

    **stack** Print debug information including: required permission, code, and stack.

    **failure** Print debug information including: required permission and code.

For a review of Java security policies and what they mean , see the Java 2 Security documentation at `http://java.sun.com/j2se/1.3/docs/guide/security/index.html` .

**Tip:** If the application is running with a Java Mail API, this message might be benign. You can update the *installed Enterprise Application root*/META-INF/was.policy file to grant the following permissions to the application:

- `permission java.io.FilePermission "${user.home}${/}.mailcap", "read";`
- `permission java.io.FilePermission "${user.home}${/}.mime.types", "read";`
- `permission java.io.FilePermission "${java.home}${/}lib${/}mailcap", "read";`
- `permission java.io.FilePermission "${java.home}${/}lib${/}mime.types", "read";`

**Error message: `CWMSG0508E: The JMS Server security service was unable to authenticate user ID:" error displayed in SystemOut.log when starting an application server`**

This error can result from installing the JMS messaging API sample and then enabling security. You can follow the instructions in the Configure and Run page of the corresponding JMS sample documentation to configure the sample to work with WebSphere Application Server security.

You can verify the installation of the message-driven bean sample by launching the installation program, selecting **Custom**, and browsing the components which are already installed in the **Select the features you like to install** panel. The JMS sample is shown as **Message-Driven Bean Sample**, under **Embedded Messaging**.

You can also verify this installation by using the administrative console to open the properties of the application server which contains the samples. Select **MDBSamples** and click uninstall.

**Error message: `CWSCJ0237E: One or more vital LTPAServerObject configuration attributes are null or not available` after enabling security and starting the application server.**

This error message can result from selecting LTPA as the authentication mechanism, but not generating the LTPA keys. The LTPA keys encrypt the LTPA token.

To resolve this problem:

1. Click **System Administration > Console users > LTPA**
2. Enter a password, which can be anything.
3. Enter the same password in **Confirm Password**.

4. Click **Apply**.
5. Click **Generate Keys**.
6. Click on **Save**.

**The exception AccessControlException, is reported in the SystemOut.log**

The problem is related to the Java 2 Security feature of WebSphere Application Server, the API-level security framework that is implemented in WebSphere Application Server Version 5. An exception similar to the following example displays. The error message and number can vary.

```
E CWSRV0020E: [Servlet Error]-[validator]: Failed to load servlet:
java.security.AccessControlException: access denied
(java.io.FilePermission
C:\WebSphere\AppServer\installedApps\maeda\adminconsole.ear\adminconsole.war\
WEB-INF\validation.xml read)
```

For an explanation of Java 2 security, how and why to enable or disable it, how it relates to policy files, and how to edit policy files, see the Java 2 security topic in the information center navigation. The topic explains that Java 2 security is not only used by this product, but developers can also implement it for their business applications. Administrators might need to involve developers, if this exception is thrown when a client tries to access a resource hosted by WebSphere Application Server.

Possible causes of these errors include:
- Syntax errors in a policy file.
- Syntax errors in permission specifications in the ra.xml file bundled in a .rar file. This case applies to resource adapters that support connector access to CICS or other resources.
- An application is missing the specified permission in a policy file, or in permission specifications in an ra.xml file bundled in a .rar file
- The class path is not set correctly, preventing the permissions for the resource.xml file for SPI from being correctly created.
- A library called by an application, or the application, is missing a doPrivileged block to support access to a resource.
- Permission is specified in the wrong policy file.

To resolve these problems:
- Check all of the related policy files to verify that the permission shown in the exception, for example java.io.FilePermission, is specified.
- Look for a related ParserException in the SystemOut.log file which reports the details of the syntax error. For example:

  `CWSCJ0189E: Caught ParserException` while creating template for application policy `C:\WAS\config\cells\xxx\node`

  The exception is com.ibm.ws.security.util.ParserException: line 18: expected ';', found 'grant'
- Look for a message similar to: `CWSCJ0325W: The permission` *`permission`* `specified in the policy file is unresolved.`
- Check the call stack to determine which method does not have the permission. Identify the class path of this method. If it is hard to identify the method, enable the Java2 security Report.
  - Configuring RAS trace by specifying `com.ibm.ws.security.core.*=all=enabled`, or specifying a Java **property.java.security.debug** property. Valid values for the **java.security.debug** property are:

    **access**
    : Print all debug information including: required permission, code, stack, and code base location.

    **stack**
    : Print debug information including: required permission, code, and stack.

    **failure**
    : Print debug information including: required permission and code.
  - The report shows:

    **Permission**
    : the missing permission.

**Code** which method has the problem.

**Stack Trace**
where the access violation occurred.

**CodeBaseLocation**
the detail of each stack frame.

Usually, Permission and Code are enough to identify the problem. The following example illustrates a report:

```
Permission:
C:\WebSphere\AppServer\logs\server1\SystemOut_02.08.20_11.19.53.log :
access denied (java.io.FilePermission
C:\WebSphere\AppServer\logs\server1\SystemOut_02.08.20_11.19.53.log delete)

Code:
 com.ibm.ejs.ras.RasTestHelper$7  in
{file:/C:/WebSphere/AppServer/installedApps/maeda/JrasFVTApp.ear/RasLib.jar
}

Stack Trace:

java.security.AccessControlException: access denied (java.io.FilePermission
C:\WebSphere\AppServer\logs\server1\SystemOut_02.08.20_11.19.53.log delete
)
        at java.security.AccessControlContext.checkPermission
                              (AccessControlContext.java(Compiled Code))
        at java.security.AccessController.checkPermission
                              (AccessController.java(Compiled Code))
        at java.lang.SecurityManager.checkPermission
                              (SecurityManager.java(Compiled Code))
                               .
Code Base Location:

com.ibm.ws.security.core.SecurityManager :
file:/C:/WebSphere/AppServer/lib/securityimpl.jar

  ClassLoader: com.ibm.ws.bootstrap.ExtClassLoader
  Permissions granted to CodeSource
(file:/C:/WebSphere/AppServer/lib/securityimpl.jar <no certificates>

  {
    (java.util.PropertyPermission java.vendor read);
    (java.util.PropertyPermission java.specification.version read);
    (java.util.PropertyPermission line.separator read);
    (java.util.PropertyPermission java.class.version read);
    (java.util.PropertyPermission java.specification.name read);
    (java.util.PropertyPermission java.vendor.url read);
    (java.util.PropertyPermission java.vm.version read);
    (java.util.PropertyPermission os.name read);
    (java.util.PropertyPermission os.arch read);
   }
   ( This list continues.)
```

- If the method is SPI, check the resources.xml file to ensure that the class path is correct.
- To confirm that all of the policy files are loaded correctly, or what permission each class path is granted, enable the trace with **com.ibm.ws.security.policy.*=all=enabled**. All loaded permissions are listed in the trace.log file. Search for the app.policy, was.policy and ra.xml files. To check the permission list for a class path, search for `Effective Policy for` *classpath*.
- If there are any syntax errors in the policy file or ra.xml file, correct them with the policytool. Avoid editing the policy manually, because syntax errors can result.
- If a permission is listed as Unresolved, it does not take effect. Verify that the specified permission name is correct.
- If the class path specified in the resource.xml file is not correct, correct it.
- If a required permission does not exist in either the policy files or the ra.xml file, examine the application code to see if you need to add this permission. If so, add it to the proper policy file or ra.xml file.

- If the permission should not be granted outside of the specific method that is accessing this resource, modify the code needs to use a doPrivileged block.
- If this permission does exist in a policy file or a ra.xml file and they were loaded correctly, but the class path still does not have the permission in its list, the location of the permission might not be correct. Read Java 2 Security in the information center navigation carefully to determine in which policy file or ra.xml file that permission should be specified.

**Tip:** If the application is running with the Java Mail API, you can update the `*installed Enterprise Application root*/META-INF/was.policy` file to grant the following permissions to the application:
- `permission java.io.FilePermission "${user.home}${/}.mailcap", "read";`
- `permission java.io.FilePermission "${user.home}${/}.mime.types", "read";`
- `permission java.io.FilePermission "${java.home}${/}lib${/}mailcap", "read";`
- `permission java.io.FilePermission "${java.home}${/}lib${/}mime.types", "read";`

**Error Message: `CWSCJ0336E: Authentication failed for user {0} because of the following exception {1}`**

This error message results if the user ID indicated is not found in the LDAP user registry. To resolve this problem:
1. Verify that your user ID and password are correct.
2. Verify that the user ID exists in the registry.
3. Verify that the base distinguished name (DN) is correct.
4. Verify that the user filter is correct.
5. Verify that the bind DN and the password for the bind DN are correct. If the bind DN and password are not specified, add the missing information and retry.
6. Verify that the host name and LDAP type are correct.

Consult with the administrator of the user registry if the problem persists.

## Errors trying to enable or configure Secure Socket Layer (SLL) encrypted access

What kind of error are you seeing?
- "The Java Cryptographic Extension (JCE) files were not found." error when launching iKeyman.

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

**"The Java Cryptographic Extension (JCE) files were not found." error when launching iKeyman.**

You may receive the following error when you attempt to start the iKeyman tool: "The Java Cryptographic Extension (JCE) files were not found. Please check that the JCE files have been installed in the correct directory". When you click OK, the iKeyman tool closes. To resolve this problem:
- Set the JAVA_HOME parameter so that is points to the Java Developer Kit that is shipped with WebSphere Application Server.
  - For example, on a Unix platform the command would be similar to: `export JAVA_HOME=/opt/WebSphere/AppServer/java`
  - On a Windows platform, if WebSphere Application Server is installed on your c: drive, the command would be: `set JAVA_HOME=c:\WebSphere\AppServer\java`
- Rename the file install_dir/java/jre/lib/ext/gskikm.jar to gskikm.jar.org.

## Errors after configuring or enabling Secure Sockets Layer

This article explains various problems you might encounter after configuring or enabling Secure Sockets Layer (SSL).

### Stopping the deployment manager after configuring Secure Sockets Layer

After configuring the Secure Sockets Layer repertoires, if you stop the deployment manager without also stopping the node agents, you might receive the following error message when you restart the deployment manager:

```
CWWMU0509I: The server "nodeagent" cannot be reached. It appears to be stopped.
CWWMU0211I: Error details may be seen in the file:
          /opt/WebSphere/AppServer/logs/nodeagent/stopServer.log
```

The error occurs because the deployment manager did not propagate the new SSL certificate to the node agents. Thus, the node agents are using an older certificate files than the deployment manager and the certificate files are incompatible. To work around this problem, you must manually stop the node agent and deployment manager processes. To end the processes on Windows platforms, use the Task Manager. On UNIX platforms, run the command to end the process.

There are some things you need to consider when identifying the specific process that should be killed. For each process being killed, WebSphere Application Server stores the process ID in a `pid` file and you need to find these `*.pid` files. For example, the `server1.pid` for a standalone install might be found at: `<WAS_root>/AppServer/logs/server1.pid`

### Accessing resources using HTTPS

If you are unable to access resources using a Secure Sockets Layer (SSL) URL (beginning with https:), or encounter error messages which indicate SSL problems, verify that your HTTP server is configured correctly for SSL by browsing the welcome page of the HTTP server using SSL by entering the URL: **https://***hostname*.

If the page works with HTTP, but not HTTPS, the problem is with the HTTP server.
- Refer to the documentation for your HTTP server for instructions on correctly enabling SSL. If you are using the IBM HTTP Server or Apache, go to:http://www.ibm.com/software/webservers/httpservers/library.html. Click **Frequently Asked Questions**> **SSL**.
- If you are use the IBM Key Management (IKeyman) tool to create certificates and keys, remember to stash the password to a file when creating the KDB file with the IBM Key Management Tool.
  1. Go to the directory where the KDB file was created, and see if there is a .sth file.
  2. If not, open the KDB file with the IBM Key Management Tool, and click **Key Database File > Stash Password**. The following message displays: `The password has been encrypted and saved in the file`.

If the HTTP server handles SSL-encrypted requests successfully, or is not involved (for example, traffic flows from a Java client application directly to an enterprise bean hosted by the WebSphere Application Server, or the problem appears only after enabling WebSphere Application Server security), what kind of error are you seeing?
- `javax.net.ssl.SSLHandshakeException - The client and server could not negotiate the desired level of security. Reason: handshake failure`
- `javax.net.ssl.SSLHandshakeException - The client and server could not negotiate the desired level of security. Reason: unknown certificate`
- `javax.net.ssl.SSLHandshakeException - The client and server could not negotiate the desired level of security. Reason: bad certificate`
- `org.omg.CORBA.INTERNAL: EntryNotFoundException or NTRegistryImp E CWSCJ0070E: No privilege id configured for:` error when programmatically creating a credential.

For general tips on diagnosing and resolving security-related problems, see “Security components troubleshooting tips” on page 968

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, see Obtaining help from IBM

**javax.net.ssl.SSLHandshakeException - The client and server could not negotiate the desired level of security. Reason: handshake failure**

If you see a Java exception stack similar to the following example:

```
[Root exception   is org.omg.CORBA.TRANSIENT:  CAUGHT_EXCEPTION_WHILE_CONFIGURING_SSL_CLIENT_SOCKET: CWWJE0080E:   javax
```

Some possible causes are:
- Not having common ciphers between the client and server.
- Not specifying the correct protocol.

To correct these problems:
1. Review the SSL settings. Click **WebSphere Administrative Console Security Settings > SSL Configuration Repertoires > DefaultSSLSettings** (or other named SSL settings).
2. Select the **Secure Sockets Layer (SSL)** option from the Additional Properties menu. You can also browse the file manually by viewing the *install_dir*/properties/sas.client.props file.
3. Check the property specified by the com.ibm.ssl.protocol file to determine which protocol is specified.
4. Check the cipher types specified by the com.ibm.ssl.enabledCipherSuites. You might want to add more cipher types to the list. To see which cipher suites are currently enabled: Fo to the properties page of the SSL settings as described above, and look for the **Cipher Suites** property. To see the list of all possible cipher suites, go to the properties page of the SSL settings as described above, then view the online help for that page. From the help page, click **Configure additional SSL settings**.
5. Correct the protocol or cipher problem by using a different client or server protocol and cipher selection. Typical protocols are SSL or SSLv3.
6. Make the cipher selection 40-bit instead of 128-bit. For CSIv2, set both of the following properties to false in the sas.client.props file, or set security level=medium in the administrative console settings:
   - com.ibm.CSI.performMessageConfidentialityRequired=false
   - com.ibm.CSI.performMessageConfidentialitySupported=false

**javax.net.ssl.SSLHandshakeException: unknown certificate**

If you see a Java exception stack similar to the following example, it might be caused by not having the personal certificate for the server in the client truststore file:

```
ERROR: Could not get the initial context or unable to look up the starting context. Exiting.  Exception received: javax.n
```

To correct this problem:
1. Check the client truststore file to determine if the signer certificate from the server personal certificate is there. For a self-signed server personal certificate, the signer certificate is the public key of the personal certificate. For a certificate authority signed server personal certificate, the signer certificate is the root CA certificate of the CA that signed the personal certificate.
2. Add the server signer certificate to the client truststore file.

**javax.net.ssl.SSLHandshakeException: bad certificate**

If you see a Java exception stack similar to the following example, it can be caused by having a personal certificate in the client keystore used for SSL mutual authentication but not having extracted the signer certificate into the server truststore file so that the server can trust it whenever the SSL handshake is made:

```
ERROR: Could not get the initial context or unable to look up the starting context. Exiting.  Exception received: javax.n
```

To verify this problem, check the server truststore file to determine if the signer certificate from the client personal certificate is there. For a self-signed client personal certificate, the signer certificate is the public

key of the personal certificate. For a certificate authority signed client personal certificate, the signer certificate is the root CA certificate of the CA that signed the personal certificate.

To correct this problem, add the client signer certificate to the server truststore file.

**org.omg.CORBA.INTERNAL: EntryNotFoundException or NTRegistryImp E CWSCJ0070E: No privilege id configured for: error when programmatically creating a credential**

If you encounter the following exception in a client application attempting to request a credential from a WebSphere Application Server using SSL mutual authentication:

```
ERROR: Could not get the initial context or unable to look up the starting context. Exiting. Exception received: org.omg.CO
```

or a simultaneous error from the WebSphere Application Server that resembles:

```
[7/31/02 15:38:48:452 CDT] 27318f5 NTRegistryImp E CWSCJ0070E: No privilege id configured for: testuser
```

The cause might be that the user ID sent by the client to the server is not in the user registry for that server.

To confirm this problem, check that an entry exists for the personal certificate that is sent to the server. Depending on the user registry mechanism, look at the native operating system user ID or Lightweight Directory Access Protocol (LDAP) server entries.

To correct this problem, add the user ID to the user registry entry (for example, operating system, LDAP directory, or other custom registry) for the personal certificate identity.

## Security components troubleshooting tips

This document explains basic resources and steps for diagnosing security related issues in the WebSphere Application Server, including:

- What “Log files” on page 969 to look at and what to look for in them.
- What “SDSF output logs” on page 970 to look at and what to look for in them.
- “General approach for troubleshooting security-related issues” on page 971 to isolating and resolving security problems.
- When and how to “Trace security” on page 975.
- An overview and table of “CSIv2 CORBA Minor Codes” on page 976.

The following security-related problems are addressed elsewhere in the information center:

- Errors and access problems after enabling security

  After enabling global security, there was a degradation in performance. See Enabling global security for information about using the unrestricted policy files.
- Errors after enabling SSL, or SSL-related error messages
- Errors trying to configure and enable security

If none of these steps solves the problem, check to see if the problem has been identified and documented using the links in Diagnosing and fixing problems: Resources for learning.

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

**Note:** For an overview of WebSphere Application Server security components such as SAS or z/SAS and how they work, see Getting started with security.

### Log files

When troubleshooting the security component, browse the JVM logs for the server that hosts the resource you are trying to access. The following is a sample of messages you would expect to see from a server in which the security service has started successfully:

```
SASRas        A CWWSA0001I: Security configuration initialized.
SASRas        A CWWSA0002I: Authentication protocol: CSIV2/IBM
SASRas        A CWWSA0003I: Authentication mechanism: SWAM
SASRas        A CWWSA0004I: Principal name: MYHOSTNAME/aServerID
SASRas        A CWWSA0005I: SecurityCurrent registered.
SASRas        A CWWSA0006I: Security connection interceptor initialized.
SASRas        A CWWSA0007I: Client request interceptor registered.
SASRas        A CWWSA0008I: Server request interceptor registered.
SASRas        A CWWSA0009I: IOR interceptor registered.
NameServerImp I CWNMS0720I: Do Security service listener registration.
SecurityCompo A CWSCJ0242A: Security service is starting
UserRegistryI A CWSCJ0136I: Custom Registry:com.ibm.ws.security.registry.nt.
NTLocalDomainRegistryImpl has been initialized
SecurityCompo A CWSCJ0202A: Admin application initialized successfully
SecurityCompo A CWSCJ0203A: Naming application initialized successfully
SecurityCompo A CWSCJ0204A: Rolebased authorizer initialized successfully
SecurityCompo A CWSCJ0205A: Security Admin mBean registered successfully
SecurityCompo A CWSCJ0243A: Security service started successfully
SecurityCompo A CWSCJ0210A: Security enabled true
```

The following is an example of messages from a server which cannot start the security service, in this case because the administrative user ID and password given to communicate with the user registry is wrong, or the user registry itself is down or misconfigured:

```
SASRas        A CWWSA0001I: Security configuration initialized.
SASRas        A CWWSA0002I: Authentication protocol: CSIV2/IBM
SASRas        A CWWSA0003I: Authentication mechanism: SWAM
SASRas        A CWWSA0004I: Principal name: MYHOSTNAME/aServerID
SASRas        A CWWSA0005I: SecurityCurrent registered.
SASRas        A CWWSA0006I: Security connection interceptor initialized.
SASRas        A CWWSA0007I: Client request interceptor registered.
SASRas        A CWWSA0008I: Server request interceptor registered.
SASRas        A CWWSA0009I: IOR interceptor registered.
NameServerImp I CWNMS0720I: Do Security service listener registration.

SecurityCompo A CWSCJ0242A: Security service is starting
UserRegistryI A CWSCJ0136I: Custom Registry:com.ibm.ws.security.
registry.nt.NTLocalDomainRegistryImpl has been initialized
Authenticatio E CWSCJ4001E: Login failed for badID/<null>
javax.security.auth.login.LoginException: authentication failed: bad user/password
```

The following is an example of messages from a server for which LDAP has been specified as the security mechanism, but the LDAP keys have not been properly configured:

```
SASRas        A CWWSA0001I: Security configuration initialized.
SASRas        A CWWSA0002I: Authentication protocol: CSIV2/IBM
SASRas        A CWWSA0003I: Authentication mechanism: LTPA
SASRas        A CWWSA0004I: Principal name: MYHOSTNAME/anID
SASRas        A CWWSA0005I: SecurityCurrent registered.
SASRas        A CWWSA0006I: Security connection interceptor initialized.
SASRas        A CWWSA0007I: Client request interceptor registered.
SASRas        A CWWSA0008I: Server request interceptor registered.
SASRas        A CWWSA0009I: IOR interceptor registered.
NameServerImp I CWNMS0720I: Do Security service listener registration.
SecurityCompo A CWSCJ0242A: Security service is starting
UserRegistryI A CWSCJ0136I: Custom Registry:com.ibm.ws.security.registry.nt.
NTLocalDomainRegistryImpl has been initialized
SecurityServe E CWSCJ0237E: One or more vital LTPAServerObject configuration
attributes are null or not available. The attributes and values are password :
LTPA password does exist, expiration time 30, private key <null>, public key <null>,
and shared key <null>.
```

A problem with the SSL configuration might lead to the following message. You should ensure that the keystore location and keystore passwords are valid. Also, ensure the keystore has a valid personal certificate and that the personal certificate public key or CA root has been extracted on put into the truststore.

```
SASRas        A CWWSA0001I: Security configuration initialized.
SASRas        A CWWSA0002I: Authentication protocol: CSIV2/IBM
SASRas        A CWWSA0003I: Authentication mechanism: SWAM
SASRas        A CWWSA0004I: Principal name: MYHOSTNAME/aServerId
SASRas        A CWWSA0005I: SecurityCurrent registered.
SASRas        A CWWSA0006I: Security connection interceptor initialized.
SASRas        A CWWSA0007I: Client request interceptor registered.
SASRas        A CWWSA0008I: Server request interceptor registered.
SASRas        A CWWSA0009I: IOR interceptor registered.
SASRas        E CWWSA0026E: [SecurityTaggedComponentAssistorImpl.register]
Exception connecting object to the ORB.  Check the SSL configuration to ensure
 that the SSL keyStore and trustStore properties are set properly.  If the problem
persists, contact support for assistance. org.omg.CORBA.OBJ_ADAPTER:
ORB_CONNECT_ERROR (5) - couldn't get Server Subcontract  minor code:
4942FB8F  completed: No
```

### SDSF output logs

When troubleshooting the security component, browse the SDSF logs for the server that hosts the resource you are trying to access. The following is a sample of messages you would expect to see from a server in which the security service has started successfully:

```
 BBOM0001I com_ibm_Server_Security_Enabled: 1.
 BBOM0001I com_ibm_CSI_claimTLClientAuthenticationSupported: 1.
 BBOM0001I com_ibm_CSI_claimTLClientAuthenticationRequired: 0.
 BBOM0001I com_ibm_CSI_claimTransportAssocSSLTLSSupported: 1.
 BBOM0001I com_ibm_CSI_claimTransportAssocSSLTLSRequired: 0.
 BBOM0001I com_ibm_CSI_claimMessageConfidentialityRequired: 0.
 BBOM0001I com_ibm_CSI_claimClientAuthenticationSupported: 1.
 BBOM0001I com_ibm_CSI_claimClientAuthenticationRequired: 0.
 BBOM0001I com_ibm_CSI_claimClientAuthenticationtype:
 SAFUSERIDPASSWORD.
 BBOM0001I com_ibm_CSI_claimIdentityAssertionTypeSAF: 0.
 BBOM0001I com_ibm_CSI_claimIdentityAssertionTypeDN: 0.
 BBOM0001I com_ibm_CSI_claimIdentityAssertionTypeCert: 0.
 BBOM0001I com_ibm_CSI_claimMessageIntegritySupported: NOT SET,DEFAULT=1.
 BBOM0001I com_ibm_CSI_claimMessageIntegrityRequired: NOT SET,DEFAULT=1.
 BBOM0001I com_ibm_CSI_claimStateful: 1.
 BBOM0001I com_ibm_CSI_claimSecurityLevel: HIGH.
 BBOM0001I com_ibm_CSI_claimSecurityCipherSuiteList: NOT SET.
 BBOM0001I com_ibm_CSI_claimKeyringName: WASKeyring.
 BBOM0001I com_ibm_CSI_claim_ssl_sys_v2_timeout: NOT SET, DEFAULT=100.
 BBOM0001I com_ibm_CSI_claim_ssl_sys_v3_timeout: 600.
 BBOM0001I com_ibm_CSI_performTransportAssocSSLTLSSupported: 1.
 BBOM0001I security_sslClientCerts_allowed: 0.
 BBOM0001I security_kerberos_allowed: 0.
 BBOM0001I security_userid_password_allowed: 0.
 BBOM0001I security_userid_passticket_allowed: 1.
 BBOM0001I security_assertedID_IBM_accepted: 0.
 BBOM0001I security_assertedID_IBM_sent: 0.
 BBOM0001I nonauthenticated_clients_allowed: 1.
 BBOM0001I security_remote_identity: WSGUEST.
 BBOM0001I security_local_identity: WSGUEST.
 BBOM0001I security_EnableRunAsIdentity: 0.

 Messages beginning with BBOO0222I are common to Java WebSphere security.
They appear in both the controller and servant but are applicable to the servant.

 BBOO0222I CWSCJ0240I: Security service initialization completed successfully
 BBOO0222I CWSCJ0215I: Successfully set JAAS login provider
 configuration class to com.ibm.ws.security.auth.login.Configuration.
```

```
BBOO0222I CWSCJ0136I: Custom
Registry:com.ibm.ws.security.registry.zOS.SAFRegistryImpl has been  initialized
BBOO0222I CWSCJ0157I: Loaded Vendor AuthorizationTable:
com.ibm.ws.security.core.SAFAuthorizationTableImpl
BBOO0222I CWSCJ0243I: Security service started successfully
BBOO0222I CWSCJ0210I: Security enabled true
```

### General approach for troubleshooting security-related issues

When troubleshooting security-related problems, the following questions are very helpful and should be considered:

**Does the problem occur when security is disabled?**
: This is a good litmus test to determine that a problem is security related. However, just because a problem only occurs when security is enabled does not always make it a security problem. More troubleshooting is necessary to ensure the problem is really security-related.

**Did security appear to initialize properly?**
: A lot of security code is visited during initialization. So you will likely see problems there first if the problem is configuration related.

The following sequence of messages generated in the SystemOut.log indicate normal code initialization of an application server. This sequence will vary based on the configuration, but the messages are similar:

```
SASRas         A CWWSA0001I: Security configuration initialized.
SASRas         A CWWSA0002I: Authentication protocol: CSIV2/IBM
SASRas         A CWWSA0003I: Authentication mechanism: SWAM
SASRas         A CWWSA0004I: Principal name: BIRKT20/pbirk
SASRas         A CWWSA0005I: SecurityCurrent registered.
SASRas         A CWWSA0006I: Security connection interceptor initialized.
SASRas         A CWWSA0007I: Client request interceptor registered.
SASRas         A CWWSA0008I: Server request interceptor registered.
SASRas         A CWWSA0009I: IOR interceptor registered.
NameServerImp I CWNMS0720I: Do Security service listener registration.
SecurityCompo A CWSCJ0242A: Security service is starting
UserRegistryI A CWSCJ0136I: Custom Registry:com.ibm.ws.security.registry.nt.
NTLocalDomainRegistryImpl has been initialized
SecurityCompo A CWSCJ0202A: Admin application initialized successfully
SecurityCompo A CWSCJ0203A: Naming application initialized successfully
SecurityCompo A CWSCJ0204A: Rolebased authorizer initialized successfully
SecurityCompo A CWSCJ0205A: Security Admin mBean registered successfully
SecurityCompo A CWSCJ0243A: Security service started successfully

SecurityCompo A CWSCJ0210A: Security enabled true
```

The following sequence of messages generated in the SDSF active log indicate normal code initialization of an application server. Non-security messages have been removed from the sequence that follows. This sequence will vary based on the configuration, but the messages are similar:

```
Trace: 2003/08/25 13:06:31.034 01 t=9EA930 c=UNK key=P8 (13007002)
  FunctionName: com.ibm.ws.security.auth.login.Configuration
  SourceId: com.ibm.ws.security.auth.login.Configuration
  Category: AUDIT
  ExtendedMessage: CWSCJ0215I: Successfully set JAAS login provider
  configuration class to com.ibm.ws.security.auth.login.Configuration.
Trace: 2003/08/25 13:06:31.085 01 t=9EA930 c=UNK key=P8 (13007002)
  FunctionName: com.ibm.ws.security.core.SecurityDM
  SourceId: com.ibm.ws.security.core.SecurityDM
  Category: INFO
  ExtendedMessage: CWSCJ0231I: The Security component's
  FFDC Diagnostic Module com.ibm.ws.security.core.SecurityDM
  registered success
fully: true.
Trace: 2003/08/25 13:06:31.086 01 t=9EA930 c=UNK key=P8 (0000000A)
  Description: Log Boss/390 Error
```

```
   from filename: ./bborjtr.cpp
   at line: 812
   error message: BBOO0222I CWSCJ0231I: The Security component's
FFDC Diagnostic Module com.ibm.ws.security.core.SecurityDM registered
  successfully: true.
 Trace: 2003/08/25 13:06:32.426 01 t=9EA930 c=UNK key=P8 (13007002)
   FunctionName: com.ibm.ws.security.core.SecurityComponentImpl
   SourceId: com.ibm.ws.security.core.SecurityComponentImpl
   Category: INFO
   ExtendedMessage: CWSCJ0309I: Java 2 Security is disabled.
 Trace: 2003/08/25 13:06:32.427 01 t=9EA930 c=UNK key=P8 (0000000A)
   Description: Log Boss/390 Error
   from filename: ./bborjtr.cpp
   at line: 812
   error message: BBOO0222I CWSCJ0309I: Java 2 Security is disabled.
 Trace: 2003/08/25 13:06:32.445 01 t=9EA930 c=UNK key=P8 (13007002)
   FunctionName: com.ibm.ws.security.core.SecurityComponentImpl
   SourceId: com.ibm.ws.security.core.SecurityComponentImpl
   Category: INFO
   ExtendedMessage: CWSCJ0212I: WCCM JAAS configuration information
successfully pushed to login provider class.
 Trace: 2003/08/25 13:06:32.445 01 t=9EA930 c=UNK key=P8 (0000000A)
   Description: Log Boss/390 Error
   from filename: ./bborjtr.cpp
   at line: 812
   error message: BBOO0222I CWSCJ0212I: WCCM JAAS configuration
 information successfully pushed to login provider class.
 Trace: 2003/08/25 13:06:32.459 01 t=9EA930 c=UNK key=P8 (13007002)
   FunctionName: SecurityComponentImpl
   SourceId: SecurityComponentImpl
   Category: WARNING
   ExtendedMessage: BBOS1000W  LTPA or ICSF are configured as the
authentication mechanism but SSO is disabled.
 Trace: 2003/08/25 13:06:32.459 01 t=9EA930 c=UNK key=P8 (0000000A)
   Description: Log Boss/390 Error
   from filename: ./bborjtr.cpp
   at line: 824
   error message: BBOS1000W  LTPA or ICSF are configured as the
authentication mechanism but SSO is disabled.
 Trace: 2003/08/25 13:06:32.463 01 t=9EA930 c=UNK key=P8 (13007002)
   FunctionName: com.ibm.ws.security.core.SecurityComponentImpl
   SourceId: com.ibm.ws.security.core.SecurityComponentImpl
   Category: INFO
   ExtendedMessage: CWSCJ0240I: Security service initialization completed
successfully
 Trace: 2003/08/25 13:06:32.463 01 t=9EA930 c=UNK key=P8 (0000000A)
   Description: Log Boss/390 Error
   from filename: ./bborjtr.cpp
   at line: 812
   error message: BBOO0222I CWSCJ0240I: Security service initialization
completed successfully
 Trace: 2003/08/25 13:06:39.718 01 t=9EA930 c=UNK key=P8 (13007002)
   FunctionName: com.ibm.ws.security.registry.UserRegistryImpl
   SourceId: com.ibm.ws.security.registry.UserRegistryImpl
   Category: AUDIT
   ExtendedMessage: CWSCJ0136I: Custom Registry:
com.ibm.ws.security.registry.zOS.SAFRegistryImpl has been initialized
 Trace: 2003/08/25 13:06:41.967 01 t=9EA930 c=UNK key=P8 (13007002)
   FunctionName: com.ibm.ws.security.core.WSAccessManager
   SourceId: com.ibm.ws.security.core.WSAccessManager
   Category: AUDIT
   ExtendedMessage: CWSCJ0157I: Loaded Vendor AuthorizationTable:
com.ibm.ws.security.core.SAFAuthorizationTableImpl
 Trace: 2003/08/25 13:06:43.136 01 t=9EA930 c=UNK key=P8 (13007002)
   FunctionName: com.ibm.ws.security.role.RoleBasedAuthorizerImpl
   SourceId: com.ibm.ws.security.role.RoleBasedAuthorizerImpl
   Category: AUDIT
```

```
   ExtendedMessage: CWSCJ0157I: Loaded Vendor AuthorizationTable:
com.ibm.ws.security.core.SAFAuthorizationTableImpl
 Trace: 2003/08/25 13:06:43.789 01 t=9EA930 c=UNK key=P8 (13007002)
   FunctionName: com.ibm.ws.security.core.SecurityComponentImpl
   SourceId: com.ibm.ws.security.core.SecurityComponentImpl
   Category: INFO
   ExtendedMessage: CWSCJ0243I: Security service started successfully
 Trace: 2003/08/25 13:06:43.789 01 t=9EA930 c=UNK key=P8 (0000000A)
   Description: Log Boss/390 Error
   from filename: ./bborjtr.cpp
   at line: 812
   error message: BBOO0222I CWSCJ0243I: Security service started successfully
 Trace: 2003/08/25 13:06:43.794 01 t=9EA930 c=UNK key=P8 (13007002)
   FunctionName: com.ibm.ws.security.core.SecurityComponentImpl
   SourceId: com.ibm.ws.security.core.SecurityComponentImpl
   Category: INFO
   ExtendedMessage: CWSCJ0210I: Security enabled true
 Trace: 2003/08/25 13:06:43.794 01 t=9EA930 c=UNK key=P8 (0000000A)
   Description: Log Boss/390 Error
   from filename: ./bborjtr.cpp
   at line: 812
   error message: BBOO0222I CWSCJ0210I: Security enabled true
 Trace: 2003/08/25 13:07:06.474 01 t=9EA930 c=UNK key=P8 (13007002)
   FunctionName: com.ibm.ws.security.core.WSAccessManager
   SourceId: com.ibm.ws.security.core.WSAccessManager
   Category: AUDIT
   ExtendedMessage: CWSCJ0157I: Loaded Vendor AuthorizationTable:
com.ibm.ws.security.core.SAFAuthorizationTableImpl
 Trace: 2003/08/25 13:07:09.315 01 t=9EA930 c=UNK key=P8 (13007002)
   FunctionName: com.ibm.ws.security.core.WSAccessManager
   SourceId: com.ibm.ws.security.core.WSAccessManager
   Category: AUDIT
   ExtendedMessage: CWSCJ0157I: Loaded Vendor AuthorizationTable:
com.ibm.ws.security.core.SAFAuthorizationTableImpl
```

**Is there a stack trace or exception printed in the SystemOut.log?**

A single stack trace tells a lot about the problem. What code initiated the code that failed? What is the failing component? Which class did the failure actually come from? Sometimes the stack trace is all that is needed to solve the problem and it can pinpoint the root cause. Other times, it can only give us a clue, and could actually be misleading. When support analyzes a stack trace, they may request additional trace if it is not clear what the problem is. If it appears to be security-related and the solution cannot be determined from the stack trace or problem description, you will be asked to gather the following trace specification: `SASRas=all=enabled:com.ibm.ws.security.*=all=enabled` from all processes involved.

**Is this a distributed security problem or a local security problem?**

- If the problem is local, that is the code involved does not make a remote method invocation, then troubleshooting is isolated to a single process. It is important to know when a problem is local versus distributed since the behavior of the ORB, among other components, is different between the two. Once a remote method invocation takes place, an entirely different security code path is entered.
- When you know that the problem involves two or more servers, the techniques of troubleshooting change. You will need to trace all servers involved simultaneously so that the trace shows the client and server sides of the problem. Try to make sure the timestamps on all machines match as closely as possible so that you can find the request and reply pair from two different processes. Enable both SAS or z/SAS and Security trace using the trace specification: `SASRas=all=enabled:com.ibm.ws.security.*=all=enabled`.

  For more information on enabling trace, see Enabling trace.

  For more information on enabling trace, see Working with Trace

**Is the problem related to authentication or authorization?**

Most security problems fall under one of these two categories. Authentication is the process of determining who the caller is. Authorization is the process of validating that the caller has the proper authority to invoke the requested method. When authentication fails, typically this is related

to either the authentication protocol, authentication mechanism or user registry. When authorization fails, this is usually related to the application bindings from assembly and/or deployment and to the caller's identity who is accessing the method and the roles required by the method.

**Is this a Web or EJB request?**

Web requests have a completely different code path than EJB requests. Also, there are different security features for Web requests than for EJB requests, requiring a completely different body of knowledge to resolve. For example, when using the LTPA authentication mechanism, the single signon feature (SSO) is available for Web requests but not for EJB requests. Web requests involve HTTP header information not required by EJB requests due to the protocol differences. Also, the Web container (or servlet engine) is involved in the entire process. Any of these components could be involved in the problem and all should be considered during troubleshooting, based on the type of request and where the failure occurs.

Secure EJB requests heavily involve the ORB and Naming components since they flow over the RMI/IIOP protocol. In addition, when work flow management (WLM) is enabled, other behavior changes in the code can be observed. All of these components interact closely for security to work properly in this environment. At times, trace in any or all of these components might be necessary to troubleshoot problems in this area. The trace specification to begin with is `SASRas=all=enabled:com.ibm.ws.security.*=all=enabled`. ORB trace is also very beneficial when the SAS/Security trace does not seem to pinpoint the problem.

**Does the problem seem to be related to the Secure Sockets Layer (SSL)?**

The Secure Socket Layer (SSL) is a totally distinct separate layer of security. Troubleshooting SSL problems are usually separate from troubleshooting authentication and/or authorization problems. There are many things to consider. Usually, SSL problems are first time setup problems because the configuration can be difficult. Each client must contain the server's signer certificate. During mutual authentication, each server must contain the client's signer certificate. Also, there can be protocol differences (SSLv3 vs. TLS), and listener port problems related to stale IORs (i.e., IORs from a server reflecting the port prior to the server restarting).

For SSL problems, we sometimes request an SSL trace to determine what is happening with the SSL handshake. The SSL handshake is the process which occurs when a client opens a socket to a server. If anything goes wrong with the key exchange, cipher exchange, etc. the handshake will fail and thus the socket is invalid. Tracing JSSE (the SSL implementation used in WebSphere Application Server) involves the following steps:

- Set the following system property on the client and server processes: `-Djavax.net.debug=true`. For the server, add this to the Generic JVM Arguments property of the Java virtual machine settings page.
- Turn on ORB trace as well.
- Recreate the problem.

  The `SystemOut.log` of both processes should contain the JSSE trace. You will find trace similar to the following:

```
SSLConnection: install <com.ibm.sslite.e@3ae78375>
>> handleHandshakeV2 <com.ibm.sslite.e@3ae78375>
>> handshakeV2 type = 1
>> clientHello: SSLv2.
SSL client version: 3.0
...
...
...
JSSEContext: handleSession[Socket[addr=null,port=0,localport=0]]

<< sendServerHello.
SSL version: 3.0
SSL_RSA_WITH_RC4_128_MD5
HelloRandom
...
...
...
<< sendCertificate.
<< sendServerHelloDone.
```

```
>> handleData <com.ibm.sslite.e@3ae78375>
>> handleHandshake <com.ibm.sslite.e@3ae78375>
>> handshakeV3 type = 16

>> clientKeyExchange.
>> handleData <com.ibm.sslite.e@3ae78375>
>> handleChangeCipherSpec <com.ibm.sslite.e@3ae78375>
>> handleData <com.ibm.sslite.e@3ae78375>
>> handleHandshake <com.ibm.sslite.e@3ae78375>
>> handshakeV3 type = 20
>> finished.
<< sendChangeCipherSpec.
<< sendFinished.
```

**Trace security**

The classes which implement WebSphere Application Server security are:
- com.ibm.ws.security.*
- com.ibm.websphere.security.*
- com.ibm.WebSphereSecurityImpl.*
- SASRas

To view detailed information on the run time behavior of security, enable trace on the following components and review the output:
- `com.ibm.ws.security.*=all=enabled:com.ibm:WebSphereSecurityImpl.*=all=enabled:com.ibm.websphere.security.*=all=enabled`. This trace statement collects the trace for the security runtime.
- `com.ibm.ws.console.security.*=all=enabled`. This trace statement collects the trace for the security center GUI.
- `SASRas=all=enabled`. This trace statement collects the trace for SAS (low-level authentication logic).

**Fine tuning SAS traces:**
If a subset of classes need to be traced for the SAS/CSIv2 component, a system property can be specified with the class names comma separated: `com.ibm.CORBA.securityTraceFilter=SecurityConnectionInterceptorImpl, VaultImpl, ...`

**Fine tuning Security traces:**
If a subset of packages need to be traced, specify a trace specification more detailed than `com.ibm.ws.security.*=all=enabled`. For example, to trace just dynamic policy code, you can specify `com.ibm.ws.security.policy.*=all=enabled`. To disable dynamic policy trace, you can specify `com.ibm.ws.security.policy.*=all=disabled`.

**Configuring CSIv2, or z/SAS Trace Settings**
Situations arise where reviewing trace for the CSIv2 and z/SAS authentication protocols can assist in troubleshooting difficult problems. This section describes how to enable to CSIv2 and z/SAS trace.

**Enabling Server-Side CSIv2 and z/SAS Trace**
To enable z/SAS trace in an application server, complete the following:
- Add the trace specification, `SASRas=all=enabled`, to the `server.xml` file or add it to the Trace settings within the WebConsole GUI.
- Typically it is best to also trace the authorization security runtime in addition to the authentication protocol runtime. To do this, use the following two trace specifications in combination: `SASRas=all=enabled:com.ibm.ws.security.*=all=enabled`.
- When troubleshooting a connection type problem, it is beneficial to trace both CSIv2 and SAS or CSIv2 and z/SAS and the ORB. To do this, use the following three trace specifications: `SASRas=all=enabled:com.ibm.ws.security.*=all=enabled:ORBRas=all=enabled`.
- In addition to adding these trace specifications, for ORB trace there are a couple of system properties that also need to be set. Go to the ORB settings in the GUI and add the following two properties: `com.ibm.CORBA.Debug=true` and `com.ibm.CORBA.CommTrace=true`.

### Configuring CSIv2, or SAS Trace Settings

Situations arise where reviewing trace for the CSIv2 or SAS authentication protocols can assist in troubleshooting difficult problems. This section describes how to enable to CSIv2 and SAS trace.

**Enabling Client-Side CSIv2 and SAS Trace**

To enable CSIv2 and SAS trace on a pure client, the following steps need to be taken:

- Edit the file TraceSettings.properties in the `/WebSphere/AppServer/properties` directory.
- In this file, change `traceFileName=` to point to the path in which you want the ouput file created. Make sure you put a double backslash (\\) between each subdirectory. For example, `traceFileName=c:\\WebSphere\\AppServer\\logs\\sas_client.log`
- In this file, add the trace specification string: `SASRas=all=enabled`. Any additional trace strings can be added on separate lines.
- Point to this file from within your client application. On the Java command line where you launch the client, add the following system property: `-DtraceSettingsFile=TraceSettings.properties`.

  **Note:** Do not give the fully qualified path to the `TraceSettings.properties` file. Make sure that the `TraceSettings.properties` file is in your class path.

**Enabling Server-Side CSIv2 and SAS Trace**

To enable SAS trace in an application server, complete the following:

- Add the trace specification, `SASRas=all=enabled`, to the `server.xml` file or add it to the Trace settings within the WebConsole GUI.
- Typically it is best to also trace the authorization security runtime in addition to the authentication protocol runtime. To do this, use the following two trace specifications in combination: `SASRas=all=enabled:com.ibm.ws.security.*=all=enabled`.
- When troubleshooting a connection type problem, it is beneficial to trace both CSIv2 and SAS or CSIv2 and z/SAS and the ORB. To do this, use the following three trace specifications: `SASRas=all=enabled:com.ibm.ws.security.*=all=enabled:ORBRas=all=enabled`.
- In addition to adding these trace specifications, for ORB trace there are a couple of system properties that also need to be set. Go to the ORB settings in the GUI and add the following two properties: `com.ibm.CORBA.Debug=true` and `com.ibm.CORBA.CommTrace=true`.

### CSIv2 CORBA Minor Codes

Whatever exceptions might occur within the security code on either the client or server, the eventual exception will become a CORBA exception. So any exception that occurs gets ″wrapped″ by a CORBA exception, because the CORBA architecture is used by the security service for its own inter-process communication. CORBA exceptions are generic, and indicate a problem in communication between two components. CORBA minor codes are more specific, and indicate the underlying reason that a component could not complete a request.

The following shows the CORBA Minor codes which a client can expect to receive after executing a security-related request such as authentication. It also includes the CORBA exception type that the minor code would appear in.

The following exception shows an example of a CORBA exception where the minor code is 49424300. From the table below, this minor code indicates Authentication Failure. Typically, a descriptive message is also included in the exception to assist in troubleshooting the problem. Here, the detailed message is ″Exception caught invoking authenticateBasicAuthData from SecurityServer for user jdoe. Reason: com.ibm.WebSphereSecurity.AuthenticationFailedException″ which indicates that the authentication failed for user ″jdoe″.

The completed field in the exception indicates whether the method was completed or not. In the case of a `NO_PERMISSION`, the method should never get invoked, so it will always be ″completed:No″. Other exceptions which are caught on the server side could have a completed status of ″Maybe″ or ″Yes″.

```
org.omg.CORBA.NO_PERMISSION: Caught WSSecurityContextException in
WSSecurityContext.acceptSecContext(),
reason: Major Code[0] Minor Code[0] Message[Exception caught invoking
authenticateBasicAuthData from SecurityServer for user jdoe.  Reason:
com.ibm.WebSphereSecurity.AuthenticationFailedException]  minor code: 49424300
completed: No

at com.ibm.ISecurityLocalObjectBaseL13Impl.PrincipalAuthFailReason.
map_auth_fail_to_minor_code(PrincipalAuthFailReason.java:83)
        at com.ibm.ISecurityLocalObjectBaseL13Impl.CSIServerRI.receive_request
               (CSIServerRI.java:1569)
        at com.ibm.rmi.pi.InterceptorManager.iterateReceiveRequest
              (InterceptorManager.java:739)
        at com.ibm.CORBA.iiop.ServerDelegate.dispatch(ServerDelegate.java:398)
        at com.ibm.rmi.iiop.ORB.process(ORB.java:313)
        at com.ibm.CORBA.iiop.ORB.process(ORB.java:1581)
        at com.ibm.rmi.iiop.GIOPConnection.doWork(GIOPConnection.java:1827)
        at com.ibm.rmi.iiop.WorkUnitImpl.doWork(WorkUnitImpl.java:81)
        at com.ibm.ejs.oa.pool.PooledThread.run(ThreadPool.java:91)
        at com.ibm.ws.util.CachedThread.run(ThreadPool.java:149)
```

The following table shows the CORBA Minor codes which a client can expect to receive after executing a security-related request such as authentication. It also includes the CORBA exception type that the minor code would appear in.

| Minor code name | Minor code value (in hex) | Exception type (all in the package of org.omg.CORBA .*) | Minor code description | Retry performed (when authenticationRe tryEnabled=true) |
|---|---|---|---|---|
| AuthenticationFailed | 49424300 | NO_PERMISSION | This is a generic authentication failed error. It does not give any details about whether the userid or password is invalid. Some registries can choose to use this type of error code, others might choose to use the next three types which are more specific. | Yes |
| InvalidUserid | 49424301 | NO_PERMISSION | This occurs when the registry returns bad userid. | Yes |
| InvalidPassword | 49424302 | NO_PERMISSION | This occurs when the registry returns bad password. | Yes |
| InvalidSecurityCredentials | 49424303 | NO_PERMISSION | This is a generic error indicating that the credentials are bad for whatever reason. It could be that they don't have the right attributes set. | Yes, if client has BasicAuth credential (token based credential was rejected in the first place). |
| InvalidRealm | 49424304 | NO_PERMISSION | This occurs when the REALM in the token received from the client does not match the server's current realm. | No |
| ValidationFailed | 49424305 | NO_PERMISSION | A validation failure occurs when a token is sent from the client or server to a target server but the token format or the expiration is invalid. | Yes, if client has BasicAuth credential (token based credential was rejected in the first place). |
| CredentialTokenExpired | 49424306 | NO_PERMISSION | This is more specific about why the validation failed. In this case, the token has a absolute lifetime, and this lifetime has expired. Therefore, it is no longer a valid token and cannot be used. | Yes, if client has BasicAuth credential (token based credential was rejected in the first place). |

| | | | | |
|---|---|---|---|---|
| InvalidCredentialToken | 49424307 | NO_PERMISSION | This is more specific about why the validation failed. In this case, the token cannot be decrypted or the data within it is not readable. | Yes, if client has BasicAuth credential (token based credential was rejected in the first place). |
| SessionDoesNotExist | 49424308 | NO_PERMISSION | This indicates that the CSIv2 session does not exist on the server. Typically, a retry occurs automatically and will successfully create a new session. | Yes |
| SessionConflictingEvidence | 49424309 | NO_PERMISSION | This indicates that a session already exists on the server which matches the context_id sent over by the client, however, the information provided by the client for this EstablishContext message is different from the information originally provided to establish the session. | Yes |
| SessionRejected | 4942430A | NO_PERMISSION | This indicates that the session referenced by the client has been previously rejected by the server. | Yes |
| SecurityServerNotAvailable | 4942430B | NO_PERMISSION | This error occurs when the server cannot contact the security server (whether local or remote) in order to authenticate or validate. | No |
| InvalidIdentityToken | 4942430C | NO_PERMISSION | This error indicates that identity cannot be obtained from the identity token when Identity Assertion is enabled. | No |
| IdentityServerNotTrusted | 4942430D | NO_PERMISSION | This indicates that the server id of the sending server is not on the target server's trusted principal list. | No |
| InvalidMessage | 4942430E | NO_PERMISSION | This indicates that the CSIv2 message format is invalid for the receiving server. | No |
| AuthenticationNotSupported | 49421090 | NO_PERMISSION | This error occurs when a mechanism does not support authentication (very rare). | No |
| InvalidSecurityMechanism | 49421091 | NO_PERMISSION | This is used to indicate that the specified security mechanism is not known. | No |
| CredentialNotAvailable | 49421092 | NO_PERMISSION | This indicates a credential is not available when it is required. | No |
| SecurityMechanismNotSupported | 49421093 | NO_PERMISSION | This error occurs when a security mechanism specified in the CSIv2 token is not implemented on the server. | No |

| | | | | |
|---|---|---|---|---|
| ValidationNotSupported | 49421094 | NO_PERMISSION | This error occurs when a mechanism does not support validation (such as LocalOS). This error should not occur since the LocalOS credential is not a forwardable credential, therefore, validation should never need to be called on it. | No |
| CredentialTokenNotSet | 49421095 | NO_PERMISSION | This is used to indicate the token inside the credential is null. | No |
| ServerConnectionFailed | 494210A0 | COMM_FAILURE | This error is used when a connection attempt fails. | Yes (via ORB retry) |
| CorbaSystemException | 494210B0 | INTERNAL | This is a generic CORBA specific exception in system code. | No |
| JavaException | 494210B1 | INTERNAL | This is a generic error that indicated an unexpected Java exception occurred. | No |
| ValueIsNull | 494210B2 | INTERNAL | This is used to indicate that a value or parameter passed in was null. | No |
| EffectivePolicyNotPresent | 494210B3 | INTERNAL | This indicates that an effective policy object for CSIv2 is not present. This object is used to determine what security configuration features have been specified. | No |
| NullPointerException | 494210B4 | INTERNAL | This is used to indicate that a NullPointerException was caught in the runtime. | No |
| ErrorGettingClassInstance | 494210B5 | INTERNAL | This indicates a problem loading a class dynamically. | No |
| MalFormedParameters | 494210B6 | INTERNAL | This indicates parameters are not valid. | No |
| DuplicateSecurityAttributeType | 494210B7 | INTERNAL | A duplicate credential attribute has been specified during the set_attributes operation. | No |
| MethodNotImplemented | 494210C0 | NO_IMPLEMENT | A method invoked has not been implemented. | No |
| GSSFormatError | 494210C5 | BAD_PARAM | This indicates that a GSS encoding or decoding routine has thrown an exception. | No |
| TagComponentFormatError | 494210C6 | BAD_PARAM | This indicates that a tag component cannot be read properly. | No |
| InvalidSecurityAttributeType | 494210C7 | BAD_PARAM | This indicates an attribute type specified during the set_attributes operation is an invalid type. | No |
| SecurityConfigError | 494210CA | INITIALIZE | A problem exists between the client and server configuration. | No |

For current information available from IBM Support on known problems and their resolution, see the IBM Support page.

IBM Support has documents that can save you time gathering information needed to resolve this problem. Before opening a PMR, see the IBM Support page.

# Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, New York 10594 USA

© Copyright IBM Corp. 2004

# Trademarks and service marks

For trademark attribution, visit the IBM Terms of Use Web site (http://www.ibm.com/legal/us/).

© Copyright IBM Corp. 2004