



Securing applications and their environment

Note

Before using this information, be sure to read the general information under “Notices” on page 1431.

Compilation date: May 23, 2006

© Copyright International Business Machines Corporation 2006. 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.	xi
Chapter 1. Overview and new features for securing applications and their environment	1
What is new for security specialists	1
Common Criteria (EAL4) support	7
Federal Information Processing Standard support.	11
Identity management capabilities	12
Security planning overview	21
Chapter 2. How do I secure applications and their environments?	29
Chapter 3. Task overview: Securing resources	31
Chapter 4. Setting up and enabling security	33
Migrating, coexisting, and interoperating – Security considerations	33
Interoperating with previous product versions	34
Interoperating with a C++ common object request broker architecture client	35
Migrating custom user registries	37
Migrating trust association interceptors	40
Migrating Common Object Request Broker Architecture programmatic login to Java Authentication and Authorization Service (CORBA and JAAS)	42
Migrating from the CustomLoginServlet class to servlet filters	45
Migrating Java 2 security policy	45
Preparing for security at installation time	48
Securing your environment before installation	49
Securing your environment after installation	49
Enabling security	50
Administrative security.	53
Application security.	63
Java 2 security	64
Enabling security for the realm	74
Testing security after enabling it	84
Chapter 5. Authenticating users	87
Selecting a registry or repository	87
User registries and repositories	89
Configuring local operating system registries	90
Configuring Lightweight Directory Access Protocol user registries	93
Configuring standalone custom registries	111
Managing the realm in a federated repository configuration.	115
Local operating system registries	165
Standalone Lightweight Directory Access Protocol registries	168
Federated repositories	171
Authentication mechanisms	173
Portlet URL security	175
Lightweight Third Party Authentication	178
Trust associations	180
Single sign-on	184
Security attribute propagation	191
Simple WebSphere authentication mechanism	205
UserRegistry interface methods	205
Authentication protocol for EJB security	212
Supported authentication protocols	215

Common Secure Interoperability Version 2 features	216
Identity assertion	216
Identity assertions with trust validation	217
Message layer authentication	218
Configuring the Lightweight Third Party Authentication mechanism	219
Authentication mechanisms and expiration	220
Generating Lightweight Third Party Authentication keys	223
Exporting Lightweight Third Party Authentication keys	223
Importing Lightweight Third Party Authentication keys	224
Disabling automatic generation of Lightweight Third Party Authentication keys	225
Managing LTPA keys from multiple WebSphere Application Server cells	225
Activating Lightweight Third Party Authentication key versions	226
Integrating third-party HTTP reverse proxy servers	227
Trust association settings	227
Trust association interceptor collection	228
Trust association interceptor settings	228
Implementing single sign-on to minimize Web user authentications	228
Configuring single sign-on capability with SPNEGO TAI	232
Configuring single sign-on capability with Tivoli Access Manager or WebSEAL	252
Propagating security attributes among application servers	267
Configuring the authentication cache	269
Security cache properties	269
Configuring IOP authentication	270
Configuring Common Secure Interoperability Version 2 inbound authentication	270
Configuring Common Secure Interoperability Version 2 outbound authentication	274
Example: Common Secure Interoperability Version 2 scenarios	280
Configuring RMI over IOP	286
Configuring inbound transports	287
Configuring outbound transports	291
Performing identity mapping for authorization across servers in different realms	294
Common Secure Interoperability Version 2 and Security Authentication Service client configuration	308
Java Authentication and Authorization Service	312
Java Authentication and Authorization Service authorization	313
Using the Java Authentication and Authorization Service programming model for Web authentication	315
Chapter 6. Authorizing access to resources	319
Authorization technology	319
Administrative roles and naming service authorization	320
Role-based authorization	324
Administrative roles	326
Enterprise bean component security	329
Authorization providers	329
Delegations	341
Programmatic login	343
Authorizing access to J2EE resources using Tivoli Access Manager	350
Using the default authorization provider	350
Enabling an external JACC provider	354
Authorizing access to administrative roles	372
Administrative user roles settings and CORBA naming service user settings	372
Administrative group roles and CORBA naming service groups	374
Assigning users to naming roles	376
Propagating administrative role changes to Tivoli Access Manager	376
The migrateEAR utility for Tivoli Access Manager	377
Chapter 7. Securing communications	381

Secure communications using Secure Sockets Layer	381
Secure Sockets Layer configurations	387
Keystore configurations	392
Dynamic outbound selection of Secure Sockets Layer configurations	395
Central management of Secure Sockets Layer configurations	396
Secure Sockets Layer node, application server, and cluster isolation	397
Default self-signed certificate configuration	402
Dynamic configuration updates	411
Certificate management using iKeyman	412
Certificate management	413
Creating a Secure Sockets Layer configuration	415
SSL certificate and key management	418
SSL configurations for selected scopes	419
SSL configurations collection	421
SSL configuration settings	421
Creating a custom trust manager configuration	422
Creating a custom key manager	428
Associating a Secure Sockets Layer configuration dynamically with an outbound protocol and remote secure endpoint	434
Quality of protection (QoP) settings	443
ssl.client.props client configuration file	445
Creating a keystore configuration	452
Changing a keystore password	452
Configuring a hardware cryptographic keystore	453
Managing keystore configurations remotely	454
Key stores and certificates collection	454
Key store settings	455
Key managers collection	456
Key managers settings	457
Creating a self-signed certificate	458
Replacing an existing self-signed certificate	458
Creating a certificate authority request	459
Certificate request settings	460
Personal certificates collection	461
Personal certificates settings	462
Personal certificate requests collection	464
Personal certificate requests settings	465
Extract certificate request	466
Receiving a certificate issued by a certificate authority	466
Replace a certificate	469
Extracting a signer certificate from a personal certificate	470
Extract certificate	470
Extract signer certificate	471
Retrieving signers using the retrieveSigners utility at the client	471
Changing the signer auto-exchange prompt at the client	472
Importing a signer certificate from a truststore to a z/OS keyring	473
Exporting a signer certificate from WebSphere Application Server for z/OS to a truststore	474
Importing a signer certificate from a truststore to a z/OS keyring	475
Exporting a signer certificate from WebSphere Application Server for z/OS to a truststore	476
Retrieving signers from a remote SSL port	477
Retrieve from port	478
Adding a signer certificate to a keystore	479
Add signer certificate	479
Signer certificates collection	480
Signer certificate settings	480
Exchanging signer certificates	481

Key stores and certificates exchange signers	482
Configuring certificate expiration monitoring	482
Manage certificate expiration settings.	483
Notifications	485
Notifications settings	485
Key management for cryptographic uses	486
Creating a key set configuration.	487
Active key history collection	489
Add key alias reference settings	489
Key sets collection	490
Key sets settings	490
Creating a key set group configuration	492
Example: Retrieving the generated keys from a key set group	493
Example: Developing a key or key pair generation class for automated key generation	494
Key set groups collection	497
Key set groups settings	497
Chapter 8. Developing extensions to the WebSphere security infrastructure	501
Developing standalone custom registries	501
Example: Standalone custom registries	502
Result.java file	502
UserRegistry.java files	503
Implementing custom password encryption	510
Developing applications that use programmatic security	510
Protecting system resources and APIs (Java 2 security)	511
Developing with programmatic security APIs for Web applications	533
Developing with programmatic APIs for EJB applications	537
Customizing Web application login.	541
Example: Form login	542
Developing servlet filters for form login processing	544
Customizing application login with Java Authentication and Authorization Service	548
Developing programmatic logins with the Java Authentication and Authorization Service	548
Configuring programmatic logins for Java Authentication and Authorization Service	552
Customizing an application login to perform an identity assertion	571
Customization of a server-side Java Authentication and Authorization Service authentication and login configuration	573
Enabling identity assertion with trust validation	592
Secure transports with JSSE and JCE programming interfaces	593
Configuring Federal Information Processing Standard Java Secure Socket Extension files	597
Implementing tokens for security attribute propagation	600
Implementing a custom propagation token	600
Implementing a custom authorization token	608
Implementing a custom single sign-on token	616
Implementing a custom authentication token	627
Propagating a custom Java serializable object	635
Plug point for custom password encryption	639
Enabling custom password encryption	640
Chapter 9. Configuring security with scripting	643
Enabling and disabling administrative security using scripting	644
Enabling and disabling Java 2 security using scripting	645
Enabling authentication in the file transfer service using scripting	645
Propagating security policy of installed applications to a JACC provider using wsadmin scripting	646
Configuring the JACC provider for Tivoli Access Manager using the wsadmin utility.	648
Disabling embedded Tivoli Access Manager client using wsadmin	649
Creating an SSL configuration at the node scope using scripting.	650

Creating self-signed certificates using scripting	652
Automating SSL configurations using scripting	653
Updating default key store passwords using scripting	656
Commands for the IdMgrConfig group of the AdminTask object	656
Commands for the IdMgrRepositoryConfig group of the AdminTask object	661
Commands for the IdMgrRealmConfig group of the AdminTask object	750
Commands for the WIMManagementCommands group of the AdminTask object	758
Commands for the KeyStoreCommands group of the AdminTask object	778
Commands for the SSLConfigCommands group of the AdminTask object	787
Commands for the DescriptivePropCommands group of the AdminTask object	800
Commands for the TrustManagerCommands group of the AdminTask object	803
Commands for the keyManagerCommands group of the AdminTask object	807
Commands for the SSLConfigGroupCommands group of the AdminTask object	811
Commands for the DynamicSSLConfigSelections group of the AdminTask object.	816
Commands for the ManagementScopeCommands group of the AdminTask object	820
Commands for the WSCertExpMonitorCommands group of the AdminTask object	824
Commands for the KeySetGroupCommands group of the AdminTask object	830
Commands for the KeySetCommands group of the AdminTask object	836
Commands for the KeyReferenceCommands group of the AdminTask object	842
Commands for the securityEnablement group of the AdminTask object	846
Commands for the CertificateRequestCommands group of the AdminTask object	852
Commands for the SignerCertificateCommands group of the AdminTask object	856
Commands for the PersonalCertificateCommands group of the AdminTask object	862
Commands for the SPNEGO TAI group of the AdminTask object.	872
Commands for the AuthorizationGroupCommands group of the AdminTask object	879
Commands for the ChannelFrameworkManagement group of the AdminTask object	893
Chapter 10. Web applications	899
Securing Web applications using an assembly tool.	899
Security constraints	901
Security settings	901
Security role references.	902
Securing applications during assembly and deployment	903
Assigning users and groups to roles	904
Updating and redeploying secured applications	912
Deploying secured applications	913
Chapter 11. SIP applications	915
Securing SIP applications	915
Configuring security for the SIP container	915
Chapter 12. EJB applications	919
Securing enterprise bean applications	919
Configuring security for EJB 2.1 message-driven beans	921
Chapter 13. Client applications	923
Accessing secure resources using SSL and applet clients	923
Applet client security requirements.	923
Chapter 14. Web services	925
Configuring HTTP outbound transport level security with the administrative console.	925
HTTP SSL Configuration collection	925
Configuring HTTP outbound transport level security with an assembly tool	926
Configuring HTTP outbound transport-level security using Java properties	927
Transport level security	927
HTTP basic authentication.	928

Configuring HTTP basic authentication with the administrative console	928
HTTP basic authentication collection	929
Configuring HTTP basic authentication with an assembly tool	929
Configuring HTTP basic authentication programmatically	930
Configuring additional HTTP transport properties using the JVM custom property panel in the administrative console	930
Configuring additional HTTP transport properties with an assembly tool	931
Configuring additional HTTP transport properties using the wsadmin command-line tool	933
Provide HTTP endpoint URL information	934
Specify endpoint URL prefixes for Web services.	934
Select default HTTP URL prefix	934
Select custom HTTP URL prefix	935
Publish WSDL zip files settings	935
Securing Web services for Version 6 and later applications based on WS-Security	935
What is new for securing Web services	940
Web services security enhancements	954
High-level architecture for Web services security	958
Overview of platform configuration and default bindings	960
Security model mixture	963
Security considerations for Web services	964
Migrating Version 5.x applications with Web services security to Version 6.1 applications.	966
Default implementations of the Web services security service provider programming interfaces	979
Default configuration	982
Basic Security Profile compliance	988
Configuring an application for Web services security with an assembly tool.	989
Configuring trust anchors for the generator binding on the application level	1083
Configuring the collection certificate store for the generator binding on the application level	1087
Username token element.	1096
Nonce, a randomly generated token.	1097
Custom security token propagation	1097
rrdSecurity.props file	1098
Configuring the token generator on the application level	1098
Configuring the key locator for the generator binding on the application level.	1115
Configuring the key information for the generator binding on the application level	1121
Configuring the signing information for the generator binding on the application level.	1132
Configuring the encryption information for the generator binding on the application level	1146
Configuring trust anchors for the consumer binding on the application level	1157
Configuring the collection certificate store for the consumer binding on the application level	1159
Binary security token	1160
Configuring token consumer on the application level.	1161
Configuring the key locator for the consumer binding on the application level.	1170
Configuring the key information for the consumer binding on the application level	1171
Configuring the signing information for the consumer binding on the application level.	1173
Configuring the encryption information for the consumer binding on the application level	1178
Hardware cryptographic device support for Web Services Security	1180
Retrieving tokens from the JAAS Subject in a server application	1182
Retrieving tokens from the JAAS Subject in an application	1184
Configuring trust anchors on the server or cell level	1184
Configuring the collection certificate store for the server or cell-level bindings	1185
Distributed nonce caching	1187
Configuring a nonce on the server or cell level	1187
Configuring token generators on the server or cell level	1188
Configuring the key locator on the server or cell level	1198
Configuring the key information for the generator binding on the server or cell level	1200
Configuring the signing information for the generator binding on the server or cell level	1202
Configuring the encryption information for the generator binding on the server or cell level	1205

Configuring trusted ID evaluators on the server or cell level	1206
Configuring token consumers on the server or cell level	1210
Configuring the key information for the consumer binding on the server or cell level	1218
Configuring the signing information for the consumer binding on the server or cell level	1219
Configuring the encryption information for the consumer binding on the server or cell level	1221
Tuning Web services security for Version 6.1 applications.	1223
Securing Web services for Version 5.x applications based on WS-Security	1224
Web services security specification—a chronology	1225
Web services security support	1226
Web services security and Java 2 Platform, Enterprise Edition security relationship	1229
Web services security model in WebSphere Application Server.	1231
Web services: Default bindings for the Web services security collection.	1234
Usage scenario for propagating security tokens	1235
Web services security constraints	1237
Overview of authentication methods.	1246
XML digital signature	1249
Securing Web services for Version 5.x applications using XML digital signature.	1253
XML encryption	1303
Securing Web services for Version 5.x applications using XML encryption.	1306
Securing Web services for Version 5.x applications using basicauth authentication	1325
Identity assertion.	1332
Securing Web services for Version 5.x applications using identity assertion authentication.	1333
Securing Web services for version 5.x applications using signature authentication.	1339
Overview of token types	1344
Security token.	1350
Securing Web services for version 5.x applications using a pluggable token	1351
Tuning Web services security for Version 5.x applications.	1362
Enabling security for WSIF	1362
Security API for the UDDI Version 3 registry.	1363
Chapter 15. Data access resources	1365
Security of lookups with component managed authentication	1365
Chapter 16. Messaging resources.	1367
Configuring authorization security for a Version 5 default messaging provider	1367
Authorization settings for Version 5 default JMS resources	1369
Securing WebSphere MQ messaging directories and log files	1371
Configuring security for EJB 2.1 message-driven beans	1372
Chapter 17. Mail, URLs, and other J2EE resources	1373
JavaMail security permissions best practices	1373
Chapter 18. Learn about WebSphere programming extensions	1375
Scheduler	1375
Securing scheduler tasks.	1375
Chapter 19. Tuning, hardening, and maintaining	1377
Tuning security configurations	1377
Secure Sockets Layer performance tips	1379
Tuning security	1381
Hardening security configurations.	1382
Securing passwords in files	1382
Encoding password in files	1382
Enabling custom password encryption	1385
Chapter 20. Troubleshooting security configurations	1387

Security components troubleshooting tips	1387
Errors when trying to configure or enable security	1398
Errors after enabling security	1400
Access problems after enabling security	1405
Errors after configuring or enabling Secure Sockets Layer	1410
Errors configuring Secure Sockets Layer encrypted access	1412
Single sign-on configuration troubleshooting tips	1414
Authorization provider troubleshooting tips	1416
SPNEGO trust association interceptor (TAI) troubleshooting tips	1420
Problem: WebSphere Application Server and the Active Directory (AD) Domain Controller's time are not synchronized within 5 minutes	1421
Problem: Getting exception: No factory available to create a name for mechanism 1.3.6.1.5.5.2	1421
Problem: Getting an exception.	1422
Problem: Single sign-on is not occurring.	1422
Problem: Credential Delegation is not working	1423
Problem: Unable to get SSO working using RC4-HMAC encryption.	1423
Problem: User receives the following message when accessing a protected URL through the SPNEGO SSO.	1424
Problem: Even with JGSS tracing disabled, some KRB_DBG_KDC messages appear in the SystemOut.log.	1424
Problem: HTTP Post parameters are lost during interaction with the SPNEGO TAI, when stepping down to userid/password login..	1424
Appendix. Directory conventions	1427
Notices	1431
Trademarks and service marks	1433

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.

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

Use the links provided in this topic to learn more about the security infrastructure.

“What is new for security specialists”

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

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.

“Security planning overview” on page 21

Several communication links are provided 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 configurations 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.

Samples

The Samples Gallery offers:

- **Login - Form Login**

The Form Login Sample demonstrates a very simple example of how to use the login facilities for WebSphere Application Server to implement and configure login applications. The Sample uses the Java 2 Platform, Enterprise Edition (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 version contains many new and changed features for those who are responsible for securing applications and the application serving environment.

New in Version 6.1! indicates new features or changes implemented at the Version 6.1 level. Unmarked items are Version 6.0 improvements that apply also to Version 6.1, which should interest anyone migrating to Version 6.1 from Version 5.x.

Deprecated and removed features describes features that are being replaced or removed in this or future releases.

Ease of use

Administrative security enabled out of box

New in Version 6.1! Access to the administrative system and its data is now protected by default. When creating a profile, whether during or after installation, you will be prompted whether to keep the default. The default is for administrative security to be enabled with the file based user repository as the user registry. The file based repository is implemented using virtual member manager. For information about this option, see “Managing the realm in a federated repository configuration” on page 115.

Simplified security configuration and administration

Rest assured that if you are migrating from a prior product version, the existing security configuration will be preserved.

New in Version 6.1!

- Simplified administrative console security panels
- New security wizard
- Security configuration reporting tool

Automatically generated server IDs

New in Version 6.1! This version distinguishes between the user identities for administrators who manage the environment and server identities for authenticating server to server communications. In most cases, server identities are automatically generated and are not stored in a repository. You can change the ID if you like.

You no longer need to specify a server user ID and password during security configuration, unless using a mixed cell environment. To maintain backwards compatibility, you must specify the server user ID.

Simplified WebSphere® key and certificate management

See “Local operating system settings” on page 92.

New in Version 6.1! Simplified WebSphere key and certificate management has been added to:

- Allow you to use the key management tools from the console
- Make it easier to configure Secure Sockets Layer (SSL) attributes
- Manage Web server and plug-in certificates from the console
- Use the TrustManager to automatically trust hosts or signers
- Make it easier to refresh an expiring certificate

Federate various repositories, so you can manage them as one

New in Version 6.1! Inclusion of virtual member manager in this release provides a single model for managing organizational entities. You can configure a realm that consists of identities in the file-based repository that is built into the system, in one or more external repositories, or in both the built-in, file-based repository and in one or more external repositories.

Currently most WebSphere Application Server applications have their own models and components for managing organizational entities, and they provide different levels of security. Most applications are dependent on specific types and brands of repositories, assume a specific schema for the data in those repositories, and are not able to use repositories with existing data. Virtual member manager helps these applications by providing them a common model, secure access to various brands and types of repositories, and the ability to use repositories with existing data. The single model includes a set of organizational entity types and their properties, a repository-independent application programming interface (API) and a Service Provider Programming Interface (SPI) for plugging in repositories. XPath is chosen as the search language in the API and SPI.

For more information, see “Federated repositories” on page 171.

Standards support and interoperability

SPNEGO support for single sign-on authentication through Windows desktop

New in Version 6.1! The Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) protocol allows flowing Kerberos tokens from web browsers such as Mozilla FireFox or Microsoft Internet Explorer. This enables seamless single-sign-on experiences on Windows desktops with web browsers that support SPNEGO.

See “Configuring SPNEGO TAI in WebSphere Application Server” on page 235.

Interoperability with other vendors on WS-Security

New in Version 6.1! The product now supports the WS-I Basic Security Profile 1.0, which promotes interoperability by addressing the most common problems encountered from implementation experience to date.

See Web Services-Interoperability Basic Profile.

Common Criteria Assurance Level 4 security

The product has been enhanced to provide Common Criteria Assurance Level 4 security functionality, with full certification available in 2005. Common Criteria is a scheme for independent assessment, analysis, and testing of IT products to a set of security requirements. Certification gives customers the confidence that products will be effective in delivering security functions such as identification and authentication, user data protection, audit, and cryptographic support. Customers gain assurance that the security functions are correctly implemented and will be effective in satisfying their security objectives.

For more information, see “Common Criteria (EAL4) support” on page 7.

Full FIPS compliance

The product has been enhanced to support an implementation of the Federal Information Processing Standards (FIPS) 140-2 government standard. The IBM Java Secure Sockets Extension (JSSE) FIPS 140-2 Cryptographic Module for multi-platforms is a scalable, multipurpose Secure Sockets provider that supports cipher suites via the Java 2 application programming interfaces (APIs) for enhanced protection of sensitive data. It enables the product and other IBM products to run in FIPS mode and help fulfill end-to-end requirements for use of FIPS-certified cryptographic module.

For more information, see “Federal Information Processing Standard support” on page 11.

JCA 1.5 support

WebSphere Application Server Version 6.0.x 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, WebSphere Application Server Version 6.0.x provides an enhanced custom principal and credential mapping programming interface and custom mapping properties at the resource reference level. The custom Java Authentication and Authorization Service (JAAS) login module, which was developed for JCA principal and credential mapping for WebSphere Application Server Version 5.x, is still supported.

Web services security

A pluggable architecture increases the extensibility of Web services security. The implementation includes many of the features that are 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 SOAP message, and specifying the order of the signing or encryption processes.

See “What is new for securing Web services” on page 940.

Messaging security

For security changes pertaining to service integration, search the information center for key word: cjr0420

When administrative security is enabled, the default behavior is for a secure bus to use secure transport protocols. To connect to a secure bus, a user must explicitly be granted the bus connector role. The default bootstrap endpoint is enhanced to use BootstrapSecureMessaging rather than BootstrapBasicMessaging.

For additional details, search the information center for keyword: cjr0009

Web authentication improvements

Separate Web authentication and authorization

New in Version 6.1! Now, Web authentication can be performed with or without Web authorization, and Web client's authenticated identity is available whether or not Web authorization is required. An authenticated identity is persisted both for protected and unprotected resources. Without the separation of Web authentication and Web authorization, a Web authenticated identity is not available when Web authorization is not required, and programmatic security can not work independently without container declarative security.

Enhanced control over Web authentication behavior

New in Version 6.1! WebSphere Application Server provides enhanced control over the authentication behavior for a Web client. Depending upon the option that you select, WebSphere Application Server can retain the authentication data for future use. Also, when you use certificate authentication and authentication fails, you can enable the Application Server to challenge the Web client for a user ID and password.

For more information, see "Authentication mechanisms" on page 173.

Portlet URL security

New in Version 6.1! The product enables direct access to portlet Uniform Resource Locators (URLs), just like servlets. For security purposes, portlets are treated similar to servlets. Most portlet security uses the underlying servlet security mechanism. However, portlet security information resides in the `portlet.xml` file, while the servlet and JavaServer Pages files reside in the `web.xml` file. Also, when you make access decisions for portlets, the security information, if any, in the `web.xml` file is combined with the security information in the `portlet.xml` file. Portlet security must support both programmatic security, that is `isUserInRole`, and declarative security

For more information, see "Portlet URL security" on page 175.

Web authentication using the Java Authentication and Authorization Service programming model

WebSphere Application Server Version 6.0.x 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 sign-on (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 313.

Expanded capabilities

Larger variety of administrative roles

New in Version 6.1! Even more administrative roles are defined to provide degrees of authority that are needed to perform certain administrative functions from either the Web-based administrative console or the system management scripting interface. The newest roles are Deployer and AdminSecurityManager, available through administrative scripting (wsadmin).

For more information, see “Administrative roles” on page 326.

Fine grained administrative role authorization

New in Version 6.1! In prior releases, users granted administrative roles could administer all of the resource instances under the cell. Now the product is more fine-grained, meaning that access can be granted to each user per resource instance.

For more information, see “Fine-grained administrative security” on page 54.

Hardware cryptographic device support for Web services security

New in Version 6.1! Web services security now supports the use of cryptographic hardware devices in two different ways. The hardware cryptographic device can be used to accelerate the cryptographic operations. Also, cryptographic keys can be stored on the hardware cryptographic device and never leave the device.

See “Hardware cryptographic device support for Web Services Security” on page 1180.

Custom password encryption

A plug point for custom password encryption must be created to encrypt and decrypt all passwords in WebSphere Application Server that are currently encoded or decoded using Base64-encoding. The implementation class of this plug point has the responsibility for managing keys, determining the encryption algorithm to use, and for protecting the master secret.

For more information, see the Technote <http://www.ibm.com/support/docview.wss?rs=180&uid=swg21210244>.

Enhanced LDAP support

In addition to support for multiple Lightweight Directory Access Protocol (LDAP) directory services binding and failover, you can dynamically update LDAP binding information without first stopping and restarting application servers.

For more information, see <http://www-1.ibm.com/support/docview.wss?rs=180&uid=swg21210243>.

Programming interfaces for implementing identity assertion with trust validation

If you want an application or system provider to perform an identity assertion with trust validation, it can be accomplished by use of the Java Authentication and Authorization Service (JAAS) login framework, where trust validation is performed in one login module and credential creation in another. These two custom login modules are used to create a JAAS login configuration that performs a login to an identity assertion.

For more information, see “Identity assertions with trust validation” on page 217.

Java 2 security manager

WebSphere Application Server Version 6.0.x provides you with greater control over the permissions 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 520.

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.

See Transport chains.

Common Criteria (EAL4) support

The National Institute of Standards and Technology (NIST) has developed Common Criteria to ensure you have a safe option for downloading software to use on your systems. Information held by IT products or systems is a critical resource that enables organizations to succeed in their mission. Additionally, individuals have a reasonable expectation that their personal information contained in IT products or systems remain private, be available to them as needed, and not be subject to unauthorized modification. IT products or systems should perform their functions while exercising proper control of the information to ensure it is protected against hazards such as unwanted or unwarranted dissemination, alteration, or loss. The term IT security is used to cover prevention and mitigation of these and similar hazards.

Many consumers of IT lack the knowledge, expertise or resources necessary to judge whether their confidence in the security of their IT products or systems is appropriate, and they may not wish to rely solely on the assertions of the developers. Consumers may therefore choose to increase their confidence in the security measures of an IT product or system by ordering an analysis of its security (in other words, a security evaluation).

To use WebSphere Application Server in the Common Criteria EAL4 evaluated configuration, obtain the WebSphere Application Server EAL4 Guidance document available from <http://www.ibm.com/support/docview.wss?rs=180&uid=swg24011697>. The document describes how to install and configure WebSphere Application Server in the evaluated configuration and how to manage and deploy applications into the evaluated configuration.

The Java 2, Enterprise Edition specification overview describes how to confirm the supported J2EE specifications and their corresponding application programming interfaces (APIs). The following J2EE specifications, as implemented by this product, require further explanation here, regarding their methods that are relevant to security.

Interoperable Naming Service (INS)

See Naming roles for the list of interface methods that are supported and are relevant to security.

Java Message Service (JMS)

The default messaging provider implements the JMS 1.1 specification (part of J2EE 1.4) and some extensions described in the product documentation. Its security model effects the JMS API developer. The following addendums apply to the JMS 1.1 specification when using the default messaging provider. Methods not listed in the table have no security relevance.

Class	Method	Messaging role required	Behavior on security exception	Notes
-------	--------	-------------------------	--------------------------------	-------

javax.jms.Session	createProducer	sender	Throws JMSSecurityException wrapping SINotAuthorizedException	3,4
javax.jms.Session	createConsumer	receiver	Throws JMSSecurityException wrapping SINotAuthorizedException	3,4
javax.jms.Session	createDurableSubscriber	receiver	Throws JMSSecurityException wrapping SINotAuthorizedException	1,3,4
javax.jms.Session	createBrowser	browser	Throws JMSSecurityException wrapping SINotAuthorizedException	3,4
javax.jms.Session	createTemporaryQueue	creator	Throws JMSSecurityException wrapping SINotAuthorizedException	3,4
javax.jms.Session	createTemporaryTopic	creator	Throws JMSSecurityException wrapping SINotAuthorizedException	3,4
javax.jms.Session	unsubscribe		Throws JMSSecurityException wrapping SINotAuthorizedException	2,3,4
javax.jms.MessageProducer	send	sender	Throws JMSSecurityException wrapping SINotAuthorizedException	3,4
javax.jms.MessageConsumer	receive	receiver	Throws JMSEException wrapping SINotAuthorizedException	3,4
javax.jms.MessageConsumer	receiveNoWait	receiver	Throws JMSEException wrapping SINotAuthorizedException	3,4
javax.jms.QueueBrowser	getEnumeration	browser	Throws JMSSecurityException wrapping SINotAuthorizedException	3,4
javax.jms.ConnectionFactory	createConnection	connector	Throws JMSSecurityException wrapping either SINotAuthorizedException or SIAuthenticationException	3,4
javax.jms.QueueSession	createReceiver	receiver	Throws JMSSecurityException wrapping SINotAuthorizedException	3,4

javax.jms.QueueSession	createSender	sender	Throws JMSSecurityException wrapping SINotAuthorizedException	3,4
javax.jms.QueueSession	createBrowser	browser	Throws JMSSecurityException wrapping SINotAuthorizedException	3,4
javax.jms.QueueSession	createTemporaryQueue	creator	Throws JMSSecurityException wrapping SINotAuthorizedException	3,4
javax.jms.QueueSender	send	sender	Throws JMSSecurityException wrapping SINotAuthorizedException	3,4
javax.jms.QueueConnectionFactory	createQueueConnection	connector	Throws JMSSecurityException wrapping either SINotAuthorizedException or SIAuthenticationException	3,4
javax.jms.QueueRequestor	<i>constructor</i>	sender, receiver, creator	Throws JMSSecurityException wrapping SINotAuthorizedException	3,4
javax.jms.QueueRequestor	request	sender, receiver	Throws JMSSecurityException or JMSEException, both wrapping SINotAuthorizedException	3,4
javax.jms.TopicSession	createSubscriber	receiver	Throws JMSSecurityException wrapping SINotAuthorizedException	3,4
javax.jms.TopicSession	createDurableSubscriber	receiver	Throws JMSSecurityException wrapping SINotAuthorizedException	3,4
javax.jms.TopicSession	createPublisher	sender	Throws JMSSecurityException wrapping SINotAuthorizedException	3,4
javax.jms.TopicSession	createTemporaryTopic	creator	Throws JMSSecurityException wrapping SINotAuthorizedException	3,4
javax.jms.TopicPublisher	publish	sender	Throws JMSSecurityException wrapping SINotAuthorizedException	3,4
javax.jms.TopicConnectionFactory	createTopicConnection	connector	Throws JMSSecurityException wrapping either SINotAuthorizedException or SIAuthenticationException	3,4

javax.jms.TopicRequestor	<i>constructor</i>	sender, receiver, creator	Throws JMSecurityException wrapping SInotAuthorizedException	3,4
javax.jms.TopicRequestor	request	sender, receiver	Throws JMSecurityException or JMSEException, both wrapping SInotAuthorizedException	3,4

1. When reconnecting to an existing subscription, must use same user ID as used when subscription was created.
2. Must use the same user ID as used when subscription was created.
3. Wrapped exceptions can be retrieved via JMSEException.getLinkedException().
4. The user ID that will be used for access control depends upon the environment from which the method is invoked, according to the following table.

Environment	User ID used
Stand-alone client	User ID specified on createConnection, otherwise null.
Application server	<ul style="list-style-type: none"> • For container managed authentication: User ID in container managed authentication alias specified in application resource reference. • For component managed authentication: User ID specified on createConnection, otherwise user ID in component managed authentication alias specified in connection factory.
Application client, using local connection factory	<ul style="list-style-type: none"> • For container managed authentication: User ID specified on createConnection, otherwise user ID specified in connection factory. • For component managed authentication: User ID specified on createConnection, otherwise null.
Application client, using server connection factory (deprecated)	User ID specified on createConnection, otherwise null.

Universal Description Discovery & Integration (UDDI)

The WebSphere UDDI Registry supports the OASIS UDDI standard 3.0.2.

Note that the WebSphere UDDI Registry supports the following UDDI APIs from the v 3.0.2 standard:

- v3 Inquiry API
- v3 Publication API
- v3 Security API
- v3 intra-node Custody Transfer API
- v3 HTTP GET services
- v1 and v2 Inquiry API
- v1 and v2 Publish API

The supported APIs require permissions, as described in **TOPIC_NAME?** of the WebSphere UDDI Registry documentation.

The WebSphere UDDI Registry does not support the following programming interfaces.

- inter-node Custody Transfer API

- Subscription API
- Replication API
- Subscription Listener API
- Value Set API

Federal Information Processing Standard support

Federal Information Processing Standards (FIPS) are standards and guidelines issued by the United States National Institute of Standards and Technology (NIST) for federal government computer systems. FIPS are developed when there are compelling federal government requirements for standards, such as for security and interoperability, but acceptable industry standards or solutions do not exist. Government agencies and financial institutions use these standards to ensure that the products conform to specified security requirements. For more information on these standards, see the National Institute of Standards and Technology.

WebSphere Application Server integrates cryptographic modules including Java Secure Socket Extension (JSSE) and Java Cryptography Extension (JCE), which have undergone FIPS 140-2 certification. In the WebSphere Application Server documentation, the IBM JSSE and JCE modules that have undergone FIPS certification are referred to as IBMJSSEFIPS and IBMJCEFIPS.

To enable FIPS for WebSphere Application Server, see *Configuring Federal Information Processing Standard Java Secure Socket Extension files*. When you enable FIPS, several components of the Application Server are affected including the cipher suites, the cryptographic providers, the load balancer, the caching proxy, the high availability manager, and the data replication service.

See “Secure transports with JSSE and JCE programming interfaces” on page 593 for more information on the impact the Federal Information Processing Standard has on WebSphere Application Server.

You can use the following IBM products with WebSphere Application Server and maintain a FIPS level of security compliance:

DB2 Version 8.2

The DB2 Universal Database uses FIPS 140-2 approved cryptographic providers.

Application Server Toolkit

The Application Server Toolkit uses FIPS 140-2 approved cryptographic providers.

IBM Tivoli Directory Server

The IBM Tivoli Directory Server provides the **Use FIPS certified implementation** option, which enables the directory server to use the FIPS-certified encryption algorithms. For more information, see “Setting the level of encryption” within the IBM Tivoli Directory Server Administration Guide.

WebSphere Application Server - Edge Component

The caching proxy contains a directive for enabling FIPS. For more information, see the *Caching Proxy Administration Guide* at the following Web site: <http://www-306.ibm.com/software/webservers/appserv/doc/v602/ec/infocenter/index.html>.

IBM WebSphere MQ

 When cryptography is required in an SSL channel, WebSphere MQ uses a cryptography package called IBM Crypto for C (ICC). On all the Windows and UNIX platforms that are supported by WebSphere MQ Version 6.x, the ICC software passed the FIPS 140-2 Cryptomodule Validation Program of the National Institute of Standards and Technology.

You can find more information about the Federal Information processing Standards (FIPS) on the Support Web site including recommended updates for WebSphere Application Server.

Identity management capabilities

Today's security infrastructure is identity driven. Secure business applications ask two key questions: who are you, and what can you access? IBM offers a complete solution for identity management across the enterprise. Learn about the identity management capabilities provided by WebSphere Application Server in conjunction with other WebSphere products and Tivoli.

For an identity management primer, refer to the identity management white paper, Help improve security and lower costs with repeatable identity management solutions. The paper describes the:

- Business landscape that makes identity management increasingly important
- Benefits of deploying an identity-driven infrastructure
- Basic functions to expect within an identity-driven infrastructure
- Known patterns into which business operations fall
- Tivoli software that provides solutions for addressing the patterns

WebSphere Application Server includes virtual member manager, for the primary purpose of providing out of box security and an additional user registry option: the federated, file-based repository. See "Federated repositories" on page 171 for details. Virtual member manager provides additional identity management capabilities. So do many Tivoli products that extend and complement the solid base for identity management now provided by the WebSphere platform.

As described later, the IBM portfolio supports the following identity management capabilities.

- Offer single sign on for your users' convenience
- Control access to Web applications
- Administer identities
- Provision users
- Federate disparate sources of identity data
- Provide standard directory services to applications
- Use strong authentication
- Control and manage access to Web applications, Web services, and your Service Oriented Architecture (SOA)
- Ensure regulatory compliance
- Secure your business portal
- Provide instance based access control for business rules
- Secure work flows containing people interactions
- Secure integration with other business applications
- Government solutions

Products that work in conjunction with WebSphere Application Server to provide a full range of identity management capabilities include, but are not limited to:

- Tivoli products for identity management
- WebSphere Process Server
- WebSphere Portal

Offer single sign on for your users' convenience

With WebSphere Application Server, Web users can authenticate once when accessing Web resources across multiple application servers. Choices for securely negotiating and authenticating HTTP requests for secured resources include trust association interceptor (TAI) that uses the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO), or using Tivoli Access Manager WebSEAL or Tivoli Access Manager plug-in for Web servers as reverse proxy servers to

provide access management and single sign-on (SSO) capability. See “Implementing single sign-on to minimize Web user authentications” on page 228 for details.

Add Tivoli Access Manager for e-business to achieve Web SSO and secure session management across e-communities, to securely extend your business processes to business partners and business affiliates. You can enable a flexible SSO to Web-based applications that can span multiple sites or domains with a range of SSO options, to help eliminate help-desk calls and other security problems associated with multiple passwords. By integrating with other SSO providers (such as Kerberos from a Microsoft domain logon, and client/server SSO solutions) Access Manager goes beyond 'reduced sign-on' to help implement a single authentication for the user across all system interactions.

Upgrade to Tivoli Federated Identity Manager to achieve standardized cross-domain SSO. The product supports a number of SSO federated identity and Web Services security standards, including Liberty Alliance specifications, SAML, WS-Federation, WS-Security and WS-Trust. This enables the company or the provider to interoperate and get SSO benefits from partners who implement any of these standards. Single Sign On (SSO) simplifies sign on for third-party users who typically have a primary relationship with their home organization. A federated business model enables a company to obtain trusted information about a third-party identity (such as customer, supplier, or a client employee) from that user's home organization without having to create, enroll, or manage a new account.

Reduce help-desk calls and other security problems associated with multiple passwords, using Tivoli Access Manager for e-business. Achieve a flexible SSO to Web-based applications that can span multiple sites or domains with a range of SSO options. By integrating with other SSO providers (such as Kerberos from a Microsoft domain logon, and client/server SSO solutions) Access Manager goes beyond 'reduced sign-on' to help implement a single authentication for the user across all system interactions.

Enhance user experience and reduced help desk costs, with one less password to remember. Use Tivoli Access Manager for e-business to achieve Microsoft desktop single sign on. Windows users can be automatically authenticated to applications protected by Access Manager for e-business.

Improve end-user experience through Single Sign On (SSO) implemented by Tivoli Federated Identity Manager.

Reduce administrative cost by delivering rapid enrollment and personalized access to end-users at their convenience with integrated self-care. The Tivoli Access Manager for e-business Self-Registration capability enables end-users to quickly self-enroll to the Enterprise Web environment without requiring manual intervention or lengthy procedures.

Administer identities

WebSphere Application Server provides fine-grained administration.

Use these products in conjunction with WebSphere Application Server for additional capabilities.

Achieve centralized administration of both access control and data protection policies across mainframe and distributed servers with Tivoli Access Manager for Business Integration. An authorized administrator can perform Web-based administration remotely without visiting a system or deploying a special administration client.

Define and manage a centralized authentication, access, and audit policy for a broad range of business initiatives with Tivoli Access Manager for e-business. Initiative include employee, customer and partner portals, CRM systems, e-procurement, cross-company single sign-on (SSO) projects, and outsourcing projects.

Manage users and groups, including dynamic and nested groups. Achieve dynamic group support with Tivoli Access Manager for e-business. An upper limit on static groups makes dynamic groups the only option in some cases, while dynamic groups may be preferred in other environments. Integrates with existing data management environments.

Use Tivoli Access Manager for e-business to achieve integrated security management for critical WebSphere applications leveraging IMS, CICS and DB2 transactions on mainframe and non-mainframe platforms.

Reduce administrative cost by delivering rapid enrollment and personalized access to end-users at their convenience with integrated self-care. The Tivoli Access Manager for e-business Self-Registration capability enables end-users to quickly self-enroll to the Enterprise Web environment without requiring manual intervention or lengthy procedures.

Use the Web Portal Manager of Tivoli Access Manager for Operating Systems for easier, graphically based management.

Access Control Lists (ACLs) help you pro-actively prevent security breaches across your enterprise, using Tivoli Access Manager for Operating Systems.

Scale to tens of millions of entries, as well as groups of hundreds of thousands of members with Tivoli Directory Server.

Enhance directory security with the password strength features of Tivoli Directory Server. Enable the pre-expiration of passwords, the definition of password rules, maintenance of password history and failed attempt account in correlation with ACL protection.

Manage organizations and entities. Lower overhead costs by automatically managing accounts, credentials, and access rights throughout the user life cycle with workflow provided by Tivoli Identity Manager.

Reduce help-desk costs and ease the burden of daily administration on help-desk and IT staff with the self-service interfaces of Tivoli Identity Manager. Enable users to perform password resets, password synchronization, and modification to personal information without administrative intervention.

Use Tivoli Identity Manager to cut elapsed turn-on time for new accounts; improve productivity by allowing end users to rapidly reset and synchronize their own passwords; and decrease errors by automating user submission and approval requests.

Take advantage of the various products' APIs to integrate and customize your identity management solution.

Provision users

Use these products in conjunction with WebSphere Application Server for additional capabilities.

Quickly connect users to appropriate resources while reducing administration workload, with Tivoli Identity Manager. Embedded provisioning engine and universal integration tools automate the implementation of administrative requests on the environment, and provide universal connectors for extending the management model to support new and custom environments.

Demonstrate enforcement of internal controls to auditors and eliminate orphan or over privileged accounts. Use Tivoli Identity Manager closed loop user provisioning to detect and correct discrepancies between approved account access and local privileges.

Implement and modify provisioning policies more quickly and accurately. Use Tivoli Identity Manager to simulate the impact of provisioning policy on user accounts before committing changes.

Centralize the definition of users and provisioning of user services with the centralized administration of Tivoli Identity Manager. Role and rule-based delegated administration enables grouping of users according to business needs and delegation of administrative privileges along organizational and geographical boundaries.

Tivoli Identity Manager provisioning is integrated with access and identity. Access to Web applications and other applications can be determined by one user profile. The product interacts directly with users and with two external types of systems: identity sources and access control mechanisms. The identity systems deliver authoritative information about the users that need

accounts. The provisioning system communicates directly with access control systems to create accounts, supply user information and passwords and define the entitlements of the account.

Pair Tivoli Identity Manager with WebSphere Process Server to achieve identity-based workflow. Streamlined automated workflow can decrease errors and inconsistency in business processes. Intelligent approval routing automates the submission and approval processes for access requests and changes to user information.

Federate disparate sources of identity data

Federated identity is a technology for brokering identities between companies or business units. Federated identity management is the set of business agreements, technical agreements and policy agreements that enable companies to partner to lower their overall identity management costs and improve user experience. It leverages the concept of a portable identity - the idea that your identity is not bound to a specific credential - to simplify the administration of users in a federated business relationship. Federation simplifies integration because there is a common way to share identities between companies and manage user sessions. Identity Federation services within a Service Oriented Architecture (SOA) ensure that users have simplified access and single sign on to the composite application environment.

Use these products in conjunction with WebSphere Application Server for additional capabilities.

Use Tivoli Access Manager for e-business to provide first-point-of-contact and session management that typically are prerequisites to federation with IBM Tivoli Federated Identity Manager.

Connect to disparate data sources with Tivoli Directory Integrator as an enterprise directory.

Ensure data availability and to maximize server response time, using Tivoli Directory Server to achieve single-master multiple replication, multiple-master replication, cascaded, gateway and partial replication.

Simplify the administration and the lifecycle management of user identities and obtain a simple, loosely-coupled model for managing identity and access to resources that span companies or security domains with the open standards and specifications support of Tivoli Federated Identity Manager, including Liberty, SAML, WS-Federation, WS-Security and WS-Trust. Works with standards based, off-the-shelf products.

Reduce administration and provisioning costs related to managing identities for third-party users, with Tivoli Federated Identity Manager. Rather than having to enroll third-party users into a company's internal identity systems, federated identity management enables IT service providers to offload the cost of user administration to their business partner companies. Because the business partner company acts like an identity provider, the service provider does not have to take on the burden of user administration costs such as user enrollment, account management, password management, password reset, help desk, or customer care costs.

Simplify integration with a common way to share identities between companies and manage user sessions. Tivoli Federated Identity Manager facilitates "straight through processing" techniques because the identity provider does not have to replicate or stage business processes on behalf of a service provider. By employing Tivoli Access Manager for e-business (included with FIM), FIM is able to provide integrated session management, significantly facilitating inter-company transactions. With a federated identity model, identity providers have an opportunity to streamline inter-company transactions, thereby reducing costs, and simplifying integration.

Obtain great flexibility for synchronization, using Tivoli Directory Integrator.

Simplify integration between your company and your partners' Web sites and business applications, including simplified session management with Tivoli Federated Identity Manager.

Provide standard directory services to applications

WebSphere Application Server supports a variety of registry and repository choices, including Lightweight Directory Access Protocol (LDAP) directories. See “Selecting a registry or repository” on page 87.

Use these products in conjunction with WebSphere Application Server for additional capabilities.

Obtain rapid time to value and save development costs with Tivoli Access Manager for Business Integration for application-level data protection for WebSphere MQ-based applications. Implement comprehensive security without writing complex security code, or modifying or recompiling existing applications.

Store credentials in Novell eDirectory with Tivoli Access Manager for Business Integration.

Achieve integration with over 70 ISV offerings, with Tivoli Access Manager for e-business. Offerings including Siebel CRM, SAP, PeopleSoft and Portal solutions from WebSphere, Plumtree, and others. Enterprises benefit from a common security model (authentication, access control, Single Sign On and audit) across the e-business, ISV and legacy applications. This reduces costly integrations and delivers rapid time to value in solution deployment because enterprises can standardize on a single identity and access management platform.

Deploy the security architecture of your choice due to the multiple directory support of Tivoli Access Manager for e-business.

Synchronize and exchange information between applications or directory sources with Tivoli Directory Integrator.

Manage data across a variety of repositories providing the consistent directory infrastructure needed for a wide variety of applications, including security and provisioning, with Tivoli Directory Integrator.

Leverages existing investments in directory and identity repositories, platforms, and operating systems with Tivoli Directory Integrator.

Avoid the time-consuming design of schemata, which can slow the deployment of the LDAP directory. Tivoli Directory Server provides comprehensive, extensible, and dynamically updatable schema.

Manage users and groups, including dynamic and nested groups. Achieve dynamic group support with Tivoli Access Manager for e-business. An upper limit on static groups makes dynamic groups the only option in some cases, while dynamic groups may be preferred in other environments. Integrates with existing data management environments.

Use Tivoli Access Manager for e-business to achieve integrated security management for critical WebSphere applications leveraging IMS, CICS and DB2 transactions on mainframe and non-mainframe platforms.

Reduce administrative cost by delivering rapid enrollment and personalized access to end-users at their convenience with integrated self-care. The Tivoli Access Manager for e-business Self-Registration capability enables end-users to quickly self-enroll to the Enterprise Web environment without requiring manual intervention or lengthy procedures.

Use strong authentication

- With Tivoli Access Manager, WebSphere Application Server supports integration with smart cards and tokens. See “Configuring single sign-on capability with Tivoli Access Manager or WebSEAL” on page 252.

Control and manage access to Web applications, Web services, and Service Oriented Architecture

WebSphere Application Server provides a solid base for protecting Web applications and Web services in a Service Oriented Architecture. See “Securing Web services for Version 6 and later applications based on WS-Security” on page 935.

Use these products in conjunction with WebSphere Application Server for additional capabilities.

Web applications

Leverage a common security policy model with Tivoli Access Manager for e-business to with the Tivoli Access Manager.

Achieve highly secure e-business with Tivoli Access Manager for e-business, which provides flexible deployment that supports proxies, plug-ins, and agents.

Enhance application and database security with Tivoli Access Manager for Operating Systems. Restricts ability to switch user IDs. Prevents deliberate or accidental loss of application data, tampering with log files, and prevents unauthorized assumption of application administrative IDs.

Eliminate many user-access problems, while still using the standard UNIX authentication mechanisms, with Tivoli Access Manager for Operating Systems. Login Policy Enforcement tracks the UNIX login process and applies policies that prevent unauthorized access, such as the number of permitted failed login attempts before the user is locked out.

Ensure application speed and user experience are not impeded by access control decision speed with the multi-threaded architecture enabled by Tivoli Access Manager for Operating Systems.

Prevent password theft with Tivoli Access Manager for Operating Systems.

Ensure security and consistent policy on your most sensitive systems with the centralized control and local autonomy provided by Tivoli Identity Manager.

Defend against the top security threat that enterprises face: misbehavior by internal users and employees with Tivoli Access Manager for Operating Systems.

Ramp up quickly to effective security with Tivoli Access Manager for Operating Systems. Fast Track Policy Modules are pre-written, customizable, best-practice policies.

Achieve centralized administration of security policy across the enterprise with Tivoli Access Manager for Operating Systems.

Create the authoritative data spaces needed to expose only trustworthy data to advanced software applications such as Web services, with Tivoli Directory Integrator.

Authorization services also help application developers use standard development tools such as Eclipse or Rational by providing a standards-based API interfaces.

Web services

Integrates seamlessly with a wide variety of repositories and technologies and enables integration with new and existing Web Services in the enterprise through the standards & Web Services support of Tivoli Directory Integrator.

Extend the reach of the directory to web services with Tivoli Directory Server. Expose the directory and deliver it to web services through XML coding. An enterprise's customers could, for example, make changes to directory data such as phone numbers or street addresses themselves over the Internet rather than calling in to customer service.

Simplify administration of security in cross-enterprise business processes by delivering "security as services" with Tivoli Federated Identity Manager.

Simplify the administration and the lifecycle management of user identities and obtain a simple, loosely-coupled model for managing identity and access to resources that span companies or security domains with the open standards and specifications support of Tivoli Federated Identity Manager, including Liberty, SAML, WS-Federation, WS-Security and WS-Trust.

SOA

SOA Management: Securing Web Services discusses the challenges of security in a Service Oriented Architecture (SOA). The security environment is still disjointedly hard wired into organizational silos segmented into network security, perimeter security, desktop security, server security and application security. Point solutions solve a partial need but they don't work in unison.

Hence, they can't appreciably lower system risk, improve platform integrity, or mitigate the risk of broadening access. SOA adoption introduces new and unforeseen challenges with security integration, identity and security management.

- Multiple Application Platforms (WebSphere, Microsoft or SAP)
- Multiple Security Domains (internal, external, business unit silos, extranet)
- Multiple Security Credentials (Kerberos, SAML, WS-Security, RACF)
- Multiple Protocols (SOAP, HTTP/S, JMS, MQ)
- Lack of "thread of identity" across the services context

Composite Applications must deal with the challenges of independent security and identity silos. The security solution needs to secure end user interactions as well service interactions (application to application). Security management needs to provide unified customer views for the composite application. The "thread" of user identity needs to be preserved end to end for auditing and compliance purposes.

Deliver policy-based integrated security management for SOA Web Services with Tivoli Federated Identity Manager.

Authorization services provided by Tivoli products in the Tivoli identity management solution ensure that SOA components can apply consistent authorization policies for Web, HTTP, and Java resources, Web Services, SOAP (WSDL resources), MQ (Queues and Queue Managers) and even core infrastructure platforms such as UNIX and Linux Servers.

Authorization services in an SOA ensures that a common authorization abstraction model enables application platforms such as WebSphere, MS .NET, BEA and SAP to apply fine-grained authorization for these resource types.

Tivoli Access Manager for e-business implements a centralized policy service for SOA elements enabling business owners to delegate authorization decisions to a Policy server deployed in the SOA environment.

As SOA transactions originate across various channels and protocols it is important to have the centralized session management service of Tivoli Access Manager for e-business to enable various SOA components to have a "common view" of the current user session, for single sign on, single sign off, auditing and reporting, and so on.

Ensure regulatory compliance

Use these products in conjunction with WebSphere Application Server for additional capabilities.

Demonstrate specific compliance with the defined security policy Tivoli Access Manager for Business Integration by obtaining message-level audit function and audit record generation.

Maintain confidentiality of message data and allow for verification of data integrity with Tivoli Access Manager for Business Integration. Securing messages both while they are being processed by WebSphere MQ and while they travel from system to system reduces exposure of data from internal employees or vendors. It can be used as part of a HIPAA compliance solution.

Obtain a central point for reporting on security events and sample reports with Tivoli Access Manager for e-business. An audit and reporting service collects audit data from multiple enforcement points, as well as from other platforms and security applications.

Achieve easier, extended auditing and reporting capabilities with Tivoli Access Manager for e-business. Audit records are written in standard XML format. An information-gathering tool allows secure, centralized collection and reporting of audit, log, statistics, and such across the extended enterprise.

Tivoli Access Manager for Operating Systems combines full-fledged intrusion prevention—host-based firewall, application and platform protection, user tracking and controls—with robust auditing and compliance checking.

Obtain mainframe-class security and auditing in a lightweight, easy-to-use product with Tivoli Access Manager for Operating Systems.

Document compliance with government regulations, corporate policy and other security mandates using the persistent universal auditing of Tivoli Access Manager for Operating Systems.

Obtain extended auditing capabilities with Tivoli Access Manager for Operating Systems. Configurable audit events can track sensitive access attempts, provide security-related information on user activity, and can send events to a centralized event management console. Verify access policy through secure logging of security events.

Improve business compliance by helping to reduce security exposure with Tivoli Federated Identity Manager.

Address policy compliance needs, using Tivoli Identity Manager to produce centralized reports on security policy, access rights, and audit events to quickly respond to internal audits and regulatory mandates.

Quickly produce reports for internal audits and ensuring regulatory compliance, with Tivoli Identity Manager auditing and reporting mechanisms.

Enforces privacy policies across your IT infrastructure with Tivoli Privacy Manager for e-business.

Monitor access to personal information and generate detailed audit logs with Tivoli Privacy Manager for e-business.

Automatically generate reports detailing compliance to corporate policies with Tivoli Privacy Manager for e-business.

Help privacy officers, legal counsel and IT staff work together to build privacy rules that integrate policy into practices, without knowledge of IT systems in order to author policies. This is provided by Tivoli Privacy Manager for e-business.

Update policies in the future with minimal impact to the environment, using Tivoli Privacy Manager for e-business. Monitor and record users' privacy preferences on a separate system from individual applications. You can author one policy and deploy it everywhere there are monitored systems, as a cost-effective alternative to modifying or rewriting existing applications in order to incorporate preferences across applications.

Comply with internal audits and regulatory reviews with Tivoli Privacy Manager for e-business. It generates enterprise-wide reports showing policies deployed, enforcement locations and audit trails that detail the management of personal information according to privacy policies.

Control policies, storage locations, audit logs, preferences and consent across the enterprise with Tivoli Privacy Manager for e-business.

Rapidly develop and customize e-business monitors for applications, middleware data repositories and other systems that persistently store privacy-sensitive information with Tivoli Privacy Manager for e-business. Monitors for LDAP and Siebel 7 are included to allow monitoring, enforcement and auditing for LDAP and Siebel 7 applications.

Use Tivoli Security Compliance Manager to automate scans of servers and desktop systems, which can help reduce the cost and time associated with manual security checks.

Provide detailed reports to security officers and compliance auditors so they can take the appropriate steps to make individual systems and departments compliant, with Tivoli Security Compliance Manager.

Improve business operations and increase efficiencies through automation and centralization with Tivoli Security Compliance Manager.

Mediate security policy violations and risks using Tivoli Security Compliance Manager in conjunction with Tivoli automated security management tools.

Automate compliance tasks, monitor correspondence, reduce human error, and tame compliance costs with Tivoli Security Compliance Manager.

Use Tivoli Security Compliance Manager to ensure consistent security audits across the organization, reducing human error.

Reduce the cost and time associated with manual security checks, using the automated scans of server and desktop systems with Tivoli Security Compliance Manager.

Identify software security vulnerabilities prior to costly damage being inflicted by security incidents, using the security vulnerability scans of Tivoli Security Compliance Manager.

Quickly produce reports for audits and ensuring regulatory compliance with the reporting mechanisms of Tivoli Security Compliance Manager.

Manage and secure your business environments from your existing hardware and operating system platforms with Tivoli Access Manager for e-business.

Secure your business portal

Use Portal in conjunction with WebSphere Application Server to satisfy customers, reduce service costs, and enable customer profiles.

- Portal includes personalization features to be able to respond to end users based on identity (and things derived from identity, such as group memberships and user profile attribute values) and context using rules.
- Portal has its own access control for protecting Portal resources (pages and portlets) that are not visible in a securable way in a URL. This access control is based on user identity and group memberships.
- Portal has a basic user interface for user self-registration and self-care after creation. It also features basic User and Group administration interfaces, not intended to replace the management interfaces of the directory server, or to be as function-rich as TIM.

Provide instance based access control for business rules

Use these products in conjunction with WebSphere Application Server for additional capabilities.

Enact rules-enabled authorization checks with WebSphere Process Server. Its rules technology for conditions enables administrators to create rules depending on the role of an invoker, for example. Although it currently does not allow for checking on role and subject information specifically, developers can use a workaround. A Java code snippet can be used to retrieve this information and feed it into the rule condition.

Dramatically improve both how quickly your applications are deployed and how quickly they adapt, achieving rules based authorization with Tivoli Access Manager for e-business. Change access-influencing policy parameters without having to rewrite and recompile applications.

Reduce administrative costs with Tivoli Identity Manager. Role and rule-based delegated administration enables grouping of users according to business needs and delegation of administrative privileges along organizational and geographical boundaries.

Streamlined automated workflow can decrease errors and inconsistency in business processes. Tivoli Identity Manager automates the submission and approval processes for access requests and changes to user information.

Secure work flows containing people interactions

The staff resolution capabilities of WebSphere Process Choreographer, part of WebSphere Process Server, provide role-based staff assignment and are compatible with various directory services.

- WebSphere Application Server Enterprise Process Choreographer: Staff Resolution Architecture

- WebSphere Application Server Enterprise Process Choreographer: Displaying work items in the Web client
- Working with WebSphere Business Integration Server Foundation Process Choreographer

Secure integration with other business applications

Use these products in conjunction with WebSphere Application Server for additional capabilities.

Leverage your J2EE investment and enable applications to be managed as part of a consistent, policy-driven strategy with Tivoli Access Manager for e-business. The product supports J2EE, Java 2 and JAAS environments, with no plug-in required, no proprietary coding needed and no pre- or post-compile necessary.

Use Tivoli Federated Identity Manager to expand the business reach of service providers creating revenue generating opportunities.

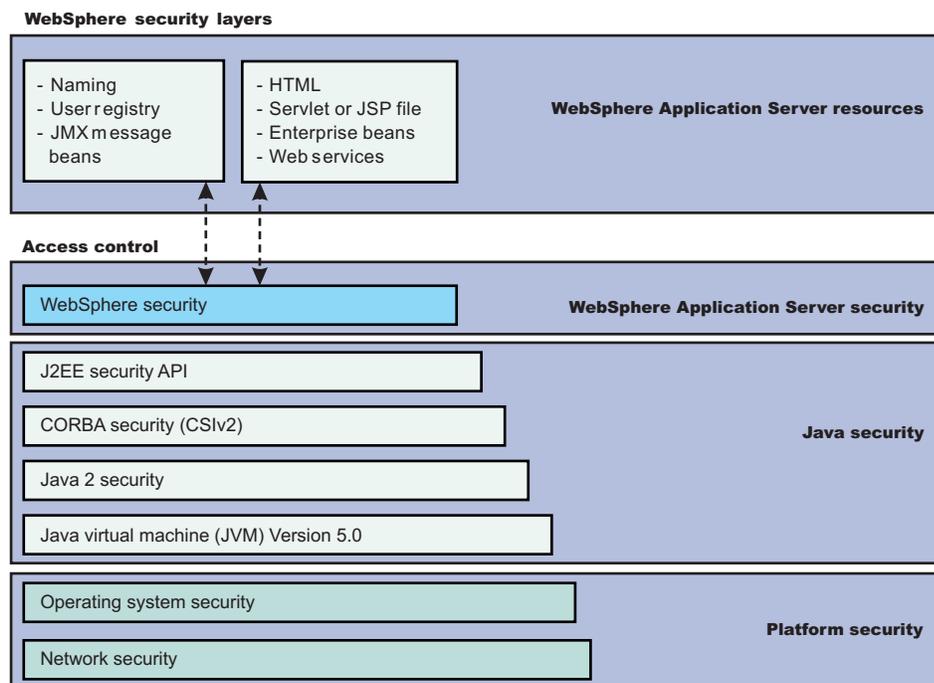
For information about Identity and Privacy Strategies Methodologies and Best Practices, see the related information link below:

Related information

http://www.burtongroup.com/research_consulting/doc.aspx?cid=739

Security planning overview

When you access information on the Internet, you connect 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 shown 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:



The following information describes each of the components of WebSphere Application Server security, Java security, and Platform security that are illustrated in the previous figure.

WebSphere Application Server security

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.

Java security

Java 2 Platform, Enterprise Edition (J2EE) security application programming interface (API)

The security collaborator enforces Java 2 Platform, Enterprise Edition (J2EE)-based security policies and supports J2EE security APIs.

CORBA security (CSIv2)

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.

V6.0.x 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.

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.

Java Virtual Machine (JVM) 5.0

The JVM security model provides a layer of security above the operating system layer. For example, JVM security protects the memory from unrestricted access, creates exceptions when errors occur within a thread, and defines array types.

Platform security

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 secures 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 confidentiality. You can configure the communication between separate application servers to use Secure Sockets Layer (SSL). Additionally, you can use IP Security and Virtual Private Network (VPN) for added message protection.

Each product application server consists of a Web container, an Enterprise Java Beans (EJB) container, and the administrative subsystem.

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 “Encoding password in files” on page 1382.

The administrative console Web application has a setup data constraint that requires access to the administrative console servlets and JavaServer Pages (JSP) files only through an SSL connection when administrative security is enabled.

In WebSphere Application Server Version 6.0.x and earlier, the administrator console HTTPS port was configured to use `DummyServerKeyFile.jks` and `DummyServerTrustFile.jks` with the default self-signed certificate. The dummy certificates and keys must be replaced immediately after WebSphere Application Server installation; the keys are common in all of the installation and are therefore insecure. WebSphere Application Server Version 6.1 provides integrated certificate and key management, which generate distinct private key and self-signed certificate with embedded server host name to enable host name verification. WebSphere Application Server Version 6.1 also enables integration with external certificate (CA) authority to use CA-issued certificates. The WebSphere Application Servers Version 6.1 installation process provides an option to enable administrative security during installation. As a result, a WebSphere Application Server process is secured immediately after installation.

Administrative security

V6.0.x WebSphere Application Servers interact with each other through CSv2 and Secure Authentication Services (SAS) security protocols as well as the HTTP and HTTPS protocols.

Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

You can configure these protocols to use Secure Sockets Layer (SSL) when you enable WebSphere Application Server administrative security. The WebSphere Application Server administrative subsystem in every server uses 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 administrative security is disabled, the SOAP JMX connector uses the HTTP protocol and the RMI/IIOP connector uses the TCP/IP protocol. When administrative security is enabled, the SOAP JMX connector always uses the HTTPS protocol. When administrative 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 administrative security and enable SSL to protect the sensitive configuration data.

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 network message 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 that is derived from the deployment descriptor and specified by deployers and administrators in a set of XML descriptor files. At runtime, 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 security to make access decisions. When administrative 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 that 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 JSP file if the user principal has one of the required security roles. Servlets and JSP files can use the `HttpServletRequest` methods, `isUserInRole` and `getUserPrincipal`.

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

The authentication occurs 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 through the Internet and the intranet. Refer to “Securing Web applications using an assembly tool” on page 899, and “Securing enterprise bean applications” on page 919.

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 Java Naming and Directory Interface (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 runtime.

Administrator role

An operator as well as a configurator, which additionally can modify sensitive security configuration and security policy such as setting server IDs and passwords, enable or disable administrative security and Java 2 security, and map users and groups to the administrator role.

iscadmins

The `iscadmins` role has administrator privileges for managing users and groups from within the administrative console only.

WebSphere Application Server defines two additional roles that are available when you use `wsadmin` scripting only

Deployer

A deployer can perform both configuration actions and run-time operations on applications.

AdminSecurityManager

An administrative security manager can map users to administrative roles. Also, when fine grained admin security is used, users granted this role can manage authorization groups.

A user with the configurator role can perform most administrative work including installing new applications and application servers. Certain configuration tasks exist that a configurator does not have sufficient authority to do when administrative security is enabled, including modifying a WebSphere Application Server identity and password, Lightweight Third-Party Authentication (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.

Enable WebSphere Application Server administrative security to protect administrative subsystem integrity. Application server security can be selectively disabled if no sensitive information is available to protect. For securing administrative security, refer to “Authorizing access to administrative roles” on page 372 and “Assigning users and groups to roles” on page 904.

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 default policies are based on the J2EE Version 1.4 specification. The specification grants 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 policies for all Java virtual machines and WebSphere Application Server processes are contained in the following policy files:

`${java.home}/jre/lib/security/java.policy`

This file is used as the default policy for the Java virtual machine (JVM).

`${USER_INSTALL_ROOT}/properties/server.policy`

This file is 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 a WebSphere Application Server subsystem. These policy files, which are an extension of the WebSphere Application Server runtime, are referred to as *Service Provider Programming Interfaces (SPI)*, and shared by multiple J2EE applications:

`profile_root/config/cells/cell_name/nodes/node_name/spi.policy`

This file is used for embedded resources defined in the `resources.xml` file, such as the Java Message Service (JMS), JavaMail, and JDBC drivers.

`profile_root/config/cells/cell_name/nodes/node_name/library.policy`

This file is used by the shared library that is defined by the WebSphere Application Server administrative console.

profile_root/config/cells/cell_name/nodes/node_name/app.policy

This file is used as the default policy for J2EE applications.

In general, applications do 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 the packaging of a `was.policy` file with each application to grant extra permissions to that application.

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

Loading libraries into WebSphere Application Server does allow applications to leave the Java sandbox. WebSphere Application Server uses a permission filtering policy file to alert you when an application installation fails because of additional permission requirements. 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 the *profile_root/config/cells/cell_name/filter.policy* file. 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.

Therefore, many applications developed for prior releases of WebSphere Application Server might not be Java 2 security ready. To quickly migrate those applications to the latest version of WebSphere Application Server, you might temporarily give those applications the `java.security.AllPermission` permission in the `was.policy` file. Test those applications to ensure that they run in an environment where Java 2 security is active. For example, identify which extra permissions, if any, are required, and grant only those permissions to a particular application. Not granting the `AllPermission` permission to applications can reduce the risk of compromising system integrity. For more information on migrating applications, refer to “Migrating Java 2 security policy” on page 45.

The WebSphere Application Server runtime uses Java 2 security to protect sensitive run-time functions. Applications that are granted the `AllPermission` permission 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 as safe, WebSphere Application Server does support having Java 2 security disabled on a per application server basis. You can enforce Java 2 security by default in the administrative console and clear the Java 2 security flag to disable it at the particular application server.

When you specify the **Enable administrative security** and **Use Java 2 security to restrict application access to local resources** options on the Secure administration, applications, and infrastructure panel of the administrative console, the information and 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.

Attention: The **Enable global security** option in previous releases of WebSphere Application Server is the same as the **Enable administrative security** option in Version 6.1. Also, the **Enable Java 2 security** option in previous releases is the same as the **Use Java 2 security to restrict application access to local resources** option in Version 6.1.

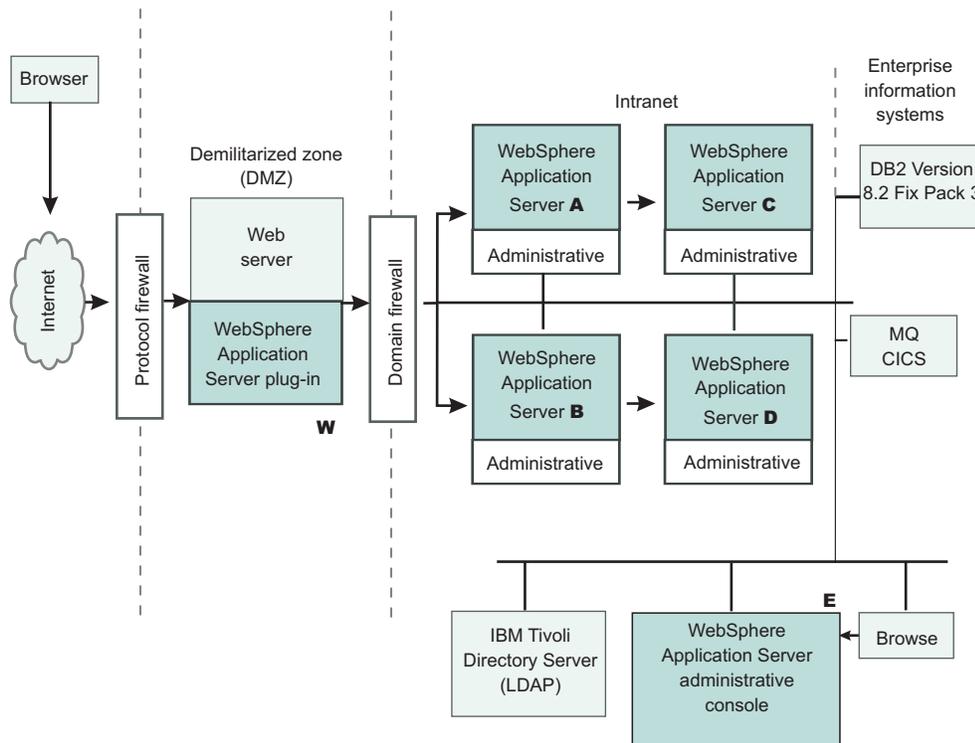
- When Java 2 security is enforced, the application code cannot access the WebSphere Application Server run-time classes that manage the configuration data unless the code is granted the required WebSphere Application Server run-time permissions.
- When Java 2 security is enforced, application code cannot access the WebSphere Application Server configuration XML files unless the code is granted the required file read and write permission.

- The JMX administrative subsystem provides SOAP over HTTP or HTTPS and a RMI/IIOP remote interface to enable application programs to extract and to modify configuration files and data. When administrative 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, the user can also modify the WebSphere Application Server configuration, including the WebSphere Application Server security identity and authentication data 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 administrative security, change server IDs and passwords, and 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 administrative security and to use Java 2 security to restrict application access to local resources. 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 set up 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 certificate, you can restrict the WebSphere Application Server plug-in to communicate with only the selected two WebSphere Application Servers as shown in the following figure. Note that you can use self-signed certificates to reduce administration and cost.



For example, 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**. To complete this task, you can generate three certificates using the IKEYMAN and the certificate management utilities. Also, you can use certificate **W** and trust certificate **A** and **B**. Configure the HTTPS server of WebSphere Application Server **A** to use certificate **A** and to trust certificate **W**. Configure the HTTPS server of WebSphere Application Server **B** to use certificate **B** and to trust certificate **W**. For more information on IKEYMAN, refer to .

The trust relationship that is depicted in the previous figure 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 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 certificates so that a password is not visible when it is passed from WebSphere Application Server to the LDAP server.

WebSphere Application Server does not provide a registry configuration or management utility. In addition, it does not dictate the registry password policy. It is recommended that you use the password policy recommended by your registry, including the password length and expiration period.

Before securing your WebSphere Application Server environment, determine which versions of WebSphere Application Server you are using, review the WebSphere Application Server security architecture, and review each of the following topics:

- “Authentication protocol for EJB security” on page 212
 - “Supported authentication protocols” on page 215
 - “Common Secure Interoperability Version 2 features” on page 216
 - “Identity assertion” on page 216
- “Authentication mechanisms” on page 173
 - “Lightweight Third Party Authentication” on page 178
 - “Trust associations” on page 180
 - “Single sign-on” on page 184
- “User registries and repositories” on page 89
 - “Local operating system registries” on page 165
 - “Standalone Lightweight Directory Access Protocol registries” on page 168
- “Java 2 security” on page 64
 - “Java 2 security policy files” on page 68
- “Java Authentication and Authorization Service” on page 312
 - “Programmatic login” on page 343
- J2EE connector security
 - “Access control exception” on page 73
 - “Role-based authorization” on page 324
 - “Administrative roles and naming service authorization” on page 320

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

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

Legend for "How do I?..." links

Detailed steps	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+

Chapter 3. Task overview: Securing resources

WebSphere Application Server supports the Java 2 Platform, Enterprise Edition (J2EE) model for creating, assembling, securing, and deploying applications. Applications are often created, assembled, and deployed in different phases and by different teams.

You can secure resources in a J2EE environment by following the required high-level steps. Consult the J2EE specifications for complete details.

- Set up and enable security. You must address several issues prior to authenticating users, authorizing access to resources, securing applications, and securing communications. These security issues include migration, interoperability, and installation. After installing WebSphere Application Server, you must determine the proper level of security that is needed for your environment. For more information, see Chapter 4, “Setting up and enabling security,” on page 33.
- Authenticate users. The process of authenticating users involves a user registry and an authentication mechanism. Optionally, you can define trust between WebSphere Application Server and a proxy server, configure single sign-on capability, and specify how to propagate security attributes between application servers. For more information, see Chapter 5, “Authenticating users,” on page 87.
- Authorize access to resources. WebSphere Application Server provides many different methods for authorizing accessing resources. For example, you can assign roles to users and configure a built-in or external authorization provider. For more information, see Chapter 6, “Authorizing access to resources,” on page 319.
- Secure communications. WebSphere Application Server provides several methods to secure communication between a server and a client. For more information, see Chapter 7, “Securing communications,” on page 381.
- Develop extensions to the WebSphere security infrastructure. WebSphere Application Server provides various plug points so that you can extend the security infrastructure. For more information, see Chapter 8, “Developing extensions to the WebSphere security infrastructure,” on page 501.
- Secure various types of WebSphere applications. See **Securing WebSphere applications** for tasks involving developing, deploying, and administering secure applications, including Web applications, Web services, and many other types. This section highlights the security concerns and tasks that are specific to each type of application.
- Tune, harden, and maintain security configurations. After you have installed WebSphere Application Server, there are several considerations for tuning, strengthening, and maintaining your security configuration. For more information, see Chapter 19, “Tuning, hardening, and maintaining,” on page 1377.
- Troubleshoot security configurations. For more information, see Chapter 20, “Troubleshooting security configurations,” on page 1387.

Your applications and production environment are secured.

See *Security: Resources* for learning for more information on the WebSphere Application Server security architecture.

Chapter 4. Setting up and enabling security

You must address several issues prior to authenticating users, authorizing access to resources, securing applications, and securing communications. These security issues include migration, interoperability, and installation.

After installing WebSphere Application Server, you can determine the proper level of security that is needed for your environment. By default, administrative security is enabled and provides the authentication of users, the use of Secure Sockets Layer (SSL), and the choice of user account repository.

The following information is covered in this section:

- Determine if any migration and interoperability issues might affect your installation. For more information, see “Migrating, coexisting, and interoperating – Security considerations.”
- Prepare your environment before and after installing WebSphere Application Server. For more information, see “Preparing for security at installation time” on page 48.
- Enable security for all your application servers or for specific application servers in your realm. For more information, see “Enabling security” on page 50.

After installing WebSphere Application Server and securing your environment, you must authenticate users. For more information, see Chapter 5, “Authenticating users,” on page 87.

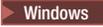
Migrating, coexisting, and interoperating – Security considerations

Use this topic to migrate the security configuration of previous WebSphere Application Server releases and its applications to the new installation of WebSphere Application Server.

This information addresses the need to migrate your security configurations from a previous release of IBM WebSphere Application Server to WebSphere Application Server Version 6.1 or later. Complete the following steps to migrate your security configurations:

- If security is enabled in the previous release, obtain the administrative server ID and password of the previous release. This information is needed in order to run certain migration jobs.
- You can optionally disable security in the previous release before migrating the installation. No logon is required during the installation.

Use the First steps wizard to access and run the Migration wizard.

1. Start the First steps wizard by launching the `firststeps.bat` or the `firststeps.sh` file. The first steps file is located in the following directory:
 -  `./app_server_root/profiles/profile_name/firststeps/firststeps.sh`
 -  `app_server_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 to migrate product configurations.

The security configuration of previous WebSphere Application Server releases and its applications are migrated to the new installation of WebSphere Application Server Version 6.1.

If a custom user registry is used in the previous version, the migration process does not migrate the class files that are used by the standalone custom registry in the previous `app_server_root/classes` directory. Therefore, after migration, copy your custom user registry implementation classes to the `app_server_root/classes` directory.

If you upgrade from WebSphere Application Server, Version 5.x to WebSphere Application Server, Version 6.1, the data that is associated with Version 5.x trust associations is not automatically migrated to Version 6.1. To migrate trust associations, see “Migrating trust association interceptors” on page 40.

Interoperating with previous product versions

IBM WebSphere Application Server inter-operates with the previous product versions. Use this topic to configure this behavior.

1. Configure WebSphere Application Server Version 6.1 with the same distributed user registry (that is, LDAP or Custom) that is configured with the previous version. Make sure that the same LDAP user registry is shared by all of the product versions.
 - a. In the administrative console, select **Security > Secure administration, applications, and infrastructure**.
 - b. Choose an available Realm definition and click **Configure**.
 - c. Enter a **Primary administrative user name**. This is the identity you use to login to the administrative console or to wsadmin. When inter-operating with a previous release, you must use the same server ID. In WebSphere Application Server Version 6.1, there are two choices for server ID. For previous releases, specify a server ID and password. However, if you choose to use the internal server ID, you must be able to override the generated value and to specify the server ID from a previous release.
 - d. Click either **Automatically generated server identity** or **Server identity that is stored in the user repository**.
 - e. If you select **Automatically generated server identity**, click **Authentication mechanisms and expiration** to change the identity to the one used by the previous release you are inter-operating with. Scroll down to the Cross-cell single sign-on section and enter the identity in **Internal server ID**.
 - f. If you select **Server identity that is stored in the user repository**, enter the **Server user id** and the associated **Password**.
 - g. Fill out the rest of the user registry settings and then click **OK**.
2. Configure the LTPA authentication mechanism. Automatic generation of the LTPA keys should be disabled. If not, keys used by a previous release are lost. Export the current LTPA keys from WebSphere Application Server Version 6.1 and import them into the previous release.
 - a. In the administrative console select **Security > Secure administration, applications, and infrastructure**.
 - b. Click **Authentication mechanisms and expiration**.
 - c. Click the **Key set groups** link, then click the key set group that displays in the Key set groups panel.
 - d. Clear the **Automatically generate keys** check box.
 - e. Click **OK**, then click **Authentication mechanisms and expiration** in the path at the top of the Key set groups panel.
 - f. Scroll down to the Cross-cell single sign-on section, and enter a password to use for encrypting the LTPA keys when adding them to the file.
 - g. Enter the password again to confirm the password.
 - h. Enter the **Fully qualified key file name** that contains the exported keys.
 - i. Click **Export keys**.
 - j. Follow the instructions provided in the previous release to import the exported LTPA keys into that configuration.
3. If you are using the default SSL configuration, extract all of the signer certificates from the WebSphere Application Server Version 6.1 common trust store. Otherwise, extract signers where necessary to import them into the previous release.
 - a. In the administrative console, click **Security > SSL certificate and key management**.

- b. Click **Key stores and certificates**.
 - c. Click **NodeDefaultTrustStore**.
 - d. Click **Signer certificates**.
 - e. Select one signer and click **Extract**.
 - f. Enter a unique path and filename for the signer (for example, c:\temp\signer1.arm).
 - g. Click **OK**. Repeat for all of the signers in the trust store.
 - h. Check other trust stores for other signers that might need to be shared with the other server. Repeat steps e through h to extract the other signers.
4. Add the exported signers to DummyServerTrustFile.jks and DummyClientTrustFile.jks in the /etc directory of the back-level product version. If the previous release is not using the dummy certificate, the signer certificate(s) from the previous release must be extracted and added into the WebSphere Application Server Version 6.1 release to enable SSL connectivity in both directions.
 - a. Open the key management utility, iKeyman, for that product version.
 - b. Start ikeyman.bat or ikeyman.sh from the \${USER_INSTALL_ROOT}/bin directory.
 - c. Select **Key Database File > Open**.
 - d. Open \${USER_INSTALL_ROOT}/etc/DummyServerTrustFile.jks.
 - e. Enter WebAS for the password.
 - f. Select **Add** and enter one of the files extracted in step 2. Continue until you have added all of the signers.
 - g. Repeat steps c through f for the DummyClientTrustFile.jks file.
 5. Verify that the application uses the correct Java Naming and Directory Interface (JNDI) name and naming bootstrap port for performing a naming lookup.
 6. Stop and restart all of the servers.

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

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 a client certificate to achieve mutual authentication on WebSphere Application Server applications.

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 (CSlv2) authentication protocol over Remote Method Invocation over the Internet Inter-ORB Protocol (RMI-IIOP). The CSlv2 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 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 a client certificate to achieve mutual authentication on WebSphere Application Server 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 presented in the following list in the file:

C++ security setting	Description
client_protocol_password	Specifies the password for the user ID.

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

- Point to the environment file using the fully qualified path name through the `WAS_CONFIG_FILE` environment variable. For example, in the `test.sh` test shell script, export:

```
/WebSphere/V6R0M0/DeploymentManager/profiles/default/config/cells
/PLEX1Network/nodes/PLEX1Manager/servers/dmgr
```

Some of the environment file terms are explained below:

default

profile name

PLEX1Network

cell name

PLEX1Manager

node name

dmgr server name

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

- 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.
- 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 `keyman.bat` or `keyman.sh` from WebSphere Application Server installation.

- Configure WebSphere Application Server to support Secure Sockets Layer (SSL) as the authentication mechanism.
 - Start the administrative console.
 - 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:
 - Click **Security > Secure administration, applications, and infrastructure**. Under RMI/IIOP security, click **CSlv2 inbound authentication**. Select **Supported** for the Basic authentication and Client certificate authentication options. Leave the rest of the options as defaults.
 - Click **Security > Secure administration, applications, and infrastructure**. Under RMI/IIOP security, click **CSlv2 inbound transport** and verify that the **SSL-supported** option is selected.
If it is a Network Deployment setting, complete the following steps:
 - Click **Servers > Application Servers > server_name_where_the_EJB_resides**.
 - Under Security, click **Server security**.
 - Select the **RMI/IIOP security for this server overrides cell settings** option.
 - Under Additional properties, click **CSlv2 inbound authentication**.

- 5) Select **Supported** for the Basic authentication and Client certificate authentication options. Leave the rest of the options as defaults.
 - 6) Click **Servers > Application Servers > *server_name_where_the_EJB_resides***.
 - 7) Under Security, click **Server security**.
 - 8) Under Additional properties, click **CSlv2 inbound transport**.
 - 9) Verify that the **SSL-Supported** option is selected.
- 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.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 exists.

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

Migrating custom user registries

If you built your own custom user registry, consider the migration items listed below. If you have a custom user registry that was provided by a Security Solution Provider, you must contact that provider to ensure that you have the correct version of their custom user registry to support WebSphere Application Server.

In WebSphere Application Server, 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.

You cannot use other WebSphere Application Server components, for example, data sources, to initialize the custom registry because other components, like the containers, are initialized after security and are not available during the registry initialization. A custom registry implementation is a pure custom implementation, independent of other WebSphere Application Server components.

The `getCallerPrincipal` enterprise bean method and the `getUserPrincipal` and `getRemoteUser` servlet methods return the security name instead of the display name. For more information, see the API documentation.

If the migration tool is used to migrate the WebSphere Application Server Version 5 configuration to WebSphere Application Server Version 6.0.x and later, this migration does not change your existing code. Because the WebSphere Application Server Version 5 custom registry works in WebSphere Application Server Version 6.0.x and later without any changes to the implementation, except when using data sources, you can use the Version 5-based custom registry after the migration without modifying the code.

In WebSphere Application Server Version 6.0.x and later, a case-insensitive authorization can occur when using an enabled custom user registry.

Setting this flag does not have any effect on the user names or passwords. Only the unique IDs that are 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 runtime.

Before proceeding, look at the UserRegistry interface. See “Developing standalone custom registries” on page 501 for a description of each of these methods in detail.

The following steps go through all the changes that are required to move your WebSphere Application Server Version 4.x custom user registry that implemented the old `com.ibm.websphere.security.CustomRegistry` interface to the `com.ibm.websphere.security.UserRegistry` interface.

Note: The sample implementation file is used as an **example** when describing the following steps.

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

```
public class FileRegistrySample implements CustomRegistry
```

to:

```
public class FileRegistrySample implements UserRegistry
```

2. Create the `java.rmi.RemoteException` exception 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 interested only in the first certificate, take the first certificate in the chain before processing. In WebSphere Application Server Version 6.0.x and later, 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.

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 that is obtained from the user 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` exception. Change the following code:

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

to:

```

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

```

In WebSphere Application Server Version 6.0.x and later, this method is not called directly by the WebSphere Application Server security component. However, other components of WebSphere Application Server, like the WebSphere Business Integration Server Foundation process choreographer, use this method when staff assignments are modeled using groups. Because this implementation is supported in WebSphere Application Server Version 6.0.x and later, 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 code:

```
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 that is obtained from the user registry whose number is limited to the value of the `limit` parameter. 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 are split onto two lines for illustrative purposes only.

11. To build the WebSphere Application Server Version 6.0.x and later implementation, make sure you have the `sas.jar` and the `wssec.jar` files in your class path.

To set the files in your class path, use the following code as a sample and substitute your environment values for the variables that are used in the example:

```

%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.

To build the WebSphere Application Server Version 5 custom registry (`CustomRegistry`) in WebSphere Application Server Version 6.0.x and later, only the `sas.jar` file is required.

12. Copy the implementation classes to the product class path.
The `%install_root%/lib/ext` directory is the preferred location.
13. Use the administrative console to set up the custom registry.
Follow the instructions in “Configuring standalone custom registries” on page 111 to set up the custom registry, including the **ignore case for authorization** option. Make sure that you add the `WAS_UseDisplayName` properties if required.

WebSphere Application Server Version 4.x based custom user registry that implemented the old `com.ibm.websphere.security.CustomRegistry` interface is migrated to the `com.ibm.websphere.security.UserRegistry` interface.

If you are enabling security, see “Enabling security” on page 50 to complete the remaining steps. When completed, save the configuration and restart all the servers. Try accessing some Java 2 Platform, Enterprise Edition (J2EE) resources to verify that the custom registry migration is successful.

Migrating trust association interceptors

Use this topic to manually migrate trust associations.

Note: Data sources are not supported for use within a Trust Association Interceptor (TAI). Data sources are intended for use within J2EE applications and designed to operate within the EJB and Web containers. Trust Association Interceptors do not run within a container, and while data sources may function in the TAI environment, they are untested and not guaranteed to function properly.

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 `com.ibm.websphere.security.webseal.ignoreProxy` property is 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 that are 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),  
HTTP/1.1 webseal1:7002, 1.1 webseal2:7001
```

and the `com.ibm.websphere.security.webseal.ignoreProxy` property is set to `true` or `yes`, the host name `Fred`, is not used when matching the host names. By default, this property is not set, which implies that any proxy host names and ports that are expected in the VIA header are listed in the host names and the ports properties to satisfy the `isTargetInterceptor` method.

The previous VIA header information was split onto two lines for illustrative purposes only.

For more information about the `com.ibm.websphere.security.webseal.ignoreProxy` property, see the article in the information center on configuring single signon using trust association interceptor ++.

Migrating product-provided trust association interceptors

The properties that are located in the `webseal.properties` and `trustedserver.properties` files are not migrated from previous versions of WebSphere Application Server. You must migrate the appropriate properties to WebSphere Application Server Version 6.0.x 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 the `com.ibm.websphere.security.WebSphereBaseTrustAssociationInterceptor` property, 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 that the interceptor is 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 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 that is required to initialize the interceptor. All of the properties set for an interceptor are 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 Zero (0) implies that the interceptor initialization is successful. Any other value implies that the initialization is not successful and the interceptor is not used.

The `init(String)` method still works if you want to use it instead of implementing the `init(Properties)` method. The only requirement is that you enter the file name containing the custom trust association properties 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 `.config` to the `com.ibm.websphere.security.trustassociation.types` property value. If the `myTAI.properties` file is located in the `app_server_root/properties` directory, set the following properties:

- `com.ibm.websphere.security.trustassociation.types = myTAItype`
- `com.ibm.websphere.security.trustassociation.myTAItype.config = app_server_root/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=  
app_server_root/properties/myTAI.properties
```

The previous line of code is split into two lines for illustrative purposes only. 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 the `init (String propsfile)` method.

Migrating custom trust association interceptors

The trust associations from previous versions of WebSphere Application Server are not automatically migrated to WebSphere Application Server Version 6.0.x and later. You 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 code:

```
%WAS_HOME%/java/bin/javac -classpath %WAS_HOME%/lib/wssec.jar;  
%WAS_HOME%/lib/j2ee.jar your_implementation_file.java
```

The previous line of code is broken into two lines for illustrative purposes only. Type the code as one continuous line.

2. Copy the custom trust association interceptor class files to a location in your product class path. Copy these class files into the %WAS_HOME%/lib/ext directory.
3. Start WebSphere Application Server.
4. Enable security to use the trust association interceptor. The properties that are located in your custom trust association properties file and in the trustedserver.properties file are not migrated from previous versions of WebSphere Application Server. You must migrate the appropriate properties to WebSphere Application Server Version 6.0.x and later using the trust association panels in the administrative console.

For more information, see Configuring trust association interceptors.

Migrating Common Object Request Broker Architecture programmatic login to Java Authentication and Authorization Service (CORBA and JAAS)

Use this topic as an example of how to perform programmatic login using the CORBA-based programmatic login APIs.

This document outlines the deprecated Common Object Request Broker Architecture (CORBA) programmatic login APIs and the alternatives that are provided by JAAS. WebSphere Application Server fully supports the Java Authentication and Authorization Service (JAAS) as programmatic login application programming interfaces (API). Refer to the *Securing applications and their environment* PDF for more details on JAAS support.

The following list includes the deprecated CORBA programmatic login APIs.

- `${user.install.root}/installedApps/sampleApp.ear/default_app.war/WEB-INF/classes/LoginHelper.java`.
- `${user.install.root}/installedApps/sampleApp.ear/default_app.war/WEB-INF/classes/ServerSideAuthenticator.java`.
- **org.omg.SecurityLevel2.Credentials**. This API is included with the product, but it is not recommended that you use the API.

The APIs that are provided in WebSphere Application Server 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` handler when the application pushes basic

authentication data (user ID, password, and security realm) or token data to product login modules. This API is recommended for server-side login.

com.ibm.websphere.security.auth.callback.WSGUICallbackHandlerImpl

Provides a login prompt CallbackHandler handler to gather basic authentication data (user ID, password, and security realm). This API is recommended for client-side login.

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 a LoginModules implementation for client and server-side login. Refer to the *Securing applications and their environment* PDF 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 interfaces, 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 Application Server 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.

Note: The LoginHelper application programming interface (API) that is used in the following example is deprecated in WebSphere Application Server Version 6.1 and will be removed in a future release. It is recommended that you use the JAAS programmatic login APIs that are shown in the next step.

```
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 the *Securing applications and their environment* PDF 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 WSPPrincipal.
                java.util.Set ps =
                    s.getPrincipals(com.ibm.websphere.security.auth.WSPPrincipal.class);
                java.util.Iterator it = ps.iterator();
                while (it.hasNext()) {
                    com.ibm.websphere.security.auth.WSPPrincipal p =
                        (com.ibm.websphere.security.auth.WSPPrincipal) 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

Use this topic to allow migration in an application that uses form-based login and servlet filters without the use of the CustomLoginServlet class.

The CustomLoginServlet class is deprecated in WebSphere Application Server 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 and 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 that is 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 JavaServer Pages (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 that contains the user name and the j_password parameter that contains 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.
Refer to the *Securing applications and their environment* PDF for details.
2. Configure the form login page and the error page for the application as described in .
Refer to the *Securing applications and their environment* PDF for details.
3. Develop servlet filters if additional processing is required before and after form login authentication.
Refer to the *Securing applications and their environment* PDF for details.
4. Configure the servlet filters that are 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.
For more information, see the *Securing applications and their environment* PDF.

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

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

Migrating Java 2 security policy

Use this topic for guidance pertaining to migrating Java 2 security policy.

Previous WebSphere Application Server releases

WebSphere Application Server uses the Java 2 security manager in the server runtime to prevent enterprise applications from calling the `System.exit` and the `System.setSecurityManager` methods. These two Java application programming interfaces (API) 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 not a beneficial operation for an application server.

To support Java 2 security properly, all the server runtime 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 that are 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 that are required by the server runtime. This situation is due to the design and algorithm that is used by Java 2 security to enforce permission checks. Refer to the Java 2 security check permission algorithm.

The following two permissions are enforced by the Java 2 security manager (hard coded) for WebSphere Application Server:

- `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 runtime is granted these permissions. All the other permission checks are not enforced.

Only two permissions are supported:

- `java.net.SocketPermission`
- `java.net.NetPermission`

However, not all the product server runtime 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 WebSphere Application Server, which means that all permissions are enforced. The default Java 2 security policy for an enterprise application is the recommended permission set defined by the Java 2 Platform, Enterprise Edition (J2EE) Version 1.4 specification. Refer to the `profile_root/config/cells/cell_name/nodes/node_name/app.policy` file for the default Java 2 security policy that is granted to enterprise applications. This policy is a much more stringent compared to previous releases.

All policy is declarative. The product security manager honors all policy that is declared in the policy files. There is an exception to this rule: enterprise applications are denied access to permissions that are declared in the `profile_root/config/cells/cell_name/filter.policy` file.

Note: The default Java 2 security policy for enterprise applications is much more stringent and all the permissions are enforced in WebSphere Application Server Version 6.0.x and later. The security policy might fail because the application code does not have the necessary permissions granted where system resources, such as file I/O, can be programmatically accessed and are now subject to the permission checking.

In application code, do not use the `setSecurityManager` permission to set a security manager. When an application uses the `setSecurityManager` permission, there is a conflict with the internal security manager within WebSphere Application Server. If you must set a security manager in an application for RMI purposes, you also must enable the **Use Java 2 security to restrict application access to local resources** option on the Secure administration, applications, and infrastructure page within the WebSphere Application Server administrative console. WebSphere Application Server then registers a

security manager. The application code can verify that this security manager is registered by using `System.getSecurityManager()` application programming interface (API).

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). This system property contains both system permissions (permissions granted to the Java virtual machine (JVM) and the product server runtime) and enterprise application permissions. Migrate the Java 2 security policy of the enterprise application to WebSphere Application Server Version 6.0.x. 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 system property is deprecated; a flag in the WebSphere configuration application programming interface (API) is used to control whether to enable Java 2 security. Enable this option through the administrative console.
- **was.home.** Expanded to the installation directory of WebSphere Application Server (action might be required). This system property is deprecated; superseded by the `${user.install.root}` and `${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

No easy way exists to migrate the Java policy file to WebSphere Application Server Version 6.0.x and later automatically because of 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 code base is used instead of an absolute code base. This process has many advantages. Grant the permissions that are defined in the `was.policy` to the specific enterprise application only, while permissions in the `app.policy` file apply to all the enterprise applications that run on the node where the `app.policy` file belongs.

Refer to the *Securing applications and their environment* PDF 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 for the `app1.ear` enterprise application and the system permissions, which are permissions that are granted to the Java virtual machine (JVM) and the product server runtime.

The default location for the Java 2 security policy file is `profile_root/properties/java.policy`. Default permissions are omitted for clarity:

```
// For product Samples
grant codeBase "file:${app_server_root}/installedApps/app1.ear/-" {
    permission java.security.SecurityPermission "printIdentity";
    permission java.io.FilePermission "${app_server_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 the file is not present, or update the `was.policy` file for migrated applications in the configuration repository 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 presented on two lines for illustrative purposes only.

The `was.policy` file is located in the `profile_root/config/cells/cell_name/applications/app.ear/deployments/app/META-INF/` directory.

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 the *Securing applications and their environment* PDF.
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 `${user.install.root}/config/cells/cell_name/nodes/node_name/app.policy` file. This validation requires code review, code inspection, application documentation review, and sandbox testing of migrated enterprise applications with Java 2 security enabled in a preproduction 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 preproduction testing of the migrated enterprise application with Java 2 security enabled. Enable trace for the WebSphere Application Server Java 2 security manager in a preproduction testing environment with the following trace string: `com.ibm.ws.security.core.SecurityManager=all=enabled`. This trace function can be helpful in debugging the `AccessControlException` exception that is created 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 that are granted to the classes on the call stack when the exception is created.

For more information, see the *Securing applications and their environment* PDF.

Note: Because the Java 2 security policy is much more stringent compared with previous releases, the administrator or deployer must 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.

Preparing for security at installation time

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

1. Secure your environment before installation. This step describes how to perform WebSphere Application Server installation with proper authority on different platforms. For more information refer to “Securing your environment before installation” on page 49.
2. Prepare the operating system for installation of WebSphere Application Server. This step describes how to prepare the different operating systems for installation of WebSphere Application Server. For more information, see *Preparing the operating system for product installation*.
3. Migrate security configurations from previous releases during installation, when you are prompted to do so. This step describes how to migrate security configurations from a previous release of WebSphere Application Server to WebSphere Application Server Version 6.1. For more information, see *Migrating product configurations*.
4. **Optional:** You can create a profile during install time. If you elect to do so, administrative security is enabled for that profile “out of the box” by default. A panel is displayed during profile creation time and **enabling administrative security** is selected by default. If you elect to keep this as the default, you

must supply an administrative user ID and password. This user ID is created in a federated repository, which is the default user registry when enabling administrative security at profile creation time.

5. Secure your environment after installation. This step provides information on how to protect password information after you install WebSphere Application Server. For more information, see “Securing your environment after installation.”

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. These instructions apply when you plan to use the local operating system as your user registry.

Windows platforms:

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

To add the rights to a user on a Windows 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:
 - Log on as a service

Securing your environment after installation

WebSphere Application Server depends on several configuration files that are 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.

Note: A Kerberos keytab configuration file contains a list of keys that are analogous to user passwords. It is important for hosts to protect their Kerberos keytab files by storing them on the local disk, which makes them readable only by authorized users.

The files in the *app_server_root/profiles/profile_name/config* and *app_server_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 need permissions as well.

The files in the *app_server_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
- sas.server.props
- server.policy

- `ssl.client.props`
- `was.policy`

Windows Secure files on a Windows system:

1. Open the browser for a view of the files and directories on the machine.
2. Locate and right-click the file or the directory that you want to protect.
3. Click **Properties**.
4. Click the **Security** tab.
5. Remove the Everyone entry and any other user or group that you do not want to have access to the file.
6. Add the users who can access the files with the proper permission.

After securing your environment, only the users with 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.

Enabling security

It is helpful to understand security from an infrastructure perspective 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 security. The following sections help you make these decisions.

Read the following article before continuing with the security configuration:

- “Administrative security” on page 53
- Security

After you understand the security components, you can proceed to configure security in WebSphere Application Server.

Note: For WebSphere Application Server Version 6.1, administrative security is enabled by default whenever a new profile is created, either during the initial install when you create a new profile or during post-install when you use the profile creation tooling. You can decide not to enable administrative security during profile creation time by instead enabling security post-profile creation using the administrative console.

1. Start the WebSphere Application Server administrative console.
If security is currently disabled, you are prompted for a user ID. Log in with any user ID. However, if security is currently enabled, you are prompted for both a user ID and a password. Log in with a predefined administrative user ID and password.
2. Click **Security > Secure administration, applications, and infrastructure**. Use the Security Configuration Wizard, which is now available in WebSphere Application Server, Version 6.1, or configure security manually. The configuration order is not important. For more information on manual configuration, see `tsec_authusers.dita`.
3. Configure the user account repository. For more information, see “Selecting a registry or repository” on page 87. On the Secure administration, applications, and infrastructure panel, you can configure user account repositories such as federated repositories, local operating system, standalone Lightweight Directory Access Protocol (LDAP) registry, and standalone custom registry.

Note: You can choose to specify either a server ID and password for interoperability or enable a WebSphere Application Server 6.1 installation to automatically generate an internal server ID. For more information about automatically generating server IDs, see “Local operating system settings” on page 92.

One of the details common to all user registries or repositories is the *Primary administrative user name*. This ID is a member of the chosen repository, but also has special privileges in WebSphere Application Server. The privileges for this ID and the privileges that are associated with the administrative role ID are the same. The Primary administrative user name can access all of the protected administrative methods.

Windows The ID must not be the same name as the machine name of your system because the repository sometimes returns machine-specific information when querying a user of the same name. In standalone LDAP registries, verify that the Primary administrative user name is a member of the repository and not just the LDAP administrative role ID. The entry must be searchable.

The Primary administrative user name does **not** run WebSphere Application Server processes. Rather, 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 local operating system registry, the process ID requires special privileges to call the operating system APIs. The process ID must have the following platform-specific privileges:

- **Windows** **Act as Part of Operating System** privileges
4. Select the **Set as current** option after you configure the user account repository. When you click **Apply** and the Enable administrative security option is set, a verification occurs to see if an administrative user ID has been configured and is present in the active user registry. The administrative user ID can be specified at the active user registry panel or from the console users link. If you do not configure an administrative ID for the active user registry, the validation fails.
 5. **Optional:** You can configure and change your External Authorization provider to either WebSphere Authorization, SAF Authorization, or an external JACC provider. For more information, see and “Enabling an external JACC provider” on page 354. To change the Authorization provider, click **Security > Secure Administration, applications, and infrastructure > External Authorization providers**.
 6. Configure the authentication mechanism.
Configure Lightweight Third-Party Authentication (LTPA), which is the default authentication mechanism, on the Authentication mechanisms and expiration panel. LTPA credentials can be forwarded to other machines. For security reasons, credential expire; however, you can configure the expiration dates on the console. LTPA credentials enable browsers to visit different product servers, which means you do not have to authenticate multiple times. For more information, see “Configuring the Lightweight Third Party Authentication mechanism” on page 219

Note: You can configure Simple WebSphere Authentication Mechanism (SWAM) as your authentication mechanism. However, SWAM is deprecated in WebSphere Application Server Version 6.1 and will be removed in a future release. SWAM credentials are not forwardable to other machines and for that reason do not expire. To use SWAM, select the **Use SWAM-no authenticated communication between servers** option.
 7. **Optional:** Import and export the LTPA keys for cross-cell single Sign-on (SSO) between cells. For more information, see the following articles:
 - “Exporting Lightweight Third Party Authentication keys” on page 223.
 - “Importing Lightweight Third Party Authentication keys” on page 224
 8. Configure the authentication protocol for special security requirements from Java clients, if needed. You can configure Common Secure Interoperability Version 2 (CSlv2) through links on the Secure administration, applications, and infrastructure panel. The Security Authentication Service (SAS) protocol is provided for backwards compatibility with previous product releases, but is deprecated. Links to the SAS protocol panels display on the Secure administration, applications, and infrastructure panel if your environment contains servers that use previous versions of WebSphere Application Server and support the SAS protocol. For details on configuring CSlv2 or SAS, see the article, “Configuring RMI over IIOP” on page 286.

Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

Attention: **V6.0.x** IBM no longer ships or supports the Secure Authentication Service (SAS) IIOp security protocol. It is recommended that you use the Common Secure Interoperability version 2 (CSiv2) protocol.

9. Modify or create a default Secure Sockets Layer (SSL) configuration. 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 registry, Web container and the authentication protocol (CSiv2 and SAS). For more information, see “Creating a Secure Sockets Layer configuration” on page 415. After you modify a configuration or create a new configuration, specify it on the **SSL configurations** panel. To get to the SSL configurations panel, complete the following steps:
 - a. Click **Security > SSL certificate and key management**.
 - b. Under Configuration settings, click **Manage endpoint security configurations > configuration_name**.
 - c. Under Related items, click **SSL configurations**.

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 > Secure administration, applications, and infrastructure**. Under RMI/IIOp security, click **CSiv2 inbound transport**.
- Click **Security > Secure administration, applications, and infrastructure**. Under RMI/IIOp security, click **CSiv2 outbound transport**.
- **V6.0.x** Click **Security > Secure administration, applications, and infrastructure**. Under RMI/IIOp security, click **SAS inbound transport**
- **V6.0.x** Click **Security > Secure administration, applications, and infrastructure**. Under RMI/IIOp security, click **SAS outbound transport**

For the 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 **New** and type sslConfig in the name field, and its value in the Value field.

For the Lightweight Directory Access Protocol (LDAP) SSL transport, you can modify the SSL configuration repertoire aliases by clicking **Security > Secure administration, applications, and infrastructure**. Under User account repository, click the **Available realm definitions** drop-down list, and select **Standalone LDAP registry**.

10. Click **Security > Secure administration, applications, and infrastructure** to configure the rest of the security settings and enable security. For information about these settings, see “Secure administration, applications, and infrastructure settings” on page 75.
11. Validate the completed security configuration by clicking **OK** or **Apply**. If problems occur, they display at the top of the console page in red type.
12. If there are no validation problems, click **Save** to save the settings to a file that the server uses when it restarts. Saving writes the settings to the configuration repository.

Important: If you do not click **Apply** or **OK** in the Secure administration, applications, and infrastructure panel before you click **Save**, your changes are not written to the repository. The server must be restarted for any changes to take effect when you start the administrative console.

13. Start the WebSphere Application Server administrative console.
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 ID is typically the server user ID that is specified when you configured the user registry.

Administrative security

Administrative security 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. Proper planning is required because incorrectly enabling administrative security can lock you out of the administrative console or cause the server to end abnormally.

Administrative security can be thought of as a “big switch” that activates a wide variety of security settings for WebSphere Application Server. Values for these settings can be specified, but they will not take effect until administrative security is activated. The settings include the authentication of users, the use of Secure Sockets Layer (SSL), and the choice of user account repository. In particular, application security, including authentication and role-based authorization, is not enforced unless administrative security is active. Administrative security is enabled by default.

Administrative security represents the security configuration that is effective for the entire security domain. A *security domain* consists of all of the servers that are configured with the same user registry *realm* name. In some cases, the realm can be the machine name of a local operating system registry. In this case, all of the application servers must reside on the same physical machine. In other cases, the realm can be the machine name of a standalone Lightweight Directory Access Protocol (LDAP) registry.

The basic requirement for a security domain is that the access ID that is returned by the registry or repository from one server within the security domain is the same access ID as that returned from the registry or repository 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.

The administrative security configuration applies to every server within the security domain.

Why turn on administrative security?

Turning on administrative security activates the settings that protect your server from unauthorized users. Administrative security is enabled by default during the profile creation time. There might be some environments where no security is needed such as a development system. On these systems you can elect to disable administrative security. However, in most environments you should keep unauthorized users from accessing the administrative console and your business applications. Administrative security must be enabled to restrict access.

What does administrative security protect?

The configuration of administrative security for a security domain involves configuring the following technologies:

- Authentication of HTTP clients
- Authentication of IIOp clients
- Administrative console security
- Naming security
- Use of SSL transports
- Role-based authorization checks of servlets, enterprise beans, and mbeans
- Propagation of identities (RunAs)
- The common user registry
- The authentication mechanism
- Other security information that defines the behavior of a security domain includes:
 - The authentication protocol (Remote Method Invocation over the Internet Inter-ORB Protocol (RMI/IIOp) security)
 - Other miscellaneous attributes

Fine-grained administrative security

In releases prior to WebSphere Application Server version 6.1, users granted administrative roles could administer all of the resource instances under the cell. WebSphere Application Server is now more fine-grained, meaning that access can be granted to each user per resource instance.

For example, users can be granted configurator access to a specific instance of a resource only (an application, an application server or a node). Users cannot access any other resources outside of the resources assigned to them. The administrative roles are now per resource instance rather than to the entire cell. However, there is a cell-wide authorization group for backward compatibility. Users assigned to administrative roles in the cell-wide authorization group can still access all of the resources within the cell.

To achieve this instance-based security or fine-grained security, resources that require the same privileges are placed in a group called the *administrative authorization group* or *authorization group*. Users can be granted access to the authorization group by assigning to them the required administrative role.

Fine-grained administrative security can also be used in single-server environments. Various applications in the single server can be grouped and placed in different authorization groups. Therefore, there are different authorization constraints for different applications. Note that the server itself cannot be part of any authorization group.

The AdminSecurityManager role is available for wsadmin users. When using **wsadmin**, users granted this role can map users to administrative roles. Also, when fine grained admin security is used, users granted this role can manage authorization groups. See “Administrative roles and naming service authorization” on page 320 for detailed explanations of all administrative roles.

There are several administrative security commands that can be used to create authorization groups, map resources to authorization groups, and to assign users to administrative roles within the authorization groups. Following are some examples:

- **Create a new authorization group:**
`$AdminTask createAuthorizationGroup {-authorizationGroupName authGroup1}`
- **Deleting an authorization group:**
`$AdminTask deleteAuthorizationGroup {-authorizationGroupName groupName}`
- **Add resources to an authorization group:**

```
$AdminTask addResourceToAuthorizationGroup
{-authorizationGroupName groupName -resourceName Application=app1}
```

- **Remove resources from an authorization group:**

```
$AdminTask removeResourceFromAuthorizationGroup
{-authorizationGroupName groupName -resourceName Application=app1}
```

- **Add user IDs to roles in an authorization group:**

```
$AdminTask mapUsersToAdminRole {-authorizationGroupName groupName
-roleName administrator -userids user1}
```

- **Add group IDs to roles in an authorization group:**

```
$AdminTask mapGroupsToAdminRole {-authorizationGroupName groupName
-roleName administrator -groupids group1}
```

- **Remove user IDs from roles in an authorization group:**

```
AdminTask removeUsersFromAdminRole {-authorizationGroupName
groupName -roleName administrator -userids user1}
```

- **Remove group IDs from roles in an authorization group:**

```
$AdminTask removeGroupsFromAdminRole {-authorizationGroupName
groupName -roleName administrator -groupids group1}
```

Resources that can be added to an authorization group

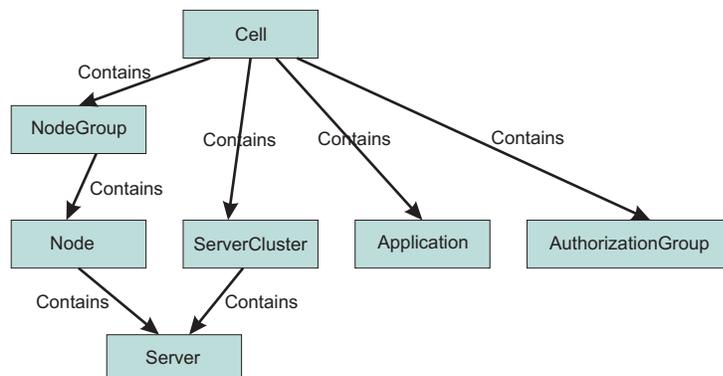
You can add only resource instances of the following types to an authorization group:

- Cell
- Node
- ServerCluster
- Server
- Application
- NodeGroup

If a resource instance is not one of the types listed above, its parent resource will be used.

A resource instance can only belong to one authorization group. However, there is a containment relationship among resource instances. If a parent resource belongs to a different authorization group than that of its child resource instance, the child resource instance implicitly will belong to multiple authorization groups. You cannot add the same resource instance to more than one authorization group.

The following diagram shows the containment relationship among resource instances:



The privileges required for actions on resource instances depend on two factors:

- The authorization group of the administrative resource instance. If a user is granted access to an authorization group, all of the resource instances in that group will be included.

- The containment relationship of the resource instance. If a user is granted access to a parent resource instance, all of the children resource instances will be included.

The privileges required to access various administrative resource instances are shown in the following table:

Resource	Action	Required roles
Server	Start, stop, runtime operations	Server-operator, node-operator, cell-operator
Server	New, delete	Node-configurator, cell-configurator
Server	Edit configuration	Server-configurator, node-configurator, cell-configurator
Server	View configuration, runtime status	Server-monitor, node-monitor, cell-monitor
Node	Restart, stop, sync	Node-operator, Cell-operator
Node	Add, delete	Cell-configurator
Node	Edit configuration	Node-configurator, cell-configurator
Node	View configuration, runtime status	Node-monitor, cell-monitor
Cluster	Start, stop, runtime operations	Cluster-operator, cell-operator
Cluster	New, delete	Cell-configurator
Cluster	Edit configuration	Cluster-configurator, cell-configurator
Cluster	View configuration, runtime status	Cluster-monitor, cell-monitor
Cluster member	Start, stop, runtime operations	Server-operator, cluster-operator, node-operator, cell-operator
Cluster member	New, delete	Node-configurator, cell-configurator
Cluster member	Edit configuration	Server-configurator, cluster-configurator, node-configurator, cell-configurator
Cluster member	View configuration, runtime status	Server-monitor, cluster-monitor, node-monitor, cell-monitor
Application	Start, stop, runtime operations	Application-deployer, cell-operator
Application	Install, uninstall	Cell-configurator application-deployer
Application	Edit configuration	Application-deployer, cell-configurator
Application	View configuration, runtime status	Application-monitor, cell-monitor
Node, cluster	Add, delete	Cell-configurator

The *server-operator* role is the operator role of the authorization group to which the server instance is part of. Similarly, the *node-operator* role is in the operator role of the authorization group to which the node instance is part of.

Fine-grained administrative security is only available for wsadmin users. It is not available for administrative console users. To log in to the administrative console, a user should be granted a monitor role at the cell level at minimum. However, to login using wsadmin, a user should be granted a monitor role for any authorization group.

If you log in to the administrative console as a cell-level administrator, operator, monitor or configurator, you can perform all operations. However, if you want to use additional administrator roles (such as deployer or AdminSecurityManager), or give users access only to specific authorization groups or permissions to non-cell authorizations groups, you must use **wsadmin**.

Fine-grained administrative security in heterogeneous and single-server environments

Fine-grained administrative security can be used in heterogeneous or single-server environments with some restrictions.

Fine-grained administrative security in a heterogeneous environment

Fine-grained administrative security in a heterogeneous environment has the following restrictions:

- Only nodes that are running WebSphere Application Server Version 6.1 can be part of an administrative authorization group.
- Only servers that are running in a WebSphere Application Server Version 6.1 node can be part of an administrative authorization group.
- Only applications that are targeted on servers running on WebSphere Application Server Version 6.1 can be part of an administrative authorization group.
- If a cluster spans nodes of multiple releases, it cannot be part of an administrative authorization group.
- If a cluster spans nodes of multiple releases, none of its members can be part of an administrative authorization group.
- If an application is targeted on a cluster that spans multiple releases, that application cannot be part of an administrative authorization group.

Fine-grained administrative security in a single-server environment

You can also use fine-grained administrative security in a single-server environment. Various applications in the single server can be grouped and placed in different authorization groups. Therefore, different authorization constraints might exist for different applications.

Life cycle of fine-grained administrative resource

An administrative resource that was once part of an authorization group continues to be part of that authorization group until one of the following events occurs:

- The administrative resource is removed from the authorization group. In this instance, the administrative resource belongs to the cell-level authorization group.
- The administrative resource is removed from the configuration. In this instance, the administrative resource does not exist in the configuration, but still exists in the authorization group. Remove this administrative resource from the authorization group.

After the administrative resource is removed from the authorization group, the administrative authorizer runtime must be notified by using the `AuthorizationManager refreshAll` MBean method.

The **refreshAll** command must be invoked after `AdminConfig.save()` and sync nodes. For example:

JACL:

```
// get AuthorizationGroup Mbean
wsadmin> set agBean [$AdminControl queryNames
Type=AuthorizationGroupManager,process=dmgr,*]
```

JYTHON:

```
// get AuthorizationGroup Mbean
wsadmin> set agBean [$AdminControl queryNames
Type=AuthorizationGroupManager,process=dmgr,*]
```

Fine-grained administrative security scenarios

The following scenarios describe the use of fine-grained administrative security, particularly the new deployment role.

Deployment role scenario 1

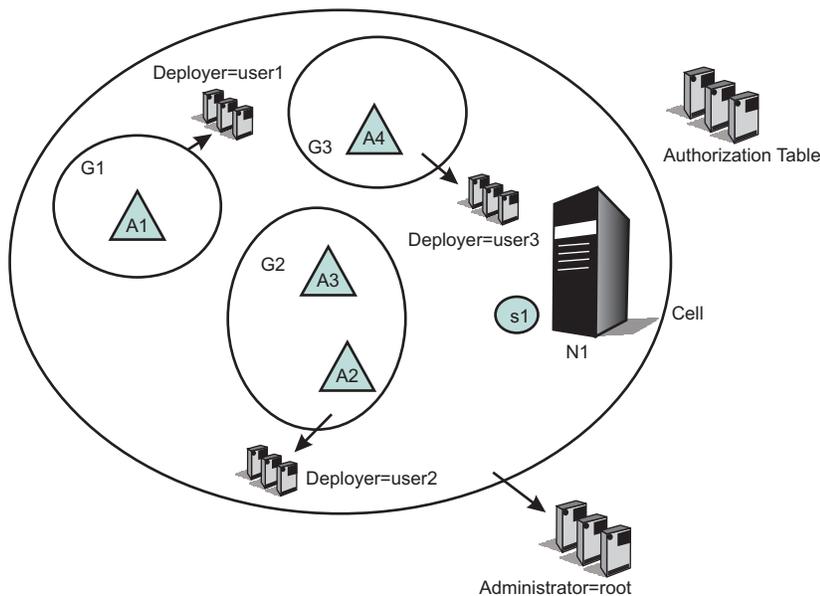
In the following scenario, there are four applications configured on server S1, as shown in the following table. Each application must be isolated so that the administrator of one application cannot modify another application. Assume that only user1 can manage application A1, user2 can manage applications A2 and A3, and only user3 can manage application A4.

Note: It is not recommended to have an application in one group and its target server in another group. However, that is not always possible. It is common to have many applications on one server. It is still sometimes necessary to isolate the administration of applications running on the same server.

One example is an Application Service Provider (ASP), where a single application server can have multiple vendor applications. In this instance the server administrator is responsible for installing all of the vendor applications. Once applications are installed, each vendor can manage their own application without interfering with other vendor's applications.

Application	Server	Node
A1	S1	N1
A2	S1	N1
A3	S1	N1
A4	S1	N1

We can configure authorization groups as shown in the diagram below:



In the diagram, application A1 is in authorization group G1, applications A2 and A3 are in authorization group G2, and application A4 is in authorization group G3.

A deployer role is assigned from authorization group G1 to user1, from authorization group G2 to user2, and from authorization group G3 to user3.

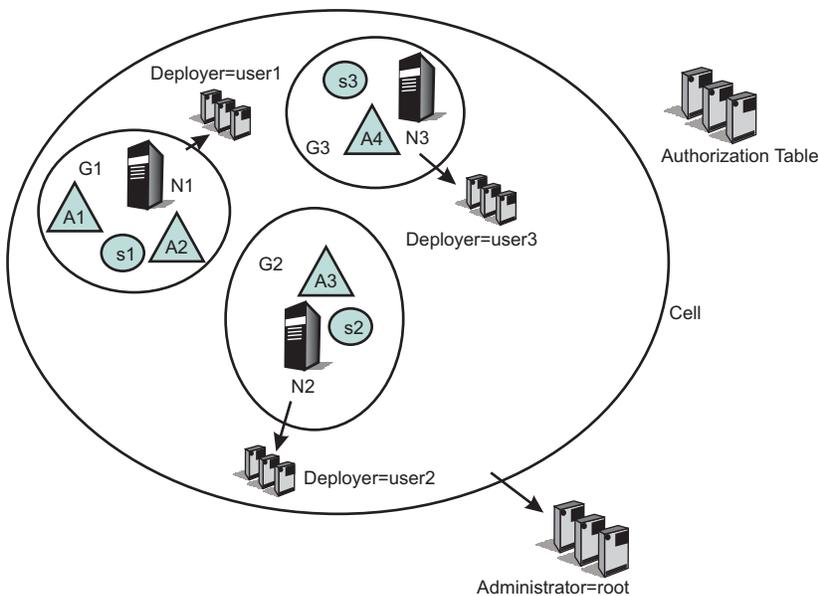
Consequently, user1 can perform all of the operations on application A1, user2 on applications A2 and A3, and user3 on application A4. Since all applications share the same server, we cannot put the same server on all authorization groups. Only a cell-level administrator can install an application. After the installation of an application is complete, the deployer of each application can modify their own. To start and stop the

server, cell-level administrative authority is required. This type of scenario is useful in an ASP environment.

Deployment role scenario 2

In the following scenario, a group of applications require the same administrative roles to one server. In this example, applications A1 and A2 are related applications, and can be administrated by one set of administrators. They are running on the same server (S1). Applications A3 and A4 require a different set of administrators, and are running on servers S2 and S3 respectively.

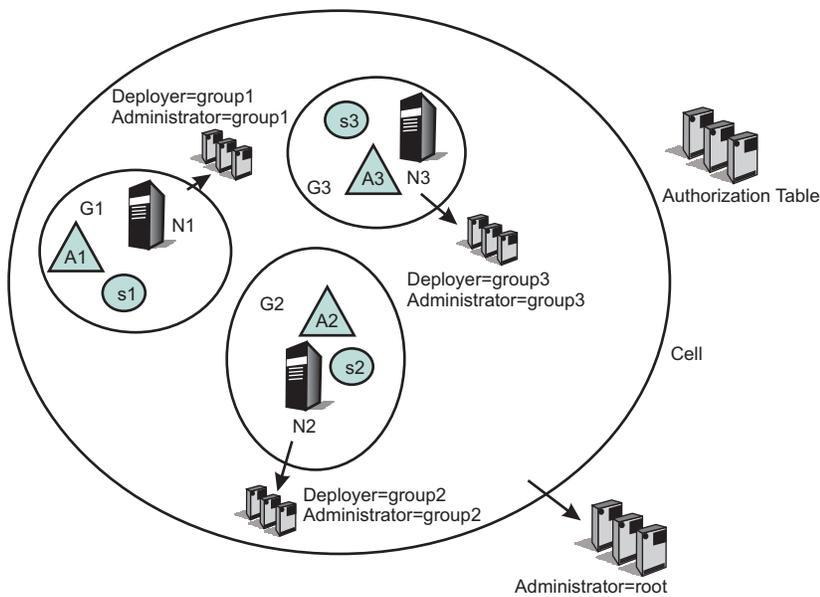
Application	Server	Node
A1	S1	N1
A2	S1	N1
A3	S2	N2
A4	S3	N3



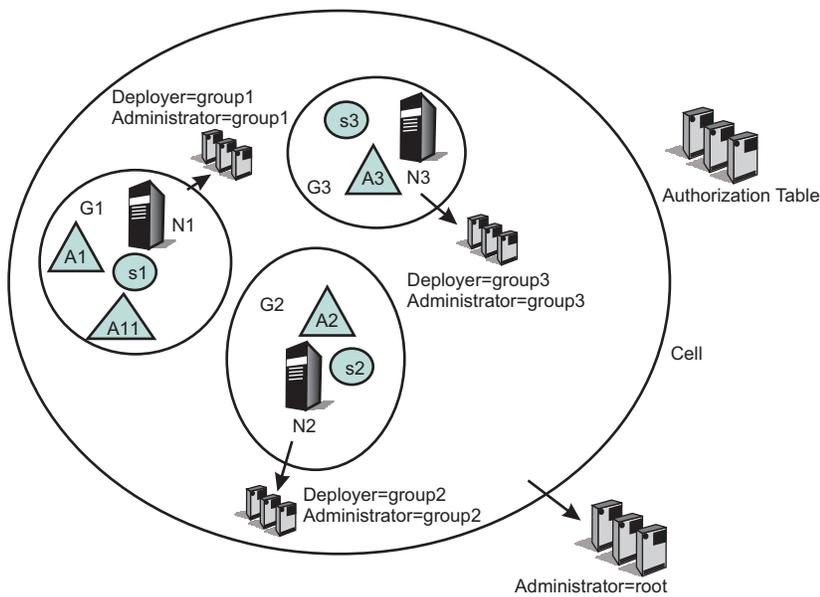
Scenarios that can be applied directly in customer environments

Each developer must be able to modify the configuration for their server, and they must be able to install their application onto that server. They also must be able to start and stop the server as well as the application on the server.

Developers also must be able to configure the server so that they can debug any problems they run into. They must have the ability to update or modify the application being developed. The administrative authorization group for this developer includes at least one server and any applications that the developer installs on that server.



In the following example, developers of authorization group G1 have a new application (A11). They can install and target that new application only on servers within authorization group G1. Also, they can place that new application in their authorization group (G1).



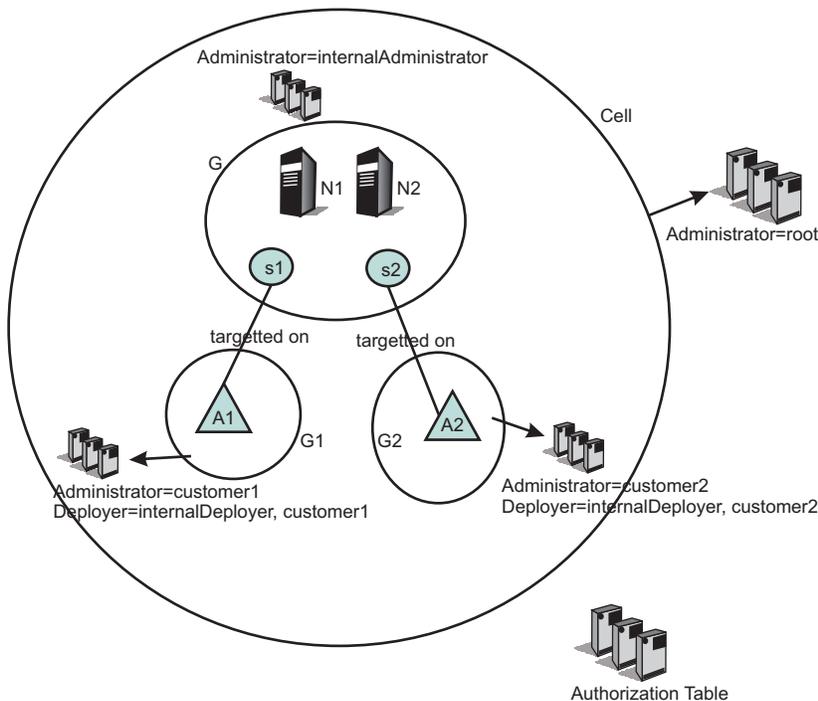
ASP environment scenario

In this scenario, the customer is an ASP. They have their own customers to whom they provide application serving function. They want to enable their customers to administer and monitor their applications, but not to see or administer applications for different customers. In this example, however, the ASP has internal staff administrators whose job it is to maintain the servers.

This internal ASP staff administrator might need to move an application from one server to another to ensure that an application remains available. The internal ASP staff administrator should be able to stop and start the servers and to change their configuration.

In contrast, the ASP customer administrator should not be able to stop or start servers. However, the ASP customer administrator should be able to update their applications running on those servers. The administrative authorization group for the internal ASP administrator can be the whole cell or can include a subset of servers, nodes, clusters and applications. The administrative authorization group for the customer administrator only includes those applications that the customer has paid to have served by this ASP.

The following diagram contains a scenario where two different customers have two different type of applications, and can manage their own applications. However, the servers and nodes on which the applications are running are isolated from their customers. The servers and nodes can only be maintained by the internal administrators. In addition, the customers cannot target their applications on a different server. This can only be performed by the internal administrator or internal deployers.

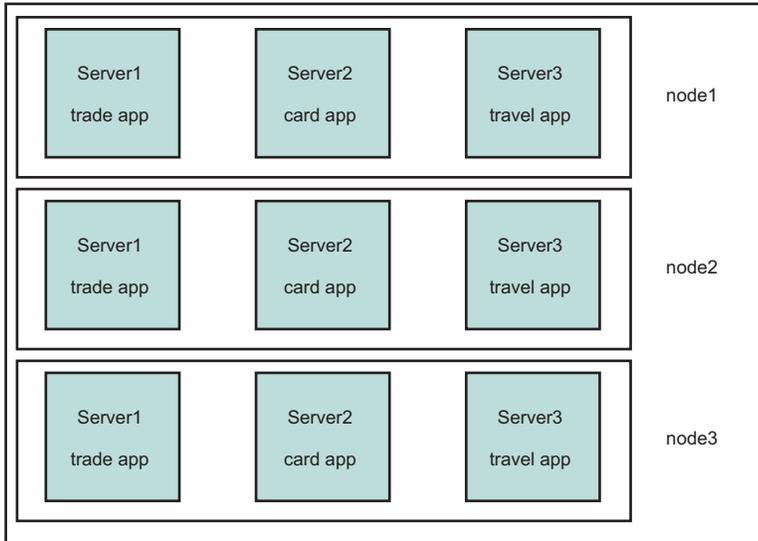


Regional organization scenario

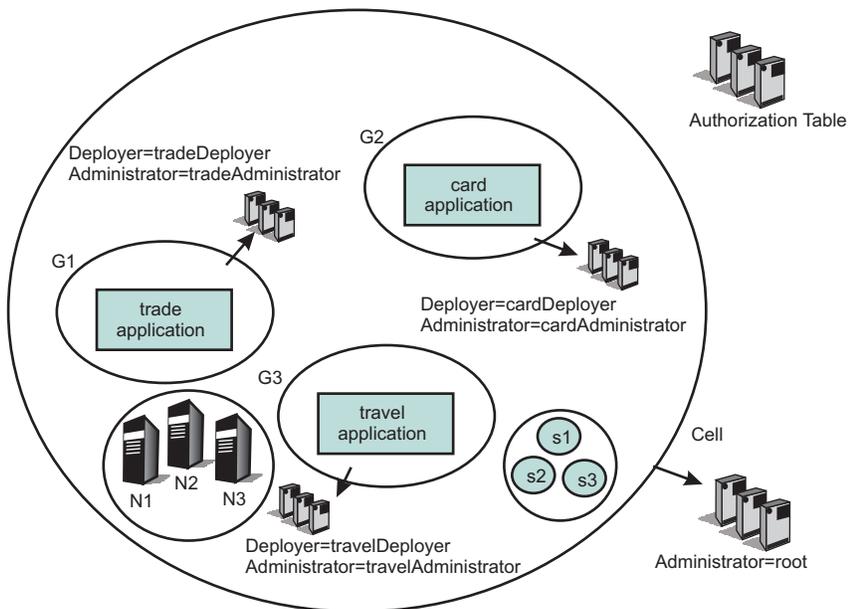
In this scenario, the customer is a large global company. The company's nodes and servers are organized so as to provide application serving for different regions (or alternatively, different lines of business). They want representatives from the different regional areas to be able to monitor and administer the nodes and servers associated with that region. However, they do not want the regional administrator to be able to effect any node and server associated with a different region.

The administrative authorization group for each regional representative includes the nodes, servers, clusters and applications associated with that region.

For example, consider a company that provides multiple services, such as a financial institution that provides services like credit card accounts, brokerage accounts, banking accounts, or travel accounts. Each of these services can be separate applications, and the administrator for each of these applications must also be different. The following figure shows one way to configure such a system:

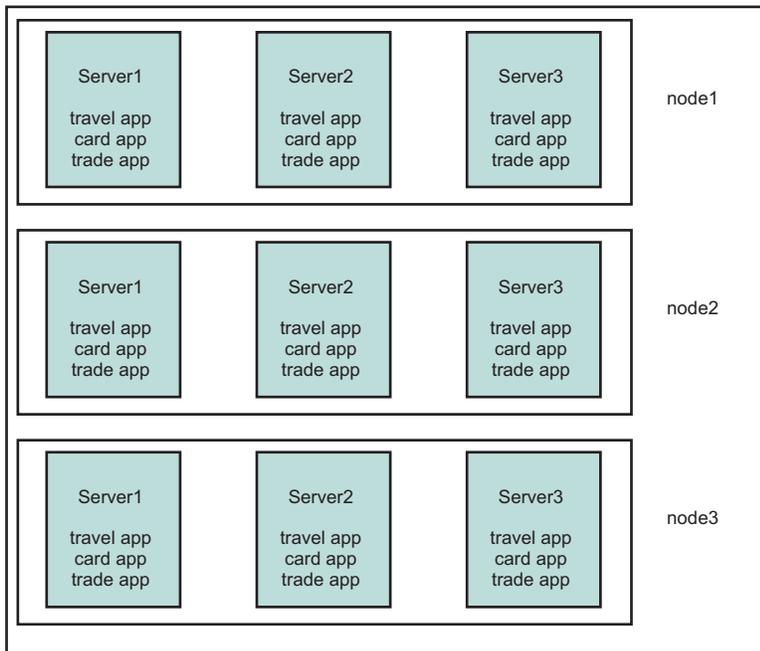


The following figure shows how the resources in such a system can be grouped to isolate administrators from each other:

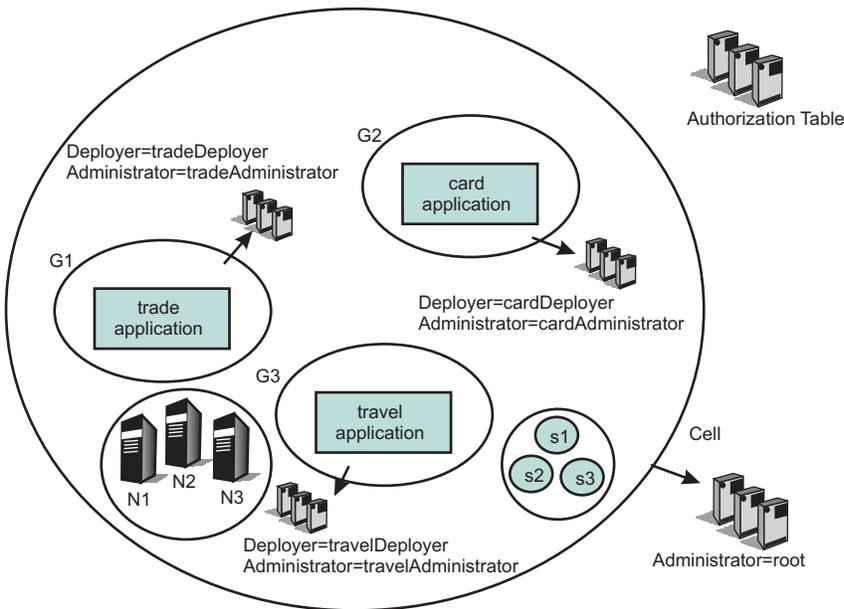


Note that the nodes are not part of any authorization group. Therefore, a trade application administrator cannot stop a server on any of the nodes, and is prevented from stopping a travel application.

The same system can be configured in another way as shown below:



The following figure shows how the resources in such a system can be grouped to isolate administrators from each other:



Application security

Application security enables security for the applications in your environment. This type of security provides application isolation and requirements for authenticating application users

In previous releases of WebSphere Application Server, when a user enabled global security, both administrative and application security were enabled. In WebSphere Application Server Version 6.1, the previous notion of global security is split into administrative security and application security, each of which you can enable separately.

As a result of this split, WebSphere Application Server clients must know whether application security is disabled at the target server. Administrative security is enabled, by default. Application security is disabled, by default. To enable application security, you must enable administrative security. Application security is in effect only when administrative security is enabled.

An Application Server Enablement Tag, which is specific to WebSphere Application Server, is imported into the Interoperable Object Reference (IOR) to indicate if application security is disabled for the server where the object lives. This tag is server-specific and enables clients to know when application security is disabled at the target server of its request.

For Web resources, when application security is enabled, authentication is prompted for lookups on the application server.

For enterprise bean resources, when application security is disabled, the client Common Secure Interoperability version 2 (CSIv2) code ignores the CSIv2 security tags for objects that are unknown system objects. When pure clients see that application security is disabled, these clients prompt for naming lookups, but do not prompt for enterprise bean operations.

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. Java 2 Platform, Enterprise Edition (J2EE) security guards access to Web resources such as servlets, JavaServer Pages (JSP) files and Enterprise JavaBeans (EJB) methods.

WebSphere Application Server security includes the following technologies:

- Java 2 Security Manager
- Java Authentication and Authorization Service (JAAS)
- Java 2 Connector authentication data entries
- **V6.0.x** Common Secure Interoperability Version 2 (CSIv2) or Security Authentication Service (SAS)

Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

- J2EE role-based authorization
- Secure Sockets Layer (SSL) configuration

Because 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 need to 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, it is disabled by default. You can configure Java 2 security and administrative security independently of one another. Disabling administrative 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 system log or trace 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.

Implications exist 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 that are recommended in the J2EE 1.4 Specification. For the details of default permissions granted to applications in the product, refer to the following policy files:

- *app_server_root/java/jre/lib/security/java.policy*
- *app_server_root/properties/server.policy*
- *profile_root/config/cells/cell_name/nodes/node_name/app.policy*

The 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 exceptions are generated from within WebSphere Application Server. The product does not support a more restrictive policy than the default that is defined in the policy files previously mentioned.

Several policy files are used to define the security policy for the Java process. These policy files are static (code base is defined in the policy file) and 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 runtime. Refer to the “Java 2 security policy files” on page 68 for more information.

Syntax errors in the policy files cause the application server process to fail, so edit these policy files carefully.

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 runtime. 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” on page 67 for information and strategies for dealing with such applications.

It is important to note when Java Security is enabled in the administrative security settings, the installed security manager does not currently check *modifyThread* and *modifyThreadGroup* permissions for non-system threads. Allowing Web and Enterprise JavaBeans (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 that are 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 *java.policy* static policy files, the *was.policy* file, 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 needs to document the required Java 2 permissions that are required 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 is never granted access to resources that are protected by the permission, it might be necessary to mark the code as **privileged**. Refer to the, `AccessControlException`, topic for more details. However, improperly inserting a `doPrivileged` call might open up security holes. Understand the implication of `doPrivileged` call to make a correct judgement.

The section on Dynamic policy files in “Java 2 security policy files” on page 68 describes how the permissions in the `was.policy` files are granted at runtime.

Developing an application to use with Java 2 security 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/1.5.0/docs/guide/security/index.html>.

Debugging Aids

The WebSphere Application Server `SYSOUT` file and the `com.ibm.websphere.java2secman.norethrow` property are the two primary aids for debugging.

The WebSphere System Log or Trace Files

The `AccessControl` exception that is logged in the system log or trace files 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 creates 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 `SYSOUT` file.

However, when the Java virtual machine `com.ibm.websphere.java2secman.norethrow` property is set and has a value of `true`, the security manager does not create 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 > *server_name***. Under Server infrastructure, click **Java and Process Management > Process definition**. Under Additional properties, click **Java Virtual Machine > Custom Properties > New**. In the **Name** field, type `com.ibm.websphere.java2secman.norethrow`. In the **Value** field, type `true`.

This property is intended for a sandbox or debug environment because it instructs the security manager not to create the `AccessControl` exception. Java 2 security is not enforced. Do not use this property 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 where the system log or trace files can be inspected for AccessControl exceptions. Because this property does not create 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 Application Server 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, no easy way exists 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 that is embedded in the application `.ear` file, for example:

```
grant codeBase "file:${application}" {
    permission java.security.AllPermission;
};
```

The server.policy file

The `server.policy` file is located in the `app_server_root/properties/` directory.

This policy defines the policy for the WebSphere Application Server 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 policy file as the value of the `java.security.policy` Java system properties. 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 the `was.policy` file (per enterprise application) to define Java 2 security policy for enterprise application resources.

The java.policy file

The file represents the default permissions that are granted to all classes. The policy of this file applies to all the processes launched by the Java Virtual Machine in the WebSphere Application Server.

The `java.policy` file is located in the `app_server_root/java/jre/lib/security` directory.

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 method reported a security exception 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 a Reliability Availability Service Ability (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. After 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.

If the application is running with Java Mail, this message might be benign. You 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.

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.

Java 2 security policy files

The Java 2 Platform, Enterprise Edition (J2EE) Version 1.3 and later specifications have 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	<i>app_server_root/java/jre/lib/security/java.policy</i> . Default permissions are granted to all classes. The policy of this file applies to all the processes launched by WebSphere Application Server.
server.policy	<i>profile_root/properties/server.policy</i> . Default permissions are granted to all the product servers.
client.policy	<i>profile_root/properties/client.policy</i> . Default permissions are granted 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	<i>profile_root/config/cells/cell_name/nodes/node_name/spi.policy</i> 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 Java database connectivity (JDBC) drivers. The code base for the embedded resources are dynamically determined from the configuration (<i>resources.xml</i> file) and run-time data, and permissions that are defined in the <i>spi.policy</i> files are automatically applied to these resources and JAR files that are specified in the class path of a resource adapter. The default permission of the <i>spi.policy</i> file is <i>java.security.AllPermissions</i> .
library.policy	<i>profile_root/config/cells/cell_name/nodes/node_name/library.policy</i> 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 <i>library.policy</i> file is empty.
app.policy	<i>profile_root/config/cells/cell_name/nodes/node_name/app.policy</i> The <i>app.policy</i> file defines the default permissions that are granted to all of the enterprise applications running on <i>node_name</i> in <i>cell_name</i> .
was.policy	<i>profile_root/config/cells/cell_name/applications/ear_file_name/deployments/application_name/META-INF/was.policy</i> This template is for application-specific permissions. The <i>was.policy</i> file is embedded in the enterprise archive (EAR) file.
ra.xml	<i>rar_file_name/META-INF/was.policy.RAR</i> . This file can have a permission specification that is defined in the <i>ra.xml</i> file. The <i>ra.xml</i> file is embedded in the RAR file.

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
<code>file:\${application}</code>	Permissions apply to all the resources within the application
<code>file:\${jars}</code>	Permissions apply to all the utility Java archive (JAR) files within the application
<code>file:\${ejbComponent}</code>	Permissions apply to the Enterprise JavaBeans (EJB) resources within the application
<code>file:\${webComponent}</code>	Permissions apply to the Web resources within the application
<code>file:\${connectorComponent}</code>	Permissions apply to the connector resources within the application

You can specify the module name for a granular setting, except for these entries that are specified by the code base symbols. 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 the resources within the application
file:\${jars}	Permissions apply to all the utility JAR files within the application
file:\${ejbComponent}	Permissions apply to the enterprise beans resources within the application
file:\${webComponent}	Permissions apply to the Web resources within the application
file:\${connectorComponent}	Permissions apply to the connector resources both within the application and in the standalone 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

Attention: 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 runtime, 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 filterMask keyword, the runtime 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 `runtimeFilterMask` keyword, the runtime 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 the `runtimeFilterMask` permission are removed from the granted permission during runtime.

If the **Warn if applications are granted custom permissions** option on the Secure administration, applications, and infrastructure 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 (`app_server_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 the IBM Developer Kit, Java Technology Edition Version 5. 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 that is loaded by the IBM Developer Kit, Java Technology Edition policy tool Version 5, 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 a 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.

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;
};
```

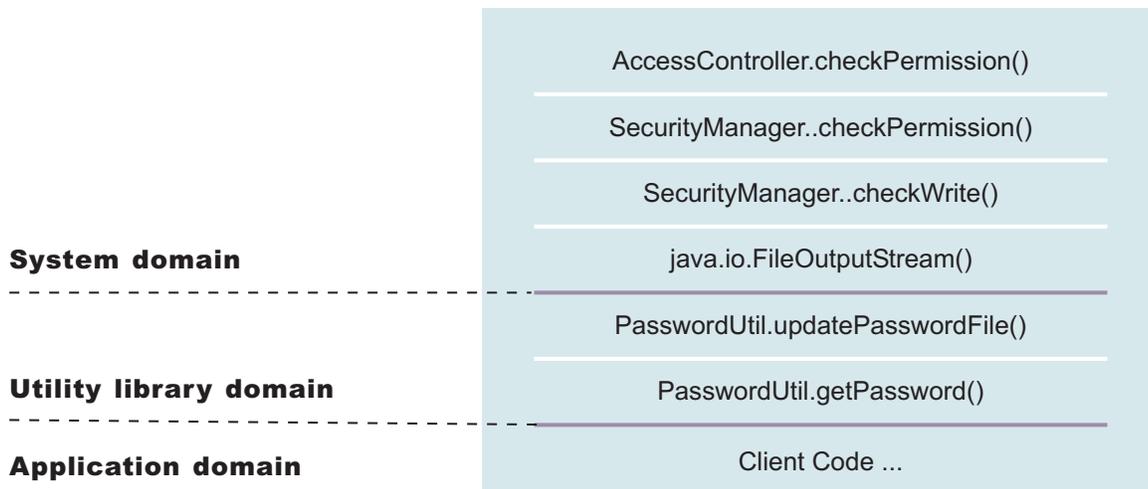
The algorithm requires that all the classes or callers on the call stack have the permissions when a `java.security.AccessController.checkPermission` method is performed or the request is denied and a `java.security.AccessControlException` exception is created. However, if the caller is marked as *privileged* 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 application programming interface (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.



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 the `PasswordUtil.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.

Enabling security for the realm

Use this topic to enable IBM WebSphere Application Server security. You must enable administrative security for all other security settings to function.

WebSphere Application Server uses cryptography to protect sensitive data and to ensure confidentiality 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 are configured for the Web Services application.

WebSphere Application Server uses Java Secure Sockets Extension (JSSE) and Java Cryptography Extension (JCE) libraries in the Software Development Kit (SDK) to perform this cryptography. The SDK provides strong but limited jurisdiction policy files. Unrestricted policy files provide the ability to perform full strength cryptography and to improve performance.

WebSphere Application Server provides a SDK 5 that contains strong, but limited jurisdiction policy files. You can download the unrestricted policy files from the following Web site: IBM developer kit: Security information. Complete the following steps to download and install the new policy files:

1. Click **J2SE 5.0**
2. Scroll down the page then click **IBM SDK Policy files**.
The Unrestricted JCE Policy files for SDK 5 Web site displays.
3. Click **Sign in** and provide your IBM.com ID and password.
4. Select **Unrestricted JCE Policy files for SDK 5** and click **Continue**.
5. View the license and click **I Agree** to continue.
6. Click **Download Now**.
7. Extract the unlimited jurisdiction policy files that are packaged in the ZIP file. The ZIP file contains a `US_export_policy.jar` file and a `local_policy.jar` file.
8. In your WebSphere Application Server installation, go to the `$JAVA_HOME/jre/lib/security` directory and back up your `US_export_policy.jar` and `local_policy.jar` files.
9. Replace your `US_export_policy.jar` and `local_policy.jar` files with the two files that you downloaded from the IBM.com Web site.
1. Enable security in the WebSphere Application Server. Make sure that all node agents within the cell are active beforehand.

For more information, see “Enabling security” on page 50. It is important to click **Security > Secure administration, applications, and infrastructure**. Select an available realm definition from the list, and then click **Set as current** so that security is enabled upon a server restart.

Note: In previous releases of WebSphere Application Server, the **Set as current** option is known as the **Enable global security** option.

2. Before restarting the server, log off the administrative console. You can log off by clicking **Logout** at the top menu bar.
3. Stop the server by going to the command line in the WebSphere Application Server `/bin` directory and issue a `stopServer server_name` command.
4. Restart the server in secure mode by issuing the command `startServer server_name`. Once the server is secure, you cannot stop the server again without specifying an administrative user name and password. To stop the server once security is enabled, issue the command, `stopServer server_name -username user_id -password password`. Alternatively, you can edit the `soap.client.props` file in the `profile_root/properties` directory, and edit the `com.ibm.SOAP.loginUserId` or `com.ibm.SOAP.loginPassword` properties to contain these administrative IDs.

If you have any problems restarting the server, review the output logs in the `profile_root/logs/server_name` directory. Check the Chapter 20, “Troubleshooting security configurations,” on page 1387 article for any common problems.

Secure administration, applications, and infrastructure settings

Use this page to configure administrative, application, and infrastructure security on a global level.

To view this administrative console page, click **Security > Secure administration, applications, and infrastructure**.

Security has some performance impacts on your applications. The performance impacts can vary depending upon the application workload characteristics. You must first determine that the needed level of security is enabled for your applications, and then measure the impact of security on the performance of your applications.

When security is configured, validate any changes to the user registry or authentication mechanism panels. Click **Apply** to validate the user registry settings. An attempt is made to authenticate the server ID

or to validate the admin ID (if internalServerID is used) to the configured user registry. Validating the user registry settings after enabling administrative security can avoid problems when you restart the server for the first time.

Security configuration wizard:

Launches a wizard that enables you to configure the basic administrative and application security settings. This process restricts administrative tasks and applications to authorized users.

Using this wizard, you can configure application security, resource or Java 2 Connector (J2C) security, and a user registry. You can configure an existing registry and enable administrative, application, and resource security.

When you apply changes made by using the security configuration wizard, administrative security is turned on by default.

Security configuration report:

Launches a security configuration report that displays the core security settings of the application server. The report also displays the administrative users and groups and the CORBA naming roles.

A current limitation to the report is that it does not display application level security information. The report also does not display information on Java Message Service (JMS) security, bus security, or Web Services security.

Enable administrative security:

Specifies whether to enable administrative security for this application server domain. Administrative security requires users to authenticate before obtaining administrative control of the application server.

For more information, see “Administrative roles” on page 326.

When enabling security, set the authentication mechanism configuration and specify a valid user ID and password (or a valid admin ID when internalServerID feature is used) in the selected registry configuration.

Note: There is a difference between the user ID (which is normally called the admin ID), which identifies administrators who manage the environment, and a server ID, which is used for server-to-server communication. You do not need to enter a server ID and password when you are using the internal server ID feature. However, optionally, you can specify a server ID and password. To specify the server ID and password, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under User accounts repository, select the repository and click **Configure**.
3. Specify the server ID and password in the Server user identity section.

Default: Enabled

Enable application security:

Enables application-level security unless the option is overwritten at the server level.

Default: Disabled

Use Java 2 security to restrict application access to local resources:

Specifies whether to enable or disable Java 2 security permission checking. By default, access to local resources is not restricted. You can choose to disable Java 2 security, even when application security is enabled.

When the **Use Java 2 security to restrict application access to local resources** option is enabled and if an application requires more Java 2 security permissions than are granted in the default policy, 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 not have all the required permissions. See “Java 2 security” on page 64 for more information about Java 2 security.

Default: Disabled

Disabling security if you have server startup problems:

If your server does not restart after you enable administrative security, you can disable security. Go to your `app_server_root/bin` directory and run 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 administrative security disabled to check any incorrect settings through the administrative console.

Warn if applications are granted custom permissions:

Specifies that during application deployment and application start, the security runtime issues a warning if applications are granted any custom permissions. Custom permissions are permissions that are defined by the user applications, not Java API permissions. Java API permissions are permissions in the `java.*` and `javax.*` packages.

The 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. No code base is defined and no relative code base is 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 you do not want an application to have according to the J2EE 1.4 specification. For more information on permissions, see “Java 2 security policy files” on page 68.

Important: You cannot enable this option without enabling the **Use Java 2 security to restrict application access to local resources** option.

Default: Disabled

Restrict access to resource authentication data:

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` permission 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 **Restrict access to resource authentication data** 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 `WSPincipalMappingLoginModule` implementation directly in the Java Authentication and Authorization Service (JAAS) login when **Use Java 2 security to restrict application access to local resources** and the **Restrict access to resource authentication data** options are enabled.

Default: Disabled

Current realm definition:

Specifies the current setting for the active user repository.

This field is read-only.

Available realm definitions:

Specifies the available user account repositories.

Set as current:

Enables the user repository after it is configured.

LDAP or a custom user registry is required when running as a UNIX non-root user or running in a multi-node environment.

You can configure settings for one of the following user repositories:

Federated repositories

Specify this setting to manage profiles in multiple repositories under a single realm. The realm can consist of identities in:

- The file-based repository that is built into the system
- One or more external repositories
- Both the built-in, file-based repository and in one or more external repositories

Note: Only a user with administrator privileges can view the federated repositories configuration.

Local operating system

You cannot use localOS in multi-node or when running as non-root on a UNIX platform.

Standalone LDAP registry

Specify this setting to use standalone LDAP registry settings when users and groups reside in an external LDAP directory. When security is enabled and any of these properties change, go to the **Security > Secure administration, applications, and infrastructure** panel and click **Apply** or **OK** to validate the changes.

Note: Since multiple LDAP servers are supported, this setting does not imply one LDAP registry.

Standalone custom registry

Specify this setting to implement your own standalone custom registry that implements the `com.ibm.websphere.security.UserRegistry` interface. When security is enabled and any of these properties change, go to the **Secure administration, applications, and infrastructure** panel and click **Apply** or **OK** to validate the changes.

Default: Disabled

Use domain-qualified user names:

Specifies that user names that are returned by methods are qualified with the security domain in which they reside.

Default: Disabled

Active protocol:

Specifies the active authentication protocol for Remote Method Invocation over the Internet Inter-ORB Protocol (RMI IOP) requests, when security is enabled.

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.x and later servers, specify CSI as your protocol.

V6.0.x If some servers are Version 3.x or Version 4.x servers, specify CSI and SAS.

Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

Note: **V6.0.x** This field displays only when a Version 6.0.x and earlier server is detected in your environment.

Default: BOTH
Range: CSI and SAS, CSI

Specify extent of protection wizard settings

Use this security wizard page to determine whether to enable application security and restrict access to local resources. When you use the wizard, admin security is enabled by default.

To view this security wizard page, click **Security > Secure administration, applications, and infrastructure > Security configuration wizard**.

Enable application security:

Enables application-level security unless the option is overwritten at the server level.

Default: Disabled

Use Java 2 security to restrict application access to local resources:

Specifies whether to enable or disable Java 2 security permission checking. By default, access to local resources is not restricted. You can choose to disable Java 2 security, even when application security is enabled.

When the **Use Java 2 security to restrict application access to local resources** option is enabled and if an application requires more Java 2 security permissions than are granted in the default policy, 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 not have all the required permissions. See “Java 2 security” on page 64 for more information about Java 2 security.

Default: Disabled

Custom properties: Security

Use this page to understand the predefined custom properties that are related to security.

To view this administrative console page, click **Security > Secure administration, applications, and infrastructure > Custom properties**. You can click **New** to add a new custom property and its associated value.

com.ibm.CSI.rmiInboundLoginConfig:

This property specifies the Java Authentication and Authorization Service (JAAS) login configuration that is used for Remote Method Invocation (RMI) requests that are received inbound.

By knowing the login configuration, you can plug in a custom login module that can handle specific cases for RMI logins.

Default system.RMI_INBOUND

com.ibm.CSI.rmiOutboundLoginConfig:

This property specifies the JAAS login configuration that is used for RMI requests that are sent outbound.

Primarily, this property prepares the propagated attributes in the Subject to be sent to the target server. However, you can plug in a custom login module to perform outbound mapping.

Default system.RMI_OUTBOUND

com.ibm.CSI.supportedTargetRealms:

This property enables credentials that are authenticated in the current realm to be sent to any realm that is specified in the Trusted target realms field. The Trusted target realms field is available on the CSiv2 outbound authentication panel. This property enables those realms to perform inbound mapping of the data from the current realm.

It is not recommended that you send authentication information to an unknown realm. Thus, this provides a way to specify that the alternate realms are trusted. To access the CSiv2 outbound authentication panel, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under RMI/IIOP security, click **CSiv2 outbound authentication**.

com.ibm.audit.auditPolicy:

This property is used by the auditing service that was introduced as a technical preview in Version 6. The auditing functionality is not available. Do not modify this property.

Default REQUIRED

com.ibm.audit.auditQueueSize:

This property is used by the auditing service that was introduced as a technical preview in Version 6. The auditing functionality is not available. Do not modify this property.

Default 5000

com.ibm.audit.auditServiceEnabled:

This property is used by the auditing service that was introduced as a technical preview in Version 6. The auditing functionality is not available. Do not modify this property.

Default false

com.ibm.audit.auditSpecification:

This property is used by the auditing service that was introduced as a technical preview in Version 6. The auditing functionality is not available. Do not modify this property.

Default J2EE=AUTHN=failure=enabled:J2EE=AUTHZ=failure=enabled

com.ibm.security.useFIPS:

Specifies that Federal Information Processing Standard (FIPS) algorithms are used. The application server uses the IBMJCEFIPS cryptographic provider instead of the IBMJCE cryptographic provider.

Default false

com.ibm.websphere.security.audit.auditEventFactory:

This property is used by the auditing service that was introduced as a technical preview in Version 6. The auditing functionality is not available. Do not modify this property.

Default J2EE=com.ibm.ws.security.audit.defaultAuditEventFactoryImpl

com.ibm.ws.security.defaultLoginConfig:

This property is the JAAS login configuration that is used for logins that do not fall under the WEB_INBOUND, RMI_OUTBOUND, or RMI_INBOUND login configuration categories.

Internal authentication and protocols that do not have specific JAAS plug points call the system login configuration that is referenced by com.ibm.ws.security.defaultLoginConfig configuration.

Default system.DEFAULT

com.ibm.ws.security.ssoInteropModeEnabled:

This property determines whether to send LtpaToken2 and LtpaToken cookies in the response to a Web request (interoperable).

When this property value is false, the application server just sends the new LtpaToken2 cookie which is stronger, but not interoperable with some other products and Application Server releases prior to Version 5.1.1. In most cases, the old LtpaToken cookie is not needed and you can set this property to false.

Default true

com.ibm.ws.security.webChallengeIfCustomSubjectNotFound:

This property determines the behavior of a single sign-on LtpaToken2 login.

When this property value is set to true, the token contains a custom cache key, and the custom Subject cannot be found, the token is used to log in directly as the custom information needs to be gathered again. A challenge occurs so that the user to login again. When this property value is set to false and the custom Subject is not found, the LtpaToken2 is used to login and gather all of the registry attributes. However, the token might not obtain any of the special attributes that downstream applications might expect.

Default true

com.ibm.ws.security.webInboundLoginConfig:

This property is the JAAS login configuration that is used for Web requests that are received inbound.

By knowing the login configuration, you can plug in a custom login module that can handle specific cases for Web logins.

Default system.WEB_INBOUND

com.ibm.ws.security.webInboundPropagationEnabled:

This property determines whether a received LtpaToken2 cookie should search for the propagated attributes locally before searching the original login server that is specified in the token. After the propagated attributes are received, the Subject is regenerated and the custom attributes are preserved.

You can configure the distributed replication service (DRS) to send the propagated attributes to front-end servers such that a local dynacache lookup can find the propagated attributes. Otherwise, an MBean request is sent to the original login server to retrieve these attributes.

Default true

com.ibm.wsspi.security.audit.auditServiceProvider:

This property is used by the auditing service that was introduced as a technical preview in Version 6. The auditing functionality is not available. Do not modify this property.

Default DEFAULT =
com.ibm.ws.security.audit.defaultAuditServiceProviderImpl

com.ibm.wsspi.security.ltpa.tokenFactory:

This property specifies the Lightweight Third Party Authentication (LTPA) token factories that can be used to validate the LTPA tokens.

Validation occurs in the order in which the token factories are specified because LTPA tokens do not have object identifiers (OIDs) that specify the token type. The Application Server validates the tokens using each token factory until validation is successful. The order that is specified for this property is the most likely order of the received tokens. Specify multiple token factories by separating them with a pipe (|) without spaces before or following the pipe.

Default com.ibm.ws.security.ltpa.LTPATokenFactory |
com.ibm.ws.security.ltpa.LTPAToken2Factory |
com.ibm.ws.security.ltpa.AuthzPropTokenFactory

com.ibm.wsspi.security.token.authenticationTokenFactory:

This property specifies the implementation that is used for an authentication token in the attribute propagation framework. The property provides an old LTPA token implementation for use as the authentication token.

Default com.ibm.ws.security.ltpa.LTPATokenFactory

com.ibm.wsspi.security.token.authorizationTokenFactory:

This property specifies the implementation that is used for an authorization token. This token factory encodes the authorization information.

Default com.ibm.ws.security.ltpa.AuthzPropTokenFactory

com.ibm.wsspi.security.token.propagationTokenFactory:

This property specifies the implementation that is used for a propagation token. This token factory encodes the propagation token information.

The propagation token is on the thread of execution and is not associated with any specific user Subjects. The token follows the invocation downstream wherever the process leads.

Default com.ibm.ws.security.ltpa.AuthzPropTokenFactory

com.ibm.wsspi.security.token.singleSignonTokenFactory:

This property specifies the implementation that is used for a Single Sign-on (SSO) token. This implementation is the cookie that is set when propagation is enabled regardless of the state of the com.ibm.ws.security.ssoInteropModeEnabled property.

By default, this implementation is the LtpaToken2 cookie.

Default com.ibm.ws.security.ltpa.LTPAToken2Factory

security.enablePluggableAuthentication:

This property is no longer used. Instead, use WEB_INBOUND login configuration.

Complete the following steps to modify the WEB_INBOUND login configuration:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Java Authentication and Authorization Service, click **System logins**.

Default true

Security custom property collection

Use this page to view and manage arbitrary name-value pairs of data, where the name is a property key and the value is a string value that can be used to set internal system configuration properties.

The administrative console contains several Custom Properties pages that work similarly. To view one of these administrative pages, click a **Custom Properties** link.

Name:

Specifies the name (or key) for the property.

Each property name must be unique. If the same name is used for multiple properties, the value specified for the first property that has that name is used.

Do not start your property names with `was.` because this prefix is reserved for properties that are predefined in the application server.

Value:

Specifies the value paired with the specified name.

Description:

Provides information about the name-value pair.

Security custom property settings

Use this page to configure arbitrary name-value pairs of data, where the name is a property key and the value is a string value that can be used to set internal system configuration properties. Defining a new property enables you to configure a setting beyond that which is available in the administrative console.

The administrative console contains several Custom property settings pages that work similarly. To view one of these administrative pages, click **Custom Properties > New**.

Name:

Specifies the name (or key) for the property.

Each property name must be unique. If the same name is used for multiple properties, the value specified for the first property that has that name is used.

Do not start your property names with `was.` because this prefix is reserved for properties that are predefined in WebSphere Application Server.

Data type String

Value:

Specifies the value paired with the specified name.

Data type String

Description:

Provides information about the name and value pair.

Data type String

Testing security after enabling it

Basic tests are available that show whether the fundamental security components are working properly. Use this task to validate your security configuration.

After configuring administrative security and restarting all of your servers in a secure mode, 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.

Basic tests are available that show whether the fundamental security components are working properly. Complete the following steps to validate your security configuration:

1. After enabling security, verify that your system comes up in secure mode.
2. Test the Web-based BasicAuth with *Snoop*, by accessing the following URL: `http://hostname.domain:9080/snoop`. A login panel is displayed. If a login panel does not display, then a problem exists. If the panel appears, type in any valid user ID and password in your configured user registry.
3. Test the Web-based form login by starting the administrative console: `http://hostname.domain:port_number/ibm/console`. A form-based login page is displayed. 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 that are used for configuring your user registry when configuring security.
4. Test Test Java Client BasicAuth with *dumpNameSpace*.
Use the `app_server_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.
5. Test all of your applications in secure mode.
6. If all the 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. For more information on common problems, see Chapter 20, "Troubleshooting security configurations," on page 1387.

The results of these tests, if successful, indicate that security is fully enabled and working properly.

Chapter 5. Authenticating users

The process of authenticating users involves a user registry and an authentication mechanism. Optionally, you can define trust between WebSphere Application Server and a proxy server, configure single sign-on capability, and specify how to propagate security attributes between application servers.

The following security topics are covered in this section:

User registries

For information on local operating system, Lightweight Directory Access Protocol (LDAP), custom user registries, and user repositories such as virtual member manager, see “User registries and repositories” on page 89.

Authentication protocol for EJB security

For more information on the authentication protocols that are used for Enterprise JavaBeans (EJB) security, see “Authentication protocol for EJB security” on page 212.

Trust associations

For more information on trust associations, see “Trust associations” on page 180.

Single sign-on

For more information on single sign-on, see “Single sign-on” on page 184.

Security attribute propagation

For more information on propagation tokens, authorization tokens, single sign-on tokens, and authentication tokens, see “Security attribute propagation” on page 191.

The following information is covered in this section:

- Configure a user registry. For more information, see “Selecting a registry or repository.”
- Configure WebSEAL or a custom trust association interceptor. For more information see, “Integrating third-party HTTP reverse proxy servers” on page 227.
- Configure single sign-on. For more information, see “Implementing single sign-on to minimize Web user authentications” on page 228.
- Propagate security attributes. For more information, see “Propagating security attributes among application servers” on page 267.
- Configure the authentication cache. For more information, see “Configuring the authentication cache” on page 269.

After completing the configuring the authentication process, you must authorize access to resources. For more information, see Chapter 6, “Authorizing access to resources,” on page 319.

Selecting a registry or repository

Information about users and groups reside in a user registry. In 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.

Note: During profile creation, either during installation or post-installation, administrative security is enabled by default. You might decide not to enable security, but if the default is accepted, the file-based federated user repository is configured as the active user registry. You can use a different user registry before the profile is created.

Before configuring the user registry or repository, decide which user registry or repository to use. Though different types of registries and repositories are supported, all of the processes in WebSphere Application Server can use only one active registry.

Configuring the correct registry or repository is a prerequisite to assigning users and groups to roles for applications. When a user registry or repository is not configured, the local operating system registry is used by default. If your choice of user registry is not the local operating system registry, you need to first configure the registry or repository, which is normally done as part of enabling security, restart the servers, and then assign users and groups to roles for all your applications.

In addition to local operating system and LDAP registries, WebSphere Application Server also provides a plug-in to support any registry by using the custom registry feature. The custom registry feature enables you to configure any user registry that is not made available through the security configuration panels of the WebSphere Application Server.

The UserRegistry interface is used to implement both the custom registry and the federated repository options for the user account repository. The interface is very helpful in situations where the current user and group information exists in some other formats, for example, a database, and cannot move to local operating system or LDAP registries. In such a case, you can implement the UserRegistry interface so that WebSphere Application Server can use the existing registry for all the security-related operations. The process of implementing a custom registry is a software implementation effort and it is expected that the implementation does not depend on other WebSphere Application Server resources, for example, data sources, for its operation.

WebSphere Application Server supports the following types of user registries:

- Federated repository
- Local operating system
- Standalone Lightweight Directory Access Protocol (LDAP) registry
- Standalone custom registry

After the applications are assigned users and groups and you need to change the user registries, 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, which uses Jacl, removes all of the users and groups 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 user registries.
- The application bindings file does not contain the access IDs which are unique for each user registry even for the same user or group name.

By default, an application does not contain access IDs 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 access IDs for the applications to improve performance, you have to remove the existing user and group information and add the information after configuring the new user registry.

For more information on updating access IDs, see `updateAccess IDs` in the AdminApp object for scripted administration article.

Complete one of the following steps to configure your user registry:

- “Configuring local operating system registries” on page 90
- “Configuring Lightweight Directory Access Protocol user registries” on page 93
- “Configuring standalone custom registries” on page 111.

- “Managing the realm in a federated repository configuration” on page 115
1. If you are enabling security, make sure that you complete the remaining steps. Verify that the User account repository on the Secure administration, applications, and infrastructure panel is set to the appropriate registry or repository. As the final step, validate the user ID and the password by clicking **Apply** on the Secure administration, applications, and infrastructure 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 **Apply** on the Secure administration, applications, and infrastructure panel. After validation, save the configuration and stop and start all WebSphere Application Servers, including the cells, nodes and all of the application servers. To avoid inconsistencies between the WebSphere Application Server processes, make sure that any changes to the registry or repository 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.

User registries and repositories

Information about users and groups reside within a registry or repository.

WebSphere Application Server provides implementations that support multiple types of registries and repositories including the local operating system registry, a standalone Lightweight Directory Access Protocol (LDAP) registry, a standalone custom registry, and federated repositories.

With WebSphere Application Server, a user registry or a repository, such as virtual member manager, authenticates a user and retrieves information about users and groups to perform security-related functions including authentication and authorization.

With WebSphere Application Server, a user registry or repository is used for:

- Authenticating a user using basic authentication, identity assertion, or client certificates
- Retrieving information about users and groups to perform security-related administrative functions, such as mapping users and groups to security roles

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.

After configuring the registry or repository, you must specify it as the active repository. Through the administration console, you can select an available realm definition for the registry or repository from the User account repository section of the Secure administration, applications, and administration panel. After selecting the registry or repository, first click **Set as current**, and then click **Apply**.

Note: WebSphere Application Server has implemented a user registry proxy by using the UserRegistry interface. However, the return values are little different from the interface. For example, `getUniqueId` returns the uniqueID with the realm name wrapped. You cannot use the return value to pass to `getUserSecurityName`, as shown in the following example:

```
// 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 userName that is specified
// in the NameCallback.
String uniqueid = reg.getUniqueId(userName);
// Strip the realm name and get real uniqueID
String uid = WSSecurityPropagationHelper.getUserFromUniqueID (uniqueID);
```

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

Configuring local operating system registries

Use these steps to configure local operating system 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 application programming interface (API) 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.

In WebSphere Application Server Version 6.1, you can use an internally-generated server ID because the Security WebSphere Common Configuration Model (WCCM) model contains a new tag, `internalServerId`. You do not need to specify a server user ID and a password during security configuration except in a mixed-cell environment. See “Administrative roles and naming service authorization” on page 320 for more detailed information about the new internal server ID.

Windows 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.

Consider the following points:

- **AIX** **Solaris** The user that the product processes run under requires the root privilege. This privilege is needed to call the 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.
- **Linux** You might need to have the password shadow file in your system.

Important: The local operating system is not a valid user account repository when you have a mixed cell environment that includes both z/OS platform and non-z/OS platform nodes.

The following steps are needed to perform this task initially when setting up security for the first time.

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Local operating system** and click **Configure**.

3. Enter a valid user name in the **Primary administrative user name** field. This value is the name of a user with administrative privileges that is defined in the registry. This user name is used to access the administrative console or used by wsadmin.
4. Click **Apply**.
5. Select either the **Automatically generated server identity** or **Server identity that is stored in the repository** option. If you select the **Server identity that is stored in the repository** option, enter the following information:

Server user ID or administrative user on a Version 6.0.x node

Specify the short name of the account that is chosen in the second step.

Server user password

Specify the password of the account that is chosen in the second step.

6. Click **OK**.

The administrative console does not validate the user ID and password when you click **OK**. Validation is only done when you click **OK** or **Apply** in the Secure administration, applications, and infrastructure panel. First, make sure that you select **Local operating system** as the available realm definition in the User account repository section, and click **Set as current**. 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 Secure administration, applications, and infrastructure panel and click **OK** or **Apply** to validate your changes. If your changes are not validated, the server might not start.

Important: Until you authorize other users to perform administrative functions, you can only access the administrative console with the server user ID and password that you specified. For more information, see “Authorizing access to administrative roles” on page 372.

For any changes in this panel to be effective, you need to save, stop, and start all the product servers, including deployment managers, nodes and application servers. If the server comes up without any problems, the setup is correct.

After completed these steps, you have configured WebSphere Application Server to use the local operating system registry to identify authorized users.

Complete any remaining steps for enabling security. For more information, see “Enabling security” on page 50.

Configuring user ID for proper privileges

Use this page to configure a user ID for proper privileges or to log on as a service on the Windows platform.

Windows

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)**. For a Windows domain controller, replace **Local Security Policy** with **Domain Security Policy** in the first step.

Note: If the machine is a standalone 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.

2. 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.

3. Add the user name by clicking **Add**.
4. Restart the machine.

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.

Local operating system settings

Use this page to configure local operating system registry settings.

To view this administrative console page, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, click the **Available realm definitions** drop-down list, select **Local operating system**.
3. Click **Configure**.

WebSphere Application Server Version 6.1 distinguishes between the user identities for administrators who manage the environment and server identities for authenticating server to server communications. In most cases, server identities are automatically generated and are not stored in a repository. However, if you are adding a Version 5.0.x or 6.0.x node to a Version 6.1 cell, you must ensure that the Version 5.x or Version 6.0.x server identity and password are defined in the repository for this cell. Enter the server user identity and password on this panel.

Primary administrative user name:

Specifies the name of a user with administrative privileges that is defined in your local operating system.

The user name is used to log on to the administrative console when administrative security is enabled.

Automatically generated server identity:

Enables the application server to generate the server identity that is used for internal process communication.

You can change this server identity on the Authentication mechanisms and expiration panel. To access the Authentication mechanisms and expiration panel, click **Security > Secure administration, applications, and infrastructure > Authentication mechanisms and expiration**. Change the value of the Internal server ID field.

Default: Disabled

Server identity that is stored in the repository:

Specifies a user identity in the repository that is used for internal process communication.

Default: Enabled

Server user ID or administrative user on a Version 6.0.x node:

Specifies the user ID that is used to run the application server for security purposes.

Password:

Specifies the password that corresponds to the server ID.

Local operating system wizard settings

Use this security wizard page to configure local operating system registry settings.

To view this security wizard page, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure > Security configuration wizard**.
2. Select your protection settings and click **Next**.
3. Select the **Local operating system** option and click **Next**.

Primary administrative user name:

Specifies the name of a user with administrative privileges that is defined in your local operating system.

The user name is used to log on to the administrative console when administrative security is enabled.

Configuring Lightweight Directory Access Protocol user registries

To access a user registry using the Lightweight Directory Access Protocol (LDAP), you must know a valid user name (ID) and password, the server host and port of the registry server, the base distinguished name (DN) and, if necessary, the bind DN and the bind password. You can choose any valid user in the user registry that is searchable. You can use any user ID that has the administrative role to log in.

In some LDAP servers, administrative users cannot be searched and thus cannot be used, for example, when `cn=root` in Tivoli Access Manager. The user is referred to as a WebSphere Application Server security server ID, server ID, or server user ID in the documentation. A server ID user has special privileges when calling some protected internal methods.

Normally, the primary administrative user name is used to log into the administrative console if security is enabled. By default, security is enabled after installation.

When security is enabled in the product, the primary administrative user name and password are authenticated with the registry during the product startup. If authentication fails, the server does not start. 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, make sure that the changes are performed when all the product servers are up and running. When changes are to be made in the registry, review the article on “Standalone Lightweight Directory Access Protocol registries” on page 168 (LDAP) before beginning this task.

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, click the **Available realm definitions** drop-down list, select **Standalone LDAP registry**, and click **Configure**.
3. Enter a valid user name in the **Primary administrative user name** 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 browsers. 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 operating system registry to authenticate and obtain privilege information about users by calling the native application programming interfaces (API) in that particular registry.
4. **Optional:** If you want to use the server ID that is stored in the repository, complete the following:
 - a. Select **Automatically generated server identity** to enable the application server to generate the server identity that is used for internal process communication. You can change this server identity on the Authentication mechanisms and expiration panel. To access the Authentication

mechanisms and expiration panel, click **Security > Secure administration, applications, and infrastructure > Authentication mechanisms and expiration**. Change the value of the **Internal server ID** field.

- b. Alternatively, specify a user identity in the repository that is used for internal process communication in the **Server identity that is stored in the repository** field.
 - c. Alternatively, specify the user ID that is used to run the application server for security purposes in the **Server user ID or administrative user on a Version 6.0.x node** field.
5. Select the type of LDAP server to use from the **Type** list. The type of LDAP server determines the default filters that are used by 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.

IBM Tivoli Directory Server users can choose IBM Tivoli Directory Server as the directory type. Use the IBM Tivoli Directory Server directory type for better performance. For a list of supported LDAP servers, see the Supported hardware, software, and APIs Web site.

Attention: IBM SecureWay Directory Server has been renamed to IBM Tivoli Directory Server in WebSphere Application Server version 6.1.

6. Enter the fully qualified host name of the LDAP server in the **Host** field. You can enter either the IP address or domain name system (DNS) name.
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.

The default value is 389. If multiple WebSphere Application Servers are installed and configured to run in the same single sign-on 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 5.x configuration, and a WebSphere Application Server at version 6.0.x is going to interoperate with the version 5.x server, then verify that port 389 is specified explicitly for the version 6.0.x server.

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`. For authorization purposes, this field is case sensitive by default. Match the case in your directory server. If a token is received (for example, from another cell or Lotus Domino) the base DN in the server must match exactly the base DN from the other cell or Domino. If case sensitivity is not a consideration for authorization, enable the **Ignore case for authorization** option.

In WebSphere Application Server, the distinguished name is normalized according to the Lightweight Directory Access Protocol (LDAP) specification. Normalization consists of removing spaces in the base distinguished name 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`.

To interoperate between WebSphere Application Server Version 5 and later versions, you must enter a normalized base distinguished name in the **Base Distinguished Name** field. In WebSphere Application Server, Version 5.0.1 or later, the normalization occurs automatically during runtime.

This field is required for all LDAP directories except the Lotus Domino Directory. The **Base Distinguished Name** field is optional for the Domino server.

9. **Optional:** Enter the bind DN name in the **Bind distinguished name** field. 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. If a name is not specified, the application server binds anonymously. See the **Base Distinguished Name** field description for examples of distinguished names.

10. **Optional:** Enter the password corresponding to the bind DN in the **Bind password** field.
11. **Optional:** Modify the Search time out value. This timeout value is the maximum amount of time that the LDAP server waits to send a response to the product client before stopping the request. The default is 120 seconds.
12. Ensure that the **Reuse connection** option is selected. This option specifies that the server should reuse the LDAP connection. Clear this option only in rare situations where a router is used to send requests to multiple LDAP servers and when the router does not support affinity. Leave this option selected for all other situations.
13. **Optional:** Verify that the **Ignore case for authorization** option is enabled. When you enable this option, 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 you use either the IBM Directory Server or the Sun ONE (formerly iPlanet) Directory Server LDAP servers, you must enable this option because the group information that is obtained from the LDAP servers is not consistent in case. This inconsistency affects the authorization check only. Otherwise, this field is optional and can be enabled when a case sensitive authorization check is required. For example, you might select this option when you use certificates and the certificate contents do not match the case of the entry in the LDAP server.

You can also enable the **Ignore case for authorization** option when you are using single sign-on (SSO) between the product and Lotus Domino. The default is enabled.

14. **Optional:** 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, you can select either the **Centrally managed** or the **Use specific SSL alias** option.

Centrally managed

Enables you to specify an SSL configuration for particular scope such as the cell, node, server, or cluster in one location. To use the **Centrally managed** option, you must specify the SSL configuration for the particular set of endpoints. The Manage endpoint security configurations and trust zones panel displays all of the inbound and outbound endpoints that use the SSL protocol. If you expand the Inbound or Outbound section of the panel and click the name of a node, you can specify an SSL configuration that is used for every endpoint on that node. For an LDAP registry, you can override the inherited SSL configuration by specifying an SSL configuration for LDAP. To specify an SSL configuration for LDAP, complete the following steps:

- a. Click **Security > SSL certificate and key management > Manage endpoint security configurations and trust zones**.
- b. Expand **Outbound > cell_name > Nodes > node_name > Servers > server_name > LDAP**.

Use specific SSL alias

Select the **Use specific SSL alias** option if you intend to select one of the SSL configurations in the menu below the option.

This configuration is used only when SSL is enabled for LDAP. The default is *DefaultSSLSettings*. To modify or create a new SSL configuration, complete the following steps:

- a. Click **Security > SSL certificate and key management**.
 - b. Under Configuration settings, click **Manage endpoint security configurations**.
 - c. Select a Secure Sockets Layer (SSL) *configuration_name* for selected scopes, such as a cell, node, server, or cluster.
 - d. Under Related items, click **SSL configurations**.
 - e. Click **New**.
15. Click **OK** and either **Apply** or **Save** until you return to the Secure administration, applications, and infrastructure panel.

This set of steps is required to set up the LDAP user 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 specified in “Enabling security for the realm” on page 74.
2. Save, stop, and restart all the product servers (deployment managers, nodes and Application Servers) for changes in this panel to take effect. If the server comes up without any problems the setup is correct.

Standalone LDAP registry 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, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, click the **Available realm definitions** drop-down list, select **Standalone LDAP registry**, and click **Configure**.

When security is enabled and any of these properties change, go to the Secure administration, applications, and infrastructure panel and click **Apply** to validate the changes.

WebSphere Application Server Version 6.1 distinguishes between the user identities for administrators who manage the environment and server identities for authenticating server to server communications. In most cases, server identities are automatically generated and are not stored in a repository. However, if you are adding a Version 5.0.x or 6.0.x node to a Version 6.1 cell, you must ensure that the Version 5.x or Version 6.0.x server identity and password are defined in the repository for this cell. Enter the server user identity and password on this panel.

Primary administrative user name:

Specifies the name of a user with administrative privileges that is defined in your custom user registry.

The user name is used to log onto the administrative console when administrative security is enabled.

Automatically generated server identity:

Enables the application server to generate the server identity that is used for internal process communication.

You can change this server identity on the Authentication mechanisms and expiration panel. To access the Authentication mechanisms and expiration panel, click **Security > Secure administration, applications, and infrastructure > Authentication mechanisms and expiration**. Change the value of the Internal server ID field.

Default: Disabled

Server identity that is stored in the repository:

Specifies a user identity in the repository that is used for internal process communication.

Default: Enabled

Server user ID or administrative user on a Version 6.0.x node:

Specifies the user ID that is used to run the application server for security purposes.

Password:

Specifies the password that corresponds to the server ID.

Type of LDAP server:

Specifies the type of LDAP server to which you connect.

IBM SecureWay Directory Server is not supported.

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 application servers are installed and configured to run in the same single sign-on domain or if the application server interoperates with a previous version, 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, verify that port 389 is specified explicitly for the Version 5 server.

Default:	389
Type:	Integer

Base distinguished name (DN):

Specifies the base distinguished name (DN) of the directory service, which indicates the starting point for LDAP searches of the directory service. In most cases, bind DN and bind password are needed. However, when anonymous bind can satisfy all of the required functions, bind DN and bind password are not needed.

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: ou=Rochester, o=IBM, c=US or o=IBM c=US or 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 Lotus Domino, the base DN in the server must match the base DN from the other cell or Lotus Domino server exactly. If case sensitivity is not a consideration for authorization, enable the **Ignore case for authorization** option. This option is required for all Lightweight Directory Access Protocol (LDAP) directories, except for the Lotus Domino Directory, IBM Tivoli Directory Server V6.0, and Novell eDirectory, where this field is optional.

If you need to interoperate between the application server Version 5 and a Version 5.0.1 or later server, you must enter a normalized base DN. A normalized base DN does not contain spaces before or after commas and equal symbols. An example of a non-normalized base DN is o = ibm, c = us or o=ibm, c=us. An example of a normalized base DN is o=ibm,c=us. In WebSphere Application Server, Version 5.0.1 or later, the normalization occurs automatically during runtime.

Bind distinguished name (DN):

Specifies the DN 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 (DN) 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 stopping a request.

Default: 120

Reuse connection:

Specifies whether the server reuses the LDAP connection. Clear this option only in rare situations where a router is used to distribute requests to multiple LDAP servers and when the router does not support affinity.

Default: Enabled
Range: Enabled or Disabled

Important: Disabling the **Reuse connection** option causes the application server to create a new LDAP connection for every LDAP search request. This situation impacts system performance if your environment requires extensive LDAP calls. This option is provided because the router is not sending the request to the same LDAP server. The option is also used when the idle connection timeout value or firewall timeout value between the application server and LDAP is too small.

If you are using WebSphere Edge Server for LDAP failover, you must enable TCP resets with the Edge server. A TCP reset causes the connection to immediately closed and a backup server to failover. For more information, see "Sending TCP resets when server is down" at <http://www-3.ibm.com/software/webservers/appserv/doc/v50/ec/infocenter/edge/LBguide.htm#HDRRESETSERVER> and the Edge Server V2 - TCP Reset feature in PTF #2 described in: <ftp://ftp.software.ibm.com/software/websphere/edgeserver/info/doc/v20/en/updates.pdf>.

Ignore case for authorization:

Specifies that a case insensitive authorization check is performed when using the default authorization.

This option is required when IBM Tivoli Directory Server is selected as the LDAP directory server.

This option 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.

This option is optional and can be enabled when a case-sensitive authorization check is required. For example, use this option when the certificates and the certificate contents do not match the case that is used for the entry in the LDAP server. You can enable the **Ignore case for authorization** option when using single sign-on (SSO) between the 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.

Centrally managed:

Specifies that the selection of an SSL configuration is based upon the outbound topology view for the Java Naming and Directory Interface (JNDI) platform.

Centrally managed configurations support one location to maintain SSL configurations rather than spreading them across the configuration documents.

Default: Enabled

Use specific SSL alias:

Specifies the SSL configuration alias to use for LDAP outbound SSL communications.

This option overrides the centrally managed configuration for the JNDI platform.

Standalone LDAP registry wizard settings

Use this security wizard page to provide the basic settings to connect the application server to an existing Lightweight Directory Access Protocol (LDAP) registry.

To view this security wizard page, click **Security > Secure administration, applications, and infrastructure > Security configuration wizard**. You can modify your LDAP registry configuration by completing the following steps:

1. Click **Security > Security administration, applications, and infrastructure**.
2. Under User account repository, click the **Available realm definitions** drop-down list, select **Standalone LDAP registry**, and click **Configure**.

Primary administrative user name:

Specifies the name of a user with administrative privileges that is defined in your custom user registry.

The user name is used to log onto the administrative console when administrative security is enabled.

Type of LDAP server:

Specifies the type of LDAP server to which you connect.

IBM SecureWay Directory Server is not supported.

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 application servers are installed and configured to run in the same single sign-on domain or if the application server interoperates with a previous version, 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, verify that port 389 is specified explicitly for the Version 5 server.

Default: 389
Type: Integer

Base distinguished name (DN):

Specifies the base distinguished name (DN) of the directory service, which indicates the starting point for LDAP searches of the directory service. In most cases, bind DN and bind password are needed. However, when anonymous bind can satisfy all of the required functions, bind DN and bind password are not needed.

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: ou=Rochester, o=IBM, c=US or o=IBM, c=US or 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 Lotus Domino, the base DN in the server must match the base DN from the other cell or Lotus Domino server exactly.

If you need to interoperate between the application server Version 5 and a Version 5.0.1 or later server, you must enter a normalized base DN. A normalized base DN does not contain spaces before or after commas and equal symbols. An example of a non-normalized base DN is o = ibm, c = us or o=ibm, c=us. An example of a normalized base DN 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 DN 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 (DN) field description for examples of distinguished names.

Bind password:

Specifies the password for the application server to use when binding to the directory service.

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 > Secure administration, applications, and infrastructure**.
2. Under User account repository, click the **Available realm definitions** drop-down list, select **Standalone LDAP registry**, and click **Configure**.
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 that is selected in the Standalone LDAP registry 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 Secure administration, applications, and infrastructure 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 and 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 and 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 display. For example, to display entries of the `objectclass = inetOrgPerson` type 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 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 (;).

Data type: String

Group member ID map:

Specifies the LDAP filter that identifies user-to-group relationships.

For directory types SecureWay, and Domino, this field takes multiple `objectclass:property` pairs, delimited by a semicolon (;). In an `objectclass:property` pair, the object class value is the same object class that is defined in the group filter, and the property is the member attribute. If the object class value does not match the object class in the 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, Sun ONE, and Active Directory, this field takes multiple `group attribute:member attribute` pairs delimited by a semicolon (;). These pairs are used to find the group memberships of a user by enumerating all the group attributes that are 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 object class stores the list of members belonging to the group represented by the object class. 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. Application server security leverages the recursive search functionality of the LDAP server to search a user's group memberships, including recursive group memberships. For example:

- IBM Directory Server is preconfigured by the application server security to recursively calculate a user's group memberships using the `ibm-allGroup` attribute.
- SunONE directory server is preconfigured 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 runtime, authentication fails because the result is an ambiguous match. The syntax or structure of this filter is:

`(&(uid=${SubjectCN})(objectclass=inetOrgPerson))`. 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}`
- `${Issuer}`
- `${NotAfter}`
- `${NotBefore}`
- `${SerialNumber}`
- `${SigAlgName}`
- `${SigAlgOID}`
- `${SigAlgParams}`
- `${SubjectCN}`
- `${Version}`

Data type:

String

Configuring Lightweight Directory Access Protocol search filters

Use this topic to configure the LDAP search filters. These steps are required to modify existing user and group filters for a particular LDAP directory type, and also to set up certificate filters to map certificates to entries in the LDAP server.

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 Standalone 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. Complete the following steps in the administrative console.

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Standalone LDAP registry** and click **Configure**.
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. The filter is also used to authenticate a user with 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 “Using specific directory servers as the LDAP server” on page 106 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 object class or the groupOfUniqueNames object class, specify the following syntax:

```
(&(cn=%v)(|(objectclass=groupOfNames)(objectclass=groupOfUniqueNames)))
```

For more information about this syntax, see the “Using specific directory servers as the LDAP server” on page 106 documentation.

6. Modify the user ID map, if necessary. This filter maps the short name of a user to an LDAP entry and specifies the piece of information that represents users when these users are displayed with their short names. For example, to display entries of 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 CN=Bob Smith, ou=austin.ibm.com, o=IBM, c=US user 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 bob user ID 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 and 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, 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 that belong 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 object class, stores the list of members that belong to the group that is represented by the object class. This field takes multiple objectclass:property pairs that are delimited by a semicolon (;). For more information about this syntax, see the “Using specific directory servers as the LDAP server” on page 106.

For the IBM Tivoli Directory Server, Sun ONE, and Active Directory, this field is used to query all users in a group with the information that is stored in the user object. 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 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** option on the Standalone LDAP registry settings to make the authorization case insensitive. To access the Standalone LDAP registry settings panel, complete the following steps:

- a. Click **Security > Secure administration, applications, and infrastructure**.
 - b. Under User account repository, click the **Available realm definitions** drop-down list, select **Standalone LDAP registry**.
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 Standalone LDAP registry panel, which contains the previous LDAP directory type, rather than the custom LDAP directory type. Clicking **OK** or **Apply** in the Standalone LDAP 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. Click **Security > Secure administration, applications, and infrastructure** and change the User account repository type to Standalone custom registry.
- Select **Custom** type from the Standalone LDAP 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 does not take place in this panel. Validation is done when you click **OK** or **Apply** on the Secure administration, applications, and infrastructure panel. If you are in the process of enabling security for the first time, complete the remaining steps and go to the Secure administration, applications, and infrastructure panel. Select **Standalone LDAP registry** as the user account repository. If security is already enabled and any information on this panel changes, go to the Secure administration, applications, and infrastructure panel and click **OK** or **Apply** to validate your changes. If your changes are not validated, the server might not start.

These steps result in the configuration of the LDAP search filters. These steps are required to modify existing user and group filters for a particular LDAP directory type. The steps are also used to set up certificate filters to map certificates to entries in the LDAP server.

1. Validate this setup by clicking **OK** or **Apply** on the Secure administration, applications, and infrastructure panel.
2. Save, stop, and start all the product servers, including the cell, nodes and all of 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.

Updating LDAP binding information

Use this information to dynamically update security LDAP binding information by switching to a different binding identity.

You can dynamically update Lightweight Directory Access Protocol (LDAP) binding information without first stopping and restarting WebSphere Application Server by using the **wsadmin** tool.

The `resetLdapBindInfo` method in `SecurityAdmin` MBean is used to dynamically update LDAP binding information at WebSphere Application Server security runtime, and it takes the bind distinguished name (DN) and bind password parameters as input. The `resetLdapBindInfo` method validates the bind information against the LDAP server. If validation passes, new binding information is stored in `security.xml`, and a copy of the information is placed in WebSphere Application Server security runtime.

If the new binding information is `null`, `null`, the `resetLdapBindInfo` method first extracts LDAP binding information, including bind DN, bind password, and target binding host from WebSphere Application Server security configuration in `security.xml`. It then pushes the binding information to WebSphere Application Server security runtime.

There are two ways to dynamically update WebSphere Application Server security LDAP binding information using the `SecurityAdmin` MBean through `wsadmin`:

- “Switching to a different binding identity” on page 106
- “Switching to a failover LDAP host” on page 106

Switching to a different binding identity:

To dynamically update security LDAP binding information by switching to a different binding identity:

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, click the **Available realm definitions** drop-down list, select **Standalone LDAP registry**, and click **Configure**.
3. Create a new bind DN. It must have the same access authority as the current bind DN.
4. Run the SecurityAdmin MBean across all of the application server processes to validate the new binding information, to save it to `security.xml`, and to push the new binding information to the runtime.

The following is a sample Jacl file for step 4:

```
proc LDAPReBind {args} {
  global AdminConfig AdminControl ldapBindDn ldapBindPassword
  set ldapBindDn [lindex $args 0]
  set ldapBindPassword [lindex $args 1]
  set secMBeans [$AdminControl queryNames type=SecurityAdmin,*]
  set plist [list $ldapBindDn $ldapBindPassword]
  foreach secMBean $secMBeans {
    set result [$AdminControl invoke $secMBean resetLdapBindInfo $plist]
  }
}
```

Switching to a failover LDAP host:

To dynamically update security LDAP binding information by switching to a failover LDAP host:

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Standalone LDAP registry** and click **Configure**.
3. Change the password for bind DN on one LDAP server (it can be the primary or the backup).
4. Update the new bind DN password to WebSphere Application security runtime by calling `resetLdapBindInfo` with the bind DN and by using its new password as a parameter.
5. Use the new bind DN password for all of the other LDAP servers. The binding information is now consistent across WebSphere Application Server and the LDAP servers.

Note: If you call `resetLdapBindInfo` with `null`, `null` as input parameters, WebSphere Application Server security runtime completes the following steps:

- a. Reads the bind DN, bind password, and target LDAP hosts from `security.xml`.
- b. Refreshes the cached connection to the LDAP server.

If you configure security to use multiple LDAP servers, this MBean call forces WebSphere Application Server security to reconnect to the first available LDAP host in the list. For example, if three LDAP servers are configured in the order of L1, L2, and L3, the reconnection process always starts with the L1 server.

Using specific directory servers as the LDAP server

This article provides important information about the directory servers that are supported as Lightweight Directory Access Protocol (LDAP) servers in WebSphere Application Server.

Microsoft Active Directory forest is not supported in the user registry in this product.

For a list of supported LDAP servers, refer to the Supported hardware and software Web site.

It is expected that other LDAP servers follow the LDAP specification. 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 that are required for that directory.

To improve performance for LDAP searches, the default filters for IBM Tivoli Directory Server, Sun ONE, 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, select the **Ignore case for authorization** option. This option 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 for authorization** option.

- **Using IBM Tivoli Directory Server as the LDAP server**

To use IBM Tivoli Directory Server, formerly IBM Directory Server, select **IBM Tivoli Directory Server** as the directory type.

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 runtime. 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 6.0 on the same machine that you install WebSphere Application Server Version 6.1. IBM Tivoli Directory Server Version 6.0 includes WebSphere Application Server Express Version 5.1.1, which the directory server uses for its administrative console. Install the Web Administration tool Version 6.0 and WebSphere Application Server Express Version 5.1.1, which are both bundled with IBM Tivoli Directory Server Version 6.0, on a different machine from WebSphere Application Server Version 6.1. You cannot use WebSphere Application Server Version 6.1 as the administrative console for IBM Tivoli Directory Server. If IBM Tivoli Directory Server Version 6.0 and WebSphere Application Server Version 6.1 are installed on the same machine, you might encounter port conflicts.

If you must install IBM Tivoli Directory Server Version 6.0 and WebSphere Application Server Version 6.1 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 5.1.1**.
- Install WebSphere Application Server Version 6.1.
- When you install WebSphere Application Server Version 6.1, change the port number for the application server.
- You might need to adjust the WebSphere Application Server environment variables on WebSphere Application Server Version 6.1 for `WAS_HOME` and `WAS_INSTALL_ROOT` (or `APP_SERVER_ROOT` for i5/OS). To change the variables using the administrative console, click **Environment > WebSphere Variables**.

- **Using a Lotus Domino Enterprise Server as the LDAP server**

If you select the Lotus Domino Enterprise Server Version 6.5.4 or Version 7.0 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 is 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.

User ID Map : person:shortname

- **Using Sun ONE Directory Server as the LDAP server**

You can select **Sun ONE Directory Server** for your Sun ONE Directory Server system. In Sun ONE Directory Server, the object class is the default 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. Create the group from the role. If you want to use the groupOfUniqueName attribute 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 that are possessed by a given entry, rather than selecting a group and browsing through the members list. When using roles, you can create a group using a:

- Managed role
- Filtered role
- Nested role

All of these roles are computable by the nsRole attribute.

- **Using Microsoft Active Directory server as the LDAP server**

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 for a given user entry, rather than browsing through the member list in each group. If you change the default behavior to browse each group, you can change the **Group Member ID Map** field from memberof:member to group:member.

The following steps describe how to set up Microsoft Active Directory as your LDAP server.

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.
3. Use the WebSphere Application Server administrative console to set up the information that is needed to use Microsoft Active Directory.
 - a. Click **Security > Secure administration, applications, and infrastructure**.
 - b. Under User account repository, select **Standalone LDAP registry** and click **Configure**.
 - c. Set up LDAP with Active Directory as the type of LDAP server. Based on the information that is determined in the previous steps, you can specify the following values on the LDAP settings panel:

Primary administrative user name

Specify the name of a user with administrative privileges that is defined in the registry. This user name is used to access the administrative console or used by wsadmin.

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 is chosen in the first step. For example: dc=ibm, dc=com

Bind distinguished name (DN)

Specify the full distinguished name of the account that is chosen in the first step. For example: cn=adminUsername, cn=users, dc=ibm, dc=com

Bind password

Specify the password of the account that is chosen in the first step.

- d. Click **OK** and **Save** to save the changes to the master configuration.
4. Click **Security > Secure administration, applications, and infrastructure**.
5. Under User account repository, click the **Available realm definitions** drop-down list, select **Standalone LDAP registry**, and click **Configure**.
6. Select either the **Automatically generated server identity** or **Server identity that is stored in the repository** option. If you select the **Server identity that is stored in the repository** option, enter the following information:

Server user ID or administrative user on a Version 6.0.x node

Specify the short name of the account that is chosen in the second step.

Server user password

Specify the password of the account that is chosen in the second step.

7. **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.
8. Click **OK** and **Save** to save the changes to the master configuration.
9. Stop and restart the administrative server so that the changes take effect.

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 the 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 these 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 not currently used in the static group membership search for WebSphere Application Server.

- Use the direct method for searching group memberships if your LDAP server has an attribute in the 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 the following methods.
 - objectclass:attribute pairs for the indirect method
 - attribute:attribute pairs for the direct method
- Use the sample entries of attribute:attribute pairs in Group member ID map fields. Note that the groupMembership attribute lists all the static groups for which a user is a member. This attribute is NOT updated whenever an object matches or does not match a dynamic group's filter. Please refer to the Novell eDirectory documentation for more information about the groupMembership attribute.
 - 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
 - groupMembership:member for eDirectory
- Use the sample entries of objectClass:attribute pairs in the Group member ID map field.
 - dominoGroup:member for Lotus 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, this attribute lists the groups beneath, which the group is directly nested only and does not contain the recursive list of nested predecessors. The Lotus Domino LDAP server 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.

Configuring dynamic and nested group support for the SunONE or iPlanet Directory Server:

Configure dynamic and nested groups to simplify WebSphere Application Server security management and increase its effectiveness and flexibility.

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 170 for more information on this topic.

1. In the administrative console for WebSphere Application Server, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, click the **Available realm definitions** drop-down list, select **Standalone LDAP registry**, and click **Configure**.
3. Select SunONE for the type of LDAP server.
4. Select the **Ignore case for authorization** option.

5. Under Additional Properties, click **Advanced Lightweight Directory Access Protocol (LDAP) user registry settings**.
6. Change the Group filter setting to `&(cn=%v)(objectclass=ldapsubentry)`.
7. Change the Group member ID map setting to `nsRole:nsRole`.
8. Click **Apply** or **OK** to validate the changes.

Configuring dynamic and nested group support for the IBM Tivoli Directory Server:

Configure dynamic and nested groups to simplify WebSphere Application Server security management and increase its effectiveness and flexibility.

When creating groups, ensure that nested and dynamic group memberships work correctly.

1. In the administrative console for WebSphere Application Server, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, click **Standalone LDAP registry**, and click **Configure**.
3. Select IBM Tivoli Directory Server for the type of LDAP server.
4. Under Additional properties, click **Advanced Lightweight Directory Access Protocol (LDAP) user registry settings**.
5. Change the Group filter value to `(&(cn=%v)(|(objectclass=groupOfNames)(objectclass=groupOfUniqueNames)(objectclass=groupOfURLs)))`.
6. Change the Group member ID map value to `ibm-allGroups:member;ibm-allGroups:uniqueMember`.
7. Click **Apply** or **OK** to validate the changes.
8. Verify that Auxiliary object class field on the Add an LDAP entry panel for your IBM Tivoli Directory server has the appropriate value. When you create a nested group, the Auxiliary object class value is `ibm-nestedGroup`. When you create a dynamic group, the Auxiliary object class value is `ibm-dynamicGroup`.

Configuring standalone custom registries

Before you begin this task, implement and build the UserRegistry interface. For more information on developing standalone custom registries refer to “Developing standalone custom registries” on page 501. The following steps are required to configure standalone custom registries through the administrative console.

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repositories, select **Standalone custom registry** and click **Configure**.
3. Enter a valid user name in the Primary administrative user name field. 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 operating system registry to authenticate and obtain privilege information about users by calling the native APIs in that particular registry.
4. Enter the complete location of the dot-separated class name that implements the `com.ibm.websphere.security.UserRegistry` interface in the Custom registry class name field. 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 `app_server_root/lib/ext` directory.
Attention: The sample provided is intended to familiarize you with this feature. Do not use this sample in an actual production environment.
5. Add your custom registry class name to the class path. It is recommended that you add the Java Archive (JAR) file that contains your custom user registry implementation to the `app_server_root/classes` directory.

6. **Optional:** Select the **Ignore case for authorization** option for the authorization to perform a case insensitive check. Enabling this option is necessary only when your user 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.
8. **Optional:** Enter additional properties to initialize your implementation.
 - a. Click **Custom properties > New**.
 - b. Enter the property name and value.

For the sample, enter the following two properties. It is assumed that the users.props file 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 choose and reference their locations through custom properties. However, make sure that the directory has the appropriate access permissions.

Property name	Property value
usersFile	<code>\${USER_INSTALL_ROOT}/customer_sample /users.props</code>
groupsFile	<code>\${USER_INSTALL_ROOT}/customer_sample /groups.props</code>

Samples of these two properties are available in users.props file and groups.props file.

The **Description**, **Required**, and **Validation Expression** fields are not used and can remain blank.

WebSphere Application Server version 4-based custom user registry is migrated to the custom user registry based on the com.ibm.websphere.security.UserRegistry interface.

- c. Click **Apply**.
- d. Repeat this step to add other additional properties.
9. Click **Security > Secure administration, applications, and infrastructure**.
10. Under User account repository, click the **Available realm definitions** drop-down list, select **Standalone custom registry**, and click **Configure**.
11. Select either the **Automatically generated server identity** or **Server identity that is stored in the repository** option. If you select the **Server identity that is stored in the repository** option, enter the following information:
 - Server user ID or administrative user on a Version 6.0.x node**
Specify the short name of the account that is chosen in the second step.
 - Server user password**
Specify the password of the account that is chosen in the second step.
12. Click **OK** and complete the required steps to turn on security.

This set of steps is required to set up the standalone custom registry and to enable security in WebSphere Application Server.

Note: The security component of WebSphere Application Server expands a selected list of variables when enabling security. See Variable settings for more detail.

1. Complete the remaining steps, if you are enabling security.
2. Validate the user and password. Save and synchronize in the cell environment.
3. After security is turned on, save, stop, and start all the product servers, including cell, nodes, and all of the application servers, for any changes to take effect. If the server comes up without any problems, the setup is correct.

Standalone custom registry settings

Use this page to configure the standalone custom registry.

To view this administrative console page, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, click the **Available realm definitions** drop-down list, select **Standalone custom registry**, and click **Configure**.

After the properties are set in this panel, click **Apply**. Under Additional Properties, click **Custom properties** to include additional properties that the custom user registry requires.

Note: Custom properties might include information such as specifying lists of users or groups.

When security is enabled and any of these custom user registry settings change, go to the Secure administration, applications, and infrastructure panel and click **Apply** to validate the changes.

WebSphere Application Server Version 6.1 distinguishes between the user identities for administrators who manage the environment and server identities for authenticating server to server communications. In most cases, server identities are automatically generated and are not stored in a repository. However, if you are adding a Version 5.0.x or 6.0.x node to a Version 6.1 cell, you must ensure that the Version 5.x or Version 6.0.x server identity and password are defined in the repository for this cell. Enter the server user identity and password on this panel.

Primary administrative user name:

Specifies the name of a user with administrative privileges that is defined in your custom user registry.

The user name is used to log onto the administrative console when administrative security is enabled.

Automatically generated server identity:

Enables the application server to generate the server identity that is used for internal process communication.

You can change this server identity on the Authentication mechanisms and expiration panel. To access the Authentication mechanisms and expiration panel, click **Security > Secure administration, applications, and infrastructure > Authentication mechanisms and expiration**. Change the value of the Internal server ID field.

Default: Disabled

Server identity that is stored in the repository:

Specifies a user identity in the repository that is used for internal process communication.

Default: Enabled

Server user ID or administrative user on a Version 6.0.x node:

Specifies the user ID that is used to run the application server for security purposes.

Password:

Specifies the password that corresponds to the server ID.

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.

Data type: String
Default: com.ibm.websphere.security.FileRegistrySample

Ignore case for authorization:

Indicates that a case-insensitive authorization check is performed when you use the default authorization.

Default: Disabled
Range: Enabled or Disabled

Standalone custom registry wizard settings

Use this page to provide the basic settings to connect the application server to an existing standalone custom registry.

To view this security wizard page, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure > Security configuration wizard**.
2. Select your protection settings and click **Next**.
3. Select the **Standalone custom registry** option and click **Next**.

You can modify your standalone custom registry configuration by completing the following steps:

1. Click **Security > Security administration, applications, and infrastructure**.
2. Under User account repository, click the **Available realm definitions** drop-down list, select **Standalone custom registry**, and click **Configure**.
3. Enter additional properties to initialize your implementation
 - Click **Custom properties > New**.
 - Enter the property name and value. For the sample, enter the following two properties. It is assumed that the `users.props` file 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 choose and reference their locations through Custom properties. However, make sure that the directory has the appropriate access permissions.

Property name	Property value
usersFile	<code>\${USER_INSTALL_ROOT}/customer_sample /users.props</code>
groupsFile	<code>\${USER_INSTALL_ROOT}/customer_sample /groups.props</code>

Samples of these two properties are available in `users.props` file and `groups.props` file.

The Description, Required, and Validation Expression fields are not used and can remain blank.

WebSphere Application Server Version 4 based custom user registry is migrated to the custom user registry based on the `com.ibm.websphere.security.UserRegistry` interface.

- Click **Apply**.

Primary administrative user name:

Specifies the name of a user with administrative privileges that is defined in your custom user registry.

The user name is used to log onto the administrative console when administrative security is enabled.

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.

Data type:	String
Default:	<code>com.ibm.websphere.security.FileRegistrySample</code>

Ignore case for authorization:

Indicates that a case-insensitive authorization check is performed when you use the default authorization.

Default:	Disabled
Range:	Enabled or Disabled

Managing the realm in a federated repository configuration

Follow this topic to manage the realm in a federated repository configuration.

The realm can consist of identities in:

- The file-based repository that is built into the system
- One or more external repositories
- Both the built-in, file-based repository and in one or more external repositories

Before you configure your realm, review [Limitations of federated repositories](#).

1. Configure your realm by using one of the following topics. You might be configuring your realm for the first time or changing an existing realm configuration.
 - “Using a single built-in, file-based repository in a new configuration under [Federated repositories](#)” on page 118
 - “Changing a federated repository configuration to include a single built-in, file-based repository only” on page 120
 - “Configuring a single, Lightweight Directory Access Protocol repository in a new configuration under [Federated repositories](#)” on page 121
 - “Changing a federated repository configuration to include a single, Lightweight Directory Access Protocol repository only” on page 122
 - “Configuring multiple Lightweight Directory Access Protocol repositories in a federated repository configuration” on page 123
 - “Configuring a single built-in, file-based repository and one or more Lightweight Directory Access Protocol repositories in a federated repository configuration” on page 124
2. Configure supported entity types using the steps described in “[Configuring supported entity types in a federated repository configuration](#)” on page 147. You must configure supported entity types before you can manage this account with Users and Groups. The Base entry for the default parent determines the repository location where entities of the specified type are placed on a create operation.
3. **Optional:** Use one or more of the following tasks to extend the capabilities of storing data and attributes in your realm:
 - a. Configure an entry mapping repository using the steps described in “[Configuring an entry mapping repository in a federated repository configuration](#)” on page 144. An entry mapping repository is used to store data for managing profiles on multiple repositories.

- b. Configure a property extension repository using the steps described in “Configuring a property extension repository in a federated repository configuration” on page 132. A property extension repository is used to store attributes that cannot be stored in your Lightweight Directory Access Protocol (LDAP) server.
 - a. Set up a database repository using wsadmin commands as described in “Setting up an entry mapping repository, a property extension repository, or a database repository using wsadmin commands” on page 136
4. **Optional:** Use one or more of the following advanced user tasks to extend the capabilities of LDAP repositories in your realm:
 - “Increasing the performance of the federated repository configuration” on page 151
 - “Configuring Lightweight Directory Access Protocol entity types in a federated repository configuration” on page 155
 - “Configuring group attribute definition settings in a federated repository configuration” on page 158
 5. **Optional:** Manage repositories that are configured in your system by following the steps described in “Managing repositories in a federated repository configuration” on page 149.
 6. **Optional:** Add an external repository into your realm by following the steps described in “Adding an external repository in a federated repository configuration” on page 131.
1. After configuring the federated repositories, click **Security > Secure administration, applications, and infrastructure** to return to the Secure administration, applications, and infrastructure panel. Verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Secure administration, applications, and infrastructure panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
 2. If you are enabling security, complete the remaining steps as specified in “Enabling security for the realm” on page 74. As the final step, validate this setup by clicking **Apply** in the Secure administration, applications, and infrastructure panel.
 3. Save, stop, and restart all the product servers (deployment managers, nodes, and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Realm configuration settings

Use this page to manage the realm. The realm can consist of identities in the file-based repository that is built into the system, in one or more external repositories, or in both the built-in, file-based repository and one or more external repositories.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.

When you finish adding or updating your federated repository configuration, go to the **Security > Secure administration, applications, and infrastructure** panel and click **Apply** to validate the changes.

A single built-in, file-based repository is built into the system and included in the realm by default.

You can configure one or more Lightweight Directory Access Protocol (LDAP) repositories to store identities in the realm. Click **Add base entry to realm** to specify a repository configuration and a base entry into the realm. You can configure multiple different base entries into the same repository.

Click **Remove** to remove selected repositories from the realm. Repository configurations and contents are not destroyed. The following restrictions apply:

- The realm must always contain at least one base entry; therefore, you cannot remove every entry.

- If you plan to remove the built-in, file-based repository from the administrative realm, verify that at least one user in another member repository is a console user with administrative rights. Otherwise, you must disable security to regain access to the administrative console.

WebSphere Application Server Version 6.1 distinguishes between the user identities for administrators who manage the environment and server identities for authenticating server to server communications. In most cases, server identities are automatically generated and are not stored in a repository. However, if you are adding a Version 5.0.x or 6.0.x node to a Version 6.1 cell, you must ensure that the Version 5.x or Version 6.0.x server identity and password are defined in the repository for this cell. Enter the server user identity and password on this panel.

Ream name:

Specifies the name of the realm. You can change the realm name.

Primary administrative user name:

Specifies the name of the user with administrative privileges that is defined in the repository, for example, adminUser.

Automatically generated server identity:

Enables the application server to generate the server identity that is used for internal process communication.

You can change this server identity on the Authentication mechanisms and expiration panel. To access the Authentication mechanisms and expiration panel, click **Security > Secure administration, applications, and infrastructure > Authentication mechanisms and expiration**. Change the value of the Internal server ID field.

Default: Disabled

Server identity that is stored in the repository:

Specifies a user identity in the repository that is used for internal process communication.

Default: Enabled

Server user ID or administrative user on a Version 6.0.x node:

Specifies the user ID that is used to run the application server for security purposes.

Password:

Specifies the password that corresponds to the server ID.

Ignore case for authorization:

Specifies that a case-insensitive authorization check is performed.

If case sensitivity is not a consideration for authorization, enable the **Ignore case for authorization** option.

Base entry:

Specifies the base entry within the realm. This entry and its descendents are part of the realm.

Repository identifier:

Specifies a unique identifier for the repository. This identifier uniquely identifies the repository within the cell.

Repository type:

Specifies the repository type, such as File or LDAP.

Using a single built-in, file-based repository in a new configuration under Federated repositories

Follow this task to use a single built-in, file-based repository in a new configuration under Federated repositories.

To use the default configuration under Federated repositories that includes a single built-in, file-based repository only, you need to know the primary administrative user name of the user who manages WebSphere Application Server resources and user accounts.

Restriction: Client certificate login is not supported in a realm that includes a single built-in, file-based repository or a single built-in, file-based repository with other repositories.

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Leave the Realm name field value as defaultWIMFileBasedRealm.
4. Enter the name of the primary administrative user in the Primary administrative user name field, for example, adminUser.
5. Leave the **Ignore case for authorization** option enabled.
6. Click **OK**.
7. Provide an administrative user password. This panel displays only when a built-in, file-based repository is included in the realm. Otherwise, the panel does not display. If a built-in, file-based repository is included, complete the following steps:
 - a. Supply a password for the administrative user in the Password field.
 - b. Confirm the password of the primary administrative user in the Confirm password field.
 - c. Click **OK**.

After completing these steps, your new configuration under Federated repositories includes a single built-in, file-based repository only.

1. Before you can manage this account with Users and Groups, configure supported entity types as described in “Configuring supported entity types in a federated repository configuration” on page 147.
2. After configuring the federated repositories, click **Security > Secure administration, applications, and infrastructure** to return to the Secure administration, applications, and infrastructure panel. Verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Secure administration, applications, and infrastructure panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
3. If you are enabling security, complete the remaining steps, as specified in “Enabling security for the realm” on page 74. As the final step, validate this setup by clicking **Apply** in the Secure administration, applications, and infrastructure panel.

4. Save, stop, and restart all the product servers (deployment managers, nodes, and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Administrative user password settings:

Use this page to set a password for the administrative user who manages the product resources and user accounts.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. If your federated repository configuration includes a built-in, file-based repository, then the **Administrative user password** panel displays when changes are applied.

When you finish adding or updating your federated repository configuration, go to the **Security > Secure administration, applications, and infrastructure** panel and click **Apply** to validate the changes.

Password:

Specifies the password of the administrative user who manages the product resources and user accounts.

Confirm password:

Confirms the password of the administrative user who manages the product resources and user accounts.

Federated repository wizard settings:

Use this security wizard page to complete the basic requirements to connect the application server to a federated repository.

To view this security wizard page, complete the following steps

1. Click **Security > Secure administration, applications, and infrastructure > Security configuration wizard**.
2. Select your protection settings and click **Next**.
3. Select the **Federated repositories** option and click **Next**.

You can modify your federated repository configuration by completing the following steps:

1. Click **Security > Security administration, applications, and infrastructure**.
2. Under User account repository, select Federated repository and click **Configure**.

Note: This wizard is used for the initial configuration of a built-in, file-based repository. The user name and password do not have to be in the federated repository because they will be created. If you have previously configured federated repositories, do not use the Security configuration wizard to modify your configuration. Instead, modify your configuration using the Federated repositories selection under User account repository on the Secure administration, applications, and infrastructure panel.

Primary administrative user name:

Specifies the name of the user with administrative privileges that is defined in the repository, for example, adminUser.

Password:

Specifies the password of the administrative user who manages the product resources and user accounts.

Confirm password:

Confirms the password of the administrative user who manages the product resources and user accounts.

Changing a federated repository configuration to include a single built-in, file-based repository only

Follow this task to change your federated repository configuration to include a single built-in, file-based repository only.

To change your federated repository configuration to include a single built-in, file-based repository only, you need to know the primary administrative user name of the user who manages WebSphere Application Server resources and user accounts.

Restriction: Client certificate login is not supported in a realm that includes a single built-in, file-based repository or a single built-in, file-based repository with other repositories.

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Enter the name of the realm in the Realm name field. If the realm contains a single built-in, file-based repository only, you must specify defaultWIMFileBasedRealm as the realm name.
4. Enter the name of the primary administrative user in the Primary administrative user name field, for example, adminUser.
5. Enable the **Ignore case for authorization** option.
6. Click **Use built-in repository** if the built-in, file-based repository is not listed in the collection.
7. Select all repositories in the collection that are not of type File and click **Remove**.
8. Click **OK**.
9. Provide an administrative user password. This panel displays only when a built-in, file-based repository is included in the realm. Otherwise, it does not display. If a built-in, file-based repository is included, complete the following steps:
 - a. Supply a password for the primary administrative user in the Password field.
 - b. Confirm the password of the primary administrative user in the Confirm password field.
 - c. Click **OK**.

After completing these steps, your federated repository configuration, which includes a single built-in, file-based repository only, is configured.

1. Before you can manage this account with Users and Groups, configure supported entity types as described in “Configuring supported entity types in a federated repository configuration” on page 147.
2. After configuring the federated repositories, click **Security > Secure administration, applications, and infrastructure** to return to the Secure administration, applications, and infrastructure panel. Verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Secure administration, applications, and infrastructure panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
3. If you are enabling security, complete the remaining steps, as specified in “Enabling security for the realm” on page 74. As the final step, validate this setup by clicking **Apply** in the Secure administration, applications, and infrastructure panel.

4. Save, stop, and restart all the product servers (deployment managers, nodes, and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Configuring a single, Lightweight Directory Access Protocol repository in a new configuration under Federated repositories

Follow this task to configure a single, Lightweight Directory Access Protocol (LDAP) repository in a new configuration under Federated repositories.

To configure an LDAP repository in a new configuration under Federated repositories, you must know a valid user name (ID), the user password, the server host and port and, if necessary, the bind distinguished name (DN) and the bind password. You can choose any valid user in the repository that is searchable. In some LDAP servers, administrative users are not searchable and cannot be used (for example, `cn=root` in SecureWay). This user is referred to as the WebSphere Application Server *administrative user name* or *administrative ID* in the documentation. Being an administrative ID means a user has special privileges when calling some protected internal methods. Normally, this ID and password are used to log in to the administrative console after you turn on security. You can use other users to log in, if those users are part of the administrative roles.

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. On the Federated repositories panel, complete the following steps:
 - a. Enter the name of the realm in the Realm name field. You can change the existing realm name.
 - b. Enter the name of the primary administrative user in the Primary administrative user name field, for example, `adminUser`.
 - c. **Optional:** Select the **Ignore case for authorization** option. When you enable this option, the authorization check is case-insensitive. Normally, an authorization check involves checking the complete DN of a user, which is unique in the realm and is case-insensitive. Clear this option when all of the member repositories in the realm are case-sensitive.

Restriction: Some repositories contain data that is case-sensitive only, and some repositories contain data that is case-insensitive only. Do not include both case-sensitive and case-insensitive repositories in the realm. For example, do not include case-sensitive repositories in the realm with a built-in, file-based repository.

- d. Click **Add base entry to realm** to add a base entry that uniquely identifies the external repository in the realm. Then complete the steps in “Adding an external repository in a federated repository configuration” on page 131.
4. On the Federated repositories panel, complete the following steps:
 - a. Select the built-in, file-based repository in the collection, and click **Remove**.

Restriction: Before you remove the built-in, file-based repository from the administrative realm, verify that at least one user in another member repository is a console user with administrative rights. Otherwise, you must disable security to regain access to the administrative console.

- b. Click **OK**.

After completing these steps, your new configuration under Federated repositories includes a single, LDAP repository only.

1. Before you can manage this account with Users and Groups, configure supported entity types as described in “Configuring supported entity types in a federated repository configuration” on page 147.
2. After configuring the federated repositories, click **Security > Secure administration, applications, and infrastructure** to return to the Secure administration, applications, and infrastructure panel. Verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is

- not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Secure administration, applications, and infrastructure panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
3. If you are enabling security, complete the remaining steps as specified in “Enabling security for the realm” on page 74. As the final step, validate this setup by clicking **Apply** in the Secure administration, applications, and infrastructure panel.
 4. Save, stop, and restart all the product servers (deployment managers, nodes and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Changing a federated repository configuration to include a single, Lightweight Directory Access Protocol repository only

Follow this task to change your federated repository configuration to include a single, Lightweight Directory Access Protocol repository (LDAP) repository only.

To configure an LDAP repository in a federated repository configuration, you must know a valid user name (ID), the user password, the server host and port and, if necessary, the bind distinguished name (DN) and the bind password. You can choose any valid user in the repository that is searchable. In some LDAP servers, administrative users are not searchable and cannot be used (for example, cn=root in SecureWay). This user is referred to as a WebSphere Application Server *administrative user name* or *administrative ID* in the documentation. Being an administrative 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 you turn on security. You can use other users to log in if those users are part of the administrative roles.

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Enter the name of the realm in the Realm name field. You can change the existing realm name.
4. Enter the name of the primary administrative user in the Primary administrative user name field, for example, adminUser.
5. **Optional:** Select the **Ignore case for authorization** option. When you enable this option, the authorization check is case-insensitive. Normally, an authorization check involves checking the complete DN of a user, which is unique in the realm and is case-insensitive. Clear this option when all of the member repositories in the realm are case-sensitive.

Restriction: Some repositories contain data that is case-sensitive only, and some repositories contain data that is case-insensitive only. Do not include both case-sensitive and case-insensitive repositories in the realm. For example, do not include case-sensitive repositories in the realm with a built-in, file-based repository.

6. **Optional:** Click **Add base entry to realm** if the LDAP repository that you need is not contained in the collection. Then complete the steps in “Adding an external repository in a federated repository configuration” on page 131.
7. On the Federated repositories panel, complete the following steps:
 - a. **Optional:** Select the repositories in the collection that you do not need in the realm and click **Remove**.

Restriction: The realm must always contain at least one base entry; therefore, you cannot remove every entry.

- b. Click **OK**.

After completing these steps, your federated repository configuration, which includes a single LDAP repository only, is configured.

1. Before you can manage this account with Users and Groups, configure supported entity types as described in “Configuring supported entity types in a federated repository configuration” on page 147.
2. After configuring the federated repositories, click **Security > Secure administration, applications, and infrastructure** to return to the Secure administration, applications, and infrastructure panel. Verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Secure administration, applications, and infrastructure panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
3. If you are enabling security, complete the remaining steps as specified in “Enabling security for the realm” on page 74. As the final step, validate this setup by clicking **Apply** in the Secure administration, applications, and infrastructure panel.
4. Save, stop, and restart all the product servers (deployment managers, nodes, and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Configuring multiple Lightweight Directory Access Protocol repositories in a federated repository configuration

Follow this task to configure multiple Lightweight Directory Access Protocol (LDAP) repositories in a federated repository configuration.

To configure an LDAP repository in a federated repository configuration, you must know a valid user name (ID), the user password, the server host and port and, if necessary, the bind distinguished name (DN) and the bind password. You can choose any valid user in the repository that is searchable. In some LDAP servers, administrative users are not searchable and cannot be used (for example, cn=root in SecureWay). This user is referred to as a WebSphere Application Server *administrative user name* or *administrative ID* in the documentation. Being an administrative 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 you turn on security. You can use other users to log in if those users are part of the administrative roles.

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Enter the name of the realm in the Realm name field. You can change the existing realm name.
4. Enter the name of the primary administrative user in the Primary administrative user name field, for example, adminUser.
5. **Optional:** Select the **Ignore case for authorization** option. When you enable this option, the authorization check is case-insensitive. Normally, an authorization check involves checking the complete DN of a user, which is unique in the realm and is case-insensitive. Clear this option when all of the member repositories in the realm are case-sensitive.

Restriction: Some repositories contain data that is case-sensitive only, and some repositories contain data that is case-insensitive only. Do not include both case-sensitive and case-insensitive repositories in the realm. For example, do not include case-sensitive repositories in the realm with a built-in, file-based repository.

6. **Optional:** Click **Add base entry to realm** if the LDAP repository that you need is not listed in the collection. Then complete the steps in “Adding an external repository in a federated repository configuration” on page 131.
7. On the Federated repositories panel, complete the following steps:
 - a. **Optional:** Repeat step 6 if the LDAP repository that you need is not listed in the collection.
 - b. **Optional:** Select the repositories in the collection that you do not need in the realm and click **Remove**. The following restrictions apply:

- The realm must always contain at least one base entry; therefore, you cannot remove every entry.
 - If you plan to remove the built-in, file-based repository from the administrative realm, verify that at least one user in another member repository is a console user with administrative rights. Otherwise, you must disable security to regain access to the administrative console.
- c. Click **OK**.

After completing these steps, your federated repository configuration, which includes multiple LDAP repositories, is configured.

1. Before you can manage this account with Users and Groups, configure supported entity types as described in “Configuring supported entity types in a federated repository configuration” on page 147.
2. After configuring the federated repositories, click **Security > Secure administration, applications, and infrastructure** to return to the Secure administration, applications, and infrastructure panel. Verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Secure administration, applications, and infrastructure panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
3. If you are enabling security, complete the remaining steps as specified in “Enabling security for the realm” on page 74. As the final step, validate this setup by clicking **Apply** in the Secure administration, applications, and infrastructure panel.
4. Save, stop, and restart all the product servers (deployment managers, nodes, and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Configuring a single built-in, file-based repository and one or more Lightweight Directory Access Protocol repositories in a federated repository configuration

Follow this task to configure a single built-in, file-based repository and multiple Lightweight Directory Access Protocol (LDAP) repositories in a federated repository configuration.

To configure a built-in, file-based repository in a federated repository configuration, you must know the primary administrative user name of the user who manages WebSphere Application Server resources and user accounts.

To configure an LDAP repository in a federated repository configuration, you must know a valid user name (ID), the user password, the server host and port and, if necessary, the bind distinguished name (DN) and the bind password. You can choose any valid user in the repository that is searchable. In some LDAP servers, administrative users are not searchable and cannot be used (for example, cn=root in SecureWay). This user is referred to as a WebSphere Application Server *administrative user name* or *administrative ID* in the documentation. Being an administrative ID means a user has special privileges when calling some protected internal methods. Normally, this ID and password are used to log in to the administrative console after you turn on security. You can use other users to log in if those users are part of the administrative roles.

Restriction: Client certificate login is not supported in a realm that includes a single built-in, file-based repository or a single built-in, file-based repository with other repositories.

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Enter the name of the realm in the Realm name field. You can change the existing realm name.
4. Enter the name of the primary administrative user in the Primary administrative user name field, for example, adminUser.

Restriction: When you configure multiple repositories that includes a single built-in, file-based repository, the primary administrative user name must exist in the file-based repository. If the primary administrative user name does not exist in the file-based repository, then the name is created in the file-based repository. The primary administrative user name cannot exist in other repositories.

5. Select the **Ignore case for authorization** option.

Attention: When the realm includes a built-in, file-based repository, you must enable the **Ignore case for authorization** option.

When you enable this option, the authorization check is case-insensitive. Normally, an authorization check involves checking the complete DN of a user, which is unique in the realm and is case-insensitive. Clear this option when all of the member repositories in the realm are case-sensitive.

Restriction: Some repositories contain data that is case-sensitive only, and some repositories contain data that is case-insensitive only. Do not include both case-sensitive and case-insensitive repositories in the realm. For example, do not include case-sensitive repositories in the realm with a built-in, file-based repository.

6. **Optional:** Click **Add base entry to realm** if the LDAP repository that you need is not contained in the collection. Then complete the steps in “Adding an external repository in a federated repository configuration” on page 131.
7. On the Federated repositories panel, complete the following steps:
 - a. **Optional:** Repeat step 6 if the LDAP repository that you need is not listed in the collection.
 - b. Click **Use built-in repository** if the built-in, file-based repository is not listed in the collection.
 - c. **Optional:** Select the repositories in the collection that you do not need in the realm and click **Remove**.

Restriction: The realm must always contain at least one base entry; therefore, you cannot remove every entry.

- d. Click **OK**.
8. Provide an administrative user password. This panel displays only when a built-in, file-based repository is included in the realm. Otherwise, the panel does not display. If a built-in, file-based repository is included, complete the following steps:
 - a. Supply a password for the administrative user in the Password field.
 - b. Confirm the password of the primary administrative user in the Confirm password field.
 - c. Click **OK**.

After completing these steps, your federated repository configuration, which includes a single built-in, file-based repository and one or more LDAP repositories, is configured.

1. Before you can manage this account with Users and Groups, configure supported entity types as described in “Configuring supported entity types in a federated repository configuration” on page 147.
2. After configuring the federated repositories, click **Security > Secure administration, applications, and infrastructure** to return to the Secure administration, applications, and infrastructure panel. Verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Secure administration, applications, and infrastructure panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
3. If you are enabling security, complete the remaining steps as specified in “Enabling security for the realm” on page 74. As the final step, validate this setup by clicking **Apply** in the Secure administration, applications, and infrastructure panel.
4. Save, stop, and restart all the product servers (deployment managers, nodes, and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Configuring Lightweight Directory Access Protocol in a federated repository configuration

Follow this topic to configure Lightweight Directory Access Protocol (LDAP) settings in a federated repository configuration.

You have chosen among various ways to configure LDAP:

- “Configuring a single, Lightweight Directory Access Protocol repository in a new configuration under Federated repositories” on page 121
- “Changing a federated repository configuration to include a single, Lightweight Directory Access Protocol repository only” on page 122
- “Configuring multiple Lightweight Directory Access Protocol repositories in a federated repository configuration” on page 123
- “Configuring a single built-in, file-based repository and one or more Lightweight Directory Access Protocol repositories in a federated repository configuration” on page 124
- “Managing repositories in a federated repository configuration” on page 149

At this point, you are viewing the LDAP repository configuration page of the administrative console.

1. Enter a unique identifier for the repository in the Repository identifier field. This identifier uniquely identifies the repository within the cell, for example: LDAP1.
2. Select the type of LDAP server that is used from the Directory type list. The type of LDAP server determines the default filters that are used by WebSphere Application Server.
IBM Tivoli Directory Server users can choose either IBM Tivoli Directory Server or SecureWay as the directory type. Use the IBM Tivoli Directory Server directory type for better performance. For a list of supported LDAP servers, see “Using specific directory servers as the LDAP server” on page 106.
3. Enter the fully qualified host name of the primary LDAP server in the Primary host name field. You can enter either the IP address or the domain name system (DNS) name.
4. Enter the server port of the LDAP directory in the Port field. The host name and the port number represent the realm for this LDAP server in a mixed version nodes cell. 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.

The default value is 389, which is not a Secure Sockets Layer (SSL) connection. Use port 636 for a Secure Sockets Layer (SSL) connection. For some LDAP servers, you can specify a different port for a non-SSL or SSL connection. If you do not know the port to use, contact your LDAP server administrator.

If multiple WebSphere Application Servers are installed and configured to run in the same single sign-on domain, or if WebSphere Application Server interoperates with a previous version of 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 5.x or 6.0.x configuration, and WebSphere Application Server at Version 6.1 is going to interoperate with the Version 5.x or 6.0.x server, then verify that port 389 is specified explicitly for the Version 6.1 server.

5. **Optional:** Enter the host name of the failover LDAP server in the Failover host name field. You can specify a secondary directory server to be used in the event that your primary directory server becomes unavailable. After switching to a secondary directory server, LDAP repository attempts to reconnect to the primary directory server every 15 minutes.
6. **Optional:** Enter the port of the failover LDAP server in the Port field and click **Add**. The default value is 389, which is not a Secure Sockets Layer (SSL) connection. Use port 636 for a Secure Sockets Layer (SSL) connection. For some LDAP servers, you can specify a different port for a non-SSL or SSL connection. If you do not know the port to use, contact your LDAP server administrator.
7. **Optional:** Select the type of *referral*. A referral is an entity that is used to redirect a client request to another LDAP server. A referral contains the names and locations of other objects. It is sent by the server to indicate that the information that the client requested can be found at another location, possibly at another server or several servers. The default value is ignore.

ignore

Referrals are ignored.

follow Referrals are followed automatically.

8. **Optional:** Enter the bind DN name in the Bind distinguished name field, for example, cn=root. The bind DN is required if anonymous binds are not possible on the LDAP server to obtain user and group information or for write operations. In most cases, bind DN and bind password are needed. However, when anonymous bind can satisfy all of the required functions, bind DN and bind password are not needed. If the LDAP server is set up to use anonymous binds, leave this field blank. If a name is not specified, the application server binds anonymously.
9. **Optional:** Enter the password that corresponds to the bind DN in the Bind password field.
10. **Optional:** Enter the property names to use to log into WebSphere Application Server in the Login properties field. This field takes multiple login properties, delimited by a semicolon (;). For example, uid;mail.

All login properties are searched during login. If multiple entries or no entries are found, an exception is thrown. For example, if you specify the login properties as uid;mail and the login ID as Bob, the search filter searches for uid=Bob or mail=Bob. When the search returns a single entry, then authentication can proceed. Otherwise, an exception is thrown.

11. **Optional:** Select the certificate map mode in the Certificate mapping field. You can use the X.590 certificates for user authentication when LDAP is selected as the repository. The Certificate mapping 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.
12. If you select **CERTIFICATE_FILTER** in the Certificate mapping field, 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 the result is 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}

13. **Optional:** Select the **Require SSL communications** option if you want to use Secure Sockets Layer communications with the LDAP server.

If you select the **Require SSL communications** option, you can select either the **Centrally managed** or **Use specific SSL alias** option.

Centrally managed

Enables you to specify an SSL configuration for a particular scope, such as the cell, node, server, or cluster in one location. To use the Centrally managed option, you must specify the

SSL configuration for the particular set of endpoints. The Manage endpoint security configurations and trust zones panel displays all of the inbound and outbound endpoints that use the SSL protocol. If you expand the Inbound or Outbound section of the panel and click the name of a node, you can specify an SSL configuration that is used for every endpoint on that node. For an LDAP registry, you can override the inherited SSL configuration by specifying an SSL configuration for LDAP. To specify an SSL configuration for LDAP, complete the following steps:

- a. Click **Security > SSL certificate and key management > Manage endpoint security configurations and trust zones**.
- b. Expand **Outbound > cell_name > Nodes > node_name > Servers > server_name > LDAP**.

Use specific SSL alias

Select the **Use specific SSL alias** option if you intend to select one of the SSL configurations in the menu that follows the option.

This configuration is used only when SSL is enabled for LDAP. The default is *DefaultSSLSettings*. To modify or create a new SSL configuration, complete the following steps:

- a. Click **Security > SSL certificate and key management**.
- b. Under Configuration settings, click **Manage endpoint security configurations and trust zones > configuration_name**.
- c. Under Related items, click **SSL configurations**.

14. Click **OK**.

After completing these steps, your LDAP repository settings are configured.

Return to the appropriate task to complete the steps for your federated repository configuration:

- “Configuring a single, Lightweight Directory Access Protocol repository in a new configuration under Federated repositories” on page 121
- “Changing a federated repository configuration to include a single, Lightweight Directory Access Protocol repository only” on page 122
- “Configuring multiple Lightweight Directory Access Protocol repositories in a federated repository configuration” on page 123
- “Configuring a single built-in, file-based repository and one or more Lightweight Directory Access Protocol repositories in a federated repository configuration” on page 124
- “Managing repositories in a federated repository configuration” on page 149

Lightweight Directory Access Protocol repository configuration settings:

Use this page to configure secure access to a Lightweight Directory Access Protocol (LDAP) repository with optional failover servers.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add** to specify a new external repository or select an external repository that is preconfigured.

When you finish adding or updating your federated repository configuration, go to the **Security > Secure administration, applications, and infrastructure** panel and click **Apply** to validate the changes.

Repository identifier:

Specifies a unique identifier for the LDAP repository. This identifier uniquely identifies the repository within the cell, for example: LDAP1.

Directory type:

Specifies the type of LDAP server to which you connect.

Expand the drop-down list to display a list of LDAP directory types.

Primary host name:

Specifies the host name of the primary LDAP server. This host name is either an IP address or a domain name service (DNS) name.

Port:

Specifies the LDAP server port.

The default value is 389, which is not a Secure Sockets Layer (SSL) connection. Use port 636 for a Secure Sockets Layer (SSL) connection. For some LDAP servers, you can specify a different port for a non-SSL or SSL connection. If you do not know the port to use, contact your LDAP server administrator.

Data type:	Integer	
Default:	389	
Range:	389, which is not a Secure Sockets Layer (SSL) connection	636, which is a Secure Sockets Layer (SSL) connection

Failover host name:

Specifies the host name of the failover LDAP server.

You can specify a secondary directory server to be used in the event that your primary directory server becomes unavailable. After switching to a secondary directory server, the LDAP repository attempts to reconnect to the primary directory server every 15 minutes.

Port:

Specifies the port of the failover LDAP server.

The default value is 389, which is not a Secure Sockets Layer (SSL) connection. Use port 636 for a Secure Sockets Layer (SSL) connection. For some LDAP servers, you can specify a different port for a non-SSL or SSL connection. If you do not know the port to use, contact your LDAP server administrator.

Data type:	Integer	
Range:	389, which is not a Secure Sockets Layer (SSL) connection	636, which is a Secure Sockets Layer (SSL) connection

Support referrals to other LDAP servers:

Specifies how referrals that are encountered by the LDAP server are handled.

A referral is an entity that is used to redirect a client request to another LDAP server. A referral contains the names and locations of other objects. It is sent by the server to indicate that the information that the client requested can be found at another location, possibly at another server or several servers. The default value is ignore.

Default: ignore
Range: **ignore** Referrals are ignored.
follow Referrals are followed automatically.

Bind distinguished name:

Specifies the distinguished name (DN) for the application server to use when binding to the LDAP repository.

If no name is specified, the application server binds anonymously. In most cases, bind DN and bind password are needed. However, when anonymous bind can satisfy all of the required functions, bind DN and bind password are not needed.

Bind password:

Specifies the password for the application server to use when binding to the LDAP repository.

Login properties:

Specifies the property names to use to log into the application server.

This field takes multiple login properties, delimited by a semicolon (;). For example, `uid;mail`. All login properties are searched during login. If multiple entries or no entries are found, an exception is thrown. For example, if you specify the login properties as `uid;mail` and the login ID as Bob, the search filter searches for `uid=Bob` or `mail=Bob`. When the search returns a single entry, then authentication can proceed. Otherwise, an exception is thrown.

Certificate mapping:

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.

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 repository.

If more than one LDAP entry matches the filter specification at run time, authentication fails because the result is 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}`

Require SSL communications:

Specifies whether secure socket communication is enabled to the LDAP server.

When enabled, the Secure Sockets Layer (SSL) settings for LDAP are used, if specified.

Centrally managed:

Specifies that the selection of an SSL configuration is based upon the outbound topology view for the Java Naming and Directory Interface (JNDI) platform.

Centrally managed configurations support one location to maintain SSL configurations, rather than spreading them across the configuration documents.

Default:	Enabled
Range:	Enabled or Disabled

Use specific SSL alias:

Specifies the SSL configuration alias to use for LDAP outbound SSL communications.

This option overrides the centrally managed configuration for the JNDI platform.

Adding an external repository in a federated repository configuration

Follow this task to add an external repository into a federated repository configuration.

1. If the Lightweight Directory Access Protocol (LDAP) repository that you want to add to your federated repository configuration is previously configured, select the corresponding Repository on the Repository reference panel. To access the Repository reference panel, complete the following steps:
 - a. Click **Security > Secure administration, applications, and infrastructure**.
 - b. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
 - c. Click **Add base entry to realm**.
2. Enter a distinguished name for the realm base entry in the Distinguished name that uniquely identifies... field. This base entry must uniquely identify the external repository in the realm. If multiple repositories are included in the realm, use this field to define an additional distinguished name (DN) that uniquely identifies this set of entries within the realm. For example, repositories LDAP1 and LDAP2 might both use `o=ibm,c=us` as the base entry in the repository. Use the DN in this field to uniquely identify this set of entries in the realm. For example: `o=ibm,c=us` for LDAP1 and `o=ibm2,c=us` for LDAP2. The specified DN in this field maps to the LDAP DN of the base entry within the repository.
3. Enter the LDAP DN of the base entry within the repository in the Distinguished name of a base entry... field. The base entry indicates the starting point for searches in this LDAP directory server. This entry and its descendents are mapped to the subtree that is identified by this unique base name entry field. For example, for a user with a DN of `cn=John Doe, ou=Rochester, o=IBM, c=US`, specify the LDAP base entry as any of the following options:
`ou=Rochester, o=IBM, c=us` or `o=IBM, c=us` or `c=us`

In most cases, this LDAP DN is the same as the distinguished name for the realm base entry.

If this field is left blank, then the subtree defaults to the root of the LDAP repository. Consult your LDAP administrator to determine if your LDAP repository provides support to search from the root, or create users and groups under the root without defining a suffix beforehand.

In WebSphere Application Server, the distinguished name is normalized according to the LDAP specification. Normalization consists of removing spaces in the base distinguished name 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`.

4. If the LDAP repository that you want to add to your realm is not previously configured, complete the following steps:
 - a. Click **Add Repository** on the Repository reference panel to configure the LDAP repository. See step 1 to access the Repository reference panel.
 - b. Configure LDAP on the LDAP configuration panel, as described in “Configuring Lightweight Directory Access Protocol in a federated repository configuration” on page 126.
 - c. Select the new Repository on the Repository reference panel.
5. Click **OK**.

You have added a new or previously configured external repository into your federated repository configuration.

1. Before you can manage this account with Users and Groups, configure supported entity types as described in “Configuring supported entity types in a federated repository configuration” on page 147.
2. After configuring the federated repositories, click **Security > Secure administration, applications, and infrastructure** to return to the Secure administration, applications, and infrastructure panel. Verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Secure administration, applications, and infrastructure panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
3. If you are enabling security, complete the remaining steps as specified in “Enabling security for the realm” on page 74. As the final step, validate this setup by clicking **Apply** in the Secure administration, applications, and infrastructure panel.
4. Save, stop, and restart all the product servers (deployment managers, nodes and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Configuring a property extension repository in a federated repository configuration

Follow this task to configure a property extension repository to store attributes that cannot be stored in your Lightweight Directory Access Protocol (LDAP) server.

For security and business reasons, you might not want to not allow write operations to your repositories. However, applications calling the federated repository configuration might need to store additional properties for the entities. A federated repository configuration provides a *property extension repository*, which is a database regardless of the type of main profile repositories, for a property-level join configuration. For example, a company that uses an LDAP directory for its internal employees and a database for external customers and business partners might not allow write access to its LDAP and its database. The company can use the *property extension repository* in a federated repository configuration to store additional properties for the people in those repositories, excluding the user ID. When an application uses the federated repository configuration to retrieve an entry for a person, the federated repository configuration transparently joins the properties of the person that is retrieved from either the LDAP or the customer’s database with the properties of the person that is retrieved from the property extension repository into a single logical person entry.

When you configure a property extension repository, you can supply a valid data source, a direct connection configuration, or both. The system first tries to connect by way of the data source. If the data source is not available, then the system uses the direct access configuration.

Restriction: You cannot configure a property extension repository in a mixed-version deployment manager cell.

1. Configure the WebSphere Application Server data source. See “Configuring the WebSphere Application Server data source” on page 143.
2. If you are adding new properties (including properties that are stored in the property extension repository) to the schema, you must do the following before you create the property extension repository.
 - a. Open or create the `wimxmlextension.xml` file under the `<WAS61>\profiles\<propfile_name>\config\cells\<cell_name>\wim\model` directory.

Attention: Make sure the editor is on the deployment manager node.
 - b. Add the schema definition of the new property. The following sample `wimxmlextension.xml` file adds a new property called `ibm-otherEmail` to both the `Person` and `PersonAccount` entity types. This new property type is “String” and it is multiple-valued.

```
<sdo:datagraph xmlns:sdo="commonj.sdo"
  xmlns:wim="http://www.ibm.com/websphere/wim">
  <wim:schema>
    <wim:propertySchema nsURI="http://www.ibm.com/websphere/wim"
      dataType="String"
      multiValued="true" propertyName="ibm-otherEmail">
      <wim:applicableEntityTypeNames>Person</wim:applicableEntityTypeNames>
      <wim:applicableEntityTypeNames>PersonAccount
      </wim:applicableEntityTypeNames>
    </wim:propertySchema>
  </wim:schema>
</sdo:datagraph>
```

Available data types are defined in `com.ibm.websphere.wim.SchemaConstants`. For example:

```
/**
 * Instance Class: java.lang.String
 */
String DATA_TYPE_STRING = "String";
/**
 * Instance Class: int
 */
String DATA_TYPE_INT = "Int";
/**
 * Instance Class: java.lang.Object
 */
String DATA_TYPE_DATE = "Date";
/**
 * Instance Class: dobjava.lang.Object
 */
String DATA_TYPE_ANY_SIMPLE_TYPE = "AnySimpleType";
/**
 * Instance Class: java.lang.String
 */
String DATA_TYPE_ANY_URI = "AnyURI";
/**
 * Instance Class: java.lang.boolean
 */
String DATA_TYPE_BOOLEAN = "Boolean";
/**
 * Instance Class: long
 */
String DATA_TYPE_LONG = "Long";
/**
 * Instance Class: double
 */
```

```
String DATA_TYPE_DOUBLE = "Double";
/**
 * Instance Class: short
 */
String DATA_TYPE_SHORT = "Short";
```

- c. Add the new property to the property extension repository. Before running the `setupIdMgrPropertyExtensionRepositoryTables` command, add the new properties into `<WAS61>\profiles\<propfile_name>\config\cells\<cell_name>\wim\config\wimlaproperties.xml`.
 - d. Follow the example inside this file to define the new property definitions. The schema file for `wimlaproperties.xml` is `wimdbproperty.xsd` and is in the same directory. It can be used for reference.
 - e. Run the `setupIdMgrPropertyExtensionRepositoryTables` command to create the property extension repository and to add the new properties.
3. Set up the property extension repository using `wsadmin` by following the procedure discussed in "Setting up an entry mapping repository, a property extension repository, or a database repository using `wsadmin` commands" on page 136; ignore the "Before you begin" options.
 4. Configure the property extension repository by completing the following steps:
 - a. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
 - b. Under User account repository, select **Federated repositories**, and click **Configure**.
 - c. Click **Property extension repository**.
 - d. Supply the name of the data source in the Data source name field.
 - e. Select the type of database that is used for the property extension repository.
 - f. Supply the name of the Java database connectivity (JDBC) driver in the JDBC driver field.

Values include:

DB2 `COM.ibm.db2.jdbc.app.DB2Driver`

Oracle
`oracle.jdbc.driver.OracleDriver`

Informix
`com.informix.jdbc.IfxDriver`

Microsoft SQL Server
`com.microsoft.jdbc.sqlserver.SQLServerDriver`

Derby `org.apache.derby.jdbc.EmbeddedDriver`

DB2 for z/OS
`com.ibm.db2.jcc.DB2Driver`

DB2 for iSeries
`com.ibm.db2.jdbc.app.DB2Driver`

- g. Supply the database URL that is used to access the property extension repository with JDBC in the Database URL field. Use an alphanumeric text string that conforms to the standard JDBC URL syntax.

Values include:

DB2 `jdbc:db2:wim`

Oracle
`jdbc:oracle:thin:@<hostname>:1521:orcl`

Derby `jdbc:derby:c:\derby\wim`

Microsoft SQL Server
`jdbc:microsoft:sqlserver://<hostname>:1433;databaseName=wim;selectmethod=cursor;`

Informix

```
jdbc:informix-sqli://<hostname>:1526/wim:INFORMIXSERVER=<IFXServerName>;
```

- h. Supply the user name of the database administrator in the Database administrator user name field.
- i. Supply the password of the database administrator in the Password field.
- j. Specify the entity retrieval limit in the Entity retrieval limit field. The entity retrieval limit is the maximum number of entities that the system can retrieve from the property extension repository with a single database query. The default value is 200.
- k. Click **OK**.

After completing these steps, your federated repository configuration, which includes a property extension repository, is configured.

1. If you are enabling security, complete the remaining steps as specified in “Enabling security for the realm” on page 74. As the final step, validate this setup by clicking **Apply** in the Secure administration, applications, and infrastructure panel.
2. Save, stop, and restart all the product servers (deployment managers, nodes, and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Property extension repository settings:

Use this page to configure a property extension repository that is used to store attributes that cannot be stored in existing repositories.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Under Additional properties, click **Property extension repository**.

When you finish adding or updating your federated repository configuration, go to the **Security > Secure administration, applications, and infrastructure** panel and click **Apply** to validate the changes.

Data source name:

Specifies the Java Naming and Directory Interface (JNDI) name of the data source that is used to access the property extension repository.

Default: jdbc/wimDS

Database type:

Specifies the type of database that is used for the property extension repository.

Default: DB2

JDBC driver:

Specifies the Java Database Connectivity (JDBC) driver that is used to access the entry mapping repository.

Values include:

DB2 COM.ibm.db2.jdbc.app.DB2Driver

Oracle`oracle.jdbc.driver.OracleDriver`**Informix**`com.informix.jdbc.IfxDriver`**Microsoft SQL Server**`com.microsoft.jdbc.sqlserver.SQLServerDriver`**Derby**`org.apache.derby.jdbc.EmbeddedDriver`**DB2 for z/OS**`com.ibm.db2.jcc.DB2Driver`**DB2 for iSeries**`com.ibm.db2.jdbc.app.DB2Driver`*Database URL:*

Specifies the Web address for the property extension repository.

Values include:

DB2 `jdbc:db2:wim`**Oracle**`jdbc:oracle:thin:@<hostname>:1521:orcl`**Derby**`jdbc:derby:c:\derby\wim`**Microsoft SQL Server**`jdbc:microsoft:sqlserver://<hostname>:1433;databaseName=wim;selectmethod=cursor;`**Informix**`jdbc:informix-sqli://<hostname>:1526/wim:INFORMIXSERVER=<IFXServerName>;`*Database administrator user name:*

Specifies the user name of the database administrator that is used to access the property extension repository.

Password:

Specifies the password that is used to enable the database administrator to access the property extension repository.

Entity retrieval limit:

Specifies the maximum number of entities that the system can retrieve from the property extension repository with a single database query.

Data type:

Integer

Default:

200

Setting up an entry mapping repository, a property extension repository, or a database repository using wsadmin commands:

You can set up an entry mapping repository, a property extension repository, or a database repository using wsadmin commands.

If you are setting up an entry mapping repository, begin with the steps described in “Configuring an entry mapping repository in a federated repository configuration” on page 144.

If you are setting up a property extension repository, begin with the steps described in “Configuring a property extension repository in a federated repository configuration” on page 132.

Three types of repositories are currently supported: DB2 repository, property extension repository, and entry mapping repository. When a repository is created, use the appropriate wsadmin command to define the database schema and to populate the database property definitions.

1. Create an empty entry mapping repository as shown in the following examples:
 - a. For DB2, open a DB2 command window or command center and enter the following:

```
db2 create database <name> using codeset UTF-8 territory US
```
 - b. Enter the following database tuning commands:

```
db2 update database configuration for <name> using applheapsz 1024
db2 update database configuration for <name> using stmtheap 4096
db2 update database configuration for <name> using app_ct1_heap_sz 2048
db2 update database configuration for <name> using locklist 1024
db2 update database configuration for <name> using indexrec RESTART
db2 update database configuration for <name> using logfilsiz 1000
db2 update database configuration for <name> using logprimary 12
db2 update database configuration for <name> using logsecond 10
db2 update database configuration for <name> using sortheap 2048
db2set DB2_RR_TO_RS=yes
```
 - c. **Optional:** For Informix databases using dbaccess, enter the following command:

```
CREATE DATABASE <name> WITH BUFFERED LOG
```
 - d. **Optional:** For Oracle databases, the database should already exist during Oracle installation (for example, orcl).
2. Run the `setupIdMgrDBTables` command by doing the following:
 - a. Start WebSphere Application Server.
 - b. Open a command window and go to the `<WAS>/Profiles/<PROFILE_NAME>bin` directory.
 - c. Start wsadmin.
 - d. Type the necessary commands as described below.

The `setupIdMgrDBTables` command can be used to:

- Specify the arguments on the command line.
- Specify the arguments in a file.

For the commands below, the `-file` option enables you to specify a file in which some or all of the parameters are specified. To use the `-file` argument on the command line, enter the full path to the file. Parameters in the file must be specified in `key=value` pairs and each must be on its own line. If a parameter is specified on both the command line and in the file, the value on the command line takes precedence.

Note: If an argument is not properly specified on the command line or in the file, a message is returned which states that the argument was not properly specified. This might mean that the argument was not specified at all or was required for a given configuration but was not specified.

If the argument was not specified at all, check that the parameter is specified on the command line or in the file, and that it is properly spelled and has matching case.

If the argument was required for a given configuration but was not specified, it is possible that a value is not required solely by the command but is required for the type of database and configuration you are setting.

For example, if you set the `dn`, `wasAdminId`, or `wasAdminPassword` parameters, you must also specify the `dbDriver` parameter. Additionally, if the `dn`, `wasAdminId` or `wasAdminPassword` parameters are specified, and the `databaseType` is not a Cloudscape 10 Version 1 database, then the `dbAdminId` and `dbAdminPassword` parameters must also be specified.

The `setupIdMgrDBTables` command:

The `setupIdMgrDBTables` command sets up the database, which includes creating and populating the tables in the database. Required arguments are prefixed by a double start (**). Arguments are case-sensitive, both through the command line and the file.

Parameters:

****schemaLocation (String)**

The location of the `<WAS>/etc/wim/setup` directory.

dbPropXML (String)

The location of database repository property definition XML file.

****databaseType (String)**

The type of database. Supported databases are `db2`, `oracle`, `informix`, `cloudscape`, `sqlserver`, `db2zos`, and `db2iseries`.

****dbURL (String)**

The database URL for direct access mode. For example: `jdbc:db2:wim`.

dbDriver (String)

The name of the database driver. For example: `COM.ibm.db2.jdbc.app.DB2Driver`.

dbAdminId (String)

The database administrator ID for direct access mode. For example: `db2admin`.

Note: For a Cloudscape 10 Version 1 embedded database, `dbAdminId` is not required.

dbAdminPassword (String)

The password associated with the `dbAdminId`.

Note: For a Cloudscape 10 Version 1 embedded database, `dbAdminPassword` is not required.

dn (String)

The default organization `uniqueName` to replace. For example: `o=yourco`. If it is not set, `o=DefaultOrganization` is used.

wasAdminId (String)

The WebSphere Application Server admin user ID. The ID should be a short name, not a `uniqueName`. For example: `wasadmin`. After creation, the `uniqueName` is `uid=wasadmin, <defaultOrg>`.

wasAdminPassword (String)

The WebSphere Application Server admin user password. If `wasAdminId` is set, then this parameter is mandatory.

saltLength (Integer)

The salt length of the randomly generated salt for password hashing.

encryptionKey (String)

The password encryption key. Set the password encryption key to match the encryption key in the `wimconfig.xml` file for the repository. If the encryption key is not set, the default is used.

derbySystemHome (String)

The home location of the Cloudscape 10 Version 1 system if you are setting up a Cloudscape 10 Version 1 database.

reportSqlError (String)

Specifies whether to report SQL errors while setting up databases.

file (String)

The full path to a file containing the input parameters. Each input parameter must match a corresponding parameter as it would be typed on the command line, and it must be placed in a key=value pair. Each pair must be on a separate line.

The deleteIdMgrDBTables command:

The deleteIdMgrDBTables command deletes the tables in the database.

Parameters:

****schemaLocation (String)**

The location of the <WAS>/etc/wim/setup directory.

****databaseType (String)**

The type of database. Supported databases are db2, oracle, informix, cloudscape, sqlserver, db2zos, and db2iseries.

****dbURL (String)**

The database URL for direct access mode. For example: jdbc:db2:wim.

dbDriver (String)

The name of the database driver. For example: COM.ibm.db2.jdbc.app.DB2Driver.

dbAdminId (String)

The database administrator ID for direct access mode. For example: db2admin.

Note: For a Cloudscape 10 Version 1 embedded database, dbAdminId is not required.

dbAdminPassword (String)

The password associated with the dbAdminId.

Note: For a Cloudscape 10 Version 1 embedded database, dbAdminPassword is not required.

derbySystemHome (String)

The home location of the Cloudscape 10 Version 1 system if you are setting up a Cloudscape 10 Version 1 database.

reportSqlError (String)

Specifies whether to report SQL errors while setting up databases.

file (String)

The full path to a file containing the input parameters. Each input parameter must match a corresponding parameter as it would be typed on the command line, and it must be placed in a key=value pair. Each pair must be on a separate line.

The setupIdMgrPropertyExtensionRepositoryTables command:

The setupIdMgrPropertyExtensionRepositoryTables command sets up the property extension repository, which includes creating and populating the tables in the database.

Parameters:

****schemaLocation (String)**

The location of the <WAS>/etc/wim/setup directory.

laPropXML (String)

The location of the property extension repository definition XML file.

****databaseType (String)**

The type of database. Supported databases are db2, oracle, informix, cloudscape, sqlserver, db2zos, and db2iseries.

****dbURL (String)**

The database URL for direct access mode. For example: jdbc:db2:wim.

dbAdminId (String)

The database administrator ID for direct access mode. For example: db2admin.

Note: For a Cloudscape 10 Version 1 embedded database, dbAdminId is not required.

dbAdminPassword (String)

The password associated with the dbAdminId.

Note: For a Cloudscape 10 Version 1 embedded database, dbAdminPassword is not required.

derbySystemHome (String)

The home location of the Cloudscape 10 Version 1 system if you are setting up a Cloudscape 10 Version 1 database.

reportSqlError (String)

Specifies whether to report SQL errors while setting up databases.

file (String)

The full path to a file containing the input parameters. Each input parameter must match a corresponding parameter as it would be typed on the command line, and it must be placed in a key=value pair. Each pair must be on a separate line.

The deleteIdMgrPropertyExtensionRepositoryTables command:

The deleteIdMgrPropertyExtensionRepositoryTables command deletes the tables in the property extension database.

*Parameters:*****schemaLocation (String)**

The location of the <WAS>/etc/wim/setup directory.

****databaseType (String)**

The type of database. Supported databases are db2, oracle, informix, cloudscape, sqlserver, db2zos, and db2iseries.

****dbURL (String)**

The database URL for direct access mode. For example: jdbc:db2:wim.

dbDriver (String)

The name of the database driver. For example: COM.ibm.db2.jdbc.app.DB2Driver.

dbAdminId (String)

The database administrator ID for direct access mode. For example: db2admin.

Note: For a Cloudscape 10 Version 1 embedded database, dbAdminId is not required.

dbAdminPassword (String)

The password associated with the dbAdminId.

Note: For a Cloudscape 10 Version 1 embedded database, dbAdminPassword is not required.

derbySystemHome (String)

The home location of the Cloudscape 10 Version 1 system if you are setting up a Cloudscape 10 Version 1 database.

reportSqlError (String)

Specifies whether to report SQL errors while setting up databases.

file (String)

The full path to a file containing the input parameters. Each input parameter must match a corresponding parameter as it would be typed on the command line, and it must be placed in a key=value pair. Each pair must be on a separate line.

The setupIdMgrEntryMappingRepositoryTables command:

The setupIdMgrEntryMappingRepositoryTables command sets up the entry mapping repository, which includes creating and populating the tables of the repository.

Parameters:

****schemaLocation (String)**

The location of the <WAS>/etc/wim/setup directory.

****databaseType (String)**

The type of database. Supported databases are db2, oracle, informix, cloudscape, sqlserver, db2zos, and db2iseries.

****dbURL (String)**

The database URL for direct access mode. For example: jdbc:db2:wim.

dbDriver (String)

The name of the database driver. For example: COM.ibm.db2.jdbc.app.DB2Driver.

dbAdminId (String)

The database administrator ID for direct access mode. For example: db2admin.

Note: For a Cloudscape 10 Version 1 embedded database, dbAdminId is not required.

dbAdminPassword (String)

The password associated with the dbAdminId.

Note: For a Cloudscape 10 Version 1 embedded database, dbAdminPassword is not required.

derbySystemHome (String)

The home location of the Cloudscape 10 Version 1 system if you are setting up a Cloudscape 10 Version 1 database.

reportSqlError (String)

Specifies whether to report SQL errors while setting up databases.

file (String)

The full path to a file containing the input parameters. Each input parameter must match a corresponding parameter as it would be typed on the command line, and it must be placed in a key=value pair. Each pair must be on a separate line.

The deleteIdMgrEntryMappingRepositoryTables command:

The deleteIdMgrEntryMappingRepositoryTables command deletes the tables in the entry mapping repository.

Parameters:

****schemaLocation (String)**

The location of the <WAS>/etc/wim/setup directory.

****databaseType (String)**

The type of database. Supported databases are db2, oracle, informix, cloudscape, sqlserver, db2zos, and db2iseries.

****dbURL (String)**

The database URL for direct access mode. For example: jdbc:db2:wim.

dbDriver (String)

The name of the database driver. For example: COM.ibm.db2.jdbc.app.DB2Driver.

dbAdminId (String)

The database administrator ID for direct access mode. For example: db2admin.

Note: For a Cloudscape 10 Version 1 embedded database, dbAdminId is not required.

dbAdminPassword (String)

The password associated with the dbAdminId.

Note: For a Cloudscape 10 Version 1 embedded database, dbAdminPassword is not required.

derbySystemHome (String)

The home location of the Cloudscape 10 Version 1 system if you are setting up a Cloudscape 10 Version 1 database.

reportSqlError (String)

Specifies whether to report SQL errors while setting up databases.

file (String)

The full path to a file containing the input parameters. Each input parameter must match a corresponding parameter as it would be typed on the command line, and it must be placed in a key=value pair. Each pair must be on a separate line.

Sample command line usage:

To set up a database using the command line, enter the following:

```
$AdminTask setupIdMgrDBTables {-schemaLocation "C:\WAS7\etc\wim\setup" -dbPropXML
"C:\WAS7\etc\wim\setup\wimdbproperties.xml" -databaseType db2
-dbURL jdbc:db2:wim -dbAdminId db2admin
-dbDriver COM.ibm.db2.jdbc.app.DB2Driver -dbAdminPassword db2adminPwd
-reportSqlError true}
```

To delete database tables using the command line, enter the following:

```
$AdminTask deleteIdMgrDBTables {-schemaLocation "C:\WAS7\etc\wim\setup"
-databaseType db2 -dbURL jdbc:db2:wim -dbAdminId db2admin
-dbDriver COM.ibm.db2.jdbc.app.DB2Driver -dbAdminPassword db2adminPwd
-reportSqlError true}
```

To set up a property extension repository using the command line, enter the following:

```
$AdminTask setupIdMgrPropertyExtensionRepositoryTables {-schemaLocation
"C:\WAS7\etc\wim\setup"
-1aPropXML "C:\WAS7\etc\wim\setup\wim1aproperties.xml" -databaseType db2
-dbURL jdbc:db2:wim -dbAdminId db2admin -dbDriver COM.ibm.db2.jdbc.app.DB2Driver
-dbAdminPassword db2adminPwd -reportSqlError true}
```

To delete a property extension repository using the command line, enter the following:

```
$AdminTask deleteIdMgrPropertyExtensionRepositoryTables {-schemaLocation "C:\WAS7\etc\wim\setup "
-databaseType db2 -dbURL jdbc:db2:wim -dbAdminId db2admin -dbDriver
COM.ibm.db2.jdbc.app.DB2Driver -dbAdminPassword db2adminPwd -reportSqlError true}
```

To set up an entry mapping repository using the command line, enter the following:

```
$AdminTask setupIdMgrEntryMappingRepositoryTables {-schemaLocation "C:\WAS7\etc\wim\setup"  
-databaseType db2 -dbURL jdbc:db2:wim -dbAdminId db2admin -dbDriver  
COM.ibm.db2.jdbc.app.DB2Driver -dbAdminPassword db2adminPwd -reportSqlError true}
```

To delete an entry mapping repository using the command line, enter the following:

```
$AdminTask deleteIdMgrEntryMappingRepositoryTables {-schemaLocation "C:\WAS7\etc\wim\setup"  
-databaseType db2 -dbURL jdbc:db2:wim -dbAdminId db2admin -dbDriver  
COM.ibm.db2.jdbc.app.DB2Driver -dbAdminPassword db2adminPwd -reportSqlError true}
```

Sample CLI Usage using -file option:

To set up a database with the -file option using the example params.txt file below, enter the following:

```
$AdminTask setupIdMgrDBTables {-file C:\params.txt -dbPropXML  
"C:\OverrideDBPropParam\wimdbproperties.xml"}
```

Params.txt

```
schemaLocation=C:\WAS7\etc\wim\setup  
dbPropXML=C:\Program Files\IBM\WebSphere\AppServer\profiles\default  
\config\cells\mycell\wim\config\wimdbproperties.xml  
laPropXML=C:\Program Files\IBM\WebSphere\AppServer\profiles\default  
\config\cells\mycell\wim\config\wimlaproperties.xml  
databaseType=db2  
dbURL=jdbc:db2:wim  
dbDriver=COM.ibm.db2.jdbc.app.DB2Driver  
reportSqlError=true  
dn=o=db.com  
dbAdminId=db2admin  
dbAdminPassword=dbPassword  
wasAdminId=wasadmin  
wasAdminPassword=wasadmin1
```

To set up a database with the -file option using a file only, enter the following:

```
$AdminTask setupIdMgrDBTables {-file C:\params.txt}
```

Note: The use of a file only works if -file is the only parameter specified on the command line. If other parameters are specified then the file is completely ignored, and only the parameters on the command line are used to execute the command.

Configuring the WebSphere Application Server data source:

This section describes how to configure the data source service in WebSphere Application Server.

1. Start the WebSphere Application Server administrative console.
2. Click **Security -> Secure administration, applications and infrastructure**.
3. On the Configuration panel, expand Java Authentication and Authorization Service and click **J2C authentication data**.
4. Click **New** and enter the Alias, User ID and Password.
5. Click **Ok**.
6. Click **Resources -> JDBC -> JDBC Providers**.
7. In the Scope section, choose the **Node level**.
8. Click **New** to create a new JDBC driver.
9. Select the Database type, Provider type, Implementation type and Name.
10. Click **Next** and configure the database class path. Click **Next**.
11. On the Summary page, click **Finish**.
12. In the Additional properties section, click **Data sources**.

13. Click **New** to create a new data source. Enter the Data source name and the JNDI name, and choose the authentication alias from the drop-down list in Component-managed authentication alias. The JNDI name should match the datasourceName value set in wimconfig.xml. By default, it is jdbc/wimDS.

Note: For Cloudscape 10 Version 1 embedded databases, leave the Component-managed authentication alias field set to NONE.

14. Click **Next**.
15. Enter the Database name and deselect **Use this data source in container managed persistence (CMP)**. Click **Next**.
16. Under Component-managed authentication alias, select the authentication alias previously created. Click **Test Connection**. The message should indicate that the connection is successful. Ignore any warnings, and then click **Next**.
17. On the Summary page, click **Finish**.
18. Save the configurations, and restart WebSphere Application Server.

Configuring an entry mapping repository in a federated repository configuration

Follow this task to configure an entry mapping repository that is used to store data for managing profiles on multiple repositories.

An *entry-level join* means that the federated repository configuration uses multiple repositories simultaneously and recognizes the entries in the different repositories as entries representing distinct entities. For example, a company might have a Lightweight Directory Access Protocol (LDAP) directory that contains entries for its employees and a database that contains entries for business partners and customers. By configuring an entry mapping repository, a federated repository configuration can use both the LDAP and the database at the same time. The federated repository configuration hierarchy and constraints for identifiers provide the aggregated namespace for both of those repositories and prevent identifiers from colliding.

When you configure an entry mapping repository, you can supply a valid data source, a direct connection configuration, or both. The system first tries to connect by way of the data source. If the data source is not available, then the system uses the direct access configuration.

Restriction: You cannot configure an entry mapping repository in a mixed-version deployment manager cell.

1. Configure the WebSphere Application Server data source. See “Configuring the WebSphere Application Server data source” on page 143.
2. Set up the entry mapping repository using wsadmin. See “Setting up an entry mapping repository, a property extension repository, or a database repository using wsadmin commands” on page 136.
3. Configure the entry mapping repository into the federated repository by doing the following:
 - a. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
 - b. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
 - c. Click **Entry mapping repository**.
 - d. Supply the name of the data source in the Data source name field.
 - e. Select the type of database that is used for the property extension repository.
 - f. Supply the name of the Java database connectivity (JDBC) driver in the JDBC driver field.

Values include:

DB2 COM.ibm.db2.jdbc.app.DB2Driver

Oracle`oracle.jdbc.driver.OracleDriver`**Informix**`com.informix.jdbc.IfxDriver`**Microsoft SQL Server**`com.microsoft.jdbc.sqlserver.SQLServerDriver`**Derby** `org.apache.derby.jdbc.EmbeddedDriver`**DB2 for z/OS**`com.ibm.db2.jcc.DB2Driver`**DB2 for iSeries**`com.ibm.db2.jdbc.app.DB2Driver`

- g. Supply the database URL that is used to access the property extension repository with JDBC in the Database URL field. Use an alphanumeric text string that conforms to the standard JDBC URL syntax.

Values include:

DB2 `jdbc:db2:wim`

Oracle

`jdbc:oracle:thin:@<hostname>:1521:orcl`

Derby `jdbc:derby:c:\derby\wim`

Microsoft SQL Server

`jdbc:microsoft:sqlserver://<hostname>:1433;databaseName=wim;selectmethod=cursor;`

Informix

`jdbc:informix-sqli://<hostname>:1526/wim:INFORMIXSERVER=<IFXServerName>;`

- h. Supply the user name of the database administrator in the Database administrator user name field.
 i. Supply the password of the database administrator in the Password field.
 j. Click **OK**.

After completing these steps, your federated repository configuration, which includes an entry mapping repository, is configured.

1. After configuring the federated repositories, click **Security > Secure administration, applications, and infrastructure** to return to the Secure administration, applications, and infrastructure panel. Verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Secure administration, applications, and infrastructure panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
2. If you are enabling security, complete the remaining steps as specified in “Enabling security for the realm” on page 74. As the final step, validate this setup by clicking **Apply** in the Secure administration, applications, and infrastructure panel.
3. Save, stop, and restart all the product servers (deployment managers, nodes, and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Entry mapping repository settings:

Use this page to configure an entry mapping repository that is used to store data for managing profiles on multiple repositories.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Under Additional properties, click **Entry mapping repository**.

When you finish adding or updating your federated repository configuration, go to the **Security > Secure administration, applications, and infrastructure** panel and click **Apply** to validate the changes.

Data source name:

Specifies the Java Naming and Directory Interface (JNDI) name of the data source that is used to access the entry mapping repository.

Default: jdbc/wimDS

Database type:

Specifies the type of database that is used to access the entry mapping repository.

Default: DB2

JDBC driver:

Specifies the Java Database Connectivity (JDBC) driver that is used to access the entry mapping repository.

Values include:

DB2 COM.ibm.db2.jdbc.app.DB2Driver

Oracle
oracle.jdbc.driver.OracleDriver

Informix
com.informix.jdbc.IfxDriver

Microsoft SQL Server
com.microsoft.jdbc.sqlserver.SQLServerDriver

Derby org.apache.derby.jdbc.EmbeddedDriver

DB2 for z/OS
com.ibm.db2.jcc.DB2Driver

DB2 for iSeries
com.ibm.db2.jdbc.app.DB2Driver

Database URL:

Specifies the Web address for the entry mapping repository.

Values include:

DB2 jdbc:db2:wim

Oracle
jdbc:oracle:thin:@<hostname>:1521:orcl

Derby jdbc:derby:c:\derby\wim

Microsoft SQL Server

```
jdbc:microsoft:sqlserver://<hostname>:1433;databaseName=wim;selectmethod=cursor;
```

Informix

```
jdbc:informix-sqli://<hostname>:1526/wim:INFORMIXSERVER=<IFXServerName>;
```

Database administrator user name:

Specifies the user name of the database administrator that is used to access the entry mapping repository.

Password:

Specifies the password that is used to enable the database administrator to access the entry mapping repository.

Configuring supported entity types in a federated repository configuration

Follow this task to configure supported entity types for user and group management.

You must configure the supported entity types before you can manage this account with Users and Groups in the administrative console. The supported entity types are Group, OrgContainer, and PersonAccount. A Group entity represents a simple collection of entities that might not have any relational context. An OrgContainer entity represents an organization, such as a company or an enterprise, a subsidiary, or an organizational unit, such as a division, a location, or a department. A PersonAccount entity represents a human being. You cannot add or delete the supported entity types, because these types are predefined.

The Base entry for the default parent determines the repository location where entities of the specified type are placed on write operations by user and group management.

Note: To manage users and groups, click **Users and Groups** in the console navigation tree. Click either **Manage Users** or **Manage Groups**.

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Click **Supported entity types** to view a list of predefined entity types.
4. Click the name of a predefined entity type to change its configuration.
5. Supply the distinguished name of a base entry in the repository in the Base entry for the default parent field. This entry determines the default location in the repository where entities of this type are placed on write operations by user and group management.
6. Supply the relative distinguished name (RDN) properties for the specified entity type in the Relative Distinguished Name properties field. Possible values are `cn` for Group, `uid` or `cn` for PersonAccount, and `o`, `ou`, `dc`, and `cn` for OrgContainer. Delimit multiple properties for the OrgContainer entity with a semicolon (;).

The following list outlines known requirements and limitations that apply to specific Lightweight Directory Access Protocol (LDAP) servers:

Using Microsoft Active Directory as the LDAP server

- Unless you modify the LDAP schema to use `uid`, you must specify `cn` in the Relative Distinguished Name (RDN) properties field for the PersonAccount entity type.
- Secure Sockets Layer communications must be enabled to create users with passwords. To select the **Require SSL communications** option, see the topic “Configuring Lightweight Directory Access Protocol in a federated repository configuration” on page 126.
- Typically the value of user is specified as the value in the Object classes field for the PersonAccount entity type and the value of group is specified as the value in the Object classes field for the Group entity type.

Using a Lotus Domino Enterprise Server as the LDAP server

- Typically, the value of cn is specified in the Relative Distinguished Name (RDN) properties field for the PersonAccount entity type. The value of uid is also acceptable.
- Typically, both inetOrgPerson and dominoPerson are used as values in the Object classes field for the PersonAccount entity type.

Using Sun ONE Directory Server as the LDAP server

- Typically, groupOfUniqueNames is specified as the value in the Object classes field for the Group entity type.

7. Click **OK**.

After completing these steps, your federated repository configuration, which uses supported entity types, is configured.

1. After configuring the federated repositories, click **Security > Secure administration, applications, and infrastructure** to return to the Secure administration, applications, and infrastructure panel. Verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Secure administration, applications, and infrastructure panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
2. If you are enabling security, complete the remaining steps as specified in “Enabling security for the realm” on page 74. As the final step, validate this setup by clicking **Apply** in the Secure administration, applications, and infrastructure panel.
3. Save, stop, and restart all the product servers (deployment managers, nodes, and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Supported entity types collection:

Use this page to list entity types that are supported by the member repositories or to select an entity type to view or change its configuration properties.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Under Additional properties, click **Supported entity types**.

You must configure the supported entity types before you can manage this account with Users and Groups in the administrative console. The Base entry for the default parent determines the repository location where entities of the specified type are placed on write operations by user and group management.

When you finish adding or updating your federated repository configuration, go to the **Security > Secure administration, applications, and infrastructure** panel and click **Apply** to validate the changes.

Related reference

“Lightweight Directory Access Protocol entity types settings” on page 157

Use this page to configure Lightweight Directory Access Protocol (LDAP) entity types that are supported by the member repositories.

Entity type:

Specifies the entity type name.

Base entry for the default parent:

Specifies the distinguished name of a base entry in the repository.

This entry determines the default location in the repository where entities of this type are placed on write operations by user and group management.

Relative Distinguished Name properties:

Specifies the relative distinguished name (RDN) properties for the specified entity type.

Possible values are `cn` for Group, `uid` or `cn` for PersonAccount, and `o`, `ou`, `dc`, and `cn` for OrgContainer. Delimit multiple properties for the OrgContainer entity with a semicolon (;).

Supported entity types settings:

Use this page to configure entity types that are supported by the member repositories.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Under Additional properties, click **Supported entity types**.
4. Click the name of a configured entity type to view or change its configuration.

You must configure the supported entity types before you can manage this account with Users and Groups in the administrative console. The Base entry for the default parent determines the repository location where entities of the specified type are placed on write operations by user and group management.

When you finish adding or updating your federated repository configuration, go to the **Security > Secure administration, applications, and infrastructure** panel and click **Apply** to validate the changes.

Entity type:

Specifies the name of the entity type.

Base entry for the default parent:

Specifies the distinguished name of a base entry in the repository.

This entry determines the default location in the repository where entities of this type are placed on write operations by user and group management.

Relative Distinguished Name properties:

Specifies the relative distinguished name (RDN) properties for the specified entity type.

Possible values are `cn` for Group, `uid` or `cn` for PersonAccount, and `o`, `ou`, `dc`, and `cn` for OrgContainer. Delimit multiple properties for the OrgContainer entity with a semicolon (;).

Managing repositories in a federated repository configuration

Follow this topic to manage repositories in a federated repository configuration.

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.

2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Under Related items, click **Manage repositories**. Repositories that are configured in the system are listed in the collection panel. This list includes repositories that are configured using the federated repository functionality as well as repositories that are created using wsadmin commands described in the topic “Commands for the IdMgrRepositoryConfig group of the AdminTask object” on page 661.
4. **Optional:** Click **Add** to configure a new external repository. The Lightweight Directory Access Protocol (LDAP) repository configuration settings are described in detail in “Configuring Lightweight Directory Access Protocol in a federated repository configuration” on page 126.

Restriction: You cannot add a database repository using the administrative console. This repository configuration is supported by using wsadmin commands only.

5. **Optional:** Click **Delete** to delete a repository that you specified previously using the administrative console or wsadmin commands.

Restriction: You cannot delete the built-in, file-based repository from the collection panel.

6. **Optional:** Select one of the LDAP repository identifier entries to view or update an external repository that is configured in the system previously. The steps to configure LDAP settings are described in detail in “Configuring Lightweight Directory Access Protocol in a federated repository configuration” on page 126.

Restriction: While database repositories that are configured in the system are listed in the collection panel, you cannot update a database repository using the administrative console. Updates to a database repository are supported by using wsadmin commands only.

7. Click **OK**.

After completing these steps, the collection panel under Managing repositories reflects a current list of repositories that are configured in your system.

1. To add one or more external repositories that are listed on this collection panel into the realm, see “Managing the realm in a federated repository configuration” on page 115.
2. After configuring the federated repositories, click **Security > Secure administration, applications, and infrastructure** to return to the Secure administration, applications, and infrastructure panel. Verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Secure administration, applications, and infrastructure panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
3. If you are enabling security, complete the remaining steps as specified in “Enabling security for the realm” on page 74. As the final step, validate this setup by clicking **Apply** in the Secure administration, applications, and infrastructure panel.
4. Save, stop, and restart all the product servers (deployment managers, nodes, and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Manage repositories collection:

Use this page to list repositories that are configured in the system or to select a repository to view or change its configuration properties. You can add or delete external repositories.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.

2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Under Related items, click **Manage repositories**.

When you finish adding or updating your federated repository configuration, go to the **Security > Secure administration, applications, and infrastructure** panel and click **Apply** to validate the changes.

Repository identifier:

Specifies a unique identifier for the repository. This identifier uniquely identifies the repository within the cell.

Repository type:

Specifies the repository type, such as File or LDAP.

Repository reference settings:

Use this page to configure a repository reference. A repository reference is a single repository that contains a set of identity entries that are referenced by a base entry into the directory information tree.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Click **Add base entry to realm**.

When you finish adding or updating your federated repository configuration, go to the **Security > Secure administration, applications, and infrastructure** panel and click **Apply** to validate the changes.

Repository:

Specifies a unique identifier for the repository. This identifier uniquely identifies the repository within the cell.

Expand the drop-down list to display a list of previously defined repository identifiers.

Distinguished name that uniquely identifies this set of entries in the realm:

Specifies the distinguished name (DN) that uniquely identifies this set of entries in the realm.

If multiple repositories are included in the realm, it is necessary to define an additional distinguished name that uniquely identifies this set of entries within the realm.

Distinguished name of a base entry in this repository:

Specifies the Lightweight Directory Access Protocol (LDAP) distinguished name (DN) of the base entry within the repository. The entry and its descendents are mapped to the subtree that is identified by the unique base name entry field.

If this field is left blank, then the subtree defaults to the root of the LDAP repository.

Increasing the performance of the federated repository configuration

Follow this page to manage the realm in a federated repository configuration.

The settings that are available on the Performance panel are independent options that pertain specifically to the federated repositories functionality. These options do not affect your entire WebSphere Application Server configuration.

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Under Related items, click **Manage repositories > repository_name**.
4. Under Additional properties, click **Performance**.
5. **Optional:** Select the **Limit search time** option and enter the maximum number of milliseconds that the Application Server can use to search through your Lightweight Directory Access Protocol (LDAP) entries.
6. **Optional:** Select the **Limit search returns** option and enter the maximum number of entries to return that match the search criteria.
7. **Optional:** Select the **Use connection pooling** option to specify whether the Application Server can store separate connections to the LDAP server for reuse.
8. **Optional:** Select the **Enable context pool** option to specify whether multiple applications can use the same connection to the LDAP server. If you select the option, specify the initial, preferred, and maximum number of entries that can use the same connection. The **Enable context pool** option can be enabled either in conjunction with the **Use connection pool** option or separately. If this option is disabled, a new connection is created for each context. You can also select the **Context pool times out** option and specify the number of seconds after which the entries in the context pool expire.
9. **Optional:** Select the **Cache the attributes** option and specify the maximum number of search attribute entries. This option enables WebSphere Application Server to save the LDAP entries so that it can search the entries locally rather than making multiple calls to the LDAP server. Click the **Cache times out** option that is associated with the **Cache the attributes** option to specify the maximum number of seconds that the Application Server can save these entries.
10. **Optional:** Select the **Cache the search results** option and specify the maximum number of search result entries. This option enables WebSphere Application Server to save the results of a search inquiry instead of making multiple calls to the LDAP server to search and retrieve the results of that search. Click the **Cache times out** option that is associated with the **Cache the search results** option to specify the maximum number of seconds that the Application Server can save the results.

These options are available to potentially increase the performance of your federated repositories configuration. However, the any increase in performance is dependant upon your specific configuration.

Lightweight Directory Access Protocol performance settings:

Use this page to minimize impacts to performance by adding opened connections and contexts to internally maintained pools and reusing them. Also minimize performance impacts by maintaining internal caches of retrieved data.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add** to specify a new external repository or select an external repository that is preconfigured.
5. Under Additional properties, click **Performance**.

When you finish adding or updating your federated repository configuration, go to the **Security > Secure administration, applications, and infrastructure** panel and click **Apply** to validate the changes.

Limit search time:

Specifies the timeout value in milliseconds for a Lightweight Directory Access Protocol (LDAP) server to respond before stopping a request.

Data type:	Integer
Units:	Milliseconds
Default:	0
Range:	Equal to or greater than 0. A value of 0 specifies that no search time limit exists.

Limit search returns:

Specifies the maximum number of entries that are returned in a search result.

Data type:	Integer
Units:	Entries
Default:	0
Range:	Equal to or greater than 0. A value of 0 specifies that no search return limit exists.

Use connection pooling:

Specifies whether to utilize the connection pooling function, which is provided in the Software Development Kit (SDK).

Connection pooling is maintained by the Java run time. It is configured by system properties.

Default:	Disabled
Range:	Enabled or Disabled

Enable context pool:

Specifies whether context pooling is enabled to the LDAP server. To improve performance, use the context pool in combination with connection pooling.

Default:	Enabled
Range:	Enabled or Disabled

Initial size:

Specifies the number of context instances in the pool when the pool is initially created by the LDAP repository.

Data type:	Integer
Default:	1
Range:	1 to 50

Preferred size:

Specifies the preferred number of context instances that the context pool maintains. Both in-use and idle context instances contribute to this number.

Data type:	Integer
Default:	3
Range:	0 to 100

Maximum size:

Specifies the maximum number of context instances that can be maintained concurrently by the context pool. Both in-use and idle context instances contribute to this number.

When the pool size reaches the maximum size, no new context instances can be created for a new request. The new request is blocked until a context instance is released or removed. The request periodically checks for context instances that are available in the pool. A request for a pooled context instance uses an existing pooled and idle context instance or a newly created pooled context instance.

A maximum pool size of 0 indicates that the context pool can maintain an infinite number of context instances.

Data type:	Integer
Default:	0

Context pool times out:

Specifies the number of seconds for the context pool to time out and remove idle context instances.

A timeout value of 0 indicates that the context pool does not time out context instances.

Data type:	Integer
Default:	0

Cache the attributes:

Specifies whether to cache the attributes that are returned from the LDAP server.

Default:	Enabled
Range:	Enabled or Disabled

Cache size:

Specifies the maximum size of the cache.

Data type:	Integer
Default:	4000
Range:	Equal to or greater than 100

Cache times out:

Specifies the maximum number of seconds that the cached search results can stay in the cache.

A timeout value of 0 indicates that the cached search results stay in the cache until update operations are made.

Data type: Integer
Units: Seconds
Default: 1200
Range: Equal to or greater than 0

Cache the search results:

Specifies whether to cache the search results that are returned from the LDAP server.

Default: Enabled
Range: Enabled or Disabled

Cache size:

Specifies the maximum size of the cache.

Data type: Integer
Default: 2000
Range: Equal to or greater than 100

Cache times out:

Specifies the maximum number of seconds that the cached search results can stay in the cache.

A timeout value of 0 indicates that the cached search results stay in the cache until update operations are made.

Data type: Integer
Units: Seconds
Default: 600
Range: Equal to or greater than 0

Configuring Lightweight Directory Access Protocol entity types in a federated repository configuration

Follow this task to configure Lightweight Directory Access Protocol (LDAP) entity types in a federated repository configuration.

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add** to specify a new external repository or select an external repository that is preconfigured.

Note: If you click **Add** to specify a new external repository, you must first complete the required fields and click **Apply** before you can proceed to the next step.

5. Under Additional properties, click **LDAP entity types**.
6. View the entity types that are supported by the member repositories, or select an entity type to view or change its configuration properties.
7. Supply the object classes that are mapped to this entity type in the Object classes field. LDAP entries that contain one or more of the object classes belong to this entity type.

8. Supply the search bases that are used to search this entity type. The search bases specified must be subtrees of the base entry in the repository. For example, you can specify the following search bases, where `o=ibm,c=us` is the base entry in the repository:

`o=ibm,c=us` or `cn=users,o=ibm,c=us` or `ou=austin,o=ibm,c=us`

In the preceding example, you cannot specify search bases `c=us` or `o=ibm,c=uk`.

Delimit multiple search bases with a semicolon (;). For example:

`ou=austin,o=ibm,c=us;ou=raleigh,o=ibm,c=us`

9. Supply the LDAP search filter that is used to search this entity type.
For example, use `(objectclass=ePerson)` to search for users or `(|(objectclass=groupOfNames)(objectclass=groupOfUniqueNames))` to search for groups in an external LDAP repository.

If a search filter is not specified, the object classes and the relative distinguished name (RDN) properties are used to generate the search filter. For information on RDN properties, see “Configuring supported entity types in a federated repository configuration” on page 147.

After completing these steps, LDAP entity types are configured for your LDAP repository.

1. After configuring the federated repositories, click **Security > Secure administration, applications, and infrastructure** to return to the Secure administration, applications, and infrastructure panel. Verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Secure administration, applications, and infrastructure panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
2. If you are enabling security, complete the remaining steps as specified in “Enabling security for the realm” on page 74. As the final step, validate this setup by clicking **Apply** in the Secure administration, applications, and infrastructure panel.
3. Save, stop, and restart all the product servers (deployment managers, nodes, and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Lightweight Directory Access Protocol entity types collection:

Use this page to list Lightweight Directory Access Protocol (LDAP) entity types that are supported by the member repositories or to select an LDAP entity type to view or change its configuration properties.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add** to specify a new external repository or select an external repository that is preconfigured.
5. Under Additional properties, click **LDAP entity types**.

When you finish adding or updating your federated repository configuration, go to the **Security > Secure administration, applications, and infrastructure** panel and click **Apply** to validate the changes.

Entity type:

Specifies the entity type name.

Object classes:

Specifies the object classes that are mapped to this entity type. LDAP entries that contain one or more of the object classes belong to this entity type.

Lightweight Directory Access Protocol entity types settings:

Use this page to configure Lightweight Directory Access Protocol (LDAP) entity types that are supported by the member repositories.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add** to specify a new external repository or select an external repository that is preconfigured.
5. Under Additional properties, click **LDAP entity types**.
6. Select an entity type to view or change its configuration properties.

When you finish adding or updating your federated repository configuration, go to the **Security > Secure administration, applications, and infrastructure** panel and click **Apply** to validate the changes.

Entity type:

Specifies the entity type.

Object classes:

Specifies the object classes that are mapped to this entity type. LDAP entries that contain one or more of the object classes belong to this entity type.

Search bases:

Specifies the search bases that are used to search this entity type.

The search bases specified must be subtrees of the base entry in the repository. For example, you can specify the following search bases, where `o=ibm,c=us` is the base entry in the repository:

```
o=ibm,c=us or cn=users,o=ibm,c=us or ou=austin,o=ibm,c=us
```

In the preceding example, you cannot specify search bases `c=us` or `o=ibm,c=uk`.

Delimit multiple search bases with a semicolon (;). For example:

```
ou=austin,o=ibm,c=us;ou=raleigh,o=ibm,c=us
```

Search filter:

Specifies the LDAP search filter that is used to search this entity type.

For example, use `(objectclass=ePerson)` to search for users or `(|(objectclass=groupOfNames)(objectclass=groupOfUniqueNames))` to search for groups in an external LDAP repository.

If a search filter is not specified, the object classes and the relative distinguished name (RDN) properties are used to generate the search filter.

Configuring group attribute definition settings in a federated repository configuration

Follow this task to configure group definition settings in a federated repository configuration.

Because group attribute definition settings apply only to a Lightweight Directory Access Protocol (LDAP) repository, you must first configure an LDAP repository. For more information, see “Managing repositories in a federated repository configuration” on page 149.

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add** to specify a new external repository or select an external repository that is preconfigured.

Note: If you click **Add** to specify a new external repository, you must first complete the required fields and click **Apply** before you can proceed to the next step.

5. Under Additional properties, click **Group attribute definition**.
6. Supply the name of the group membership attribute in the Name of group membership attribute field. Only one membership attribute can be defined for each LDAP repository.

Every LDAP entry should have this attribute to indicate the groups to which this entry belongs. For example, `memberOf` is the name of the membership attribute that is used in Active Directory. The group membership attribute contains values that reference groups to which this entry belongs. If `UserA` belongs to `GroupA`, then the value of the `memberOf` attribute of `UserA` should contain the distinguished name of `GroupA`.

If your LDAP server does not support the group membership attribute, then do not specify this attribute. The LDAP repository can look up groups by searching the group member attributes, though the performance might be slower.

7. Select the scope of the group membership attribute. The default value is `Direct`.

Direct The membership attribute contains direct groups only. Direct groups are the groups that contain the member. For example, if `Group1` contains `Group2` and `Group2` contains `User1`, then `Group2` is a direct group of `User1`, but `Group1` is not a direct group of `User1`.

Nested

The membership attribute contains both direct groups and nested groups.

All The membership attribute contains direct groups, nested groups, and dynamic members.

After completing these steps, group attribute definition settings are configured for your LDAP repository.

1. After configuring the federated repositories, click **Security > Secure administration, applications, and infrastructure** to return to the Secure administration, applications, and infrastructure panel. Verify that `Federated repositories` is identified in the Current realm definition field. If `Federated repositories` is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Secure administration, applications, and infrastructure panel. If `Federated repositories` is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
2. If you are enabling security, complete the remaining steps as specified in “Enabling security for the realm” on page 74. As the final step, validate this setup by clicking **Apply** in the Secure administration, applications, and infrastructure panel.
3. Save, stop, and restart all the product servers (deployment managers, nodes, and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Group attribute definition settings:

Use this page to specify the name of the group membership attribute. Every Lightweight Directory Access Protocol (LDAP) entry includes this attribute to indicate the group to which this entry belongs.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add** to specify a new external repository or select an external repository that is preconfigured.
5. Under Additional properties, click **Group attribute definition**.

When you finish adding or updating your federated repository configuration, go to the **Security > Secure administration, applications, and infrastructure** panel and click **Apply** to validate the changes.

Name of group membership attribute:

Specifies the name of the group membership attribute. Only one membership attribute can be defined for each Lightweight Directory Access Protocol (LDAP) repository.

Every LDAP entry should have this attribute to indicate the groups to which this entry belongs. For example, `memberOf` is the name of the membership attribute that is used in Active Directory. The group membership attribute contains values that reference groups to which this entry belongs. If UserA belongs to GroupA, then the value of the `memberOf` attribute of UserA should contain the distinguished name of GroupA.

If your LDAP server does not support the group membership attribute, then do not specify this attribute. The LDAP repository can look up groups by searching the group member attributes, though the performance might be slower.

Scope of group membership attribute:

Specifies the scope of the group membership attribute.

Default:	Direct
Range:	Direct The membership attribute contains direct groups only. Direct groups are the groups that contain the member. For example, if Group1 contains Group2 and Group2 contains User1, then Group2 is a direct group of User1, but Group1 is not a direct group of User1.
	Nested The membership attribute contains both direct groups and nested groups.
	All The membership attribute contains direct groups, nested groups, and dynamic members.

Configuring member attributes in a federated repository configuration

Follow this task to configure member attributes in a federated repository configuration.

Because member attributes apply only to a Lightweight Directory Access Protocol (LDAP) repository, you must first configure an LDAP repository. For more information, see “Managing repositories in a federated repository configuration” on page 149.

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add** to specify a new external repository or select an external repository that is preconfigured.

Note: If you click **Add** to specify a new external repository, you must first complete the required fields and click **Apply** before you can proceed to the next step.

5. Under Additional properties, click **Group attribute definition**.
6. Under Additional properties, click **Member attributes**.
7. Click **New** to specify a new member attribute or **Delete** to remove a preconfigured member attribute.
8. Accept the default, or supply the name of the member attribute in the Name of member attribute field. For example, member and uniqueMember are two commonly used names of member attributes.
The member attribute is used to store the values that reference members that the group contains. For example, a group type with an object class groupOfNames has a member attribute named member; group type with object class groupOfUniqueNames has a member attribute named uniqueMember. An LDAP repository supports multiple group types if multiple member attributes and their associated group object classes are specified.
9. Supply the object class of the group that uses this member attribute in the Object class field. If this field is not defined, this member attribute applies to all group object classes.
10. Select the scope of the member attribute. The default value is Direct.

Direct The member attribute contains direct members only. Direct members are members that are directly contained by the group. For example, if Group1 contains Group2 and Group2 contains User1, then User1 is a direct member of Group2, but User1 is not a direct member of Group1.

Nested

The member attribute contains both direct members and nested members.

All

The member attribute contains direct members, nested members, and dynamic members.

After completing these steps, member attributes are configured for your LDAP repository.

1. After configuring the federated repositories, click **Security > Secure administration, applications, and infrastructure** to return to the Secure administration, applications, and infrastructure panel. Verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Secure administration, applications, and infrastructure panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
2. If you are enabling security, complete the remaining steps as specified in “Enabling security for the realm” on page 74. As the final step, validate this setup by clicking **Apply** in the Secure administration, applications, and infrastructure panel.
3. Save, stop, and restart all the product servers (deployment managers, nodes, and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Member attributes collection:

Use this page to list Lightweight Directory Access Protocol (LDAP) member attributes or to select a member attribute to view or change its configuration properties.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add** to specify a new external repository or select an external repository that is preconfigured.
5. Under Additional properties, click **Group attribute definition**.
6. Under Additional properties, click **Member attributes**.

When you finish adding or updating your federated repository configuration, go to the **Security > Secure administration, applications, and infrastructure** panel and click **Apply** to validate the changes.

Name:

Specifies the name of the member attribute in LDAP. For example, member and uniqueMember are two commonly used names of member attributes.

The member attribute is used to store the values that reference members that the group contains. For example, a group type with an object class groupOfNames has a member attribute named member; group type with object class groupOfUniqueNames has a member attribute named uniqueMember. An LDAP repository supports multiple group types if multiple member attributes and their associated group object classes are specified.

Scope:

Specifies the scope of the member attribute.

Default:

Direct

Range:

- Direct** The member attribute contains direct members only. Direct members are members that are directly contained by the group. For example, if Group1 contains Group2 and Group2 contains User1, then User1 is a direct member of Group2, but User1 is not a direct member of Group1.
- Nested** The member attribute contains both direct members and nested members.
- All** The member attribute contains direct members, nested members, and dynamic members.

Object class:

Specifies the object class of the group that uses this member attribute. If this field is not defined, this member attribute applies to all group object classes.

Member attributes settings:

Use this page to configure Lightweight Directory Access Protocol (LDAP) member attributes.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.

3. Under Related items, click **Manage repositories**.
4. Click **Add** to specify a new external repository or select an external repository that is preconfigured.
5. Under Additional properties, click **Group attribute definition**.
6. Under Additional properties, click **Member attributes**.
7. Click **New** to specify a new member attribute.

When you finish adding or updating your federated repository configuration, go to the **Security > Secure administration, applications, and infrastructure** panel and click **Apply** to validate the changes.

Name of member attribute:

Specifies the name of the member attribute in LDAP. For example, member and uniqueMember are two commonly used names of member attributes.

The member attribute is used to store the values that reference members that the group contains. For example, a group type with an object class groupOfNames has a member attribute named member; group type with object class groupOfUniqueNames has a member attribute named uniqueMember. An LDAP repository supports multiple group types if multiple member attributes and their associated group object classes are specified.

Object class:

Specifies the object class of the group that uses this member attribute. If this field is not defined, this member attribute applies to all group object classes.

Scope:

Specifies the scope of the member attribute.

Default:	Direct
Range:	<p>Direct The member attribute contains direct members only. Direct members are members that are directly contained by the group. For example, if Group1 contains Group2 and Group2 contains User1, then User1 is a direct member of Group2, but User1 is not a direct member of Group1.</p> <p>Nested The member attribute contains both direct members and nested members.</p> <p>All The member attribute contains direct members, nested members, and dynamic members.</p>

Configuring dynamic member attributes in a federated repository configuration

Follow this task to configure dynamic member attributes in a federated repository configuration.

Because dynamic member attributes apply only to a Lightweight Directory Access Protocol (LDAP) repository, you must first configure an LDAP repository. For more information, see “Managing repositories in a federated repository configuration” on page 149.

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add** to specify a new external repository or select an external repository that is preconfigured.

Note: If you click **Add** to specify a new external repository, you must first complete the required fields and click **Apply** before you can proceed to the next step.

5. Under Additional properties, click **Group attribute definition**.
6. Under Additional properties, click **Dynamic member attributes**.
7. Click **New** to specify a new dynamic member attribute or **Delete** to remove a preconfigured dynamic member attribute.
8. Accept the default, or supply the name of the dynamic member attribute in the Name of dynamic member attribute field. The name of the dynamic member attribute defines the filter for dynamic group members in LDAP, for example, memberURL is the name of a commonly used dynamic member attribute.

If both member and dynamic member attributes are specified for the same group type, this group type is a hybrid group with both static and dynamic members.

A dynamic group defines its members differently than a static group. Instead of listing the members individually, the dynamic group defines its members using an LDAP search. The filter for the search is defined in a dynamic member attribute. For example, the dynamic group uses the structural objectclass groupOfURLs, or auxiliary objectclass ibm-dynamicGroup, and the attribute memberURL, to define the search using a simplified LDAP URL syntax:

```
ldap:///<base DN of search> ? ? <scope of search> ? <searchfilter>
```

The following is an example of the LDAP URL that defines all entries that are under o=Acme with the objectclass=person:

```
ldap:///o=Acme,c=US??sub?objectclass=person
```

9. Supply the object class of the group that contains the dynamic member attribute in the Dynamic object class field, for example, groupOfURLs. If this property is not defined, the dynamic member attribute applies to all group object classes.

After completing these steps, dynamic member attributes are configured for your LDAP repository.

1. After configuring the federated repositories, click **Security > Secure administration, applications, and infrastructure** to return to the Secure administration, applications, and infrastructure panel. Verify that Federated repositories is identified in the Current realm definition field. If Federated repositories is not identified, select **Federated repositories** from the Available realm definitions field and click **Set as current**. To verify the federated repositories configuration, click **Apply** on the Secure administration, applications, and infrastructure panel. If Federated repositories is not identified in the Current realm definition field, your federated repositories configuration is not used by WebSphere Application Server.
2. If you are enabling security, complete the remaining steps as specified in “Enabling security for the realm” on page 74. As the final step, validate this setup by clicking **Apply** in the Secure administration, applications, and infrastructure panel.
3. Save, stop, and restart all the product servers (deployment managers, nodes, and Application Servers) for changes in this panel to take effect. If the server comes up without any problems, the setup is correct.

Dynamic member attributes collection:

Use this page to manage Lightweight Directory Access Protocol (LDAP) dynamic member attributes.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add** to specify a new external repository or select an external repository that is preconfigured.
5. Under Additional properties, click **Group attribute definition**.

6. Under Additional properties, click **Dynamic member attributes**.

When you finish adding or updating your federated repository configuration, go to the **Security > Secure administration, applications, and infrastructure** panel and click **Apply** to validate the changes.

Name:

Specifies the name of the attribute that defines the filter for dynamic group members in LDAP. For example, memberURL is the name of a commonly used dynamic member attribute.

If both member and dynamic member attributes are specified for the same group type, this group type is a hybrid group with both static and dynamic members.

A dynamic group defines its members differently than a static group. Instead of listing the members individually, the dynamic group defines its members using an LDAP search. The filter for the search is defined in a dynamic member attribute. For example, the dynamic group uses the structural objectclass groupOfURLs, or auxiliary objectclass ibm-dynamicGroup, and the attribute memberURL, to define the search using a simplified LDAP URL syntax:

```
ldap:///<base DN of search>??<scope of search>?<searchfilter>
```

The following is an example of the LDAP URL that defines all entries that are under o=Acme with the objectclass=person:

```
ldap:///o=Acme,c=US??sub?objectclass=person
```

Object class:

Specifies the object class of the group that contains this dynamic member attribute, for example, groupOfURLs. If this property is not defined, the dynamic member attribute applies to all group object classes.

Dynamic member attributes settings:

Use this page to configure Lightweight Directory Access Protocol (LDAP) dynamic member attributes.

To view this administrative console page, complete the following steps:

1. In the administrative console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, select **Federated repositories** from the Available realm definitions field and click **Configure**.
3. Under Related items, click **Manage repositories**.
4. Click **Add** to specify a new external repository or select an external repository that is preconfigured.
5. Under Additional properties, click **Group attribute definition**.
6. Under Additional properties, click **Dynamic member attributes**.
7. Click **New** to specify a new dynamic member attribute.

When you finish adding or updating your federated repository configuration, go to the **Security > Secure administration, applications, and infrastructure** panel and click **Apply** to validate the changes.

Name of dynamic member attribute:

Specifies the name of the attribute that defines the filter for dynamic group members in LDAP. For example, memberURL is the name of a commonly used dynamic member attribute.

If both member and dynamic member attributes are specified for the same group type, this group type is a hybrid group with both static and dynamic members.

A dynamic group defines its members differently than a static group. Instead of listing the members individually, the dynamic group defines its members using an LDAP search. The filter for the search is defined in a dynamic member attribute. For example, the dynamic group uses the structural objectclass `groupOfURLs`, or auxiliary objectclass `ibm-dynamicGroup`, and the attribute `memberURL`, to define the search using a simplified LDAP URL syntax:

```
ldap:///<base DN of search>??<scope of search>?<searchfilter>
```

The following is an example of the LDAP URL that defines all entries that are under `o=Acme` with the `objectclass=person`:

```
ldap:///o=Acme,c=US??sub?objectclass=person
```

Dynamic object class:

Specifies the object class of the group that contains this dynamic member attribute, for example, `groupOfURLs`. If this property is not defined, the dynamic member attribute applies to all group object classes.

Local operating system registries

With the registry implementation for the local operating system, the WebSphere Application Server authentication mechanism can use the user accounts database of the local operating system.

Lightweight Directory Access Protocol (LDAP) is a centralized registry. Most local operating system registries are not centralized registries.

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 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 2003 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 a Windows 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 supported by Windows 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 Windows NT domain native call returns the support group only without an error.

Do not use a local operating system 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.

The Windows domain registry and Network Information Services (NIS) are exceptions. Both the Windows domain registry and Network Information Services (NIS) are centralized registries. The Windows domain registry is supported by WebSphere Application Server; however, NIS is not supported.

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.

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 standalone machine or a machine that is part of a domain or is the domain controller, the access requirements vary.

- For a standalone 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.

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 user 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 user registry and the local operating system registry

When the machine that hosts the WebSphere Application Server process is a member of a domain, both the local and the domain user registries are used by default. The following section describes more on this topic and recommends some best practices to avoid unfavorable consequences.

Note: Although this section does not directly describe z/OS considerations, you should be aware that overall security operations are affected by how well you set up these registries.

- **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 user registry or the domain user registry to map to the roles.

In cases where the same users or groups exist in both the local user registry and the domain user 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 user registry tries to authenticate the user first. If authentication fails, the local user 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 user registry. In case of failure, the local user 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, only that user 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.

Tip: A user, Bob, on one machine in the local OS user registry, for example, is not the same as the user Bob on another machine in the domain user registry, for example, because the unique ID of Bob, which is the security identifier [SID] in this case, is different in different user registries.

- **Examples**

The MyMachine machine is part of the MyDomain domain. The MyMachine machine contains the following users and groups:

- MyMachine\user2
- MyMachine\user3
- MyMachine\group2

The MyDomain domain 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 user registry is used for authentication. If the authentication fails because the password is different, for example, the local user registry is used.
2. If the MyMachine\user2 user is mapped to a role, only the user2 user in MyMachine machine has access. Thus, if the user2 password is the same on both the local and the domain user registries, the user2 user cannot access the resource because the user2 user is always authenticated using the domain user registry. If both user registries have common users, it is recommended that you have different passwords.
3. If the group2 group is mapped to a role, only the users who are members of the MyDomain\group2 group can access the resource because group2 information is first obtained from the domain user registry.
4. If the MyMachine\group2 group is mapped to a role, only the users who are members of the MyMachine\group2 group can access the resource. A specific group is mapped to the role (MyMachine\group2 instead of just group2).
5. Use either the user3 user or the MyMachine\user3 user to map to a role because the user3 user is unique as it exists in one user 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 user registry. If you want to access users and groups from either the local or the domain user registry, instead of both, set the `com.ibm.websphere.registry.UseRegistry` property. This property can be set to either `local` or `domain`. When this property is set to `local` (case insensitive) only the local user registry is used. When this property is set to `domain`, (case insensitive) only the domain user registry is used.

Set this property by completing the following steps to access the **Custom Properties** panel in the administrative console:

1. Click **Security > Secure administration, applications, and infrastructure**
2. Under User account repository, click the **Available realm definitions** drop-down list, select **Local operating system**, and click **Configure**.
3. Under Additional properties, click **Custom properties**.

You can also use `wsadmin` to configure this property. 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 Linux and Solaris system user registries

For WebSphere Application Server local operating system 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 administrative security and configuring the 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 enable local operating system security successfully.

Standalone Lightweight Directory Access Protocol registries

A Standalone Lightweight Directory Access Protocol (LDAP) registry performs authentication using an LDAP binding.

WebSphere Application Server security provides and supports the 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 for authenticating a user and other security-related tasks. For example, the servers are used to retrieve 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 administrative user name that is defined in the registry, 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 and have administrative privileges. 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, the primary administrative user name and password are authenticated with the registry during the product startup. If authentication fails, the server does not start. 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, 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 that are described in “Configuring Lightweight Directory Access Protocol user registries” on page 93. 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 Secure administration, applications, and infrastructure panel. When the server is up, you can change the ID, password, and other configuration information and then enable security.

You can use the custom Lightweight Directory Access Protocol (LDAP) feature to support any LDAP server by setting up the correct configuration. However, support is not extended to these custom LDAP servers because many configuration possibilities exist.

The users and groups and security role mapping information is used by the configured authorization engine to perform access control decisions.

Dynamic groups and nested group support

Dynamic and nested groups simplify WebSphere Application Server security management and increase its effectiveness and flexibility.

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 so an application does not need 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.

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 run-time authorization table.

Dynamic groups and nested group support for the IBM Tivoli Directory Server:

Dynamic and nested groups simplify WebSphere Application Server security management and increase its effectiveness and flexibility.

WebSphere Application Server supports all Lightweight Directory Access Protocol (LDAP) dynamic and nested groups when using IBM Tivoli Directory Server. This function is enabled by default by taking advantage of a new feature in IBM Tivoli Directory Server. IBM Tivoli Directory Server 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.

For more information, see “Configuring dynamic and nested group support for the IBM Tivoli Directory Server” on page 111.

Dynamic and nested group support for the SunONE or iPlanet Directory Server:

Dynamic and nested groups simplify WebSphere Application Server security management and increase its effectiveness and flexibility.

The SunONE or iPlanet Directory Server uses two grouping mechanisms:

Groups

Entries that name other entries as a list of members or as a filter for members.

Roles 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

Depends upon the attributes that are contained in each entry. Entries are members, if they match a specified Lightweight Directory Access Protocol (LDAP) filter. 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” on page 110 for more information.

Security failover among multiple LDAP servers

WebSphere Application Server security can be configured to attempt failovers between multiple Lightweight Directory Access Protocol (LDAP) hosts.

If the current active LDAP server is unavailable, WebSphere Application Server security attempts a failover to the first available LDAP host in the specified host list. The multiple LDAP servers can be replicas of the same master LDAP server, or they can be any LDAP host with the same schema, which contain data that is imported from the same LDAP Data Interchange Format (LDIF) file.

Whenever a failover occurs, WebSphere Application Server security always uses the first available LDAP server in the specified host list. For example, if there are four LDAP servers configured in the order of L1, L2, L3, and L4, L1 is treated as the primary LDAP server. The preference of connection is from L1 to L4. If, for example, WebSphere Application Server security is currently connected to L4, and failover or reconnection is necessary, WebSphere Application Server security first attempts to connect to L1, L2, and then L3 in that order until the connection is successful.

The current LDAP host name is logged in message CWSCJ0419I in the WebSphere Application Server log file, `SystemOut.log`. If you want to reconnect to the primary LDAP host, run the WebSphere Application Server MBean method, `resetLDAPBindInfo`, with `null,null` as the input.

To configure LDAP failover among multiple LDAP hosts, you must use `wsadmin` or `ConfigService` to include the backup LDAP host, which does not have a number limitation. The LDAP host that is displayed in the administrative console is the primary LDAP host, and is the first item listed in the LDAP host list in `security.xml`.

The WebSphere Application Server security realm name defaults to the primary LDAP host name that is displayed in the administrative console. It includes a trailing colon and a port number (if one exists). However, the custom property, `com.ibm.websphere.security.ldap.logicRealm`, can be added to override the default security realm name. Use the `logicRealm` name to configure each cell to have its own LDAP host for interoperability and backward compatibility, and to provide flexibility for adding or removing the LDAP host dynamically. If migrating from a previous installation, the new `logicRealm` name does not take effect until administrative security is enabled again. To be compatible with a previous release that does not support logic realm, the `logicRealm` name has to be the same as that used by the previous installation (the LDAP host name, including a trailing colon and port number).

The following example shows how to use `wsadmin` to add a backup LDAP host for failover:

```
proc LDAPAdd {args} {
    global AdminConfig AdminControl ldapServer ldapPort
    set ldapServer [lindex $args 0]
    set ldapPort [lindex $args 1]
    global ldapUserRegistryId
    if {[catch {$AdminConfig list LDAPUserRegistry} result]} {
        puts stdout "\$AdminConfig list LDAPUserRegistry caught an exception $result\n"
        return
    } else {
        if {$result != {}} {
            set ldapUserRegistryId [lindex $result 0]
        } else {
            return;
        }
    }
    set secMbean [$AdminControl queryNames type=SecurityAdmin,*]
    set Attrs2 [list [list hosts [list [list [list host $ldapServer]
        [list port $ldapPort]]]]]
    $AdminConfig modify $ldapUserRegistryId $Attrs2
    $AdminConfig save
}
```

Federated repositories

Federated repositories enable you to use multiple repositories with WebSphere Application Server. These repositories, which can be file-based repositories, LDAP repositories, or a sub-tree of an LDAP repository, are defined and theoretically combined under a single realm. All of the user repositories that are configured under the federated repository functionality are invisible to WebSphere Application Server.

When you use the federated repositories functionality, all of the configured repositories, which you specify as part of the federated repository configuration, become active. It is recommended that the user ID, and the distinguished name (DN) for an LDAP repository, be unique in multiple user repositories that are configured under the same federated repository configuration. For example, there might be three different repositories that are configured for the federated repositories configuration: Repository A, Repository B, and Repository C. When user1 logs in, the federated repository adapter searches each of the repositories for all of the occurrences of that user. If multiple instances of that user are found in the combined repositories, an error message displays.

In addition, the federated repositories functionality in WebSphere Application Server supports the logical joining of entries across multiple user repositories when the Application Server searches and retrieves

entries from the repositories. For example, when an application calls for a sorted list of people whose age is greater than twenty, WebSphere Application searches all of the repositories in the federated repositories configuration. The results are combined and sorted before the Application Server returns the results to the application.

Unlike the local operating system, standalone LDAP registry, or custom registry options, federated repositories provide user and group management with read and write capabilities. When you configure federated repositories, you can use one of the following methods to add, create, and delete users and groups:

Important: If you configure multiple repositories under the federated repositories realm, you must also configure supported entity types and specify a base entry for the default parent. The base entry for the default parent determines the repository location where entities of the specified type are placed on write operations by user and group management. See “Configuring supported entity types in a federated repository configuration” on page 147 for details.

- Use the user management application programming interfaces (API). For more information, refer to articles under “Developing with virtual member manager” in this information center.
- Use the administrative console. To manage users and groups within the administrative console, click **Users and Groups > Manage Users** or **Users and Groups > Manage Groups**. For information on user and group management, click the Help link that displays in the upper right corner of the window. From the left navigation pane, click **Users and Groups**.
- Use the wsadmin commands. For more information, see “Commands for the WIMManagementCommands group of the AdminTask object” on page 758.

If you do not configure the federated repositories functionality or do not enable federated repositories as the active repository, you cannot use the user management capabilities that are associated with federated repositories. You can configure an LDAP server as the active user registry and configure the same LDAP server under federated repositories, but not select federated repositories as the active user repository. With this scenario, authentication takes place using the LDAP server, and you can use the user management functionality for the LDAP server that is available for federated repositories.

The following table compares the federated repository functionality that is available in WebSphere Application Server Version 6.1 with the registry functionality that remains unchanged from previous versions of the Application Server.

Table 3. Federated repositories versus user registry implementations

Federated repositories	User registry
Supports multiple types of repositories such as file-based, LDAP, database, and custom. In WebSphere Application Server Version 6.1, file-based and LDAP repositories are supported by the administrative console. However, the federated repositories functionality does not support local operating system implementations. For database and custom repositories, you can use the wsadmin command-line interface or the configuration application programming interfaces (API).	Supports multiple types of registries such as the local operating system, a standalone LDAP registry, and a standalone custom registry.
Supports multiple repositories in a realm within a cell.	Supports one registry only in a realm within a cell.
Provides read and write capabilities for the repositories that are defined in the federated repository configuration.	Provides read only capability for the registries.
Provides account and password policy support as defined by the registry type. However, this support is not provided by the federated repository functionality.	Provides account and password policy support as defined by the registry type.
Supports identity profiles.	Does not support identity profiles.
Uses the custom UserRegistry implementation.	Uses the custom UserRegistry implementation.

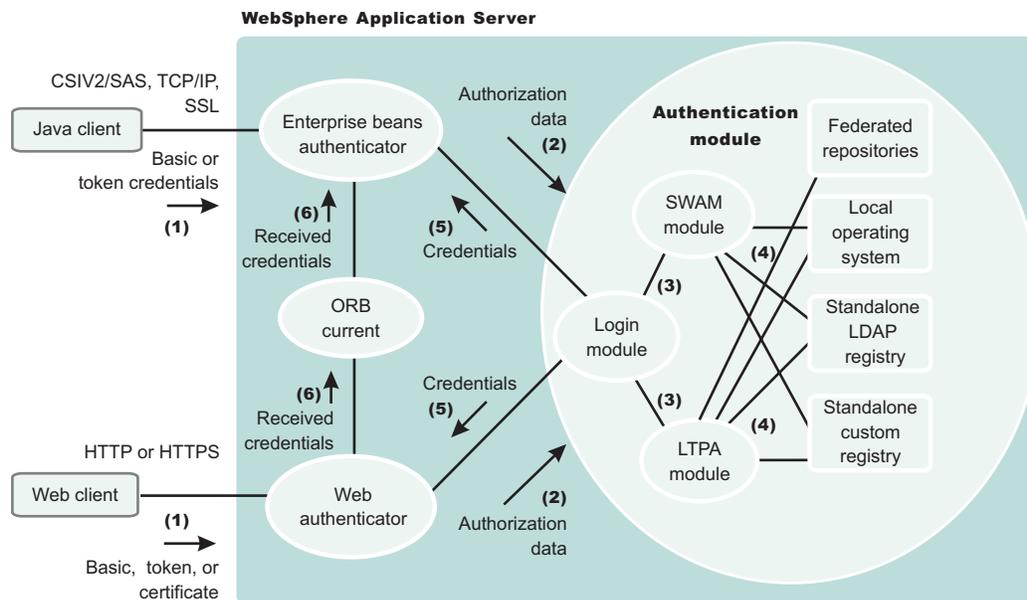
Authentication mechanisms

An *authentication mechanism* defines rules about security information, such as 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 who or what it claims to be 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.

WebSphere Application Server provides two authentication mechanisms: Lightweight Third Party Authentication (LTPA) and Simple WebSphere Authentication Mechanism (SWAM). You configure LTPA, which is the default authentication mechanism, in the administrative console by clicking **Security > Secure administration, applications, and infrastructure > Authentication mechanisms and expiration**. SWAM is deprecated in Version 6.1. SWAM does not provide authenticated communication between different servers. To use SWAM instead of LTPA, select the **Use SWAM-no authenticated communication between servers** option on the Authentication mechanisms and expiration panel. However, if you select the **Use SWAM-no authenticated communication between servers** option, the other information on the Authentication mechanisms and expiration panel will be ignored.

Authentication



Authentication process

The figure demonstrates the authentication process. Authentication is required for enterprise bean clients and Web clients when they access protected resources. Enterprise bean clients, like 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: **V6.0.x**

- Common Secure Interoperability Version 2 (CSIv2)
- Secure Authentication Service (SAS)

Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

Web clients use the HTTP or HTTPS protocol to send the authentication information, as shown in the previous figure.

The authentication information can be basic authentication (user ID and password), a credential token (in the case of Lightweight Third Party Authentication (LTPA)), or a client certificate. The Web authentication is performed by the Web Authentication module.

You can configure Web authentication for a Web client by using the administrative console. Click **Security > Secure administration and applications**. Under Authentication, expand **Web security** and click **General settings**. The following options exist for Web authentication:

Authenticate only when the URI is protected

Specifies that the Web client can retrieve an authenticated identity only when it accesses a protected Uniform Resource Identifier (URI). WebSphere Application Server challenges the Web client to provide authentication data when the Web client accesses a URI that is protected by a J2EE role. This default option is also available in previous versions of WebSphere Application Server.

Use available authentication data when an unprotected URI is accessed

Specifies that the Web client is authorized to call the `getRemoteUser`, `isUserInRole`, and `getUserPrincipal` methods; retrieves an authenticated identity from either a protected or an unprotected URI. Although the authentication data is not used when you access an unprotected URI, the authentication data is retained for future use. This option is available when you select the **Authenticate only when the URI is protected** check box.

Authenticate when any URI is accessed

Specifies that the Web client must provide authentication data regardless of whether the URI is protected.

Default to basic authentication when certificate authentication for the HTTPS client fails.

Specifies that WebSphere Application Server challenges the Web client for a user ID and password when the required HTTPS client certificate authentication fails.

V6.0.x 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:

- LTPA
- Simple WebSphere Authentication Mechanism (SWAM)

Note: SWAM is deprecated in WebSphere Application Server Version 6.1 and will be removed in a future release. LTPA is the default authentication mechanism.

The authentication module uses the registry that is configured on the system to perform the authentication (4). Four types of registries are supported:

- Federated repositories
- Local operating system
- Standalone Lightweight Directory Access Protocol (LDAP) registry

- Standalone custom registry

External registry implementation following the registry interface that is specified by IBM can replace either the local operating system or the LDAP registry.

The login module creates a JAAS subject after authentication and stores the credential that is derived from the authentication data in the public credentials list of the subject. The credential is returned to the Web authenticator or to the enterprise beans authenticator (5).

The Web authenticator and the enterprise beans authenticator store the received credentials in the Object Request Broker (ORB) current for the authorization service to use in performing further access control checks. If the credentials are forwardable, they are sent to other application servers.

Portlet URL security

WebSphere Application Server enables direct access to portlet Uniform Resource Locators (URLs), just like servlets. This section describes security considerations when accessing portlets using URLs.

For security purposes, portlets are treated similar to servlets. Most portlet security uses the underlying servlet security mechanism. However, portlet security information resides in the `portlet.xml` file, while the servlet and JavaServer Pages files reside in the `web.xml` file. Also, when you make access decisions for portlets, the security information, if any, in the `web.xml` file is combined with the security information in the `portlet.xml` file.

Portlet security must support both programmatic security, that is `isUserInRole`, and declarative security. The programmatic security is exactly the same as for servlets. However, for portlets, the `isUserInRole` method uses the information from the `security-role-ref` element in `portlet.xml`. The other two methods used by programmatic security, `getRemoteUser` and `getUserPrincipal`, behave the same way as they do when accessing a servlet. Both of these methods return the authenticated user information accessing the portlet.

The declarative security aspect of the portlets is defined by the `security-constraint` information in the `portlet.xml` file. This is similar to the `security-constraint` information used for the servlets in the `web.xml` file with the following differences:

- The `auth-constraint` element, which lists the names of the roles that can access the resources, does not exist in the `portlet.xml` file. The `portlet.xml` file contains only the `user-data-constraint` element, which indicates what type of transport layer security (HTTP or HTTPS) is required to access the portlet.
- The `security-constraint` information in the `portlet.xml` file contains the `portlet-collection` element, while the `web.xml` file contains the `web-resource-collection` element. The `portlet-collection` element contains only a list of simple portlet names, while the `web-resource-collection` contains the `url-patterns` as well as the HTTP methods that need protection.

The portlet container does not deal with the user authentication directly. For example, it does not prompt you to collect the credential information. The portlet container must, instead, use the underlying servlet container for the user authentication mechanism. As a result, there is no `auth-constraint` element in the `security-constraint` information in the `portlet.xml` file.

In WebSphere Application Server, when a portlet is accessed using a URL, the user authentication is processed based on the `security-constraint` information for that portlet in the `web.xml` file. This implies that to authenticate a user for a portlet, the `web.xml` file must contain the `security-constraint` information for that portlet with the relevant `auth-constraints` contained in it. If a corresponding `auth-constraint` for the portlet does not exist in the `web.xml` file, it indicates that the portlet is not required to have authentication. In this case, unauthenticated access is permitted just like a URL pattern for a servlet that does not contain any `auth-constraints` in the `web.xml` file. An `auth-constraint` for a portlet can be specified directly by using the portlet name in the `url-pattern` element, or indirectly by a `url-pattern` that implies the portlet.

Note: You cannot have a servlet or JSP with the same name as a portlet for WebSphere Application Server security to work with portlet.

The following examples demonstrate how the security-constraint information contained in the `portlet.xml` and `web.xml` files in a portlet application are used to make security decisions for portlets. The `security-role-ref` element, which is used for `isUserInRole` calls, is not discussed here because it is used the same way for servlets.

In the examples below (unless otherwise noted), there are four portlets (`MyPortlet1`, `MyPortlet2`, `MyPortlet3`, `MyPortlet4`) defined in `portlet.xml`. The portlets are secured by combining the information, if any, in the `web.xml` file when they are accessed directly through URLs.

All of the examples show the contents of the `web.xml` and `portlet.xml` files. Use the correct tools when creating these deployment descriptor files as you normally would when assembling a portlet application.

Example 1: The `web.xml` file does not contain any security-constraint data

In the following example, the security-constraint information is contained in `portlet.xml`:

```
<security-constraint id="SecurityConstraint_1">
  <web-resource-collection id="WebResourceCollection_1">
    <web-resource-name>Protected Area</web-resource-name>
    <url-pattern>/*</url-pattern>
  </web-resource-collection>
  <auth-constraint id="AuthConstraint_1">
    <role-name>Manager</role-name>
  </auth-constraint>
</security-constraint>
```

In this example, when you access anything under `MyPortlet1` and `MyPortlet3`, and these portlets are accessed using the unsecured HTTP protocol, you are redirected through the secure HTTPS protocol. The `transport-guarantee` is set to use secure connections. For `MyPortlet2` and `MyPortlet4`, unsecured (HTTP) access is permitted because the `transport-guarantee` is not set. There is no corresponding security-constraint information for all four portlets in the `web.xml` file. Therefore, all of the portlets can be accessed without any user authentication and role authorization. The only security involved in this instance is the transport-layer security using Secure Sockets Layer (SSL) for `MyPortlet1` and `MyPortlet3`.

The following table lists the security constraints that are applicable to the individual portlets.

URL	Transport Protection	User Authentication	Role Based Authorization
/MyPortlet1/*	HTTPS	None	None
/MyPortlet2/*	None	None	None
/MyPortlet3/*	HTTPS	None	None
/MyPortlet4/*	None	None	None

Example 2: The `web.xml` file contains portlet specific security-constraint data

In the following example, the security-constraint information that corresponds to the portlet is contained in `web.xml`. The `portlet.xml` file is the same as that shown in the previous example.

```
<security-constraint id="SecurityConstraint_1">
  <web-resource-collection id="WebResourceCollection_1">
    <web-resource-name>Protected Area</web-resource-name>
    <url-pattern>/MyPortlet1/*</url-pattern>
    <url-pattern>/MyPortlet2/*</url-pattern>
  </web-resource-collection>
```

```

    <auth-constraint id="AuthConstraint_1">
      <role-name>Employee</role-name>
    </auth-constraint>
  </security-constraint>

```

The security-constraint information contained in the web.xml file in this example indicates that the user authentication must be performed when accessing anything under the MyPortlet1 and MyPortlet2 portlets. When you attempt to access these portlets directly using URLs, and there is no authentication information available, you are prompted to enter their credentials. After you are authenticated, the authorization check is performed to see if you are listed in the Employee role. The user/group to role mapping is assigned during the portlet application deployment. In the web.xml file listed above, note the following:

- Because the web.xml file uses url-pattern, the portlet names have been modified slightly. MyPortlet1 is now /MyPortlet1/*, which indicates that everything under the MyPortlet1 URL is protected. This matches the information in the portlet.xml file because the security runtime code converts the portlet-name element in the portlet.xml file to url-pattern (for example, MyPortlet1 to /MyPortlet1/*), even for the transport-guarantee.
- The http-method element in the web.xml file is not used in the example because all HTTP methods must be protected.

The following table lists the new security constraints that are applicable to the individual portlets.

URL	Transport Protection	User Authentication	Role Based Authorization
MyPortlet1/*	HTTPS	Yes	Yes (Employee)
MyPortlet2/*	None	Yes	Yes (Employee)
MyPortlet3/*	HTTPS	None	None
MyPortlet4/*	None	None	None

Example 3: The web.xml file contains generic security-constraint data implying all portlets.

In the following example, the security-constraint information is contained in the web.xml file that corresponds to the portlet. The portlet.xml file is the same as that shown in the first example.

```

<security-constraint id="SecurityConstraint_1">
  <web-resource-collection id="WebResourceCollection_1">
    <web-resource-name>Protected Area</web-resource-name>
    <url-pattern>/*</url-pattern>
  </web-resource-collection>
  <auth-constraint id="AuthConstraint_1">
    <role-name>Manager</role-name>
  </auth-constraint>
</security-constraint>

```

In this example, /* implies that all resources that do not contain their own explicit security-constraints should be protected by the Manager role as per the URL pattern matching rules. Because the portlet.xml file contains explicit security-constraint information for MyPortlet1 and MyPortlet3, these two portlets are not protected by the Manager role, only by the HTTPS transport. Because the portlet.xml file cannot contain the auth-constraint information, any portlets that contain security-constraints in it are rendered unprotected when an implying URL (//* for example) is listed in the web.xml file because of the URL matching rules.

In the case above, both MyPortlet1 and MyPortlet3 can be accessed without user authentication. However, because MyPortlet2 and MyPortlet4 do not have security-constraints in the portlet.xml file, the /* pattern is used to match these portlets and are protected by the Manager role, which requires user authentication.

The following table lists the new security constraints that are applicable to the individual portlets with this setup.

URL	Transport Protection	User Authentication	Role Based Authorization
MyPortlet1/*	HTTPS	None	None
MyPortlet2/*	None	Yes	Yes (Manager)
MyPortlet3/*	HTTPS	None	None
MyPortlet4/*	None	Yes	Yes (Manager)

If in the example above, if you must also protect a portlet contained in the `portlet.xml` file (for example, MyPortlet1), the `web.xml` file should contain an explicit security-constraint entry in addition to `/*` as shown in the following example:

```
<security-constraint id="SecurityConstraint_1">
  <web-resource-collection id="WebResourceCollection_1">
    <web-resource-name>Protected Area</web-resource-name>
    <url-pattern>*</url-pattern>
  </web-resource-collection>
  <auth-constraint id="AuthConstraint_1">
    <role-name>Manager</role-name>
  </auth-constraint>
</security-constraint>
<security-constraint id="SecurityConstraint_2">
  <web-resource-collection id="WebResourceCollection_2">
    <web-resource-name>Protection for MyPortlet1</web-resource-name>
    <url-pattern>/MyPortlet1/*</url-pattern>
  </web-resource-collection>
  <auth-constraint id="AuthConstraint_1">
    <role-name>Manager</role-name>
  </auth-constraint>
</security-constraint>
```

In this case, MyPortlet1 is protected by the Manager role and requires authentication. The data-constraint of CONFIDENTIAL is also applied to it because the information in the `web.xml` file and the `portlet.xml` file are combined. Because MyPortlet3 is not explicitly listed in the `web.xml` file, it is still not protected by the Manager role and does not require user authentication.

The following table shows the effect of this change.

URL	Transport Protection	User Authentication	Role Based Authorization
MyPortlet1/*	HTTPS	Yes	Yes (Manager)
MyPortlet2/*	None	Yes	Yes (Manager)
MyPortlet3/*	HTTPS	None	None
MyPortlet4/*	None	Yes	Yes (Manager)

Lightweight Third Party Authentication

Lightweight Third Party Authentication (LTPA) is intended for distributed, multiple application server and machine environments. LTPA supports forwardable credentials and single sign-on (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.

Application servers distributed in multiple nodes and cells can securely communicate using the LTPA protocol. It also provides the single sign-on (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 DNS domain are case sensitive and must match identically.

Windows For local OS, the realm name is the domain name, if a domain is in use or the realm name is the machine name.

For the Lightweight Directory Access Protocol (LDAP), the realm name is the host:port value of the LDAP server.

The LTPA protocol uses cryptographic keys to encrypt and decrypt user data that passes between the servers. These keys must 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 that participate 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 protocol layer 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 that it has not expired and that 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 of the WebSphere Application Server processes in a cell (deployment manager, 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.

WebSphere Application Server supports the LTPA and the Simple WebSphere Authentication Mechanism (SWAM) protocols.

Note: SWAM is deprecated in WebSphere Application Server Version 6.1 and will be removed in a future release.

When security is enabled during profile creation time, LTPA is configured by default.

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	Local OS user registry	LDAP user registry	Custom user registry
SWAM	No	No	Yes	Yes	Yes
LTPA	Yes	Yes	Yes	Yes	Yes

Lightweight Third Party Authentication key sets and key set groups

Key set groups contain lists of key sets and Lightweight Third Party Authentication (LTPA) key generation schedules. Each key set contains key references to keys in key stores. To generate keys automatically, each key set must be a member of a key set group.

The keys for some key configurations must be generated together. The LTPA key pair is referenced in one key set while the secret or private key is in a separate key set. When the key set group is created, the two

key sets are added as members of the key set group. Key set group settings determine whether the keys for both key sets are generated together automatically or manually.

The key set group contains the following attributes:

- Member key sets
- Choice of either manual or automatic key generation in the member key sets
- Schedule for automatically generating keys

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 that are 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.

In this setup, 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 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 that are 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 the WebSphere Application Server TAI interface is provided to support WebSEAL Version 4.1. If you plan to use WebSEAL 5.1 or a later version of WebSEAL, 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 and later, but does not support WebSEAL Version 4.1. For an explanation of security attribute propagation, see “Security attribute propagation” on page 191

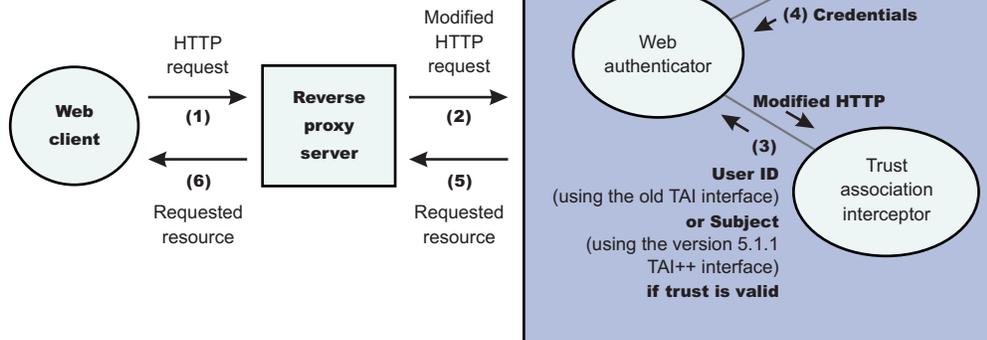
Trust association model

HTTP Request:

User ID and password in basic authentication data

Modified HTTP Request:

Trusted server ID and password in basic authentication data and user ID in the HTTP request header



IBM WebSphere Application Server: WebSEAL Integration

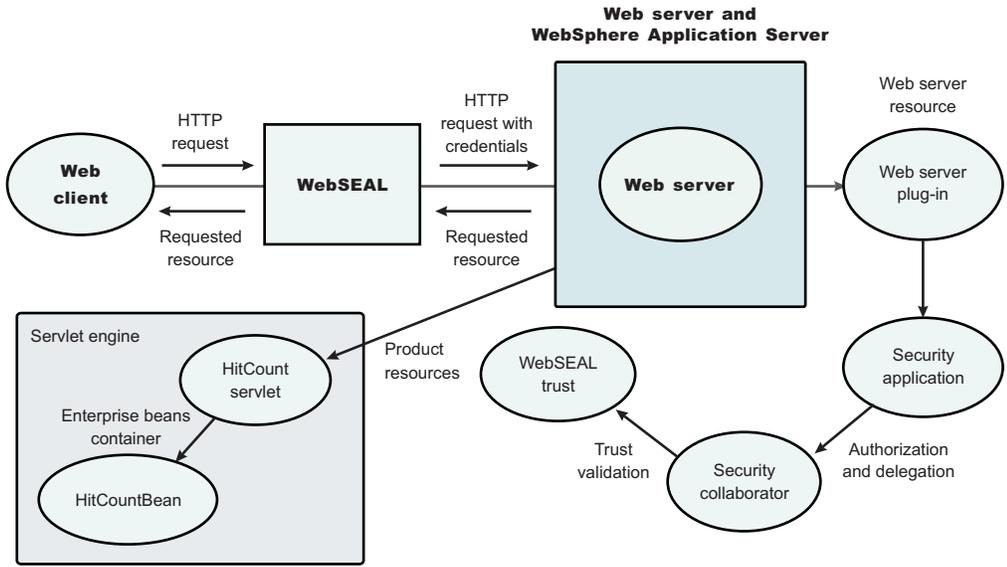
The integration of WebSEAL and WebSphere Application Server security is achieved by placing the WebSEAL server at the front-end as a reverse proxy server. From a WebSEAL management perspective, a junction is created with WebSEAL on one end, and the product Web server on the other end. A junction is a logical connection that is created to establish a path from the WebSEAL server to another server.

In this setup, a request for Web resources that are 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, 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, 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 the resource to the client user.

WebSEAL server

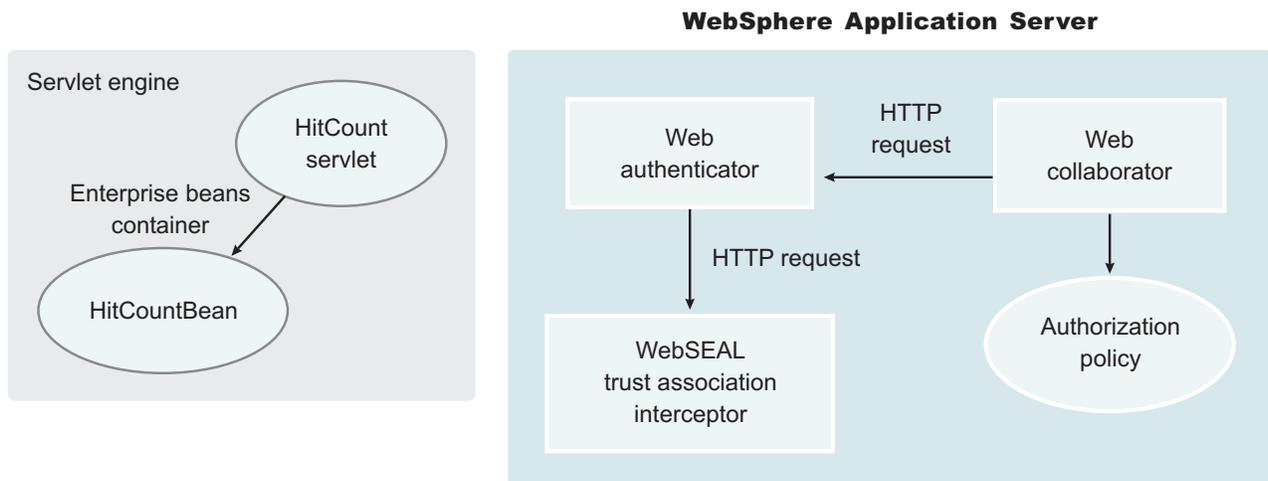
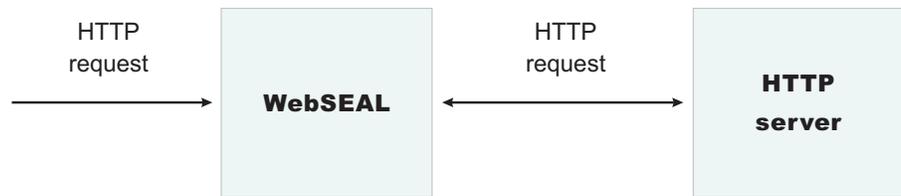
The policy director delegates all of the Web requests to its Web component, the WebSEAL server. One of the major functions of the server is to perform authentication of the requesting user. The WebSEAL server consults a Lightweight Directory Access Protocol (LDAP) directory. It can also map the original user ID to another user ID, such as when global single sign-on (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 that is 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. The Web collaborator 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 interceptor (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.

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 6.0.x and later supports the `com.ibm.wsspi.security.tai.TrustAssociationInterceptor` interface.

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 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, existing 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 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.

Single sign-on

With single sign-on (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.

Application servers distributed in multiple nodes and cells can securely communicate using the Lightweight Third Party Authentication (LTPA) protocol. LTPA is intended for distributed, multiple application server and machine environments. 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.

LTPA also provides the 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. Web users can authenticate once to a WebSphere Application Server or to a Domino server. This authentication is accomplished by configuring WebSphere Application Servers and the Domino servers to share authentication information.

Without logging in again, Web users can access other WebSphere Application Servers or Domino servers in the same DNS domain that are enabled for SSO. You can 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 SSO 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. The realm names on each system in the DNS domain are case sensitive and must match identically. 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 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 standalone custom registry. Domino servers do not support standalone custom registries, but you can use a Domino-supported registry as a standalone custom registry within WebSphere Application Server.

You can use a Domino directory that is configured for LDAP access or other LDAP directories for the registry. The LDAP directory product must have WebSphere Application Server support. Supported products include both Domino and LDAP servers, such as IBM Tivoli Directory Server. Regardless of the choice to use an LDAP or a standalone 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 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 6.5.4 for iSeries and 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 are supported.
 - You can use any HTTP Web server that is 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.

Single sign-on for HTTP requests using SPNEGO

WebSphere Application Server provides a trust association interceptor (TAI) that uses the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) to securely negotiate and authenticate HTTP requests for secured resources in WebSphere Application Server.

SPNEGO is a standard specification defined in The Simple and Protected GSS-API Negotiation Mechanism (IETF RFC 2478).

When WebSphere Application Server administrative security is enabled, the SPNEGO TAI is initialized. While processing inbound HTTP requests, the Web authenticator component interacts with the SPNEGO TAI, which is defined and enabled in the security configuration repository. One interceptor is selected and is responsible for authenticating access to the secured resource that is identified in the HTTP request.

Important: The use of TAIs is an optional feature. If no TAI is selected, the authentication process continues normally.

HTTP users log in and authenticate only once at their desktop and are subsequently authenticated (internally) with WebSphere Application Server. The SPNEGO TAI is invisible to the end-user of WebSphere applications. The SPNEGO TAI is only visible to the Web administrator who is responsible for ensuring a proper configuration, capacity, and maintenance of the Web environment.

In addition to WebSphere Application Server security runtime services, some external components are required to completely enable operation of the SPNEGO TAI. The external components include:

- Microsoft's Windows 2000 or Windows 2003 Servers with Active Directory domain and associated Kerberos Key Distribution Center (KDC).
- A client application, for example, a browser or .NET client, that supports the SPNEGO authentication mechanism, as defined in IETF RFC 2478. Microsoft Internet Explorer Version 5.5 or later and Mozilla Firefox Version 1.0 are browser examples. Any browser needs to be configured to use the SPNEGO mechanism. For more information on performing this configuration, see "Configuring SPNEGO TAI in WebSphere Application Server" on page 235

The authentication of HTTP requests is triggered by the requestor (the client-side), which generates a SPNEGO token. WebSphere Application Server receives this token and validates trust between the requester and WebSphere Application Server. Specifically, the SPNEGO TAI decodes and retrieves the requester's identity from the SPNEGO token. The identity is used to establish a secure context between the requester and the application server.

Remember: The SPNEGO TAI is a server-side solution in WebSphere Application Server. Client-side applications are responsible for generating the SPNEGO token for use by the SPNEGO TAI. The requester's identity in WebSphere Application Server security registry must be identical to that identity the SPNEGO TAI retrieves. An identical match does occur when Microsoft Windows Active Directory server is the Lightweight Directory Access Protocol (LDAP) server that is used in WebSphere Application Server. A custom login module is available as a plug-in to support custom mapping of the identity from the Active Directory to the WebSphere Application Server security registry. See "Mapping user Ids from client to server for SPNEGO" on page 249 for details on using this custom login module.

WebSphere Application Server validates the identity against its security registry and, if the validation is successful, produces a Lightweight Third Party Authentication (LTPA) security token and places and returns a cookie to the requester in the HTTP response. Subsequent HTTP requests from this same requester to access additional secured resources in WebSphere Application Server use the LTPA security token previously created, to avoid repeated login challenges.

The challenge-response handshake process is illustrated in the following graphic:

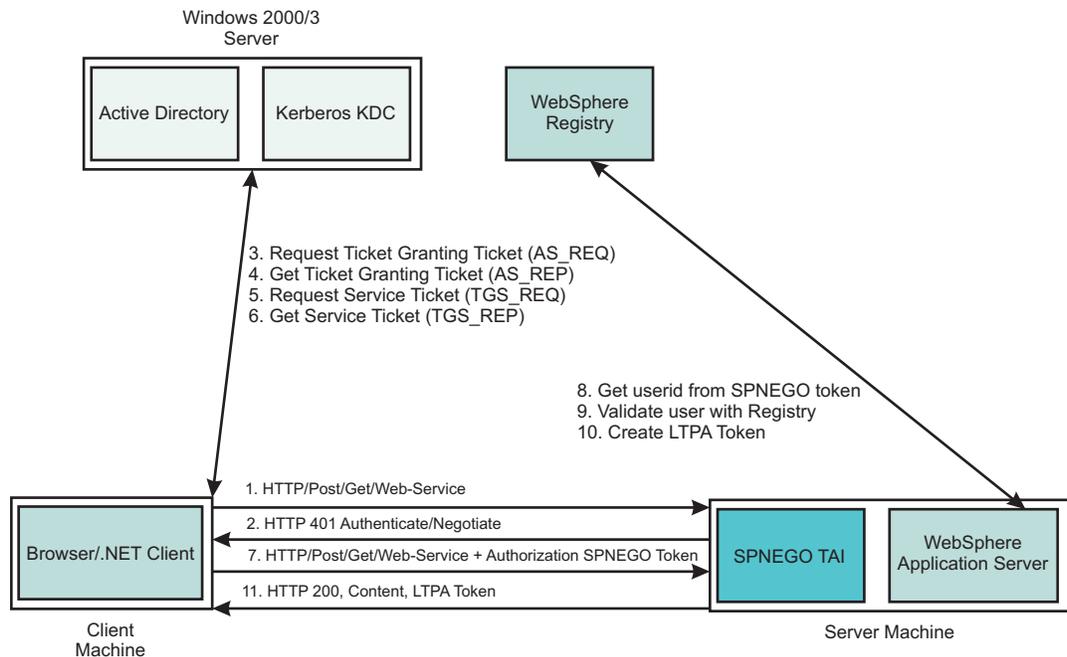


Figure 1. HTTP request processing, WebSphere Application Server - SPNEGO TAI

The SPNEGO TAI can be enabled for all or for selected WebSphere Application Servers in a WebSphere Application Server cell configuration. Also, the behavior of each SPNEGO TAI instance is controlled by custom configuration properties that are used to identify, for example, the criteria used to filter HTTP requests, such as the host name and security realm name used to construct the Kerberos Service Principal Name (SPN). For more information regarding establishing and setting the SPNEGO TAI custom configuration properties, see the following topics:

- Setting up the Kerberos configuration properties. See “Kerberos configuration requirements for SPNEGO TAI” on page 189.
- Setting or adjusting the SPNEGO TAI custom attributes. See SPNEGO TAI custom configuration attributes.
- Adjusting the SPNEGO TAI filter settings. See Filtering HTTP requests for SPNEGO TAI
- Using the custom login module to map the identity from the Active Directory to the WebSphere Application Server registry. See Mapping user Ids from client to server for SPNEGO.
- Setting the major and additional Java virtual machine (JVM) attributes. See SPNEGO TAI JVM configuration attributes

The Web administrator has access to the following SPNEGO TAI security components and associated configuration data, as illustrated in the following graphic.

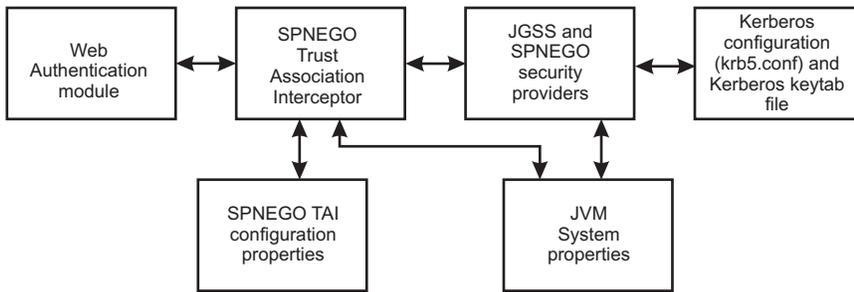


Figure 2. SPNEGO TAI security and configuration elements

- The Web authentication module and the Lightweight Third Party Authentication (LTPA) mechanism provide the plug-in runtime framework for trust association interceptors. See “Configuring the Lightweight Third Party Authentication mechanism” on page 219 for more detail is configuring the LTPA mechanism for use with the SPNEGO TAI.
- The Java Generic Security Service (JGSS) provider is included in the Java SDK (`jre/lib/ibmjgssprovider.jar`) and used to obtain the Kerberos security context and credentials that are used for authentication. IBM JGSS 1.0 is a Java Generic Security Service Application Programming Interface (GSSAPI) framework with Kerberos V5 as the underlying default security mechanism. GSSAPI is a standardized abstract interface under which can be plugged different security mechanisms based on private-key, public-key and other security technologies. GSSAPI shields secure applications from the complexities and peculiarities of the different underlying security mechanisms. GSSAPI provides identity and message origin authentication, message integrity, and message confidentiality. For more information, see JGSS
- The Kerberos configuration properties (`krb5.conf` or `krb5.ini`) and Kerberos encryption keys (stored in a Kerberos keytab file) are used to establish secure mutual authentication.
The Kerberos key table manager (Ktab), which is part of JGSS, allows you to manage the principal names and service keys stored in a local Kerberos keytab file. Principal name and key pairs listed in the Kerberos keytab file allow services running on a host to authenticate themselves to the Kerberos Key Distribution Center (KDC). Before a server can use Kerberos, a Kerberos keytab file must be initialized on the host that runs the server.
“Kerberos configuration requirements for SPNEGO TAI” on page 189 highlights the Kerberos configuration requirements for the SPNEGO TAI as well as the use of Ktab.
- The SPNEGO provider supplies the implementation of the SPNEGO authentication mechanism, located at `/$WAS_HOME/java/jre/lib/ext/ibmspnego.jar`.
- The custom configuration properties control the runtime behavior of the SPNEGO TAI. Configuration operations are performed with the administrative console or scripting facilities. Refer to “SPNEGO TAI custom configuration attributes” on page 236 for more information about these custom configuration properties.
- Java virtual machine (JVM) system properties control diagnostic trace information for problem determination of the JGSS security provider and use of the property reload feature. “SPNEGO TAI JVM configuration attributes” on page 247 describes these JVM configuration attributes

The benefits of having WebSphere Application Server use the SPNEGO TAI include:

- An integrated single sign-on environment with Microsoft Windows 2000 or 2003 Servers using Active Directory domain is established.
- The cost of administering a large number of ids and passwords is reduced.
- A secure and mutually authenticated transmission of security credentials from the Web browser or .NET clients is established.
- Interoperability with Web services and .NET applications that use SPNEGO authentication at the transport level is achieved.

Using the SPNEGO TAI in your WebSphere Application Server environment requires planning then implementation. See “Configuring single sign-on capability with SPNEGO TAI” on page 232 for the steps to take in planning for SPNEGO TAI. Implementing the use of the SPNEGO TAI is divided into the following areas responsibility:

End user

The end user must configure the Web browser or .NET application to issue HTTP requests that are processed by the SPNEGO TAI.

Web administrator

The Web administrator is responsible for configuring the SPNEGO TAI of WebSphere Application Server to respond to HTTP requests of the client.

WebSphere Application Server administrator

The WebSphere Application Server administrator is responsible for configuring WebSphere Application Server and the SPNEGO TAI for optimum installation performance.

See “Configuring WebSphere Application Server environment to use SPNEGO” on page 233 for an explanation of the tasks required to use the SPNEGO TAI and the responsible party associated with each task.

Kerberos configuration requirements for SPNEGO TAI

Kerberos configuration settings, the Kerberos key distribution center (KDC) name, and realm settings for the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) are provided in the Kerberos configuration file or through `java.security.krb5.kdc` and `java.security.krb5.realm` system property files.

The Web administrator creates the Kerberos configuration file with the appropriate settings that allow HTTP requests to be processed by the SPNEGO TAI. See “Creating the Kerberos configuration file for use with the SPNEGO TAI” on page 234 for more information.

The Web administrator can provide the same Kerberos configuration system properties in separate files: `java.security.krb5.kdc` and `java.security.krb5.realm`. See Kerberos Requirements for information on how this is accomplished.

The Kerberos key table manager command (Ktab) allows the Web administrator to manage the principal names and service keys stored in a local Kerberos keytab file. Kerberos service principal (SPN) name and keys listed in the Kerberos keytab file allow services running on the host to authenticate themselves to the KDC. Before SPNEGO TAI can use Kerberos, the WebSphere Application Server administrator must setup a Kerberos keytab file on the host running WebSphere Application Server.

Important: It is very important to protect the keytab files, making them readable only by the authorized WebSphere users.

Important: Any updates to the Kerberos keytab file using Ktab do not affect the Kerberos database. If you change the keys in the Kerberos keytab file, you must also make the corresponding changes to the Kerberos database.

Below is an example of how Ktab is used on a LINUX platform to add new principal names to the Kerberos keytab file.

```
[root@wssecjibe bin]# ./java com.ibm.security.krb5.internal.tools.Ktab -a
HTTP/wssecjibe.austin.ibm.com@WSSEC.AUSTIN.IBM.COM ot56prod -k /etc/krb5.keytab
Done!
Service key for principal HTTP/wssecjibe.austin.ibm.com@WSSEC.AUSTIN.IBM.COM saved
[root@wssecjibe bin]# ./java com.ibm.security.krb5.internal.tools.Ktab
1 entries in keytab, name: /etc/krb5.keytab
  KVNO      Principal
  ----      -
1          HTTP/wssecjibe.austin.ibm.com@WSSEC.AUSTIN.IBM.COM
```

```

[root@wssecjibe bin]# ls /etc/krb5.*
/etc/krb5.conf      /etc/krb5.ini.orig  /etc/krb5.keytab.good
/etc/krb5.conf.orig /etc/krb5.keytab
[root@wssecjibe bin]# ./java com.ibm.security.krb5.internal.tools.Ktab -a
HTTP/wssecdhat.austin.ibm.com@WSSEC.AUSTIN.IBM.COM ot56prod -k /etc/krb5.keytab
Done!
Service key for principal HTTP/wssecdhat.austin.ibm.com@WSSEC.AUSTIN.IBM.COM
saved
[root@wssecjibe bin]# ./java com.ibm.security.krb5.internal.tools.Ktab
2 entries in keytab, name: /etc/krb5.keytab
      KVNO      Principal
      ----      -
1      HTTP/wssecjibe.austin.ibm.com@WSSEC.AUSTIN.IBM.COM
1      HTTP/wssecdhat.austin.ibm.com@WSSEC.AUSTIN.IBM.COM

```

Tip: On WebSphere Application Server, Ktab is located at:

```
<install root>/java/jre/bin
```

Global single sign-on principal mapping

You can use the Java Authorization Contract for Containers (JACC) provider for Tivoli Access Manager to manage authentication to enterprise information systems (EIS) such as databases, transaction processing systems, and message queue systems that are located within the WebSphere Application Server security domain. Such authentication is achieved using the global single sign-on (GSO) principal mapper Java Authentication and Authorization Service (JAAS) login module for Java 2 Platform, Enterprise Edition (J2EE) Connector Architecture resources.

With GSO principal mapping, a special-purpose JAAS login module inserts a credential into the subject header. This credential is used by the resource adapter to authenticate to the 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 JACC provider for Tivoli Access Manager bypasses the credential that is stored in the Extensible Markup Language (XML) configuration files and uses the Tivoli Access Manager global sign-on (GSO) database instead to provide the authentication information for the EIS security domain.

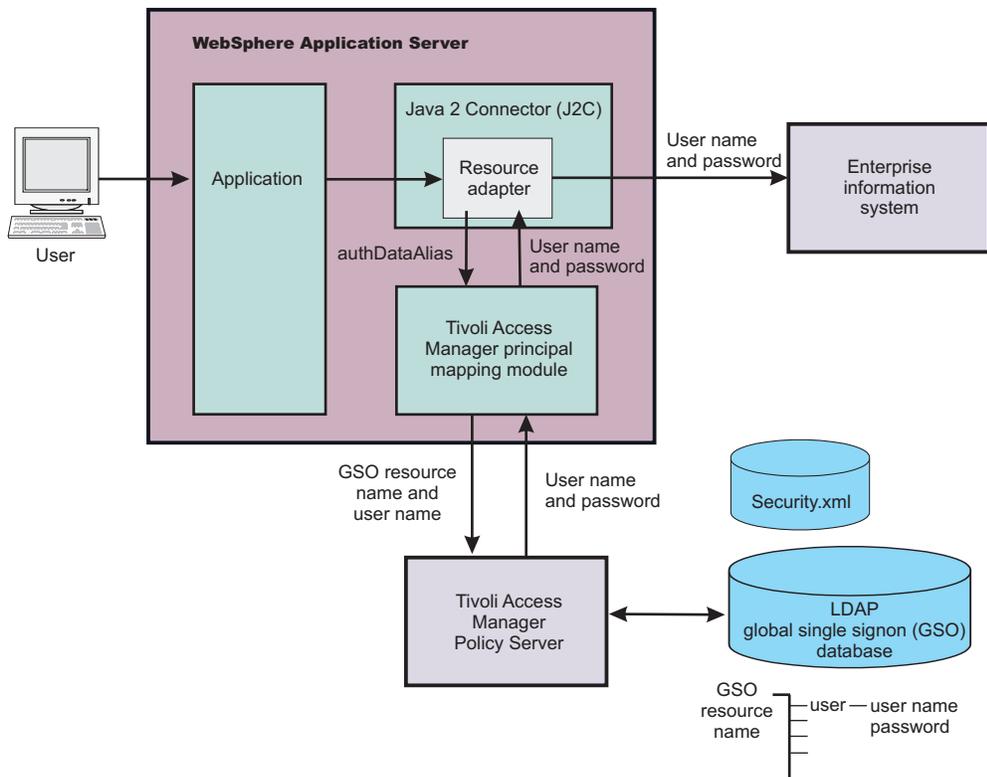
WebSphere Application Server provides a default principal mapping module that associates user credential information with EIS resources. The default mapping module is defined in the WebSphere Application Server administrative console on the Application login panel. To access the panel, click **Security > Secure administration, applications, and infrastructure**. Under Java Authentication and Authorization Service, 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; this attribute contains an alias that refers to a user name and password pair that is defined elsewhere.

The Tivoli Access Manager principal mapping module uses the authDataAlias attribute to determine the GSO resource name and the user name that is required to perform the lookup on the Tivoli Access Manager GSO database. The Tivoli Access Manager Policy Server retrieves the GSO data from the user registry.

Tivoli Access Manager stores authentication information on the Tivoli Access Manager GSO database against a resource and user name pair.

GSO principal mapping architecture



Security attribute propagation

With *Security attribute propagation*, WebSphere Application Server can 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 that is 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 deserialize 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 runtime 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 Version 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 request 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 a Lightweight Third Party Authentication (LTPA) token, and then deserializing a series of tokens that constitute both custom objects and token framework objects known to 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 propagation token” on page 195 and “Implementing a custom propagation token” on page 600.

Single sign-on (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 with the getVersion method defines the cookie name. WebSphere Application Server defines a default SingleSignonToken token 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 sign-on scenario. For more information on SSO tokens, see “Implementing single sign-on to minimize Web user authentications” on page 228. For information on how to use and implement this token type, see “Default single sign-on token” on page 203 and “Implementing a custom single sign-on token” on page 616.

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 run by the target server. The attributes that are 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 propagation token” on page 195 and “Implementing a custom propagation token” on page 600.

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 does 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 how the custom data uses the token to perform custom authentication decisions. For information on how to use and implement this token type, see “Default authentication token” on page 204 and “Implementing a custom authentication token” on page 627.

Horizontal propagation versus downstream propagation

In WebSphere Application Server, both horizontal propagation, which uses single sign-on 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 among 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 sign-on (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. To enable horizontal propagation, you must configure the single sign-on token and the Web inbound security attribute propagation features. You can configure both of these features using the administrative console.

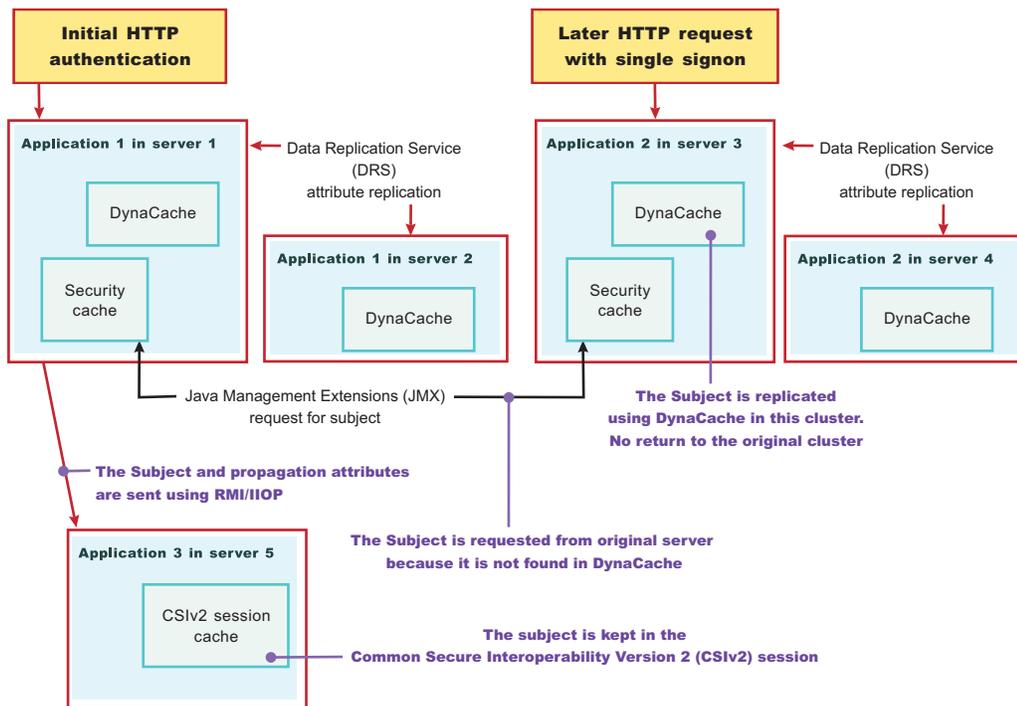
When front-end servers are configured and in the same distributed replication service (DRS) replication domain, the application server automatically propagates the serialized information to all of the servers within the same domain. In figure 1, application 1 is deployed on server 1 and server 2, and 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 five to six separate remote calls.

Figure 1

1. User authenticates to server 1.
2. Server 1 makes an RMI request to server 5.
3. User accesses another Web application on server 3.



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 WSPincipal 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 WSPincipal 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 getUniqueld() 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.

Default propagation token

A default propagation token is located on the running thread for applications and the security infrastructure to use. WebSphere Application Server propagates this default propagation token downstream and the token stays on the thread where the invocation lands at each hop.

The data is available from within the container of any resource where the propagation token lands. Remember that you must enable the propagation feature at each server where a request is sent for propagation to work. Make sure that you enable 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 topic documents the usage scenarios and includes examples. A close relationship exists between the propagation token and the work area feature. The main difference between these features is that after you add attributes to the propagation token, you cannot change the attributes. You cannot change these attributes so that the security runtime 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 object is stored to hold that attribute. Any new attribute that is added with the same key

is added to the ArrayList object. When you call `getAttributes`, the ArrayList object is converted to a String array and the order is preserved. The first element in the String array is the first attribute added for that specific key.

In the default propagation token, 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 send the authentication information downstream again 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 propagation token changes, a new session is generated and subsequently a new authentication occurs. Frequent changes to the propagation token during a method cause frequent downstream calls. If you change the token prior to making many downstream calls or you change the token between each downstream call, you might impact security performance.

Getting the server list from the default propagation token

Every time the propagation token is propagated and used to create the authenticated Subject, either horizontally or downstream, the name of the receiving application server is logged into the propagation token. 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_name:server_name*. The output, for example, is: `myManager:node1:server1`

Getting the caller list from the default propagation token

A default propagation token is generated any time an authenticated user is set on the running thread or anyone tries to add attributes to the propagation token. Whenever an authenticated user is set on the thread, the user is logged in the default propagation token. 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 that is added to the thread gets logged into the propagation token:

- The current Subject must be authenticated. For example, an unauthenticated Subject is not logged.
- The current authenticated Subject is logged if a Subject is not previously logged.
- The current authenticated Subject is logged if the last authenticated Subject that is logged does not contain the same user.
- The current authenticated Subject is logged on each unique application server that is 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_name:server_name:realm:port_number/securityName*. The output, for example, is: `myManager:node1:server1:ldap.austin.ibm.com:389/jsmith`.

Getting the first caller from the default propagation token

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 security name of the caller only. If you need to know more than the security name, call the `getCallerList` method and retrieve the first entry in the String array. This entry provides all the caller information. The following code sample retrieves the security name 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 application server name from the default propagation token

Whenever you want to know what the first application server is for this request, 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 propagation token

You can add custom attributes to the default propagation token for application usage. This token follows the request downstream so that the attributes are available when needed. When you use the default propagation token to add attributes, you must understand the following issues:

- Adding information to the propagation token 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 you need. However, all of the values must be available from a returned String array in the order that they were added.
- The propagation token is available only on servers where propagation and security are enabled.
- The Java 2 Security `javax.security.auth.AuthPermission wssecurity.addPropagationAttribute` attribute is needed to add attributes to the default propagation token.

- 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 propagation token” to retrieve attributes using the `getPropagationAttributes` application programming interface (API).

Getting custom attributes from the default propagation token

Custom attributes are added to the default propagation token using the `addPropagationAttribute` API. Retrieve these attributes using the `getPropagationAttributes` API. This token follows the request downstream so the attributes are available when needed. When you use the default propagation token to retrieve attributes, you must understand the following issues:

- The propagation token is available only on servers where propagation and security are enabled.
- The Java 2 Security `javax.security.auth.AuthPermission "wssecurity.getPropagationAttributes"` permission is needed to retrieve attributes from the default propagation token.

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 token factory that is associated with the default propagation token

When WebSphere Application Server generates a default propagation token, 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 > Secure administration, applications, and infrastructure**.
2. Under Additional properties, click **Custom properties**.

The default token factory that is specified for this property is called com.ibm.ws.security.ltpa.AuthzPropTokenFactory. This token factory encodes the data in the propagation token and does not encrypt the data. Because the propagation token typically flows over CSIV2 using Secure Sockets Layer (SSL), encrypting the token is not required. However, if you need additional security for the propagation token, you can associate a different token factory implementation with this property to get encryption. For example, if you choose to associate the com.ibm.ws.security.ltpa.LTPAToken2Factory token factory 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 propagation token is a good reason to change the token factory implementation to something that encrypts rather than just encodes.

If you want to perform your own signing and encryption of the default propagation token, you must implement the following classes:

- com.ibm.wsspi.security.ltpa.Token
- com.ibm.wsspi.security.ltpa.TokenFactory

Your token factory implementation instantiates and validates your token implementation. You can choose to use the Lightweight Third Party Authentication (LTPA) keys and have them pass into the initialize method of the token factory, or you can use your own keys. If you use your own keys, they must be the same everywhere to validate the tokens that are generated using those keys. See the API documentation, available through a link on the front page of the information center, for more information on implementing your own custom token factory. To associate your token factory with the default propagation token, using the administrative console, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.

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 token factory implementation.
4. Verify that your implementation classes are put into the `app_server_root/classes` directory so that the WebSphere Application Server class loader can load the classes.
5. Verify that your implementation classes are located in the `${USER_INSTALL_ROOT}/classes` directory so that the WebSphere Application Server class loader can load the classes.

Related tasks

“Propagating security attributes among application servers” on page 267

Use the security attribute propagation feature of WebSphere Application Server to send security attribute information regarding the original login to other servers using a token. This topic will help to configure WebSphere Application Server to propagate security attributes to other servers.

Default authorization token

This topic explains how WebSphere Application Server uses the default authorization token. Consider using the default authorization token when you are looking for a place to add string attributes that get propagated downstream.

However, make sure that the attributes you add to the authorization token are specific to the user that is associated with the authenticated Subject. If they are not specific to a user, the attributes probably belong in the propagation token, which is also propagated with the request. For more information on the propagation token, see “Default propagation token” on page 195. To add attributes into the authorization token, 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.Im.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 authorization token is created in the `wsMapDefaultInboundLoginModule` login module after the login occurs in the `ltpaLoginModule` login module. You can obtain a reference to the default authorization token 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 authorization token.

For more information on the Java Authentication and Authorization Service (JAAS) programming model, see *Security: Resources for learning*.

Important: Whenever you plug 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 `app_server_root/classes` directory, it has Java 2 Security AllPermissions permissions. 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 that are needed to run the application programming interfaces (API) that are 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 authorization token 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 579.) // 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 do during login, see // “Custom login
module development for a system login configuration” on page 579.) // 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); // Get the 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 579.) } private java.util.Map _sharedState = null;
private com.ibm.wsspi.security.token.AuthorizationToken defaultAuthzToken = null; }

```

Changing the token factory that is associated with the default authorization token

When WebSphere Application Server generates a default authorization token, 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 > Secure administration, applications, and infrastructure**.
2. Under Additional properties, click **Custom properties**.

The com.ibm.ws.security.ltpa.AuthzPropTokenFactory token factory is the default. This token factory encodes the data, but does not encrypt the data in the authorization token. Because the authorization token typically flows over Common Secure Interoperability Version 2 (CSIv2) using Secure Sockets Layer (SSL), encrypting the token is not necessary. However, if you need additional security for the authorization token, you can associate a different token factory implementation with this property to get encryption. For example, if you associate the com.ibm.ws.security.ltpa.LTPAToken2Factory token factory with this property,

the token uses Advanced Encryption Standard (AES) encryption. However, you need to weigh the performance impacts against your security needs. Adding sensitive information to the authorization token is one reason to change the token factory implementation to something that encrypts rather than just encodes.

If you want to perform your own signing and encryption of the default authorization token, you must implement the following classes:

- `com.ibm.wsspi.security.ltpa.Token`
- `com.ibm.wsspi.security.ltpa.TokenFactory`

Your token factory 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 token factory or you can use your own keys. If you use your own keys, they must be the same everywhere to validate the tokens that are generated using those keys. See the API documentation, that is available through a link on the front page of the information center, for more information on implementing your own custom token factory. To associate your token factory with the default authorization token, using the administrative console, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
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 token factory implementation.
4. Verify that your implementation classes are put into the `app_server_root/classes` directory so that the WebSphere Application Server class loader can load the classes.
5. Verify that your implementation classes are put into the `${USER_INSTALL_ROOT}/classes` directory so that the WebSphere Application Server class loader can load the classes.

Default single sign-on token

Do not use the default single sign-on token in service provider code. This default token is used by the WebSphere Application Server run-time code only.

Size limitations exist 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 single sign-on token. To implement a custom single sign-on token see “Implementing a custom single sign-on token” on page 616 for more information.

Changing the token factory that is associated with the default single sign-on token

When the default single sign-on token 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 > Secure administration, applications, and infrastructure**.
2. Under Additional properties, click **Custom properties**.

The `com.ibm.ws.security.ltpa.LTPAToken2Factory` token factory is the default that is specified for this property. This token factory creates a single sign-on (SSO) token called `LtpaToken2`, which WebSphere Application Server uses for propagation. This token factory uses the AES/CBC/PKCS5Padding cipher. If you change this token factory, you lose the interoperability with any servers running a version of WebSphere Application Server prior to version 5.1.1 that use the default token factory. Only servers running WebSphere Application Server Version 5.1.1 or later with propagation enabled are aware of the `LtpaToken2` cookie. If all of your application servers use WebSphere Application Server Version 5.1.1 or later and all of your servers use your new token factory this awareness is not a problem.

If you need to perform your own signing and encryption of the default single sign-on token, you must implement the following classes:

- `com.ibm.wsspi.security.ltpa.Token`

- `com.ibm.wsspi.security.ltpa.TokenFactory`

Your token factory implementation instantiates (`createToken`) and validates (`validateTokenBytes`) your token implementation. You can use the Lightweight Third-Party Authentication (LTPA) keys passed into the `initialize` method of the token factory or you can use your own keys. If you use your own keys, they must be the same everywhere to validate the tokens that are generated using those keys. See the API documentation, available through a link on the front page of the information center, for more information on implementing your own custom token factory. To associate your token factory with the default single sign-on token using the administrative console, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
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 `app_server_root/classes` directory so that the WebSphere Application Server class loader can load the classes.
5. Verify that your implementation classes are located in the `${USER_INSTALL_ROOT}/classes` directory so that the WebSphere Application Server class loader can load the classes.

Related tasks

“Propagating security attributes among application servers” on page 267

Use the security attribute propagation feature of WebSphere Application Server to send security attribute information regarding the original login to other servers using a token. This topic will help to configure WebSphere Application Server to propagate security attributes to other servers.

“Implementing a custom single sign-on token” on page 616

You can create your own single sign-on token implementation. The single sign-on token implementation is set in the login Subject and added to the HTTP response as an HTTP cookie.

Default authentication token

Do not use the default authentication token 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 authentication token” on page 627 for more information.

Changing the token factory that is associated with the default authentication token

When WebSphere Application Server generates a default authentication token, 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 > Secure administration, applications, and infrastructure**.
2. Under Additional properties, click **Custom properties**.

The `com.ibm.ws.security.ltpa.LTPATokenFactory` token factory is the default for this property. The `LTPATokenFactory` token factory uses the `DESede/ECB/PKCS5Padding` cipher. This token factory creates an interoperable Lightweight Third Party Authentication (LTPA) token. If you change this token factory, 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 token factory implementation. However, if all of your application servers use WebSphere Application Server Version 5.1.1 or later and all of your servers use your new token factory, this interoperability is not a problem.

If you associate the `com.ibm.ws.security.ltpa.LTPAToken2Factory` token factory with the `com.ibm.wsspi.security.token.authenticationTokenFactory` property, the token is Advanced Encryption

Standard (AES) encrypted. However, you need to weigh the performance against your security needs. You might add additional attributes to the authentication token in the Subject during a login that are available downstream.

If you need to perform your own signing and encryption of the default authentication token, you must implement the following classes:

- `com.ibm.wsspi.security.ltpa.Token`
- `com.ibm.wsspi.security.ltpa.TokenFactory`

Your token factory implementation instantiates (`createToken`) and validates (`validateTokenBytes`) your token implementation. You can use the LTPA keys that are passed into the `initialize` method of the token factory or you can use your own keys. If you use your own keys, they must be the same everywhere to validate the tokens that are generated using those keys. See the API documentation, available through a link on the front page of the information center, for more information on implementing your own custom token factory. To associate your token factory with the default authentication token using the administrative console, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
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 token factory implementation.
4. Verify that your implementation classes are put into the `install_dir/classes` directory so that the WebSphere Application Server class loader can load the classes.

Simple WebSphere authentication mechanism

The Simple WebSphere authentication mechanism (SWAM) is intended for simple, non-distributed, single application server runtime environments.

Note: SWAM is deprecated in WebSphere Application Server Version 6.1 and will be removed in a future release.

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.

Because SWAM is intended for a single application server process, single sign-on (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.

UserRegistry interface methods

Implementing this interface enables WebSphere Application Server security to use custom registries. This capability extends 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 that are defined in the Custom User Registry panel propagate to this method.

For the FileRegistrySample.java sample file, the initialize method retrieves the names of the registry files that contain the user and group information.

This method is called during server bringup 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 security name. A credential is created for the user for authorization purposes. This user name is also returned for the getCallerPrincipal enterprise bean call and the servlet calls the getUserPrincipal and getRemoteUser methods. 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, you must verify that the user is valid in the registry.

For the FileRegistrySample.java sample file, 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 user registry and, if they match, the method returns the user being authenticated.

This method is called for various scenarios, for example, by the administrative console to validate the user information after the user registry is initialized. This method 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 that is 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 create the `CertificateMapNotSupportedException` exception. 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 transport layer authentication. When the identity assertion token, using the `CSlv2` 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 is the `X509Certificate` certificate. 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 is called for Web applications only, but in version 5.x and later, you 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 `customRealm` default name is used.

For the `FileRegistrySample.java` sample file, the `getRealm` method returns the `customRealm` string. One of the calls to this method occurs when the user registry information is validated. This method is the same method 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 zero (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 asterisk (*) returns all the users and a pattern of (b*) returns the users starting with *b*.

The return parameter is an object with a `com.ibm.websphere.security.Result` type. This object contains two attributes, a `java.util.List` and a `java.lang.boolean` attribute. The list contains the users that are returned and the Boolean flag indicates if more users are available in the user 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 `FileRegistrySample.java` sample file, the `getUsers` method retrieves the required number of users from the user registry and sets them as a list in the `Result` object. To find out if more users are presented 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 the 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 user 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 changes to take the limit value and limits the users that are returned. The return is changed to return a Result object, which consists of the list and a flag that indicates if more entries exist. When the list returns, use the Result.setList(List) method to set the list in the Result object. If more entries exist than requested in the limit parameter, set the Boolean attribute to true in the result object, using the 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 descriptive name is 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.

If display names existed for any user in WebSphere Application Server Version 4.x, these names were useful for the Enterprise JavaBean (EJB) method call getCallerPrincipal and the servlet calls getUserPrincipal and getRemoteUser. If the display names are 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 in some situations because the display names might not be unique in the user registry. Avoid this problem by changing the default behavior to return the user security name instead of the user display name in this version of the product. For more information on how to set properties for the custom registry, see the section on *Setting Properties for Custom Registries*.

In the FileRegistrySample.java sample file, this method returns the display name of the user whose name matches the user name that is provided. If the display name does not exist, this method returns an empty string.

This method can be called by the product to present the display names in the administrative console, or by using the command line and the **wsadmin** tool. Use this method for display purposes only. This method is the same as in Version 4.x.

```
public String getUniqueUserId(String userSecurityName)
    throws EntryNotFoundException,
           CustomRegistryException,
           RemoteException;
```

This method returns the unique ID of the user, given the security name.

In the FileRegistrySample.java sample file, this method returns the uniqueUserId value 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 `FileRegistrySample.java` sample file, 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 `FileRegistrySample.java` sample file, 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, when the name is obtained from the certificate if the user is not found as a valid user in the user 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 user 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 user registry. A limit of zero (0) implies to return all the groups that match the pattern and can cause problems for large user registries. Use this limit with care. The custom registry implementations are expected to support at least the wildcard search (*). For example, a pattern of asterisk (*) returns all the users and a pattern of (b*) returns the users starting with *b*.

The return parameter is an object of the `com.ibm.websphere.security.Result` type. This object contains the `java.util.List` and `java.lang.Boolean` attributes. The list contains the groups that are returned and the Boolean flag indicates whether more groups are available in the user 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 `FileRegistrySample.java` sample file, the `getUsers` method retrieves the required number of groups from the user registry and sets them as a list in the `Result` object. To find out if more groups are presented 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 the asterisk (*) and question mark (?) characters.

This method is called by the administrative console to add groups to roles in the various `map-groups-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 user 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 the limit parameter. Change to take the limit value and limit the users that are returned. The return is changed to return a `Result` object, which consists of the list and a flag that indicates whether more entries exist. Use the `Result.setList(List)` method to set the list in the `Result` object. If more entries exist than requested

in the `limit` parameter, set the Boolean attribute to true in the Result object using the `Result.setHasMore` method. 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 that describes the group that you can set in some user 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 `FileRegistrySample.java` sample file, this method returns the display name of the group whose name matches the group name that is 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 the command line using the **wsadmin** tool. This method is used for display purposes only.

```
public String getUniqueGroupId(String groupSecurityName)
    throws EntryNotFoundException,
           CustomRegistryException,
           RemoteException;
```

This method returns the unique ID of the group that is given the security name.

In the `FileRegistrySample.java` sample file, 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 `FileRegistrySample.java` sample file, this method returns the unique ID of all the groups that contain this `uniqueUserID` ID. This method is called when creating the credential for the user. As part of creating the credential, all the `groupUniqueIds` IDs 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 `FileRegistrySample.java` sample file, 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` ID.

```
public boolean isValidGroup(String groupSecurityName)
    throws CustomRegistryException,
           RemoteException;
```

This method indicates if the given group is a valid group in the registry.

In the `FileRegistrySample.java` sample file, 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 `FileRegistrySample.java` sample file, this method returns all the group security names that contain the `userSecurityName` name.

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. Because 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 zero (0) indicates to return all of the users in that group. This method is not directly called by the WebSphere Application Server security component. However, this method can be called by other components. In rare situations, if you are working with a user registry where getting all the users from any of your groups is not practical, you can create the `NotImplementedException` exception for the particular groups. In this case, verify that if the process choreographer is installed the staff assignments are not modeled using these particular groups. If no concern exists about returning the users from groups in the user registry, it is recommended that you do not create the `NotImplemented` exception when implementing this method.

The return parameter is an object with a `com.ibm.websphere.security.Result` type. This object contains the `java.util.List` and `java.lang.boolean` attributes. The list contains the users that are returned and the Boolean flag, which indicates whether more users are available in the user registry for the search pattern. This Boolean flag indicates to the client whether users are available in the user 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 zero (0). If the method succeeds in getting one more user, the Boolean flag is set to `true`.

In WebSphere Application Server Version 4, this `getUsers` method is mandatory for the product. For WebSphere Application Server Version 5.x or later, this method can create the `NotImplementedException`

exception in situations where it is not practical to get the requested set of users. However, create this exception in rare situations when as other components can be affected. In Version 4, this method accepts only the pattern parameter and returns a list. In Version 5, this method accepts the limit parameter. Change your implementation to take the limit value and limit the users that are returned. The return changes to return a Result object, which consists of the list and a flag that indicates whether more entries exist. When the list is returned, use the Result.setList(List) method to set the list in the Result object. If more entries than requested are 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.

Attention: The first two lines of the following code sample are split for illustrative purposes only.

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

In this release the WebSphere Application Server, the createCredential method is not called. You can return *null*. In the example, a *null* value is returned.

Authentication protocol for EJB security

WebSphere Application Server Version 6.1 servers support the CSiv2 authentication protocol only. SAS is only supported between Version 6.0.x and earlier version servers that have been federated in a Version 6.1 cell. The option to select between SAS, CSiv2, or both is only available in the administration console when a Version 6.0.x or earlier release has been federated in a Version 6.1 cell.

V6.0.x 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 is considered the strategic protocol.

Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

Invoking Enterprise Java Beans (EJB) methods in a secure WebSphere Application Server environment requires an authentication protocol to determine the level of security and the type of authentication that 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 that are determined by the object Interoperable Object Reference (IOR) with the client authentication requirements that are 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--Secure Sockets Layer (SSL) or TCP/IP?
- If 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?

V6.0.x 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.

V6.0.x 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

V6.0.x 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

V6.0.x 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 that are used by WebSphere Application Server are add-on Interoperable Inter-ORB Protocol (IIOP) services. IIOP is a request-and-reply communications protocol that is 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 that is located on the server. As mentioned previously, the authentication policy is established here 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 that are authenticated by the server, an authentication mechanism-specific token that is 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 re-authenticate or re-validate 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.

V6.0.x The fact that a service context contains a unique identity is another way for WebSphere Application Server to support both 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.

V6.0.x 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 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 that is 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 that are applied to the method and the roles given to the user.

If a service context is not found by the CSiv2 server request interceptor, the interceptor process looks at the transport connection to see if a client certificate chain is sent. This process 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 valid security information is not 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. 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 that is 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 that 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

Step 1:

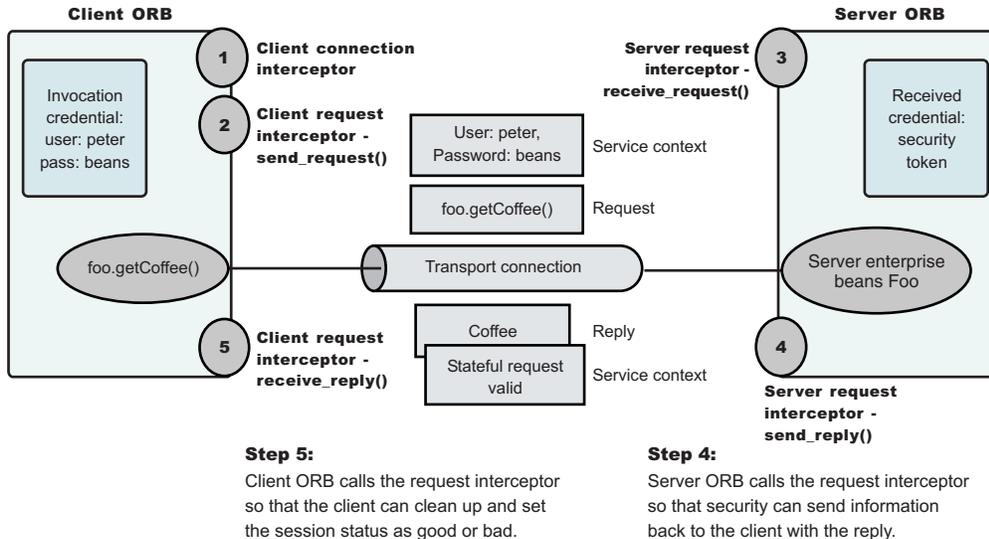
Client ORB calls the connection interceptor to create the connection.

Step 2:

Client ORB calls the request interceptor to get client security information.

Step 3:

Server ORB calls the request interceptor to receive the security information, authenticate, and set the received credential.



. 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 talk only to servers that either require or support that feature. When a server requires a feature, it can talk only 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 that requires 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 runtime 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 CSv2 specification requirements.

Supported authentication protocols

Use this page to reference information regarding supported authentication protocols.

V6.0.x Beginning with WebSphere Application Server Version 6.1, the WebSphere Application Server Version 6.1 servers only support the Common Secure Interoperability Version 2 (CSlv2) authentication protocol. Secure Authentication Service (SAS) is only supported between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell. The option to select between SAS, CSlv2, or both will only be made available in the administration console when a Version 6.0.x or previous release has been federated in a Version 6.1 cell.

V6.0.x In future releases, IBM will no longer ship or support the Secure Authentication Service (SAS) IIOP security protocol. It is recommended that you use the Common Secure Interoperability version 2 (CSlv2) protocol.

V6.0.x You can configure both protocols to work simultaneously between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell. If a server supports both protocols, it exports an interoperable object reference (IOR) that contains tagged components describing the configuration for SAS and CSlv2. If a client supports both protocols, it reads tagged components for both CSlv2 and SAS. If the client and server support both protocols, CSlv2 is used. However, if the server supports SAS (for example, the server 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.

Common Secure Interoperability Version 2 features

The following Common Secure Interoperability Version 2 (CSlv2) features are available in IBM WebSphere Application Server: Secure Sockets Layer (SSL) client certificate authentication, message layer authentication, identity assertion, and security attribute propagation.

- 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.

- Message Layer Authentication.

Authenticates credential information and sends that information across the network so that a receiving server can interpret it.

- 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 user registry 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. This information cannot be located using the user registry 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 the authorization engine 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 identity 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. 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 `ITTPPrincipal` 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.

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 (CSlv2) protocol.

Identity assertions with trust validation

If you want an application or system provider to perform an identity assertion with trust validation, it can be accomplished by use of the Java Authentication and Authorization Service (JAAS) login framework, where trust validation is performed in one login module and credential creation in another. These two custom login modules are used to create a JAAS login configuration that performs a login to an identity assertion.

Two custom login module are required:

- A user-implemented trust association login module. This login module performs whatever trust verification the user requires. When trust is verified, the trust verification status and the login identity must be placed in a map in the share state of the login module to enable the credential creation login

module to use that information. The map must be stored in the `com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.state` property. State maps contain the following information:

- `com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.trusted` – set to `true`, if trusted, and `false`, if not trusted.
- `com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.principal` – contains the principal of the identity.
- `com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.certificates` – contains the certificate of the identity
- The `com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule` module performs the credential creation. It requires that the trust state information be in the login context's shared state. This login module is protected by the Java 2 security runtime permissions for the following:
 - `com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.initialize`
 - `com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.login`

`IdentityAssertionLoginModule` searches for the trust information in the shared state property, `com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.state`. This is a map that contains the trust status and the identity used to login. The map includes the following:

- `com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.trusted` – if set to `true` it is trusted, `false` if not trusted.
- `com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.principal` – if a principal is used, it contains the principal of the identity necessary to login.
- `com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.certificates` – if a certificate is used, it contains an array of a certificate chain that includes the identity necessary to login.

A `WSLoginFailedException` is returned if the state, trust, or identity information is missing. The login module then performs a login of the identity. The subject now contains the new identity.

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 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, Generic Security Services Username Password (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, Lightweight Third Party Authentication (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

Note: SWAM is deprecated in WebSphere Application Server Version 6.1 and will be removed in a future release.

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.

Note: When **perform basic authentication** is enabled, if the client is not similarly configured (and does not pass a credential such as a user ID and password), the server object request broker (ORB) does not.

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.

Configuring the Lightweight Third Party Authentication mechanism

You must configure Lightweight Third Party Authentication (LTPA) when you set up security for the first time. LTPA is the default authentication mechanism for WebSphere Application Server.

1. Open the administrative console.

Type `http://fully_qualified_host_name:port_number/ibm/console` to access the administrative 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 > Secure administration, applications, and infrastructure > Authentication mechanisms and expiration**.
3. Select the appropriate group from the **Key set group** field that contains your public, private, and shared LTPA keys. These keys are used to encrypt and decrypt data that is sent between servers. You can access these key set group configurations using the Key set group link. In the Key set group configuration, you can indicate whether to automatically generate new keys and when to generate them.
4. Enter a positive integer value in the **Authentication cache 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 must log in again. An optimal value for this field depends on your configuration. However, the default value is 10 minutes.

5. Enter a positive integer in the **Timeout value for forwarded credentials between servers** field. This value refers to how long the server credentials from another server are valid before they expire. The default value is 120 minutes. The value in the **Timeout value for forwarded credentials between servers** field must be greater than the value in the **Authentication cache timeout** field.
6. 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 that are required to enable security, and start with single sign-on (SSO), if it is required.
7. Complete the information in the Security > Secure administration, applications, and infrastructure panel and click **OK**. The LTPA keys are generated automatically the first time. Do not generate the keys manually.

The previous steps configured LTPA.

After configuring LTPA, you can also complete the following tasks:

1. Generate key files. For more information, see “Generating Lightweight Third Party Authentication keys” on page 223.
2. Export key files. For more information, see “Exporting Lightweight Third Party Authentication keys” on page 223.
3. Import key files. For more information, see “Importing Lightweight Third Party Authentication keys” on page 224.
4. Manage LPTA keys from multiple cells. For more information, see `tsec_sslmanagelptakeys.dita`.
5. If you are enabling security, you can also enable single sign-on (SSO). See:
 - “Configuring single sign-on capability with Tivoli Access Manager or WebSEAL” on page 252
6. If you generated a new set of keys or imported a new set of keys, verify that the keys are saved to the master configuration 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 of the product servers that are participating in the protected domain. Changes to the time, date, and time zone are done independently from WebSphere Application Server. If the clock skew is too high between servers, the LTPA token seems prematurely expired and causes authentication or validation failures.

Authentication mechanisms and expiration

Use this page to specify the shared keys and configure the authentication mechanism that is used to exchange information between servers. You can also use this page to specify the amount of time that the authentication information remains valid and specify the single sign-on configuration.

To view this administrative console page, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Authentication, click **Authentication mechanisms and expiration**.

After you configure the properties on this page, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Verify that the appropriate registry is configured.
3. Click **Apply**. When security is enabled and any of these properties change, return to the Secure administration, applications, and infrastructure panel and click **Apply** to validate the changes.

Key set group

Specifies groups of public, private, and shared keys. These key groups enable the application server to manage multiple sets of Lightweight Third Party Authentication (LTPA) keys.

Generate Keys

Specifies whether to generate a new set of LTPA keys in the configured keystore, and to update the runtime with the new keys. By default, LTPA keys are regenerated on a schedule every 90 days, configurable to the day of the week.

Each new set of LTPA keys is stored in the keystore associated with the key set group. A maximum number of keys (or even one) can be configured. However, it is recommended to have at least two keys; the old keys can be used for validation while the new keys are being distributed.

This step is not necessary during security enablement. A default set of keys is created during the first server startup. If any nodes are down during a key generation event, the nodes should be synchronized with the Deployment Manager before restart.

Authentication cache timeout

Specifies the time period in minutes at which an LTPA token expires. Verify that this time period is less than the value for the Timeout value for forwarded credentials between servers field.

If the application server infrastructure 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 not accessed within the timeout period is purged from the cache. 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.

The default security cache timeout value is 10 minutes. If you have a small number of users, it should be set higher than that, or if a large number of users, it should be set lower.

The LTPA timeout value should not be set lower than the security cache timeout. It is also recommended that the LTPA timeout value should be set higher than the orb request timeout value. However, there is no relation between the security cache timeout value and the orb request timeout value.

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	Minutes and seconds
Default	10 minutes
Range:	Greater than 30 seconds

Timeout value for forwarded credentials between servers

Specifies the period of time after which forwarded credentials expire.

Specify a value for this field that is greater than the authentication cache timeout value.

Default	120 minutes
----------------	-------------

Password

Enter a password which will be used to encrypt and decrypt the LTPA keys from the SSO properties file. During import, this password should match the password used to export the keys at another LTPA server (for example, another application server Cell, Lotus Domino Server, and so on). During export, remember this password in order to provide it during the import operation.

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 that is used to encrypt and decrypt the LTPA keys.

Use this password when importing these keys into other application server administrative domain configurations and when configuring SSO for a Lotus Domino server.

Data type String

Fully qualified key file name

Specifies the name of the file that is used when importing or exporting keys.

Enter a fully qualified key file name, and click **Import Keys** or **Export Keys**.

Data type String

Internal server ID

Specifies the server ID that is used for interprocess communication between servers. The server ID is protected with an LTPA token when sent remotely. You can edit the internal server ID to make it identical to server IDs across multiple application server administrative domains (cells). By default this ID is the cell name.

This internal server ID should only be used in a Version 6.1 or higher environment. For mixed-version Cells, you should convert to using a server user ID and server password for interoperability.

To switch back to the server user ID and password for interoperability, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, click the **Available realm definitions** drop-down list, select a user registry, and click **Configure**.
3. Select the **Server identity that is stored in the repository** option and type a valid registry ID and password.

Data type String

Import Keys

Specifies whether the server imports new LTPA keys.

To support single sign-on (SSO) in the application server product across multiple application server 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, complete the following steps:

1. Enter the appropriate password in the Password and Confirm password fields.
2. Click **OK** and click **Save**.
3. Enter the directory location where the LTPA keys are located in the Fully qualified key file name field prior to clicking **Import keys**.
4. Do not click **OK** or **Apply**, but save the settings.

Export Keys

Specifies whether the server exports LTPA keys.

To support single sign-on (SSO) in the WebSphere product across multiple 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 Fully qualified key file name field and click **Export Keys**. The encrypted keys are stored in the specified file.

Use SWAM-no authenticated communication between servers

Specifies the Simple WebSphere Authentication Mechanism (SWAM). Unauthenticated credentials are forwarded between servers. When a caller process invokes a remote method, its identity is not verified. Depending upon the security permissions for the EJB methods, authentication failures might occur.

SWAM is a deprecated feature and will be removed in a future release. It is recommend that you use LTPA for authenticated communication between servers.

Generating Lightweight Third Party Authentication keys

WebSphere Application Server generates Lightweight Third Party Authentication (LTPA) keys automatically during the first server startup. You can generate additional keys as you need them in the Authentication mechanisms and expiration panel.

Complete the following steps to generate new LTPA keys in the administrative console.

1. Access the administrative console.
Type `http://fully_qualified_host_name:port_number/ibm/console` to access the administrative console in a Web browser.
2. Verify that all the WebSphere Application Server processes are running, including the cell, nodes, and application servers.

Important: If any of the servers are down at the time of key generation and then restarted later, these servers might contain old keys. Copy the new set of keys to these servers to restart them after you generate them.

3. Click **Security > Secure administration, applications, and infrastructure > Authentication mechanisms and expiration**.
4. Click **Generate keys** to generate a new set of LTPA keys in the local keystore and update the runtime with the new keys. By default, LTPA keys are regenerated on a schedule every 90 days, configurable to the day of the week. Each new set of LTPA keys is stored in the keystore that is associated with the key set group. The same password that is already stored in the configuration is used when you generate new keys.

Tip: This step is not necessary when you enable security because, by default, a set of keys is created during the first server startup. However, the keystore should have at least two keys: the old keys can be used for validation while the new keys are being distributed. If any nodes are down during a key generation event, the nodes should be synchronized with the Deployment Manager before restarting the server.

5. Restart the server for the changes to become active.

After WebSphere Application Server generates and saves a new set of keys, the generated keys are not used in the configuration until WebSphere Application Server is restarted. Token generation uses the keys that were last imported. To view the latest key version, see “Activating Lightweight Third Party Authentication key versions” on page 226.

Exporting Lightweight Third Party Authentication keys

To support single sign-on (SSO) in WebSphere Application Server across multiple WebSphere Application Server domains or cells, you must share the Lightweight Third Party Authentication (LTPA) keys and the password among the domains.

Make sure that the time in the domains is similar so that you do not mistakenly interpret the tokens as expired between the cells.

Complete the following steps in the administrative console to export key files for LTPA so that they can be shared across domains:

1. Type `http://server_name:port_number/ibm/console` in a Web browser to access the administrative console.
2. Click **Security > Secure administration, applications, and infrastructure > Authentication mechanisms and expiration**.
3. In the Password and Confirm password fields, enter the password that is used to encrypt the LTPA keys. Remember the password so that you can use it later when the keys are imported into the other cell.
4. In the Fully qualified key file name field, specify the fully qualified path to the location where you want the exported LTPA keys to reside. You must have write permission to this file.
5. Click **Export keys** to export the keys to the location that you specified in the **Fully qualified key file name** field.
6. Specify the **Internal server ID** that is used for interprocess communication between servers. The server ID is protected with an LTPA token when sent remotely. You can edit the internal server ID to make it identical to server IDs across multiple application server administrative domains (cells). By default this ID is the cell name.
7. Click **OK** and **Save**.

You can share LTPA keys and passwords among domains on WebSphere Application Server.

After exporting the keys from one cell, you must import those keys into the other cell. For more information, see “Importing Lightweight Third Party Authentication keys”

Importing Lightweight Third Party Authentication keys

To support single sign-on (SSO) in WebSphere Application Server across multiple WebSphere Application Server domains or cells, you must share the LTPA keys and the password among the domains. You can import LTPA keys from other domains and export keys to other domains.

After you export LTPA keys from one cell, you must import these keys into another cell. To import keys, you must know the password for the exported key file to access the LTPA keys. Verify that key files are exported from one of the cells into a file.

Complete the following steps in the administrative console to import key files for LTPA.

1. Access the administrative console for the cell that will receive the imported keys by typing `http://server_name:port_number/ibm/console` in a Web browser.
2. Click **Security > Secure administration, applications, and infrastructure > Authentication mechanisms and expiration**.
3. In the **Password** and **Confirm password** fields, enter the password that is used to decrypt the LTPA keys. This password must match the password that was used in the cell from which you are importing the keys.
4. In the **Fully qualified key file name** field, specify the fully qualified path to the location where the signer keys reside. You must have write permission to this file.
5. Click **Import keys** to import the keys to the location that you specified in the **Fully qualified key file name** field.
6. Click **OK** and **Save** to save the changes to the master configuration. It is important to save the new set of keys to match the new password so that no problems are encountered when starting the servers later.

After a new set of keys is generated and saved, the generated keys are not used in the configuration until WebSphere Application Server is restarted.

Important: After you enter the password in the Password and Confirm password fields and click **Save**, the password is not redisplayed on the administrative console panel.

Disabling automatic generation of Lightweight Third Party Authentication keys

You can disable the automatic generation of new Lightweight Third Party Authentication (LTPA) keys for key sets that are members of a key set group. Automatic generation creates new keys on a schedule that you specify when you configure a key set group, which manages one or more key sets. WebSphere Application Server uses key set groups to automatically generate cryptographic keys or multiple synchronized key sets.

You must know the name of the key set group and the management scope where the key set group is defined.

LTPA keys are used to encrypt the LTPA token. You might want to disable the auto-generation of these keys so that you can generate them on a schedule. The following steps are needed to complete this task in the administrative console.

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations**.
2. Expand the tree to the inbound or outbound management scope that contains the key set group, and then click the scope link.
3. Under Related Items, click **Key Set Groups**.
4. Click the key set group that you want to disable.
5. Clear the **Automatically generate keys** option.
6. Click **OK** and **Save** to save the changes to the master configuration.
7. Start the server again for the changes to become active.

You have disabled the automatic generation of LTPA keys for the key sets in the key set group.

Tip: You can generate keys manually at any time by completing the following steps:

1. Open the key set group collection.
2. Select the check box beside the key set group.
3. Click **Generate keys**.

Managing LTPA keys from multiple WebSphere Application Server cells

You can specify the shared keys and configure the authentication mechanism that is used to exchange information between servers to import and export LTPA keys across multiple WebSphere Application Server cells.

You must be sure that the exported key file for the multiple cells is accessible on the host where WebSphere Application Server is running. Also, you must know the password that was used when the keys were exported.

Complete the following steps to manage LTPA keys using the administrative console.

1. Access the administrative console.
Type `http://fully_qualified_host_name:port_number/ibm/console` to access the administrative console in a Web browser.
2. Verify that all of the WebSphere Application Server processes are running, including cells, 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, then bring them back up.

3. Click **Security > Secure administration, applications, and infrastructure > Authentication mechanisms and expiration**.
4. Type the password for the LTPA keys in the **Password** field. Enter a password that is used to encrypt and decrypt the LTPA keys from the single sign-on (SSO) properties file. During import, this password should match the password that is used to export the keys at another LTPA server. During export, remember this password in order to provide it during the import operation.
5. Type the password again in the **Confirm password** field.
6. Select from among the following options:
 - To support SSO in the WebSphere product across multiple application server domains (cells), you can share the LTPA keys and the password among the domains. Before exporting, make sure that security is enabled and using LTPA on the system that is running. For more information, see “Exporting Lightweight Third Party Authentication keys” on page 223.
 - To support SSO in the application server product across multiple application server domains (cells), you can share the LTPA keys and the password among the domains. For more information, see “Importing Lightweight Third Party Authentication keys” on page 224.
 - To import LTPA keys for the current cell if they were previously exported, see “Importing Lightweight Third Party Authentication keys” on page 224.
7. Start the server again for any changes you make to become active.

The shared LTPA keys are now available for WebSphere Application Server to use for secure connections.

After the keys are generated or imported, they are used to encrypt and decrypt the LTPA token. To view the latest key version, see “Activating Lightweight Third Party Authentication key versions.”

Activating Lightweight Third Party Authentication key versions

Key sets manage Lightweight Third Party Authentication (LTPA) keys in a key store that is based on a key alias prefix. A key alias prefix is automatically generated when you generate a new key and store it in a key store. Key stores can contain multiple versions of keys for any given key alias prefix. You can specify a maximum number of active keys in the key set configuration.

You must know the name of the key set group and the management scope where the key set group is defined. Complete the following steps in the administrative console.

LTPA keys are used to encrypt the LTPA token. You might want to set a specific number of active keys that WebSphere Application Server returns when the server queries for keys for a particular key set. The following steps are needed to complete this task in the administrative console.

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations**.
2. Expand the tree to the inbound or outbound management scope that contains the key set group, and then click the scope link.
3. Under Related Items, click **Key Sets**.
4. Click the key set that you want to modify.
5. In the **Maximum number of keys referenced** field, type a numerical value for the maximum number of keys that you want to activate.
6. Click **OK** and **Save** to save the changes to the master configuration.
7. Start the server again for the changes to become active. WebSphere Application Server activates only the number of recent keys that you specified.

The **Maximum number of keys referenced** value determines how many active keys are returned when the server queries for keys for the selected key set.

You can click **Active key history** in the Key set panel to display the keys that are active for this key set.

Integrating third-party HTTP reverse proxy servers

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.
Type `http://fully_qualified_host_name: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 > Secure administration, applications, and infrastructure**.
3. Under Web security, click **Trust association**.
4. Select the **Enable trust association** option.
5. Under Additional properties, click **Interceptors**. The default value appears.
6. Verify that the appropriate trust association interceptors are listed. If you need to use a WebSEAL trust association interceptor, see “Configuring single sign-on using the trust association interceptor” on page 260 or “Configuring single sign-on using trust association interceptor ++” on page 261. 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

Use this page to enable trust association, which integrates 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.

To view this administrative console page, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Authentication, expand Web security and click **Trust association**.

When security is enabled and any of these properties change, go to the Secure administration, applications, and infrastructure 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 > Secure administration, applications, and infrastructure**.
2. Under Authentication, expand Web security and click **Trust association**.
3. Under Additional Properties, click **Interceptors**.

When security is enabled and any of these properties are changed, go to the Secure administration, applications, and infrastructure 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`

Trust association interceptor settings

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 > Secure administration, applications, and infrastructure**.
2. Under Authentication, expand Web security and click **Trust association**.
3. Under Additional Properties, click **Interceptors > New**.

When security is enabled and any of these properties are changed, go to the Secure administration, applications, and infrastructure 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`

Implementing single sign-on to minimize Web user authentications

With single sign-on (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. Use this topic to configure single sign-on for the first time.

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, the realm name is the domain name if a domain is in use. If a domain is not used, the realm name is the machine name.

Linux The realm 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 sign-on is a subset of LTPA, it is recommended that you read “Lightweight Third Party Authentication” on page 178 for more information.

When you enable security attribute propagation, the following cookies are added to the response:

LtpaToken

LtpaToken is used for inter-operating 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 that are 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 191.

Note: LtpaToken is generated for releases prior to WebSphere Application Server Version 5.1.1. LtpaToken2 is generated for WebSphere Application Server Version 5.1.1 and beyond.

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, which is located in the SSO configuration panel in the administrative console. To access this panel, complete the following steps: <ol style="list-style-type: none"> 1. Click Security > Secure administration, applications, and infrastructure. 2. Under Web security, click Single sign-on (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 supports multiple attributes that are 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: <ol style="list-style-type: none"> 1. Click Security > Secure administration, applications, and infrastructure. 2. Under Web security, click Single sign-on (SSO).

Token type	Purpose	How to specify
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: <ol style="list-style-type: none"> 1. Click Security > Secure administration, applications, and infrastructure. 2. Under Web security, click Single sign-on (SSO).

The following steps are required to configure SSO for the first time.

1. Open the administrative console.

Type `http://localhost:port_number/ibm/console` to access the administrative 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 > Secure administration, applications, and infrastructure**.
3. Under Web security, click **Single sign-on (SSO)**.
4. Click the **Enabled** option if SSO is disabled. After you click the **Enabled** option, make sure that you complete the remaining steps to enable security.
5. Click **Requires SSL** if all of the requests are expected to use HTTPS.
6. Enter the fully qualified domain names in the **Domain name** field where SSO is effective. 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.

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.

You can configure the Domain name field using any of the following values:

Domain name value type	Example	Purpose
Blank		The domain is not set. This causes the browser to set the domain to the request host name. The sign-on is valid on that single host only.
Single domain name	<code>austin.ibm.com</code>	If the request is to a host within the configured domain, the sign-on is valid for all hosts within that domain. Otherwise, it is valid on the request host name only.

Domain name value type	Example	Purpose
UseDomainFromURL	UseDomainFromURL	If the request is to a host within the configured domain, the sign-on is valid for all hosts within that domain. Otherwise, it is valid on the request host name only.
Multiple domain names	austin.ibm.com;raleigh.ibm.com	The sign-on is valid for all hosts within the domain of the request host name.
Multiple domain names and UseDomainFromURL	<ul style="list-style-type: none"> austin.ibm.com;raleigh.ibm.com; UseDomainFromURL 	The sign-on is valid for all hosts within the domain of the request host name.

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.

For more information, see “Single sign-on settings” on page 253.

7. **Optional:** Enable the **Interoperability mode** option if you want to support 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 token 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 tokens are added to the response. Otherwise, only the LtpaToken2 token is added to the response. If the **Web inbound security attribute propagation** option is disabled, then only the LtpaToken token is added to the response.
8. **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 191.

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 token using WSSecurityHelper application programming interfaces (APIs).

If you find that 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.

9. Click **OK**.

For the changes to take effect, save, stop, and restart all the product servers.

Configuring single sign-on capability with SPNEGO TAI

WebSphere Application Server provides a trust association interceptor (TAI) that uses the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) to securely negotiate and authenticate HTTP requests for secured resources in WebSphere Application Server. To deploy and use the SPNEGO TAI you need to examine your installation and decide on how best to configure the SPNEGO TAI.

Lightweight Third Party Authentication (LTPA) is the default authentication mechanism for WebSphere Application Server. However, you may need to configure LTPA prior to configuring the SPNEGO TAI. LTPA is the required authentication mechanism for all trust association interceptors. You can configure LTPA by clicking **Security > Secure administration, applications, and infrastructure > Authentication mechanisms and expiration**.

Note: Enabling Web security single sign-on (SSO) is optional when you configure the SPNEGO TAI. For more information, see “Implementing single sign-on to minimize Web user authentications” on page 228.

Answer the following questions to establish how the SPNEGO TAI is deployed.

1. What is your criteria for intercepting HTTP requests?
 - You must decide if the SPNEGO TAI deployment will use the HTTPHeaderFilter class as the default. If you do use this class, then you must specify the exact filter properties for this class. The default behavior of the SPNEGO TAI is to use the com.ibm.ws.spnego.HTTPHeaderFilter class to intercept all requests.
 - If you do not use the sample com.ibm.ws.spnego.HTTPHeaderFilter class, then you must define a new class that implements the com.ibm.wsspi.security.spnego.SpnegoTAIFilter interface.
 - You can decide to further control what HTTP requests are intercepted using the Service Provider Programming Interface (SPI), “Filtering HTTP requests for SPNEGO TAI” on page 250

See “SPNEGO TAI custom configuration attributes” on page 236 for descriptions of

- com.ibm.ws.security.spnego.SPN<id>.filterClass
- com.ibm.ws.security.spnego.SPN<id>.filter

2. Is user Id mapping to be used? If not, why not? WebSphere Application Server enables you to define or develop a custom login module to map user IDs. See “Mapping user IDs from client to server for SPNEGO” on page 249 for more detail about performing this mapping.

You must decide, before deploying the TAI, whether or not to use this custom login module to perform the SPNEGO TAI identity mapping

3. What type of encryption is to be used to process the SPNEGO tokens? Microsoft Windows Active Directory supports two different Kerberos encryption types: RC4-HMAC and DES-CBC-MD5. The IBM Java Generic Security Service (JGSS) library (and SPNEGO library) support both of these encryption types.

Restriction: RC4-HMAC encryption is only supported with a Windows 2003 Server key distribution center (KDC). RC4-HMAC encryption is not supported when using a Windows 2000 Server as a Kerberos KDC.

4. How will you handle credential delegation? Kerberos supports the delegation of credentials. A server that receives Kerberos credentials from a client can impersonate that client to other servers by using delegated credentials. Since SPNEGO TAI tokens are a wrapping of a Kerberos credential, a server that receives Kerberos credentials within an SPNEGO token can use those Kerberos credentials to impersonate the original user. That server can interact using SPNEGO over HTTP as a SPNEGO client to other SPNEGO servers by composing an appropriate HTTP Authorization header.

5. Will the SPNEGO TAI be deployed in a single or multiple domain name service (DNS) domain environment?

Web browsers running on Windows are sensitive to DNS domains. They only send a SPNEGO token when the target host name identifies a host name defined in the DNS domain of the client machine.

You can use HTTP redirection to support this configuration with the creation of a pseudo Kerberos service principal name (SPN) in each DNS domain. All SPNs that WebSphere Application Server supports must have their secret keys available in Kerberos keytab files. To enable single sign-on across multiple DNS domains, a separate Kerberos keytab file is generated for each SPN per domain. These individual Kerberos keytab files must be merged before they can be used by WebSphere Application Server.

6. How frequently will application servers reload the SPNEGO TAI properties The SPNEGO TAI has an optional property reload feature that allows the reloading of the TAI properties without restarting the Java virtual machine (JVM). This reload feature is controlled by the system properties `com.ibm.ws.security.spnego.propertyReloadFile` and `com.ibm.ws.security.spnego.propertyReloadTimeout`. These properties taken together enable the SPNEGO TAI internal properties to be reloaded from a file on the file system after a certain time period. If the `com.ibm.ws.security.spnego.propertyReloadTimeout` attribute is set to a valid integer value, and the `com.ibm.ws.security.spnego.propertyReloadFile` attribute points to a file on the file system, then each JVM reloads the SPNEGO TAI properties from the file after the timeout period expires. Also, the SPNEGO TAI properties are reloaded only if the date on the file has changed. If these reload properties are not set, then the SPNEGO TAI properties are only loaded once, at JVM initialization, from the SPNEGO TAI custom properties that are defined in WebSphere Application Server configuration data. See “SPNEGO TAI JVM configuration attributes” on page 247 for more information about these reload properties.

The Windows Active Directory (Web) administrator, the WebSphere Application Server administrator, and the application team review and answer these questions to determine the best deployment and configuration settings for the SPNEGO TAI.

Configuring WebSphere Application Server environment to use SPNEGO

The objective of the Web administrator is to configure and administer the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) in WebSphere Application Server to provide users who successfully authenticated in a Microsoft Active Directory domain with a single sign-on capability. The administrator specifies additional criteria that selects what Web transactions to authenticate in the single sign-on environment.

Verify that the Web browser is configured to use the SPNEGO authentication mechanism. “Configuring the Web browser to use SPNEGO” on page 252 describes what the user needs to do to configure the Web browser.

The Web administrator configures and enables the SPNEGO TAI. The process to configure and enable the SPNEGO TAI operation in WebSphere Application Server requires some tasks that use tools not supplied by WebSphere Application Server. Refer to those appropriate documents that describe these tasks and tools. These documents are supplied by the appropriate supplier.

1. Create a user account in the Microsoft Active Directory.
2. Map the user account to the Kerberos service principal name (SPN). This user account represents the WebSphere Application Server as being a Kerberize'd service with the Kerberos key distribution center (KDC). Use the **setspn** tool to establish WebSphere Application Server as the user. This user account is not the account name of the user. More information about the **setspn** tool can be found here, Windows 2003 Technical Reference (setspn command)
3. Create the Kerberos keytab file and make it available to WebSphere Application Server. Use the **ktpass** tool to create the Kerberos keytab file (krb5.keytab). Windows 2003 Technical Reference (Kerberos keytab file and ktpass command) provides more information on creating the Kerberos keytab file.
4. Configure and enable the application server and the associated SPNEGO TAI using the administrative console or using the `wsadmin` command to perform command tasks. See “Configuring SPNEGO TAI in WebSphere Application Server” on page 235.
5. Select Lightweight Third-Party Authentication (LTPA) as the authentication mechanism. See “Configuring the Lightweight Third Party Authentication mechanism” on page 219.

6. Enable the SPNEGO TAI in each application server in which it is defined.
7. Install the Kerberos keytab file (krb5.keytab).
8. Update the associated Kerberos configuration (krb5.conf).
9. Configure JVM properties and enable SPNEGO TAI. See “Configuring JVM properties and enabling SPNEGO TAI in WebSphere Application Server” on page 245.

WebSphere Application Server is configured to use the SPNEGO TAI.

Configuring the Kerberos configuration properties:

The Kerberos configuration properties, or krb5.ini and krb5.conf files, must be configured on every WebSphere Application Server instance in a cell in order to use the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) for WebSphere Application Server.

A configuration file for a Kerberos configuration on a UNIX platform follows:

```
[libdefaults]
default_realm = WSSEC.AUSTIN.IBM.COM
    default_keytab_name = FILE:/etc/krb5.keytab
    default_tkt_enctypes = des-cbc-md5
    default_tgs_enctypes = des-cbc-md5

[realms]
WSSEC.AUSTIN.IBM.COM = {
kdc = axel.austin.ibm.com:88
    default_domain = austin.ibm.com
admin_server = axel.austin.ibm.com
    default domain = austin.ibm.com
}

[domain_realm]
.austin.ibm.com = WSSEC.AUSTIN.IBM.COM
```

In the above example, the Kerberos key distribution center (KDC) is axel.austin.ibm.com:88. The Kerberos keytab file is located at: FILE:/etc/krb5.keytab. The Kerberos realm name is WSSEC.AUSTIN.IBM.COM, which is also the Microsoft domain controller. After you update your Kerberos configuration properties for your particular system deployment, your Kerberos configuration is ready for use with the SPNEGO TAI.

The Kerberos configuration is configured for use with the SPNEGO TAI.

Creating the Kerberos configuration file for use with the SPNEGO TAI:

You use the wsadmin utility to create the Kerberos keytab configuration file for use with the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) for WebSphere Application Server.

Note: A Kerberos keytab configuration file contains a list of keys that are analogous to user passwords. It is important for hosts to protect their Kerberos keytab files by storing them on the local disk. The krb5.conf file permission must be 644, which means that you can read and write the file; however, members of the group that the file belongs to, and all others can only read the file.

Note: A Kerberos keytab configuration file contains a list of keys that are analogous to user passwords. It is important for hosts to protect their Kerberos keytab files by storing them on the local disk. The krb5.conf file permission must be 644, which means that you can read and write the file; however, members of the group that the file belongs to, and all others can only read the file. The user ID that runs adjnt, control, and servants must have read access to the krb5.conf and krb5.keytab files.

Use the wsadmin utility to configure the SPNEGO TAI for WebSphere Application Server:

1. Start WebSphere Application Server.

2. Start the command-line utility by running the **wsadmin** command from the `app_server_root/bin` directory.
3. At the **wsadmin** prompt, enter the following command:

```
$AdminTask createKrbConfigFile
```

You can use the following parameters with this command:

Option	Description
<krbPath>	This parameter is required. It provides the fully qualified file system location of the Kerberos configuration (krb5.ini or krb5.conf) file.
<realm>	This parameter is required. It provides the Kerberos realm name. The value of this attribute is used by the SPNEGO TAI to form the Kerberos service principal name for each of the hosts specified with the property <code>com.ibm.ws.security.spnego.SPN<id>.hostName</code> .
<kdcHost>	This parameter is required. It provides the host name of the Kerberos Key Distribution Center (KDC).
<kdcPort>	This parameter is optional. It provides the port number of the KDC. The default value, if not specified, is 88.
<dns>	This parameter is required. It provides the default domain name service (DNS) that is used to produce a fully qualified host name.
<keytabPath>	This parameter is required. It provides the file system location of the Kerberos keytab file.
<encryption>	This parameter is optional. It identifies the list of supported encryption types, separated by a space. The specified value is used for the <code>default_tkt_encypes</code> and <code>default_tgs_encypes</code> . The default encryption types, if not specified, are <code>des-cbc-md5</code> and <code>rc4-hmac</code> .

In the following example, the `wsadmin` command creates the `krb5.ini` file in the `c:\winnt` directory. The default Kerberos keytab file is also in `c:\winnt`. The actual Kerberos realm name is `WSSEC.AUSTIN.IBM.COM` and the KDC host name is `host1.austin.ibm.com`.

```
wsadmin>$AdminTask createKrbConfigFile {-krbPath
c:\winnt\krb5.ini -realm WSSEC.AUSTIN.IBM.COM -kdcHost host1.austin.ibm.com
-dns austin.ibm.com -keytabPath c:\winnt\krb5.keytab}
```

The Kerberos keytab configuration file is created for use with the SPNEGO TAI.

Note: The default Kerberos `krb5.ini` file on Windows is `/winnt/krb5.ini` and on a distributed environment is `/etc/krb5`. If you specify another location path, then you must also specify the `java.security.krb5.conf` JVM property.

For example, if your `krb5.conf` file is specified at `/opt/IBM/WebSphere/profiles/AppServer/etc/krb5.conf`, then you need to specify `-Djava.security.krb5.conf=/opt/IBM/WebSphere/profiles/AppServer/etc/krb5.conf`.

Configuring SPNEGO TAI in WebSphere Application Server:

Performing this task helps you, as Web administrator, to ensure that WebSphere Application Server is properly configured to enable the operation of the Simple and Protected GSS-API Negotiation (SPNEGO) trust association interceptor (TAI).

You need to know how to use the WebSphere Application Server administrative console to manage the security configuration and have the proper authority to modify the security configuration of the application server.

Note: It is recommended that you use wsadmin to manage the SPNEGO TAI properties.

Complete the following steps to enable the operation of the SPNEGO TAI.

1. Log on to the WebSphere Application Server administrative console.
2. Click **Security > Secure administration, applications, and infrastructure**.
3. Expand **Web security** and click **Trust association**.
4. Under the General Properties heading, select the **Enable trust association** check box, then click **Interceptors**.
5. Select the SPNEGO TAI in the list of interceptors, then click **Custom properties**.
6. Click **New** and then fill in the **Name** and **Value** fields. Click **OK**. Repeat this step for each custom property that you want to apply to the SPNEGO TAI.
7. After you finish defining your custom properties, click **Save** to store the updated SPNEGO TAI configuration.

Your SPNEGO TAI configuration is now configured for WebSphere Application Server. You must ensure that:

- A user account is created in the Microsoft Active Directory and mapped to a Kerberos principal name.
- A Kerberos keytab file (krb5.keytab) is created and made available to the WebSphere Application Server. The Kerberos keytab file contains keys WebSphere Application Server uses to authenticate the user in the Microsoft Active Directory and the Kerberos account.

SPNEGO TAI custom configuration attributes:

The Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) custom configuration attributes control different operational aspects of the SPNEGO TAI. You can specify different attribute values for each application server.

Each of the attributes defined in the following table is specified in the Custom Properties panel for the SPNEGO TAI using the administrative console facility. For convenience, you can optionally place these attributes in a properties file. In this case, the SPNEGO TAI loads the configuration attributes from the file instead of the Custom Properties panel definition. Refer to com.ibm.ws.security.spnego.propertyReloadFile property as defined in “SPNEGO TAI JVM configuration attributes” on page 247.

To assign unique attribute names that identify each possible SPN, an SPN<id> is embedded in the attribute name and used to group the attributes that are associated with each SPN. The SPN<id> s are numbered sequentially for each attribute group.

Table 4.

Attribute Name	Required	Default Value
com.ibm.ws.security.spnego.SPN<id>.hostName	Yes	None
com.ibm.ws.security.spnego.SPN<id>.filterClass	No	See the description that follows.
com.ibm.ws.security.spnego.SPN<id>.filter	No	See the description that follows.
com.ibm.ws.security.spnego.SPN<id>.enableCredDelegate	No	false
com.ibm.ws.security.spnego.SPN<id>.spnegoNotSupportedPage	No	See the description that follows.

Table 4. (continued)

Attribute Name	Required	Default Value
com.ibm.ws.security.spnego.SPN<id>.NTLMTokenReceivedPage	No	See the description that follows.
com.ibm.ws.security.spnego.SPN<id>.trimUserName	No	true

com.ibm.ws.security.spnego.SPN<id>.hostName

This attribute is required. It specifies the hostname in the SPN used by the SPNEGO TAI to establish a Kerberos secure context.

Note: The hostname is the long form of hostname. For example, myHostName.austin.ibm.com. The Kerberos SPN is a string of the form HTTP/hostname@realm. The complete SPN is used with the Java Generic Security Service (JGSS) by the SPNEGO provider to obtain the security credential and security context that are used in the authentication process.

com.ibm.ws.security.spnego.SPN<id>.filterClass

This attribute is optional. It specifies the name of the Java class that is used by the SPNEGO TAI to select which HTTP requests are subject to SPNEGO authentication. If no class is specified, the default com.ibm.ws.security.spnego.HTTPHeaderFilter implementation class is used. The Java class that is specified must implement the com.ibm.wsspi.security.spnego.SpnegoFilter interface. A default implementation of this interface is provided. Specify the com.ibm.ws.security.spnego.HTTPHeaderFilter class to use the default implementation. This class uses the selection rules specified with the com.ibm.ws.security.spnego.SPN<id>.filter property.

com.ibm.ws.security.spnego.SPN<id>.filter

This attribute is optional. It defines the filtering criteria that is used by the specified class with the previous attribute. It defines arbitrary criteria that is meaningful to the implementation class used. The com.ibm.ws.security.spnego.HTTPHeaderFilter default implementation class uses this attribute to define a list of selection rules that represent conditions that are matched against the HTTP request headers to determine whether or not the HTTP request is selected for SPNEGO authentication.

Each condition is specified with a key-value pair, separated from each other by a semicolon. The conditions are evaluated from left to right, as they display in the specified attribute. If all conditions are met, the HTTP request is selected for SPNEGO authentication.

The key and value in the key-value pair are separated by an operator that defines which condition is checked. The key identifies an HTTP request header to extract from the request and its value is compared with the value that is specified in the key-value pair according to the operator specification. If the header that is identified by the key is not present in the HTTP request, the condition is treated as not being met.

Any of the standard HTTP request headers can be used as the key in the key-value pairs. Refer to the HTTP specification for the list of valid headers. In addition, two keys are defined to extract information from the request, also useful as a selection criterion, which is not available through standard HTTP request headers. The remote-address key is used as a pseudo header to retrieve the remote TCP/IP address of the client application that sent the HTTP request. The request-URL key is used as a pseudo header to retrieve the URL that is used by the client application to make the request. The interceptor uses the result of the getRequestURL operation in the javax.servlet.http.HttpServletRequest interface to construct the Web address. If a query string is present, the result of the getQueryString operation in the same interface is also used. In this case, the complete URL is constructed as follows:

```
String url = request.getRequestURL() + '?' + request.getQueryString();
```

The following operators and conditions are defined:

Table 5. Filter conditions and operations

Condition	Operator	Example
Match exactly	= = Arguments are compared as equal.	host=host.my.company.com
Match partially (includes)	%= Arguments are compared with a partial match being valid.	user-agent%=IE 6
Match partially (includes one of many)	^= Arguments are compared with a partial match being valid for one of many arguments specified.	request-url^=webApp1 webApp2 webApp3
Does not match	!= Arguments are compared as not equal.	request-url!=noSPNEGO
Greater than	> Arguments are compared lexogaphically as greater than.	remote-address>192.168.255.130
Less than	< Arguments are compared lexogaphically as less than.	remote-address<192.168.255.135

com.ibm.ws.security.spnego.SPN<id>.enableCredDelegate

This attribute is optional. It indicates whether or not the Kerberos SPNEGO delegated credentials are stored by the SPNEGO TAI. This attribute enables the capability for an application to retrieve the stored credentials and propagate them to other applications downstream for additional SPNEGO authentication.

This attribute requires use of the advanced Kerberos credential delegation feature and requires development of custom logic by the application developer. The developer must interact directly with the Kerberos Ticket Granting Service (TGS) to obtain a Ticket Granting Ticket (TGT) using the delegated Kerberos credentials on behalf of the end-user who originated the request. The developer must also construct the appropriate Kerberos SPNEGO token and include it in the HTTP request to continue the downstream SPNEGO authentication process, including handling additional SPNEGO challenge-response exchange, if necessary.

com.ibm.ws.security.spnego.SPN<id>.spnegoNotSupportedPage

This attribute is optional. It specifies the Web address of a resource that contains the content that the SPNEGO TAI includes in the HTTP response that the (browser) client application displays if it does not support SPNEGO authentication. It can specify a Web (http://) or a file (file://) resource. If this attribute is not specified or the interceptor cannot find the specified resource, the following content is used:

```
<html><head><title>SPNEGO authentication is not supported</title></head>
<body>SPNEGO authentication is not supported on this client</body></html>;
```

com.ibm.ws.security.spnego.SPN<id>.NTLMTokenReceivedPage

This attribute is optional. It specifies the Web address of a resource that contains the content that the SPNEGO TAI includes in the HTTP response that the (browser) client application displays when the SPNEGO token is received by the interceptor when the challenge-response handshake contains a NT

LAN Manager (NTLM) token instead of the expected SPNEGO token. It can specify a Web (http://) or a file (file://) resource. If this attribute is not specified or the interceptor cannot find the specified resource, the following content is used:

```
<html><head><title>An NTLM Token was received.</title></head>
<body>Your browser configuration is correct, but you have not logged into a supported
Microsoft(R) Windows(R) Domain.
<p>Please login to the application using the normal login page.</html>
```

com.ibm.ws.security.spnego.SPN<id>.trimUserName

This attribute is optional. It specifies whether (true) or not (false) the SPNEGO TAI is to remove the suffix of the principal user name, starting from the "@" that precedes the Kerberos realm name. If this attribute is set to true, the suffix of the principal user name is removed. If this attribute is set to false, the suffix of the principal name is retained. The default value used is true. For example,

When `com.ibm.ws.security.spnego.SPN<id>.trimUserName = true`

`bobsmith@myKerberosRealm` becomes `bobsmith`

When `com.ibm.ws.security.spnego.SPN<id>.trimUserName = false`

`bobsmith@myKerberosRealm` remains `bobsmith@myKerberosRealm`

Note: The following commands tasks can be used to operate on these SPNEGO TAI attributes:

- "Adding SPNEGO TAI properties using the wsadmin utility"
- "Deleting SPNEGO TAI properties using the wsadmin utility" on page 244
- "Modifying SPNEGO TAI properties using the wsadmin utility" on page 242
- "Displaying SPNEGO TAI properties using the wsadmin utility" on page 244

Related concepts

"Single sign-on for HTTP requests using SPNEGO" on page 185

WebSphere Application Server provides a trust association interceptor (TAI) that uses the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) to securely negotiate and authenticate HTTP requests for secured resources in WebSphere Application Server.

Related tasks

"Adding SPNEGO TAI properties using the wsadmin utility"

You use the wsadmin utility to add properties for the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) in the security configuration for WebSphere Application Server.

"Deleting SPNEGO TAI properties using the wsadmin utility" on page 244

You use the wsadmin utility to delete properties in the configuration of the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) for WebSphere Application Server.

"Modifying SPNEGO TAI properties using the wsadmin utility" on page 242

You use the wsadmin utility to modify the properties in the configuration of the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) for WebSphere Application Server.

"Displaying SPNEGO TAI properties using the wsadmin utility" on page 244

You use the wsadmin utility to display the properties in the configuration of the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) for WebSphere Application Server.

"Mapping user Ids from client to server for SPNEGO" on page 249

You can use a system programming interface to customize the behavior of the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) by implementing arbitrary mappings of the end-user's identity, which is retrieved from Microsoft Active Directory to the identity that is used in the WebSphere Application Server security registry.

Adding SPNEGO TAI properties using the wsadmin utility:

You use the `wsadmin` utility to add properties for the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) in the security configuration for WebSphere Application Server.

Use the `wsadmin` utility to configure the SPNEGO TAI for WebSphere Application Server:

1. Start WebSphere Application Server.
2. Start the command-line utility by running the `wsadmin` command from the `app_server_root/bin` directory.
3. At the `wsadmin` prompt, enter the following command:
`$AdminTask addSpnegoTAIProperties`

You can use the following parameters with this command:

Option	Description
<code><spnId></code>	This is the SPN identifier for the group of custom properties that are to be defined with this command. If you do not specify this parameter, an unused SPN identifier is assigned.
<code><host></code>	It specifies the host name portion in the SPN used by the SPNEGO TAI to establish a Kerberos secure context. This parameter is required.
<code><filter></code>	This attribute is optional. It defines the filtering criteria used by the class specified with the above attribute. If no filter is specified, all HTTP requests are subject to SPNEGO authentication.
<code><filterClass></code>	This attribute is optional. It specifies the name of the Java class used by the SPNEGO TAI to select which HTTP requests will be subject to SPNEGO authentication. If no filter class is specified, the default filter class, <code>com.ibm.ws.security.spnego.HTTPHeaderFilter</code> , is used.
<code><noSpnegoPage></code>	This attribute is optional. It specifies the URL of a resource that contains the content the SPNEGO TAI will include in the HTTP response to be displayed by the (browser) client application if it does not support SPNEGO authentication. If you do not specify the <code>noSpnegoPage</code> attribute then the default is used: <pre>"<html><head><title>SPNEGO authentication is not supported. </title></head>" + "<body>SPNEGO authentication is not supported on this client. </body></html>";</pre>

Option	Description
<ntlmTokenPage>	<p>This attribute is optional. It specifies the URL of a resource that contains the content the SPNEGO TAI will include in the HTTP response to be displayed by the (browser) client application when the SPNEGO token received by the interceptor after the challenge-response handshake contains a NT LAN manager (NTLM) token instead of the expected SPNEGO token.</p> <p>If you do not specify the ntlmTokenPage attribute then the default is used:</p> <pre>"<html><head><title>An NTLM Token was received.</title></head>" + "<body>Your browser configuration is correct, but you have not logged into a supported Windows Domain." + "<p>Please login to the application using the normal login page.</html>";</pre>
<trimUserName>	<p>This parameter is optional. It specifies whether (true) or not (false) the SPNEGO TAI is to remove the suffix of the principal user name, starting from the "@" that precedes the Kerberos realm name. If this attribute is set to true, the suffix of the principal user name is removed. If this attribute is set to false, the suffix of the principal name is retained. The default value used is true.</p>

SPNEGO TAI properties have been added for this WebSphere Application Server.

Example 1

The following example configures the SPNEGO TAI to intercept HTTP request that contain IE 6 in the user agent request header. The SPNEGO TAI uses the SPN of HTTP/myhost.ibm.com@<default_realm> to authenticate the request originator.

```
$AdminTask addSpnegoTAIProperties -host myhost.ibm.com -filter user-agent%=IE 6
```

Example 2

The following is an example of adding SPNEGOTAIProperties for SPN1 to use the default filterClass and to intercept all requests for the host, central01.austin.ibm.com.

```
wsadmin>$AdminTask addSpnegoTAIProperties -interactive
Add SPNEGO TAI properties
```

Add SPNEGO TAI configuration properties.

```
*Host name in Service Principal Name (host): central01.austin.ibm.com
Service Principal Name identifier (spnId): 1
HTTP header filter rule (filter):
Name of class used to filter HTTP requests (filterClass):
SPNEGO not supported browser response (noSpnegoPage):
NTLM Token received browser response (ntlmTokenPage):
Trim User Name browser response (trimUserName):
```

Add SPNEGO TAI properties

```
F (Finish)
C (Cancel)
```

Select [F, C]: [F] f

```
WASX7278I: Generated command line: $AdminTask addSpnegoTAIProperties {-host central01.austin.ibm.com}
com.ibm.ws.security.spnego.SPN1.hostName=central01.austin.ibm.com
wsadmin>
```

Modifying SPNEGO TAI properties using the wsadmin utility:

You use the wsadmin utility to modify the properties in the configuration of the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) for WebSphere Application Server.

You use the wsadmin utility to configure the SPNEGO TAI for WebSphere Application Server:

1. Start WebSphere Application Server.
2. Start the command-line utility by running the **wsadmin** command from the *app_server_root/bin* directory.
3. At the **wsadmin** prompt, enter the following command:
\$AdminTask modifySpnegoTAIProperties

You can use the following parameters with this command:

Option	Description
<spnId>	The SPN identifier for the group of custom properties that are to be defined with this command. You must specify this parameter.
<host>	This parameter is optional. It specifies the host name portion in the SPN used by the SPNEGO TAI to establish a Kerberos secure context.
<filter>	This parameter is optional. It defines the filtering criteria used by the class specified with the above attribute.
<filterClass>	This parameter is optional. It specifies the name of the Java class used by the SPNEGO TAI to select which HTTP requests will be subject to SPNEGO authentication. If no class is specified, all HTTP requests will be subject to SPNEGO authentication.
<noSpnegoPage>	This parameter is optional. It specifies the URL of a resource that contains the content the SPNEGO TAI will include in the HTTP response to be displayed by the (browser) client application if it does not support SPNEGO authentication. If you do not specify the noSpnegoPage attribute then the default is used: "html"<head><title>SPNEGO authentication is not supported.</title></head>" + "body">SPNEGO authentication is not supported on this client.</body></html>";

Option	Description
<ntlmTokenPage>	<p>This parameter is optional. It specifies the URL of a resource that contains the content the SPNEGO TAI will include in the HTTP response to be displayed by the (browser) client application when the SPNEGO token received by the interceptor after the challenge-response handshake contains a NT LAN manager (NTLM) token instead of the expected SPNEGO token.</p> <p>If you do not specify the ntlmTokenPage attribute then the default is used:</p> <pre>"<html><head><title>An NTLM Token was received.</title></head>" + "<body>Your browser configuration is correct, but you have not logged into a supported Windows Domain." + "<p>Please login to the application using the normal login page.</html>";</pre>
<trimUserName>	<p>This parameter is optional. It specifies whether (true) or not (false) the SPNEGO TAI is to remove the suffix of the principal user name, starting from the "@" that precedes the Kerberos realm name. If this attribute is set to true, the suffix of the principal user name is removed. If this attribute is set to false, the suffix of the principal name is retained. The default value used is true.</p>

SPNEGO TAI properties are modified for this WebSphere Application Server.

Example 1

The following example configures the SPNEGO TAI to intercept HTTP request that contain IE 6 in the user agent request header. The SPNEGO TAI uses the SPN of HTTP/myhost.ibm.com@<default_realm> to authenticate the request originator. Then the example modifies the value of the filter custom property that was defined and changes it from user-agent%=IE 6 to host==myhost.company.com.

```
$AdminTask addSpnegoTAIProperties -host myhost.ibm.com -filter user-agent%=IE 6
$AdminTask modifySpnegoTAIProperties -spnId 1 -filter host==myhost.company.com
```

Example 2

This is an example of modifying the SPNEGO TAI for SPN1 properties to add a filter for host central01.austin.ibm.com.

```
wsadmin>$AdminTask modifySpnegoTAIProperties -interactive
Modify SPNEGO TAI properties
```

Modify SPNEGO TAI configuration properties

```
*Service Principal Name identifier (spnId): 1
Host name in Service Principal Name (host): central01.austin.ibm.com
HTTP header filter rule (filter): request-url!=noSPNEGO;request-url%=snoop
Name of class used to filter HTTP requests (filterClass):
SPNEGO not supported browser response (noSpnegoPage):
NTLM Token received browser response (ntlmTokenPage):
Trim User Name browser response (trimUserName):
```

Modify SPNEGO TAI properties

F (Finish)

C (Cancel)

Select [F, C]: [F] f

```
WASX7278I: Generated command line: $AdminTask modifySpnegoTAIProperties {-spnId
1 -host w2003secdev.austin.ibm.com -filter request-url!=noSPNEGO;request-url%=snoo
oop}
com.ibm.ws.security.spnego.SPN1.filter=request-url!=noSPNEGO;request-url%=snoop
com.ibm.ws.security.spnego.SPN1.hostName=central01.austin.ibm.com
wsadmin>
```

Deleting SPNEGO TAI properties using the wsadmin utility:

You use the wsadmin utility to delete properties in the configuration of the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) for WebSphere Application Server.

You use the wsadmin utility to configure the SPNEGO TAI for WebSphere Application Server:

1. Start WebSphere Application Server.
2. Start the command-line utility by running the **wsadmin** command from the *app_server_root/bin* directory.
3. At the **wsadmin** prompt, enter the following command:

```
$AdminTask deleteSpnegoTAIProperties
```

You can use the following parameters with this command:

Option	Description
<spnId>	This is an optional parameter. The SPN identifier for the group of custom properties that are to be deleted with this command. If you do not specify this parameter, all SPNEGO TAI custom properties are deleted.

SPNEGO TAI properties are deleted for this WebSphere Application Server.

Example 1

The following example deletes all the SPNEGO TAI properties for SPN2

```
wsadmin>$AdminTask deleteSpnegoTAIProperties {-spnId 2}
```

Example 2

The following example deletes all SPNEGO TAI properties

```
wsadmin>$AdminTask deleteSpnegoTAIProperties
com.ibm.ws.security.spnego.SPN1.filter=request-url!=noSPNEGO;request-url%=snoop
com.ibm.ws.security.spnego.SPN1.hostName=central01.austin.ibm.com
com.ibm.ws.security.spnego.SPN2.hostName=wssecpd.austin.ibm.com
wsadmin>
```

Displaying SPNEGO TAI properties using the wsadmin utility:

You use the wsadmin utility to display the properties in the configuration of the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) for WebSphere Application Server.

You use the wsadmin utility to configure the SPNEGO TAI for WebSphere Application Server:

1. Start WebSphere Application Server.
2. Start the command-line utility by running the **wsadmin** command from the *app_server_root/bin* directory.
3. At the **wsadmin** prompt, enter the following command:

```
$AdminTask showSpnegoTAIProperties
```

You can use the following parameters with this command:

Option	Description
<spnId>	This is an optional parameter. The service principal name (SPN) identifier for the group of custom properties that are to be displayed with this command. If you do not specify this parameter, all SPNEGO TAI custom properties are displayed.

SPNEGO TAI properties are displayed for this WebSphere Application Server.

Example 1

The following example displays all SPNEGO TAI properties.

```
wsadmin>$AdminTask showSpnegoTAIProperties
com.ibm.ws.security.spnego.SPN1.filter=request-url!=noSPNEGO;request-url%=snoop
com.ibm.ws.security.spnego.SPN1.hostName=central01.austin.ibm.com
com.ibm.ws.security.spnego.SPN2.hostName=wssecpd.austin.ibm.com
wsadmin>
```

Example 2

The following example displays SPNEGO TAI properties for SPN1 and host, central01.austin.ibm.com.

```
wsadmin>$AdminTask showSpnegoTAIProperties -interactive
Show SPNEGO TAI configuration properties.
```

Display SPNEGO TAI configuration properties.

Service Principal Name identifier (spnId): 1

Show SPNEGO TAI configuration properties.

F (Finish)

C (Cancel)

Select [F, C]: [F]

WASX7278I: Generated command line: \$AdminTask showSpnegoTAIProperties {-spnId 1}

```
com.ibm.ws.security.spnego.SPN1.filter=request-url!=noSPNEGO;request-url%=snoop
com.ibm.ws.security.spnego.SPN1.hostName=central01.austin.ibm.com
com.ibm.ws.security.spnego.SPN1.trimUserName=true
wsadmin>
```

Configuring JVM properties and enabling SPNEGO TAI in WebSphere Application Server:

Performing this task helps you, as Web administrator, to ensure that WebSphere Application Server is configured to enable the operation of the Simple and Protected GSS-API Negotiation mechanism (SPNEGO) trust association interceptor (TAI) with the required Java virtual machine (JVM) property.

You need to know how to use the WebSphere Application Server administrative console to manage the security configuration and have the proper authority to modify the security configuration of the application server.

Complete the following steps to enable the operation of the SPNEGO TAI by setting the JVM required property.

1. Log on to WebSphere Application Server administrative console.
2. Click **Servers > Application servers**.

3. Select the appropriate servers and click **Java and process management > Process Definition**.
4. Click **Java virtual machine**. In the **Generic JVM arguments** field, type `-Dcom.ibm.ws.security.spnego.isEnabled=true`.
5. Click **Apply > OK** to save the configuration

The application server is configured and ready to provide a single sign-on environment for end users who have successfully authenticated in a Microsoft Active Directory domain. You must restart each application server that is configured for SPNEGO Web authentication.

Enabling the SPNEGO TAI using scripting:

You use the wsadmin utility to enable the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) for WebSphere Application Server.

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 the SPNEGO TAI:

1. Identify the server and assign it to the server1 variable:

- Using Jacl:

```
set server1 [$AdminConfig getid /Cell:mycell/Node:mynode/Server:server1/]
```

- Using Jython:

```
server1 = AdminConfig.getid("Cell:mycell/Node:mynode/Server:server1/")
print server1
```

Example output:

```
server1(cells/mycell/nodes/mynode|servers/seerver1|server.xml#Server_1)
```

2. Identify the Java virtual machine (JVM) belonging to this server and assign it to the jvm variable:

- Using Jacl:

```
set jvm [$AdminConfig list JavaVirtualMachine $server1]
```

- Using Jython:

```
jvm = AdminConfig.list('JavaVirtualMachine',server1')
```

Example output:

```
(cells/mycell/nodes/mynode/servers/server1:server.xml#JavaVirtualMachine_1)
(cells/mycell/nodes/mynode/servers/server1:server.xml#JavaVirtualMachine_2)
```

3. Identify the controller JVM of the server:

- Using Jacl:

```
set cjvm [lindex $jvm 0]
```

Using Jython:

```
# get line separator
import java
lineSeparator = java.lang.System.getProperty('line.separator')
arrayJVMs = jvm.split(lineSeparator)
cjvm = arrayJVMs[0]
```

4. Modify the generic JVM arguments to enable SPNEGO TAI:

- Using Jacl:

```
$AdminConfig modify $cjvm { {genericJvmArguments "-Dcom.ibm.ws.security.spnego.isEnabled=true"} }
```

- Using Jython:

```
AdminConfig.modify(cjvm, [['genericJvmArguments', "-Dcom.ibm.ws.security.spnego.isEnabled=true"]])
```

5. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.

- In a network deployment environment only, synchronize the node. See the Synchronizing nodes with the wsadmin tool article for more information.

SPNEGO TAI JVM configuration attributes:

Java virtual machine (JVM) attributes control the operation of the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI).

The following JVM attributes control operation of the SPNEGO TAI. Different attribute values can be specified for each application server.

Table 6. JVM configuration attributes

Attribute Name	Required	Value Type	Default Value	Recommended Value
com.ibm.ws.security.spnego.isEnabled	No	Boolean	False	True
com.ibm.ws.security.spnego.propertyReloadFile	No	String	None	For Windows C:\temp\TAI.props For UNIX /tmp/TestTAI.Properties
com.ibm.ws.security.spnego.propertyReloadTimeout	No	Integer	None	120

com.ibm.ws.security.spnego.isEnabled

Use this attribute to enable or disable operation of the SPNEGO TAI in a given application server. When set to `false`, the SPNEGO TAI is disabled and not used by the Web authentication module for authenticating any Web requests. When set to `true`, the SPNEGO TAI is enabled and used by the Web authentication module for authenticating any Web requests.

com.ibm.ws.security.spnego.propertyReloadFile

Use this attribute to identify the file that contains configuration properties for the SPNEGO TAI, when it is not convenient to stop and restart the application server. The properties contained in this file can be reloaded to configure the SPNEGO TAI.

Important: the properties that are defined in the specified file override any properties defined using the administrative console.

A sample of this reload file follows:

```
#####
# Template properties files for SPNEGO TAI
#
# Where possible defaults have been provided.
#
#####

#-----
# Hostname
#-----
#com.ibm.ws.spnego.SPN1.HostName=wsecurity.austin.ibm.com

#-----
# (Optional) SpnegoNotSupportedPage
#-----
#com.ibm.ws.spnego.SPN1.SpnegoNotSupportedPage=

#-----
# (Optional) NTLMTokenReceivedPage
#-----
#com.ibm.ws.spnego.SPN1.NTLMTokenReceivedPage=
```

```
#-----
# (Optional) FilterClass
#-----
#com.ibm.ws.spnego.SPN1.FilterClass=com.ibm.ws.spnego.HTTPHeaderFilter

#-----
# (Optional) Filter
#-----
#com.ibm.ws.spnego.SPN1.Filter=
```

Important: If `com.ibm.ws.security.spnego.propertyReloadFile` attribute is set, but the `com.ibm.ws.security.spnego.propertyReloadTimeout` attribute is not, then the SPNEGO TAI is not initialized.

com.ibm.ws.security.spnego.propertyReloadTimeout

Use this attribute to specify a time interval in seconds that elapses after which the SPNEGO TAI reloads the configuration properties. Also, the SPNEGO TAI reloads the configuration properties if the file that is identified by the `com.ibm.ws.security.spnego.propertyReloadFile` attribute changed since the last time the configuration attributes were retrieved. This time interval in seconds must be specified as a positive integer.

Important:

- If the `com.ibm.ws.security.spnego.propertyReloadFile` attribute and the `com.ibm.ws.security.spnego.propertyReloadTimeout` attribute are not set, then the SPNEGO TAI properties are only loaded once from the SPNEGO TAI custom properties defined in the WebSphere Application Server configuration data. This one time loading occurs when the JVM is initialized.
- If `com.ibm.ws.security.spnego.propertyReloadTimeout` attribute is set, but the `com.ibm.ws.security.spnego.propertyReloadFile` attribute is not, then the SPNEGO TAI is not initialized.

The following examples show how to enable operation of the SPNEGO TAI by setting the `com.ibm.ws.security.spnego.isEnabled` JVM property to `true` using the scripting that is available in WebSphere Application Server for AdminConfig commands.

Using JACL:

```
set server [$AdminConfig getid /Cell:mycell/Node:mynode/Server:myserver]
set jvm [$AdminConfig list JavaVirtualMachine $server]
$AdminConfig modify $jvm {{genericJvmArguments "-Dcom.ibm.ws.security.spnego.isEnabled =true"}}
$AdminConfig save
```

Using Jython:

```
server = AdminConfig.getid('/Cell:mycell/Node:mynode/Server:myserver')
jvm = AdminConfig.list('JavaVirtualMachine', server)
AdminConfig.modify(jvm, [['genericJvmArguments', "Dcom.ibm.ws.security.spnego.isEnabled =true"]])
AdminConfig.save()
```

Remember: You can also use the `wsadmin` command for the AdminConfig scripting object to interactively set the `com.ibm.ws.security.spnego.isEnabled` attribute. See “Enabling the SPNEGO TAI using scripting” on page 246 for more information.

The following attributes are not used directly by the SPNEGO TAI; however, they affect the operation of the core security runtime and can also be used for problem determination.

Table 7. JVM configuration attributes

Attribute Name	Required	Value Type	Default Value	Recommended Value
<code>java.security.properties</code>	No	String	None	
<code>com.ibm.security.jgss.debug</code>	No	String	None	"off" or "all"

Table 7. JVM configuration attributes (continued)

Attribute Name	Required	Value Type	Default Value	Recommended Value
com.ibm.security.krb5.Krb5Debug	No	String	None	"off" or "all"
javax.security.auth.useSubjectCredsOnly	Yes	Boolean	True	False

java.security.properties

This property is optional. It can be used when different application servers in a cell have different security requirements and it is not convenient to modify the global `java.security` file for the entire cell. In such situations, the `java.security.properties` attribute is used to specify the location of the `java.security` file used by the JVM for each application server.

com.ibm.security.jgss.debug

This attribute is optional. It can be used to collect diagnostic trace information for problem determination in the Java Generic Security Service (JGSS) application programmer interface (API) implementation. The value can be set to `all` or `off` to enable or disable tracing, respectively. See Java Generic Security Service User's Guide for specific JGSS API information.

com.ibm.security.krb5.Krb5Debug

This attribute is optional. It can be used to collect additional diagnostic trace information for problem determination in the JGSS implementation. The value can be set to `all` or `off` to enable or disable tracing, respectively.

javax.security.auth.useSubjectCredsOnly

JGSS includes an optional Java Authentication and Authorization Service (JAAS) login facility that saves `Principal` credentials and secret keys in the `Subject` of the application's JAAS login context. JGSS retrieves credentials and secret keys from the `Subject` by default. This feature can be disabled by setting the Java property `javax.security.auth.useSubjectCredsOnly` to `false`.

Attention: The SPNEGO TAI does not use the optional JAAS login module. The `javax.security.auth.useSubjectCredsOnly` property must be set to `false`.

Mapping user Ids from client to server for SPNEGO:

You can use a system programming interface to customize the behavior of the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) by implementing arbitrary mappings of the end-user's identity, which is retrieved from Microsoft Active Directory to the identity that is used in the WebSphere Application Server security registry.

You need to perform some administrative tasks in the WebSphere Application Server environment to use SPNEGO TAI and to ensure that the requester's identity matches the identity in the WebSphere Application Server user registry.

In the simplest deployment of the SPNEGO TAI, it is assumed that the requester's identity in the WebSphere Application Server user registry is identical to the identity retrieved. This is the case when Microsoft Windows Active Directory server is the lightweight directory access protocol (LDAP) server used in WebSphere Application Server. This is default behavior of the SPNEGO TAI.

You do not need to use this simple deployment of the SPNEGO TAI. WebSphere Application Server can use a different registry, such as a local OS, LDAP, or custom registry instead of the Microsoft Active Directory. If WebSphere Application Server uses a different registry than the Microsoft Active Directory, then a mapping from the Microsoft Windows user Id to a WebSphere Application Server user Id is necessary.

1. Configure the Web browser to use SPNEGO.
2. Configure Java virtual machine (JVM) properties and custom SPNEGO TAI properties.
3. Enable the SPNEGO TAI.

4. Use the custom login module to perform any custom mapping of user IDs from the user registry to the user registry of WebSphere Application Server. The custom login module is a plug-in mechanism that is defined for authenticating incoming and outgoing requests in WebSphere Application Server. The custom login module is inserted before the `ItpaLoginModule` and maps the name in the `com.ibm.wsspi.security.tai.TAIResult` (which was returned to the Web authenticator) to the corresponding name in the user registry. The `ItpaLoginModule` then uses the mapped identity to create a `WSCredential`.

The custom login module can also supply the full set of security attributes in the `javax.security.auth.Subject` in the `com.ibm.wsspi.security.tai.TAIResult` to fully assert the mapped identity. When the identity is fully asserted, the `wsMapDefaultInboundLoginModule` maps those security attributes to a `WSCredential`.

Using the custom login module, Microsoft Active Directory identities are mapped to the WebSphere Application Server's security registry.

Filtering HTTP requests for SPNEGO TAI:

You can use a system programming interface to customize the behavior of the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) by specifying whether or not a particular HTTP request should be intercepted.

Before you begin, you need to fully understand the deployment of the SPNEGO TAI in your installation.

Verify the configuration of your SPNEGO TAI. The deployment of the SPNEGO TAI can vary from a single WebSphere Application Server system on which a single application is running to a large multinode WebSphere Application Server Network Deployment (ND) cell, with dozens of application servers, hosting many applications. Every SPNEGO TAI is installed at the cell level. You must be aware of your particular SPNEGO TAI configuration.

The default behavior of the SPNEGO TAI is to not intercept HTTP requests. This default behavior ensures that the SPNEGO TAI can be installed into an existing cell, configured for a single application server and not change any other application servers in the cell. Other WebSphere Application Servers can run exactly as before within a given configuration.

Then decide whether or not to use the sample `SPN<id>.filter` class and determine the exact filter properties to use.

Note: The default behavior of the SPNEGO TAI is to use the `com.ibm.ws.security.spnego.SPN<id>.filter` class and intercept all requests.

If the default behavior is not appropriate, you can use a customer provided class, or extend or modify the sample class as required. The system programmer interface, `com.ibm.ws.security.spnego.SpnegoFilter` allows you to implement a custom filter to determine whether or not to intercept a particular HTTP request. With the default implementation, you can set filter rules for coarse as well as fine-grained criteria in selecting which HTTP requests to intercept.

1. Set the `com.ibm.ws.security.spnego.isEnabled` Java virtual machine (JVM) custom property to `true` to enable the SPNEGO TAI on any JVM.
2. Identify when the SPNEGO TAI intercepts a given request. A set of filter properties is provided, but you must determine what is appropriate and modify the `com.ibm.ws.security.spnego.SPN<id>.filter` class accordingly.

Your SPNEGO TAI is set to filter HTTP requests when it is operating.

Kerberos configuration requirements for SPNEGO TAI:

Kerberos configuration settings, the Kerberos key distribution center (KDC) name, and realm settings for the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) are provided in the Kerberos configuration file or through `java.security.krb5.kdc` and `java.security.krb5.realm` system property files.

The Web administrator creates the Kerberos configuration file with the appropriate settings that allow HTTP requests to be processed by the SPNEGO TAI. See “Creating the Kerberos configuration file for use with the SPNEGO TAI” on page 234 for more information.

The Web administrator can provide the same Kerberos configuration system properties in separate files: `java.security.krb5.kdc` and `java.security.krb5.realm`. See Kerberos Requirements for information on how this is accomplished.

The Kerberos key table manager command (Ktab) allows the Web administrator to manage the principal names and service keys stored in a local Kerberos keytab file. Kerberos service principal (SPN) name and keys listed in the Kerberos keytab file allow services running on the host to authenticate themselves to the KDC. Before SPNEGO TAI can use Kerberos, the WebSphere Application Server administrator must setup a Kerberos keytab file on the host running WebSphere Application Server.

Important: It is very important to protect the keytab files, making them readable only by the authorized WebSphere users.

Important: Any updates to the Kerberos keytab file using Ktab do not affect the Kerberos database. If you change the keys in the Kerberos keytab file, you must also make the corresponding changes to the Kerberos database.

Below is an example of how Ktab is used on a LINUX platform to add new principal names to the Kerberos keytab file.

```
[root@wssecjibe bin]# ./java com.ibm.security.krb5.internal.tools.Ktab -a
HTTP/wssecjibe.austin.ibm.com@WSSEC.AUSTIN.IBM.COM ot56prod -k /etc/krb5.keytab
Done!
Service key for principal HTTP/wssecjibe.austin.ibm.com@WSSEC.AUSTIN.IBM.COM saved
[root@wssecjibe bin]# ./java com.ibm.security.krb5.internal.tools.Ktab
1 entries in keytab, name: /etc/krb5.keytab
    KVNO    Principal
    ----    -
1          HTTP/wssecjibe.austin.ibm.com@WSSEC.AUSTIN.IBM.COM
[root@wssecjibe bin]# ls /etc/krb5.*
/etc/krb5.conf      /etc/krb5.ini.orig  /etc/krb5.keytab.good
/etc/krb5.conf.orig /etc/krb5.keytab
[root@wssecjibe bin]# ./java com.ibm.security.krb5.internal.tools.Ktab -a
HTTP/wssecredhat.austin.ibm.com@WSSEC.AUSTIN.IBM.COM ot56prod -k /etc/krb5.keytab
Done!
Service key for principal HTTP/wssecredhat.austin.ibm.com@WSSEC.AUSTIN.IBM.COM
saved
[root@wssecjibe bin]# ./java com.ibm.security.krb5.internal.tools.Ktab
2 entries in keytab, name: /etc/krb5.keytab
    KVNO    Principal
    ----    -
1          HTTP/wssecjibe.austin.ibm.com@WSSEC.AUSTIN.IBM.COM
1          HTTP/wssecredhat.austin.ibm.com@WSSEC.AUSTIN.IBM.COM
```

Tip: On WebSphere Application Server, Ktab is located at:

```
<install root>/java/jre/bin
```

Configuring the Web browser to use SPNEGO

You can configure your browser to utilize the Simple and Protected GSS-API Negotiation (SPNEGO) mechanism. Authentication of your browser requests are processed by the SPNEGO trust association interceptor (TAI) in the WebSphere Application Server.

You need to know how to display and set options in the Microsoft Internet Explorer browser or any other browser (such as Firefox). You must have a browser installed that supports SPNEGO authentication.

Complete the following steps to ensure that your Microsoft Internet Explorer browser is enabled to perform SPNEGO authentication.

1. At the desktop, log in to the windows active directory domain.
2. Activate Internet Explorer.
3. In the Internet Explorer window, click **Tools > Internet Options > Security** tab.
4. Select the **Local intranet** icon and click **Sites**.
5. In the Local intranet window, ensure that the "check box" to include all local (intranet) not listed in other zones is selected, then click **Advanced** .
6. In the **Local intranet** window, fill in the Add this Web site to the zone field with the Web address of the host name so that the single sign-on (SSO) can be enabled to the list Web sites shown in the Web sites field. Your site information technology staff provides this information. Click **OK** to complete this step and close the Local intranet window.
7. On the **Internet Options** window, click the **Advanced** tab and scroll to **Security settings**. Ensure that the **Enable Integrated Windows Authentication (requires restart)** box is selected.
8. Click **OK**. Restart your Microsoft Internet Explorer to activate this configuration.

Complete the following steps to ensure that your Firefox browser is enabled to perform SPNEGO authentication.

1. At the desktop, log in to the windows active directory domain.
2. Activate Firefox.
3. At the address field, type **about:config**.
4. In the Filter, type **network.n**
5. If the deployed SPNEGO solution is using the advanced Kerberos feature of Credential Delegation double click on **network.negotiate-auth.delegation-uris**. This preference lists the sites for which the browser may delegate user authorization to the server.
6. Enter a comma delimited list of trusted domains or URLs.
7. Click **OK**. The configuration appears as updated.
8. Restart your Firefox browser to activate this configuration.

Your Internet browser is properly configured for SPNEGO authentication. You can use applications that are deployed in WebSphere Application Server that use secured resources without being repeatedly requested for an ID and password.

Configuring single sign-on capability with Tivoli Access Manager or WebSEAL

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 sign-on (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 sign-on are available, the TAI interface and the TAI++ interface, so named as both use WebSphere Application Server trust association interceptors (TAI). With the TAI, the end-user name is extracted from the HTTP header and forwarded to embedded Tivoli Access Manager where the end-user

name is used to construct the client credential information and authorize the user. With the TAI++, all of the user credential information is available in the HTTP header and not just the user name. The TAI++ is the more efficient of the two solutions because a Lightweight Directory Access Protocol (LDAP) call is not required. TAI functionality is retained for backwards compatibility.

Complete the following tasks to enable single sign-on 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. Create a trusted user account for Tivoli Access Manager in the shared Lightweight Directory Access Protocol (LDAP) user registry. For more information, see “Creating a trusted user account in Tivoli Access Manager” on page 258.
2. Configure either WebSEAL or the Tivoli Access Manager plug-in for Web servers to work with WebSphere Application Server. For more information, see either of the following articles:
 - “Configuring WebSEAL for use with WebSphere Application Server” on page 259
 - “Configuring Tivoli Access Manager plug-in for Web servers for use with WebSphere Application Server” on page 260
3. Configure single sign-on using either the TAI or TAI++ interface. For more information, see either of the following articles:
 - “Configuring single sign-on using the trust association interceptor” on page 260
 - “Configuring single sign-on using trust association interceptor ++” on page 261

Single sign-on settings

Use this page to set the configuration values for single sign-on (SSO).

To view this administrative console page, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Authentication, click **Web security > Single sign-on (SSO)**.

Enabled:

Specifies that the single sign-on function is enabled.

Web applications that use J2EE FormLogin style login pages, such as the administrative console, require single sign-on (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 sign-on 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 sign-on hosts.

The 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 sign-on 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, the 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 the application server does not set a domain for the `LtpaToken` cookie.

If you specify the `UseDomainFromURL` value, the application server sets the SSO domain name value to the domain of the host that is used in the Web address. For example, if an HTTP request comes from `server1.raleigh.ibm.com`, the 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 sign-on (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 sign-on (SSO) token is used to log in and recreate the Subject from the user registry.

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.

com.tivoli.pd.jcfg.PDJrteCfg utility for Tivoli Access Manager single sign-on

The `com.tivoli.pd.jcfg.PDJrteCfg` utility configures the Java Runtime Environment component for Tivoli Access Manager. This component enables WebSphere Application Server to use Tivoli Access Manager security.

Purpose

Syntax

```
java com.tivoli.pd.jcfg.PDJrteCfg -action {config | unconfig} -cfgfiles_path  
configuration_file_path -host policy_server_host jre_path
```

Parameters

-action {config|unconfig}

Specifies the action to be performed. Actions include:

config Use to configure the Access Manager Java Runtime Environment component.

unconfig

Use to reconfigure the Access Manager Java Runtime Environment component.

-host *policy_server_host*

Specifies the policy server host name.

Valid values for *policy_server_host* include any valid IP host name.

Examples include:

```
host = libra
host = libra.dallas.ibm.com
```

Comments

This command copies Tivoli Access Manager Java libraries to a library extensions directory that exists for a Java runtime that has already been installed on the system.

You can install more than one Java Runtime Environment (JRE) on a given machine. The `pdjrtecfg` command can be used to configure the Tivoli Access Manager Java Runtime Environment component independently for each of the JRE configurations.

```
${JAVA_HOME}/bin/java
-Dfile.encoding=ISO8859-1 \
-Dws.output.encoding=CP1047 \
-Xnoargsconversion \
-Dpd.home=${WAS_HOME}/java/jre/PolicyDirector \
-cp ${WAS_HOME}/java/jre/lib/ext/PD.jar \
  com.tivoli.pd.jcfg.PDJrteCfg \
-action config \
  -cfgfiles_path ${WAS_HOME}/java/jre \
  -host gary.us.ibm.com \
```

com.tivoli.pd.jcfg.SvrSslCfg utility for Tivoli Access Manager single sign-on

The utility is used to configure and remove the configuration information associated with WebSphere Application Server and the Tivoli Access Manager server.

Purpose

Syntax

```
java com.tivoli.pd.jcfg.SvrSslCfg
-action {config | unconfig} -admin_id admin_user_ID
-admin_pwd admin_password -appsvr_id application_server_name
-appsvr_pwd application_server_password -mode{local|remote}
-host host_name_of_application_server
-policysvr policy_server_name:port:rank [...]
-authzsvr authorization_server_name:port:rank [...]
-cfg_file fully_qualified_name_of_configuration_file
-domain Tivoli_Access_Manager_domain
-key_file fully_qualified_name_of_keystore_file
-cfg_action {create|replace}
```

Parameters

-action {config | unconfig}

Specifies the configuration action that is performed by the script. The following options apply:

-action config

Configuring a server creates user and server information in the user registry and creates local configuration and key store files on the application server. Use the `-action unconfig` option to reverse this operation.

If this action is specified, the following options are required: `-admin_id`, `-admin_pwd`, `-appsvr_id`, `-port`, `-mode`, `-policysvr`, `-authzsvr`, and `-key_file`.

-action unconfig

Reconfigures an application server to complete the following actions:

- Remove the user and server information from the user registry
- Delete the local key store file
- Remove information for this application from the configuration file without deleting the file

The reconfiguration operation fails only if the caller is unauthorized or the policy server cannot be contacted.

This action can succeed when a configuration file does not exist. When the configuration file does not exist, it is created and used as a temporary file to hold configuration information during the operation, and then the file is deleted completely.

If this action is specified, the following options are required: `-admin_id`, `-admin_pwd`, `-appsvr_id`, and `-policysvr`.

-admin_id *admin_user_ID*

Specifies the Tivoli Access Manager administrator name. If this option is not specified, `sec_master` is the default.

A valid administrative ID is an alphanumeric, case-sensitive string. String values are expected to be characters that are part of the local code set. You cannot use a space in the administrative ID.

For example, for U.S. English the valid characters are the letters a-Z, the numbers 0-9, a period (`.`), an underscore (`_`), a plus sign (`+`), a hyphen (`-`), an at sign (`@`), an ampersand (`&`), and an asterisk (`*`). The minimum and maximum lengths of the administrative ID, if there are limits, are imposed by the underlying registry.

-admin_password *admin_password*

Specifies the password of the Tivoli Access Manager administrator user that is associated with the `-admin_id` parameter. The password restrictions depend upon the password policy for your Tivoli Access Manager configuration.

-appsvr_id *application_server_name*

Specifies the name of the application server. The name is combined with the host name to create unique names for Tivoli Access Manager objects created for your application. The following names are reserved for Tivoli Access Manager applications: `ivacl`, `secmgr`, `ivnet`, and `ivweb`.

-appsvr_pwd *application_server_password*

Specifies the password of the application server. This option is required. A password is created by the system and the configuration file is updated with the password created by the system.

If this option is not specified, the server password will be read from standard input.

-authzsvr *authorization_server_name*

Specifies the name of the Tivoli Access Manager authorization server with which the application server communicates. The server is specified by fully qualified host name, the SSL port number, and the rank. The default SSL port number is 7136. For example: `myauth.mycompany.com:7136:1`. You can specify multiple servers if the entries are separated by a comma (`,`).

-cfg_action {**create** | **replace**}

Specifies the action to take when creating the configuration and key files. Valid values are **create** or

replace. Use the **create** option to initially create the configuration and keystore files. Use the **replace** option if these files already exist. If you use the **create** option and the configuration or keystore files already exist, an exception is created.

Options are as follows:

create Specifies to create the configuration and key store files during server configuration. Configuration fails if either of these files already exists.

replace

Specifies to replace the configuration and key store files during server configuration. Configuration deletes any existing files and replaces them with new ones.

-cfg_file *fully_qualified_name_of_configuration_file*
Specifies the configuration file path and name.

A file name should be an absolute file name (fully qualified file name) to be valid.

-domain *Tivoli_Access_Manager_domain*

Specifies the Tivoli Access Manager domain name to which the administrator is authenticated. This domain must exist and an the administrator ID and password must be valid for this domain. The application server is specified in this domain.

If not specified, the local domain that was specified during Tivoli Access Manager runtime configuration will be used. The local domain value will be retrieved from the configuration file.

A valid domain name is an alphanumeric, case-sensitive string. String values are expected to be characters that are part of the local code set. You cannot use a space in the domain name.

For example, for U.S. English the valid characters for domain names are the letters a-Z, the numbers 0-9, a period (.), an underscore (_), a plus sign (+), a hyphen (-), an at sign (@), an ampersand (&), and an asterisk (*). The minimum and maximum lengths of the domain name, if there are limits, are imposed by the underlying registry.

-host *host_name_of_application_server*

Specifies the TCP host name used by the Tivoli Access Manager policy server to contact this server. This name is saved in the configuration file using the `azn-app-host` key.

The default is the local host name returned by the operating system. Valid values for `host_name` include any valid IP host name.

Examples:

```
host = libra  
host = libra.dallas.ibm.com
```

-key_file *fully_qualified_name_of_keystore_file*

Specifies the directory that is to contain the key files for the server. A valid directory name is determined by the operating system. Use a fully qualified file name that contains the application server certificate and key file.

Make sure that server user (for example, `ivmgr`) or all users have permission to access the `.kdb` file and the folder that contains the `.kdb` file.

This option is required.

-mode *server_mode*

Specifies the mode in which the application operates. This value must be either `local` or `remote`.

-policysvr *policy_server_name*

Specifies the name of the policy server.

Comments

After the successful configuration of a Tivoli Access Manager Java application server, SvrSslCfg creates a user account and server entries representing the Java application server in the Tivoli Access Manager user registry. In addition, SvrSslCfg creates a configuration file and a Java key store file, which securely stores a client certificate, locally on the application server. This client certificate permits callers to make authenticated use of Tivoli Access Manager services. Conversely, reconfiguration removes the user and server entries from the user registry and cleans up the local configuration and keystore files.

The contents of an existing configuration file can be modified by using the SvrSslCfg utility. The configuration file and the key store file must already exist when calling SvrSslCfg with all options other than `-action config` or `-action unconfig`.

The following options are parsed and processed into the configuration file, but are otherwise ignored in this version of Tivoli Access Manager:

The host name is used to build a unique name (identity) for the application. The `pdadmin` user list command displays the application identity name in the following format:

```
server_name/host_name
```

Note that the `pdadmin` server list command displays the server name in a slightly different format:

```
server_name-host_name
```

```
CLASSPATH=${WAS_HOME}/java/jre/lib/ext/PD.jar:${WAS_CLASSPATH}
java \
-cp ${CLASSPATH} \
-Dpd.cfg.home= ${WAS_HOME}/java/jre \
-Dfile.encoding=ISO8859-1 \
-Dws.output.encoding=CP1047 \
-Xnoargsconversion \
  com.tivoli.pd.jcfg.SvrSslCfg \
-action config \
-admin_id sec_master \
-admin_pwd $TAM_PASSWORD \
-appsvr_id $APPSVR_ID \
-policysvr ${TAM_HOST}:7135:1 \
-port 7135 \
-authzsvr ${TAM_HOST}:7136:1 \
-mode remote \
-cfg_file ${CFG_FILE} \
-key_file ${KEY_FILE} \
-cfg_action create
```

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 account includes the ID and password that WebSEAL uses to identify itself to WebSphere Application Server. To prevent potential vulnerabilities, do not use the `sec_master` ID as the trusted user account and ensure that the password you use is unique and generated randomly. Use the trusted user account should for the TAI or TAI++ only.

1. 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.
2. Reference the code listed below as an example for creating a trusted user account.

3. Reference the following additional resources for more information:
 - a. “Configuring WebSEAL for use with WebSphere Application Server”
 - b. “Configuring Tivoli Access Manager plug-in for Web servers for use with WebSphere Application Server” on page 260

```
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

Use this topic to set the SSO password in WebSEAL for single sign-on to WebSphere Application Server.

A junction must be created between WebSEAL and WebSphere Application Server. This junction carries the iv-credentials (for TAI++) or iv-user (for TAI) and the HTTP basic authentication headers with the request. You can configure WebSEAL to pass the end user identity in other ways, the iv-credentials header is the only one supported by the TAI++ and the iv-user is the only one supported by TAI.

We recommend that communications over the junction use Secure Sockets Layer (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 activity 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 WebSphere Application Server using the **-c iv_creds** option for TAI++ and **-c iv_user** for TAI. Enter either of the following commands as one line using the variables that are appropriate for your environment:

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 recreate the junction.
2. The junction can be created as `-t tcp` or `-t ssl`, depending on your requirements.

For single sign-on (SSO) to WebSphere Application Server the SS) 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`. 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 258.
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/tprodlist.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 Server resources.

With such an arrangement the plug-in authorizes all user requests before passing the credentials of the authorized user to 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 sign-on (SSO) user password.

1. The Tivoli Access Manager plug-in for Web servers configuration shows IV headers configured for post-authorization processing, and basic authentication that is configured as the authentication mechanism and for post-authorization processing, as shown in the example below.
2. After a request is authorized, the basic authentication header is removed from the request (`strip-hdr=always`) and a new one is added (`add-hdr=supply`).
3. Included in this new header is the password that is set when the SSO user is created in “Creating a trusted user account in Tivoli Access Manager” on page 258.
4. Specify this password 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.
5. An iv-creds header is also added (`generate=iv-creds`), which contains the credential information of the user passed onto WebSphere Application Server. 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 sign-on using the trust association interceptor” or “Configuring single sign-on using trust association interceptor ++” on page 261

Configuring single sign-on using the trust association interceptor

This task is performed to enable single sign-on using the trust association interceptor. These steps involve setting up trust association and creating the interceptor properties.

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.

Lightweight Third Party Authentication (LTPA) is the default authentication mechanism for WebSphere Application Server. You can configure LTPA prior to configuring single sign-on (SSO) by clicking **Security > Secure administration, applications, and infrastructure > Authentication mechanisms and expiration**. Although you can use Simple WebSphere Authentication Mechanism (SWAM) by selecting the **Use SWAM-no authenticated communication between servers** option on the Authentication mechanisms and expiration panel, single sign-on (SSO) requires LTPA as the configured authentication mechanism.

1. From the administrative console for WebSphere Application Server, click **Security > Secure administration, applications, and infrastructure**.
2. Under Web security, click **Trust association**.
3. Select the **Enable trust association** option.
4. Under Additional properties, click the **Interceptors** link.
5. Click **com.ibm.ws.security.web.WebSealTrustAssociationInterceptor** to use the WebSEAL interceptor. This interceptor is the default.
6. Under Additional properties, click **Custom Properties**.
7. Click **New** to enter the property name and value pairs. Ensure the following parameters are set:

Table 8. Trust association interceptor properties

Option	Description
com.ibm.websphere.security.trustassociation.types	Ensure that <i>webseal</i> 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 258 The format of the username is the short name representation. This property is mandatory. If the property is not set in the WebSphere Application Server, TAI initialization fails.
com.ibm.websphere.security.webseal.id	The <i>iv-user</i> header, which is com.ibm.websphere.security.webseal.id=iv-user
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) are trusted and expected in the request header. For example: com.ibm.websphere.security.webseal.hostnames=host1 This includes the proxy host names unless the com.ibm.websphere.security.webseal.ignoreProxy is set to <i>true</i> . Obtain a list of servers 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 are in the request header. This includes the proxy ports unless the com.ibm.websphere.security.webseal.ignoreProxy is set to <i>true</i> . For example: com.ibm.websphere.security.webseal.ports=80,443
com.ibm.websphere.security.webseal.ignoreProxy	An optional property that if set to <i>true</i> or <i>yes</i> ignores the proxy host names and ports in the IV header. By default this property is set to <i>false</i> .

8. Click **OK**.
9. Save the configuration and log out.
10. Restart WebSphere Application Server.

Configuring single sign-on using trust association interceptor ++

Perform this task to enable single sign-on using trust association interceptor ++. The steps involve setting up trust association and creating the interceptor properties.

Lightweight Third Party Authentication (LTPA) is the default authentication mechanism for WebSphere Application Server. However, you may need to configure LTPA prior to configuring the TAMTrustAssociationInterceptorPlus. LTPA is the required authentication mechanism for all trust association interceptors. You can configure LTPA by clicking **Security > Secure administration, applications, and infrastructure > Authentication mechanisms and expiration**.

Note: Enabling Web security single sign-on (SSO) is optional when you configure the TAMTrustAssociationInterceptorPlus. For more information, see “Implementing single sign-on to minimize Web user authentications” on page 228.

Although you can use Simple WebSphere Authentication Mechanism (SWAM) by selecting the **Use SWAM-no authenticated communication between servers** option on the Authentication mechanisms and expiration panel, single sign-on (SSO) requires LTPA as the configured authentication mechanism.

1. From the administrative console for WebSphere Application Server, click **Security > Secure administration, applications, and infrastructure**.
2. Under Web security, click **Trust association**.
3. Click **Enable Trust Association**.
4. Click **Interceptors**.
5. Click **com.ibm.ws.security.web.TAMTrustAssociationInterceptorPlus** to use the WebSEAL interceptor. This interceptor is the default.
6. Click **Custom Properties**.
7. Click **New** to enter the property name and value pairs. Verify that the following parameters are set:

Table 9. Custom properties

Option	Description
com.ibm.websphere.security.webseal.checkViaHeader	<p>You can configure TAI so that the via header can be ignored when validating trust for a request. Set this property to <i>false</i> if none of the hosts in the via header need to be trusted. When set to <i>false</i> you do not need to set the trusted host names and host ports properties. The only mandatory property to check when via header is <i>false</i> is com.ibm.websphere.security.webseal.loginId.</p> <p>The default value of the check via header property is <i>false</i>. When using Tivoli Access Manager plug-in for Web servers, set this property to <i>false</i>.</p> <p>Note: The via header is part of the standard HTTP header that records the server names the request that passed through.</p>
com.ibm.websphere.security.webseal.loginId	<p>The WebSEAL trusted user as created in “Creating a trusted user account in Tivoli Access Manager” on page 258 The format of the username is the short name representation. This property is mandatory. If it is not set in WebSphere Application Server, the TAI initialization fails.</p>
com.ibm.websphere.security.webseal.id	<p>A comma-separated list of headers that exists in the request. If all of the configured headers do not exist in the request, trust cannot be established. The default value for the ID property is <i>iv-creds</i>. Any other values set in WebSphere Application Server are added to the list along with <i>iv-creds</i>, separated by commas.</p>
com.ibm.websphere.security.webseal.hostnames	<p>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 <i>false</i> then the trusted host names property has no influence. If the checkViaHeader property is set to <i>true</i>, and the trusted host names property is not set, TAI initialization fails.</p>
com.ibm.websphere.security.webseal.ports	<p>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 that arrive from unlisted ports might not be trusted. If the checkViaHeader property is not set, or is set to <i>false</i> this property has no influence. If the checkViaHeader property is set to <i>true</i>, and the trusted host ports property is not set in WebSphere Application Server, the TAI initialization fails.</p>

Table 9. Custom properties (continued)

Option	Description
<code>com.ibm.websphere.security.webseal.viaDepth</code>	<p>A positive integer that specifies 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 host is not trusted, trust cannot be established. The via depth property is used when only some of the hosts in the via header have to be trusted. The setting indicates the number of hosts that are required to be trusted.</p> <p>As an example, consider the following header: Via: HTTP/1.1 webseal1:7002, 1.1 webseal2:7001</p> <p>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: <code>com.ibm.websphere.security.webseal.hostnames = webseal1,webseal2</code> <code>com.ibm.websphere.security.webseal.ports = 7002,7001</code></p> <p>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: <code>com.ibm.websphere.security.webseal.hostnames = webseal2</code> <code>com.ibm.websphere.security.webseal.ports = 7001</code></p> <p>The viaDepth property is set to 0 by default, which means all of the hosts in the via header are checked for trust.</p>
<code>com.ibm.websphere.security.webseal.ssoPwdExpiry</code>	<p>After trust is established for a request, the single sign-on user password is cached, eliminating the need to have the TAI re-authenticate the single sign-on user with Tivoli Access Manager for every request. You can modify the cache timeout period by setting the single sign-on password expiry property to the required time in seconds. If the password expiry property is set to 0, the cached password never expires. The default value for the password expiry property is 600.</p>
<code>com.ibm.websphere.security.webseal.ignoreProxy</code>	<p>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. Remember 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.</p>
<code>com.ibm.websphere.security.webseal.configURL</code>	<p>For the TAI to establish trust for a request, it requires that the SvrSslCfg run for the Java Virtual Machine on the Application Server and result in the creation of a properties file. If this properties file is not at the default URL, which is <code>file://java.home/PdPerm.properties</code>, the correct URL of the properties file must be set in the configuration 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 <code>file://\${WAS_INSTALL_ROOT}/java/jre/PdPerm.properties</code>.</p>

8. Click **OK**.
9. Save the configuration and log out.
10. Restart WebSphere Application Server.

Configuring global sign-on principal mapping

You can create a new application login that uses the Tivoli Access Manager GSO database to store the login credentials.

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Authentication, click **Java Authentication and Authorization Service > Application logins**.

3. Click **New** to create a new Java Authentication and Authorization Service (JAAS) login configuration.
4. Enter the alias name of the new application login. Click **Apply**.
5. Under Additional properties, click **JAAS login modules** to define the JAAS Login Modules.
6. Click **New** and enter the following information:

Module class name: com.tivoli.pdwas.gso.AMPrincipalMapper
Use Login Module Proxy: enable
Authentication strategy: REQUIRED

7. Click **Apply**
8. Under Additional Properties section, click **Custom Properties** to define login module-specific values that are passed directly to the underlying login modules.
9. Click **New**.

The Tivoli Access Manager principal mapping module uses the authDataAlias configuration string to retrieve the correct user name and password from the security configuration.

The authDataAlias attribute that is passed to the module is configured for the J2C connection factory. Because the authDataAlias attribute is an arbitrary string that is entered at configuration time, the following scenarios are possible:

- The authDataAlias attribute contains both the global sign-on (GSO) resource name and the user name. The format of this string is "Resource/User".
- The authDataAlias attribute contains the GSO Resource name only. The user name is determined by using the Subject of the current session.

The scenario to use is determined by a JAAS configuration option, as shown here:

Name: com.tivoli.pd.as.gso.AliasContainsUserName

Value: True, if the alias contains the user name; false, if the user name must be retrieved from the security context

When entering authDataAlias attributes through the WebSphere Application Server administrative console, the node name is automatically pre-pended to the alias. The JAAS configuration entry determines whether this node name is removed or included as part of the resource name, as shown here:

Name: com.tivoli.pd.as.gso.AliasContainsNodeName

Value: True, if the alias contains the node name

Note: If the PdPerm.properties configuration file is not located in the JAVA_HOME/PdPerm.properties default location, then you also need to add the following property:

Name: com.tivoli.pd.as.gso.AMCfgURL

Value: file:///path to PdPerm.properties

Enter each new parameter using the following scenario information as a guide, then click **Apply**.

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 WebSphere Application Server 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 is 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 WebSphere Application Server 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

10. Create the Java 2 Connector (J2C) authentication aliases. The user name and password that are assigned to these alias entries are irrelevant because Tivoli Access Manager is responsible for providing user names and passwords. However, the user name and password that are assigned to the J2C authentication aliases need to exist so that they can be selected for the J2C connection factory in the administrative console.

To create the J2C authentication aliases, from the WebSphere Application Server administrative console, click **Security > Secure administration, applications, and infrastructure**. Under Authentication, click **Java Authentication and Authorization Service > J2C authentication data**, and then click **New** for each new entry. Refer to the previous table for scenario inputs.

The connection factories for each resource adapter that need to use the GSO database must be configured to use the Tivoli Access Manager Principal mapping module:

- a. From the WebSphere Application Server administrative console, click **Applications > Enterprise Applications > application_name > Resource references**. Note that J2C connection factories must be already configured for the selected application. To configure a new J2C connection factory, see tdat_confconfac.dita.
- b. Under Additional properties, click **Resource Adapter**.
The resource adapter can be standalone and does not need to be packaged with the application. The resource adapter is configured from **Resources > Resource Adapters** for standalone scenarios.
- c. Under Additional properties, click **J2C Connection Factories**.
- d. Click **New** and enter the connection factory properties.
- e. When finished, click **Apply > Save**.

Attention:

Custom mapping configuration for the connection factory is deprecated in WebSphere Application Server Version 6. To configure the GSO credential mapping, use the Map Resource References to Resources panel on the administrative console. For more information, see J2EE connector security.

Propagating security attributes among application servers

Use the security attribute propagation feature of WebSphere Application Server to send security attribute information regarding the original login to other servers using a token. This topic will help to configure WebSphere Application Server to propagate security attributes to other servers.

To fully enable security attribute propagation, you must configure the single sign-on (SSO), Common Secure Interoperability Version 2 (CSlv2) inbound, and CSlv2 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**.

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:port_number/ibm/console`. The administrative console address might differ if you have previously changed the port number.
2. Click **Security > Secure administration, applications, and infrastructure**.
3. Under Web security, click **Single sign-on (SSO)**.
4. **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 Propagation token, but ignore the security attribute information that they do not understand.
5. 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 CSlv2 Inbound authentication and CSlv2 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. These attributes include any custom attributes that might be set in the PropagationToken token 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 measure the performance of your environment both 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.

6. Click **Security > Secure administration, applications, and infrastructure**. Under RMI/IIOP security, click **CSlv2 inbound authentication**. The Login configuration field specifies `RMI_INBOUND` as the system login configuration that is used for inbound requests. To add custom Java Authentication and Authorization Service (JAAS) login modules, complete the following steps:
 - a. Click **Security > Secure administration, applications, and infrastructure**. Under Java Authentication and Authorization Service, click **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.

Note: SWAM is deprecated in WebSphere Application Server Version 6.1 and will be removed in a future release.

- 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 that are 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.

You can specify the order in which the login modules are processed by clicking **Set Order**.

7. 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.
8. Click **Security > Secure administration, applications, and infrastructure**. Under RMI/IOP security, click **CSiv2 Outbound authentication**. The CSiv2 outbound authentication panel is displayed. The **Login configuration** field specifies RMI_OUTBOUND as the 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 that are listed previously for CSiv2 Inbound authentication.
9. **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 CSiv2 protocol to send the Subject and PropagationToken token 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.

10. **Optional:** Select the **Custom Outbound Mapping** option if you clear the **Security Attribute Propagation** option and you want to use the RMI_OUTBOUND login configuration. If neither 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.
11. **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 support inbound mapping that is 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 having the request rejected because of 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.

12. **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
```

Note: The `sas.client.props` file is located at `<WAS-HOME>/profiles/<ProfileName>/properties`.

After completing these steps, you have configured WebSphere Application Server to propagate security attributes to other servers.

If you need to disable security attribute propagation, determine whether you need to disable it for either the server level or the cell level.

Attention: Changes to the server-level settings override the cell settings.

To disable security attribute propagation on the server level, complete the following steps:

1. Click **Server > Application Servers > *server_name***.
2. Under Security, click **Server security**.
3. Select the **RMI/IIOP security for this server overrides cell settings** option.
4. Disable security attribute propagation for inbound requests by clicking **CSI inbound authentication** under Additional Properties and clearing the **Security attribute propagation** option.
5. Disable security attribute propagation for outbound requests by clicking **CSI outbound authentication** under Additional Properties and clearing the **Security attribute propagation** option.

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.

Configuring the authentication cache

The security authentication cache affects the frequency of rehashing and the distribution of the hash algorithms.

To configure the authentication cache properties, complete the following steps:

1. Click **Servers > Application Servers > *server_name***.
2. Under Server infrastructure, click **Java and Process Management > Process definition**.
3. Under Additional properties, click **Java Virtual Machine > Custom Properties**.
4. Click **New** to specify a new custom property.

For information on the supported authentication cache properties, see “Security cache properties.”

Security cache properties

The following Java virtual machine (JVM) security cache custom properties determine whether the authentication cache is enabled or disabled. If the authentication cache is enabled, as recommended, these custom properties specify the initial size of the primary and secondary hash table caches, which affect the frequency of rehashing and the distribution of the hash algorithms.

Important: The `com.ibm.websphere.security.util.tokenCacheSize` and `com.ibm.websphere.security.util.LTPAValidationCacheSize` properties were replaced with the `com.ibm.websphere.security.util.authCacheSize` property.

You can specify these system properties by completing the following steps:

1. Click **Servers > Application servers > *server_name***.
2. Click **Java and Process Management > Process Definition**.

3. Under Additional properties, click **Java Virtual Machine**.
4. Specify the property name and its value in the Generic JVM arguments field. You can specify multiple property name and value pairs delimited by a space.

WebSphere Application Server includes the following security cache custom properties:

com.ibm.websphere.security.util.authCacheEnabled

Specifies whether to disable the authentication cache. It is recommended that you leave the authentication cache enabled for performance reasons. However, you can disable the authentication cache for debug or measurement purposes.

Default:	True
----------	------

com.ibm.websphere.security.util.authCacheSize

Specifies the initial size of the primary and secondary hash table caches. A higher number of available hash values might decrease the occurrence of hash collisions. A hash collision results in a linear search for the hash bucket, which might decrease the retrieval time. If several entries compose a hash table cache, you create a table with a larger capacity that supports more efficient hash entries instead of allowing automatic rehashing determine the growth of the table. Rehashing causes every entry to move each time.

Default:	200
Type:	Integer

Configuring IIOP authentication

WebSphere Application Server enables you to specify Internet Inter-ORB Protocol (IIOP) authentication for both inbound and outbound authentication requests. For inbound requests, you can specify the type of accepted authentication, such as basic authentication. For outbound requests, you can specify properties such as type of authentication, identity assertion or login configurations that are used for requests to downstream servers.

The following topics are covered in this section:

- Configuring Common Secure Interoperability Version 2 inbound authentication
- Configuring Common Secure Interoperability Version 2 outbound authentication

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 > Secure administration, applications, and infrastructure**.
3. Under RMI/IIOP security, click **CSlv2 inbound authentication**.
4. Consider the following 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 authenticates 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.

For more information, refer to Identity assertion.

- 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.

For more information, refer to User ID and password.

- Secure Sockets Layer client certificate authentication (transport layer).

The SSL client certificate is used to authenticate instead of using user ID and Password. If 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), and not from the transport layer through the client certificate authentication.

A client has an SSL client certificate that is stored in the keystore file 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 user 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 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, 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, a principal identity is sent. If authentication is done with a certificate, 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 gathers 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 a J2EE application 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. For more information, see *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, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Authentication, click **RMI/IIOP security > CSiv2 inbound authentication**.

You can also view this administrative console page on the server page by completing the following steps:

1. Click **Servers > Application servers > *server_name***.
2. Under Security, click **Server security**.
3. 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.

Basic authentication occurs in the message layer. 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 click **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 trusts 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 local OS 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.

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.

Trusted identities:

Specifies a pipe-separated (|) list of trusted server administrator user IDs, which are trusted to perform identity assertion to this server. For example, `serverid1|serverid2|serverid3`. The application server supports the comma (,) character as the list delimiter for backwards compatibility. The 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:

Select this option to enable stateful sessions, which 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 > Secure administration, applications, and infrastructure**. Under Authentication, click **Java Authentication and Authorization Service > System logins**.

Security attribute propagation:

Select this option to support security attribute propagation during login requests. When you select this option, the 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 > Secure administration, applications, and infrastructure**. Under Authentication, click **Authentication mechanisms and expiration**.

If you do not select this option, the application server does not accept any additional login information to propagate to downstream servers.

Configuring Common Secure Interoperability Version 2 outbound authentication

The following choices are available when configuring the Common Secure Interoperability Version 2 (CSlv2) Outbound Authentication panel.

Outbound authentication refers to the configuration that determines the type of authentication that is 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 runtime is responsible for preventing access to protected resources. However, if you choose to prevent an unauthenticated credential from going 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.

- Select **Identity Assertion** (attribute layer). When selected, this server sends 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 from the inbound socket. The same scenario is true for other mechanisms of authentication. Read the Identity Assertion topic for more information.
- 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.
- 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.

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.GIOPIImpl.locate(GIOPIImpl.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 the Lightweight Third Party Authentication (LTPA). 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, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Authentication, click **RMI/IIOP security > CSiv2 outbound authentication**.

You also can view this administrative console page by completing the following steps:

1. Click **Servers > Application Servers > server_name**.
2. Under Security, click **Server security**.
3. 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 - the attribute layer, the message layer, and the transport layer - 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.

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 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 inbound 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.

Use server trusted identity

Specifies the server identity that the application server uses to establish trust with the target server. The server identity can be sent using one of the following methods:

- A server ID and password when the server password is specified in the registry configuration.
- A server ID in a Lightweight Third Party Authentication (LTPA) token when the internal server ID is used.

For interoperability with application servers other than WebSphere Application Server, use of the following methods:

- Configure the server ID and password in the registry.
- Select the **Specify an alternative trusted identity** option and specify the trusted identity and password so that an interoperable Generic Security Services Username Password (GSSUP) token is sent instead of an LTPA token.

Specify an alternative trusted identity

Specifies an alternative user as the trusted identity that is sent to the target servers instead of sending the server identity. This option is recommended for identity assertion. The identity is automatically trusted when it is sent within the same cell and does not need to be in the trusted identities list within the same cell. However, this identity must be in the registry of the target servers in an external cell and the user ID must be on the trusted identities list or the identity is rejected during trust evaluation.

Trusted identity

Specifies the trusted identity that is sent from the sending server to the receiving server.

If you specify an identity in this field, it can be selected on the panel for your configured user account repository. If you do not specify an identity, a Lightweight Third Party Authentication (LTPA) token is sent between the servers.

Password

Specifies the password that is associated with the trusted identity.

Confirm password

Confirms the password that is associated with the trusted identity.

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 completing the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Authentication, click **Java Authentication and Authorization Service > System logins > New**.

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, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Authentication, click **Java Authentication and Authorization Service > System logins > New**.

Security attribute propagation:

Enables the 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, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Authentication, click **Authentication mechanisms and expiration**.

By default, the Security attribute propagation option is enabled and outbound login configuration is invoked. If you clear this option, the application server does not propagate any additional login information to downstream servers. If you select the **Use SWAM-no authenticated communication between servers** option on the Authentication mechanisms and expiration panel, the security attribute propagation feature is not supported.

Note: SWAM is deprecated in the application server Version 6.1 and will be removed in a future release.

Trusted target realms:

Specifies a list of trusted target realms, separated by a pipe character (|), that differ from the current realm.

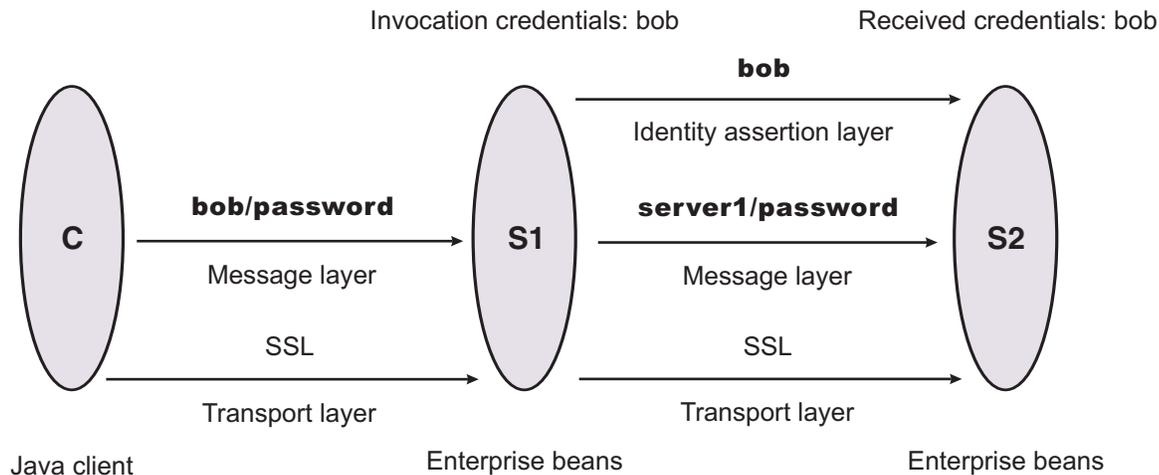
Prior to WebSphere Application Server, Version 5.1.1, if the current realm does not match the target realm, the authentication request is not sent outbound to other application servers.

Example: Common Secure Interoperability Version 2 scenarios

The articles included in this section provide specific scenarios demonstrating how to configure Common Secure Interoperability Version 2 (CSIv2).

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 following steps take you through the configuration of C, S1, and S2.



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.
Use 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.

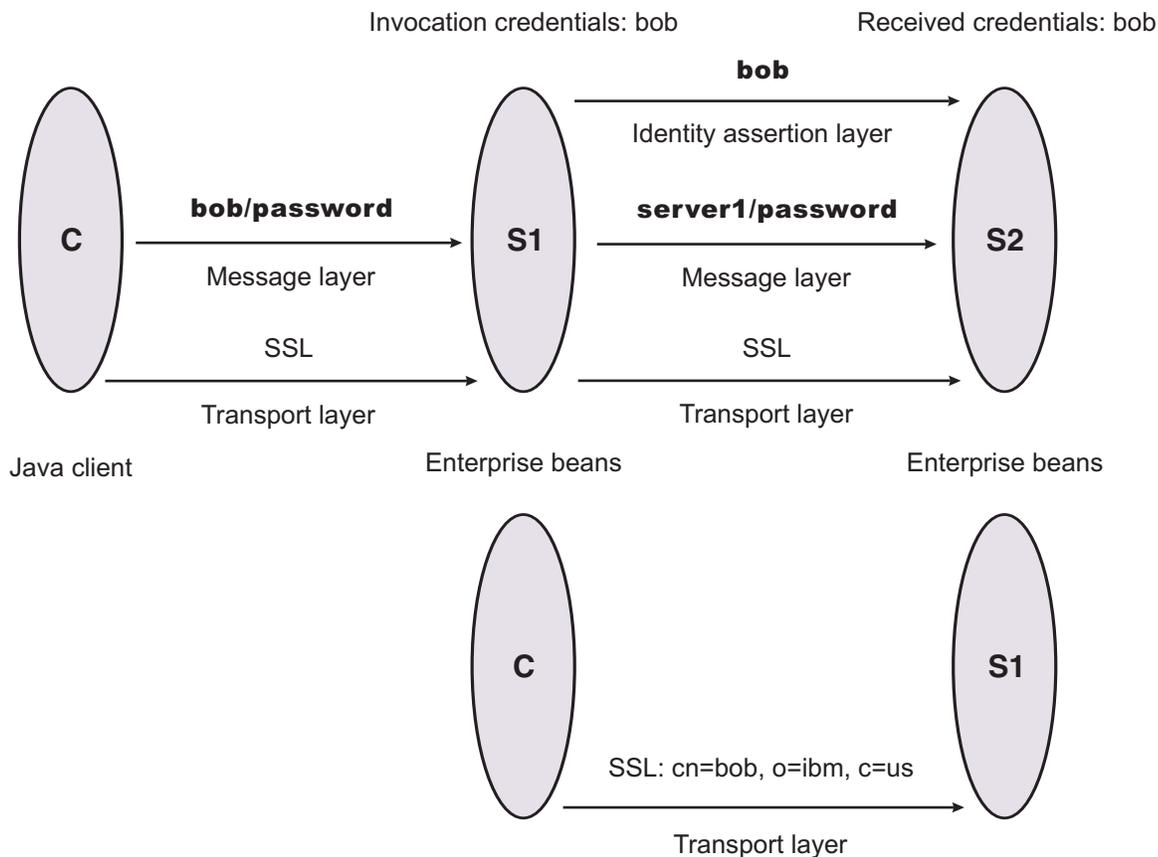
Configuring server, S2

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.

Use 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.

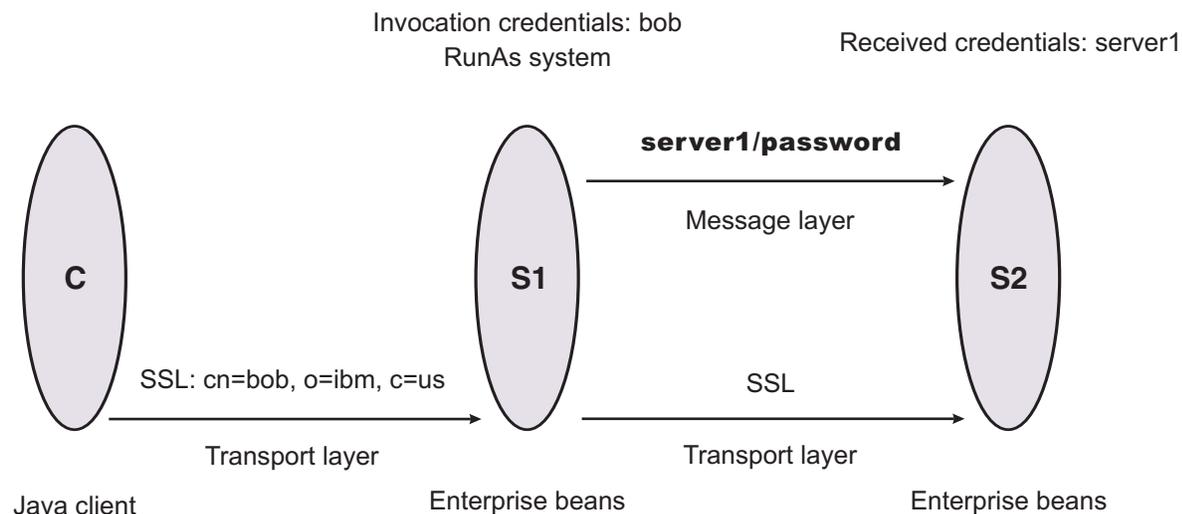
2. Disable user ID and password authentication.
3. Enable SSL.
4. Enable SSL client authentication.

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.
Use 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

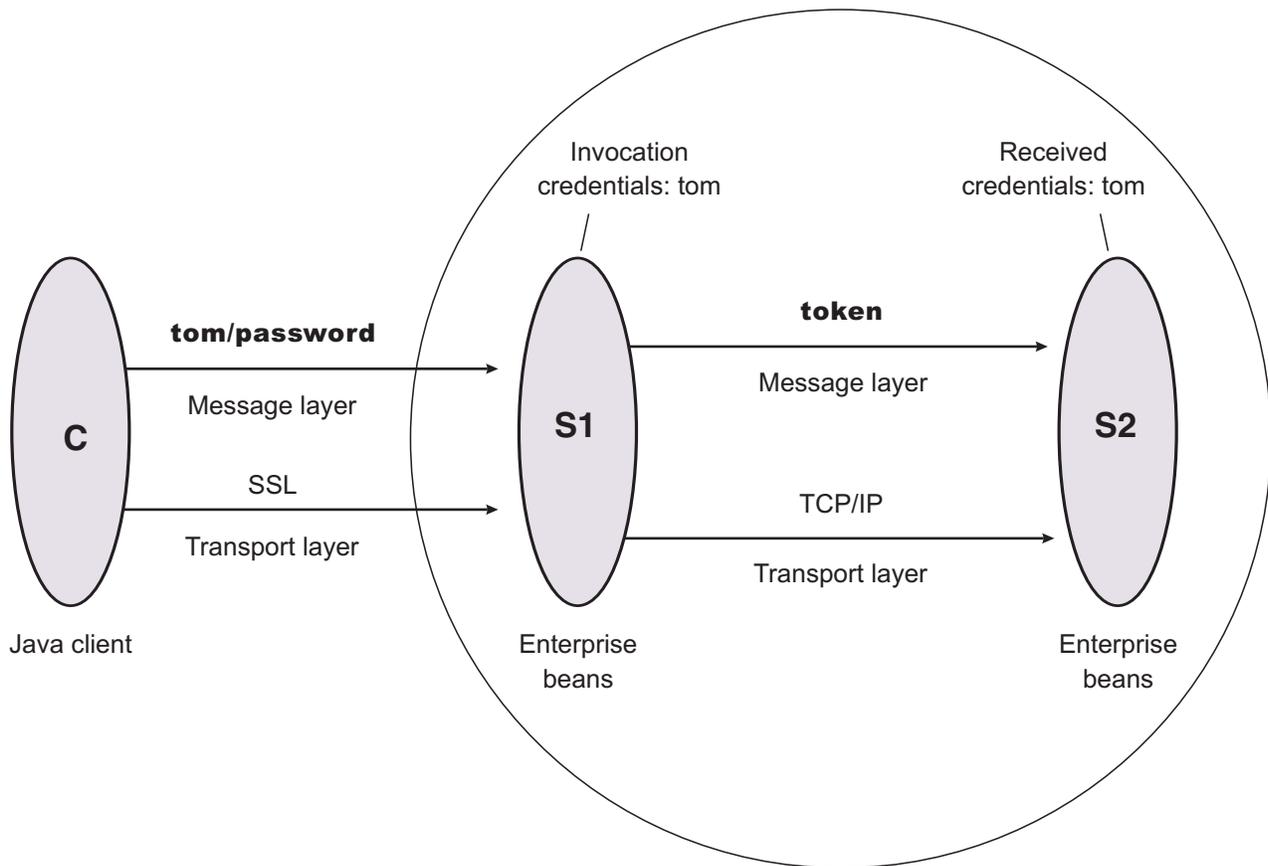
In the administrative console, the S2 server is configured for incoming requests to support message layer authentication over SSL. Configuration for outgoing requests is not relevant for this scenario.

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.

Virtual Private Network



Configuring C

C requires message layer authentication with an Secure Sockets Layer (SSL) transport:

1. Point the client to the `sas.client.props` file.
Use 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.

Configuring RMI over IIOP

Complete the following steps to configure Common Secure Interoperability Version 2 (CSIV2) and Security Authentication Service (SAS).

Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

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 must configure outbound security only.

The upstream EJB application server configures inbound security to handle the correct 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, 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 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, 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 or scripting, 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.

Configuring inbound transports

By using this configuration, you can configure a different transport for inbound security versus outbound security.

V6.0.x *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 (CSlv2) and Secure Authentication Service (SAS) have the ability to configure the transport.

Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

However, the following differences between the two protocols exist: **V6.0.x**

- CSlv2 is much more flexible than SAS, which requires Secure Sockets Layer (SSL); CSlv2 does not require SSL.
- SAS does not support SSL client certificate authentication, while CSlv2 does.
- CSlv2 can require SSL connections, while SAS only supports SSL connections.
- SAS always has two listener ports open: TCP/IP and SSL.
- CSlv2 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 > Secure administration, applications, and infrastructure**.
2. Under RMI/IIOP security, click **CSlv2 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. You can define these settings by accessing the SSL configurations panel. For more information, see `tsec_sslconfiguration.dita`.
4. Click **Apply** in the CSlv2 inbound transport panel.
5. Consider fixing the listener ports that you configured.

You complete this action in a different panel, but think about this action now. Most endpoints 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 - CSlv2 Client Authentication SSL Port
- CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS - CSlv2 SSL Port
- **V6.0.x** 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, and is statically specified using configuration dialogs or during migration. 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.
6. Click **Security > Secure administration, applications, and infrastructure**. Under RMI/IIOP security, click **CSlv2 inbound transport** to select the SSL settings that are used for inbound requests from CSlv2 clients. Remember that the CSlv2 protocol is used to inter-operate with previous releases. When configuring the keystore and truststore files in the SSL configuration, these files need the right information for inter-operating with previous releases of WebSphere Application Server. For example, a previous release has a different truststore file than the Version 6 release. If you use the Version 6 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, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Authentication, click **RMI/IIOP security > CSlv2 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: **V6.0.x** 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.

V6.0.x 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.

Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

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.

V6.0.x 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**. Under Additional properties, click **Endpoint listeners**.

Provide a fixed port number for the following ports. A zero port number indicates that a dynamic assignment is made at runtime.

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

SSL settings:

Specifies a list of predefined SSL settings to choose from for inbound connections.

Note: **V6.0.x** This option is available for non-z/OS platform servers when there is a version 6.0.x server in your environment. However, if your environment contains only Version 6.1 servers, this option does not apply.

These settings are configured at the SSL Repertoire panel. To access the SSL Repertoire panel, complete the following steps:

1. Clicking **Security > SSL certificate and key management**.
2. Under configuration settings, click **Manage endpoint security configurations and trust zones**.
3. Expand Inbound and click *inbound_configuration*.
4. Under Related items, click **SSL configurations**.

Data type:	String
Default:	DefaultSSLSettings
DefaultIOPSSL	
Range:	Any SSL settings configured in the SSL Configuration Repertoire

Centrally managed:

Specifies that the selection of an SSL configuration is based upon the outbound topology view for the Java Naming and Directory Interface (JNDI) platform.

Centrally managed configurations support one location to maintain SSL configurations rather than spreading them across the configuration documents.

Default:	Enabled
-----------------	---------

Use specific SSL alias:

Specifies the SSL configuration alias to use for LDAP outbound SSL communications.

This option overrides the centrally managed configuration for the JNDI platform.

z/OS SSL settings:

Specifies a list of predefined Secure Sockets Layer (SSL) settings for inbound connections. Configure these settings on the SSL panel by clicking Secure communications on the administrative console.

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 application server.

To view this administrative console page, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Authentication, expand RMI/IOP security and click **SAS inbound transport**.

Attention: V6.0.x The panel associated with this article displays only when you have a Version 6.0.x server in your environment. SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

SSL Settings:

Specifies a list of predefined SSL settings to choose from for inbound connections.

These settings are configured on the Secure Sockets Layer (SSL) configuration panel. To access the SSL configuration panel, complete the following steps:

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations and trust zones**.
2. Expand Inbound > *configuration_name*.
3. Under Related Items, click **SSL configurations**.

Data type:	String
Default:	DefaultSSLSettings

Configuring outbound transports

By using this configuration, you can configure a different transport for inbound security versus outbound security.

Outbound transports refers to the transport that is 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 > Secure administration, applications, and infrastructure**.
2. Under RMI/IOP security, click **CSlv2 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 > Secure administration, applications, and infrastructure**. Under RMI/IOP security, click **CSlv2 outbound transport**. By selecting the type of transport, you choose 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 that is 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. Select the **SSL required** option if you want to use Secure Sockets Layer communications with the outbound transport.

If you select the **SSL required** option, you can select either the **Centrally managed** or **Use specific SSL alias** option.

Centrally managed

Enables you to specify an SSL configuration for particular scope such as the cell, node, server, or cluster in one location. To use the **Centrally managed** option, you must specify the SSL configuration for the particular set of endpoints. The Manage endpoint security configurations and trust zones panel displays all of the inbound and outbound endpoints that use the SSL protocol. If you expand the Inbound or Outbound section of the panel and click the name of a node, you can specify an SSL configuration that is used for every endpoint on that node. For an outbound transport, you can override the inherited SSL configuration by specifying an SSL configuration for a particular endpoint. To specify an SSL configuration for an outbound transport, click **Security > SSL certificate and key management > Manage endpoint security configurations and trust zones** and expand **Outbound**.

Use specific SSL alias

Select the **Use specific SSL alias** option if you intend to select one of the SSL configurations in the menu below the option.

This configuration is used only when SSL is enabled for LDAP. The default is *DefaultSSLSettings*. To modify or create a new SSL configuration, complete the steps described in “Creating a Secure Sockets Layer configuration” on page 415.

3. **V6.0.x** Select the SSL that are settings used for outbound requests to downstream Secure Authentication Service (SAS) servers. Click **Security > Secure administration, applications, and infrastructure**. Under RMI/IOP security, click **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 inter-operating with previous releases of WebSphere Application Server. For example, a previous release has a different personal certificate than the Version 6.x release. If you use the keystore file from the Version 6.x release, you must add the signer to the truststore file of the previous release. Also, you must extract the signer for the Version 6.x release and import that signer into the truststore file of the previous release.

Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

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, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Authentication, click **RM/IIOP security > CSiv2 outbound transport**.

You also can view this administrative console by completing the following steps:

1. Click **Servers > Application servers > server_name**.
2. Under Security, click **Server security**.
3. Under Additional properties, click **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, complete the following steps:

1. Click **Security > SSL certificate and key management**.
2. Under Configuration settings, click **Manage endpoint security configurations and trust zones**.
3. Expand Outbound > *outbound_configuration_name*.
4. Under Related items, click **SSL configurations**.

Data type: String
Range: Any SSL settings that are configured in the SSL Configuration Repertoires panel

Note: **V6.0.x** This field is available only if a Version 6.0.x server exists in your environment.

SSL enabled:

Specifies whether secure socket communication is enabled to the server.

Centrally managed:

Specifies that the selection of an SSL configuration is based upon the outbound topology view for the Java Naming and Directory Interface (JNDI) platform.

Centrally managed configurations support one location to maintain SSL configurations rather than spreading them across the configuration documents.

Default: Enabled

Use specific SSL alias:

Specifies the SSL configuration alias that you want to use for outbound SSL communications.

This option overrides the centrally managed configuration for the JNDI (LDAP) protocol.

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.

To view this administrative console page, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Authentication, expand RMI/IIOP security and click **SAS outbound transport**.

Attention: **V6.0.x** The panel associated with this article displays only when you have a Version 6.0.x server in your environment.

SSL settings:

Specifies a list of predefined Secure Sockets Layer (SSL) settings to choose from for outbound connections.

These settings are configured on the SSL configuration panel. To access the SSL configuration panel, complete the following steps:

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations and trust zones**.
2. Expand Outbound > *configuration_name*.
3. Under Related Items, click **SSL configurations**.

Data type: String
Default: DefaultSSLSettings

Performing identity mapping for authorization across servers in different realms

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.

The following topics are covered in this section:

- Configuring inbound identity mapping
- Configuring outbound identity mapping to a different target realm

Configuring inbound identity mapping

For inbound identity mapping, 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 NameCallback and the WSCredTokenCallback callbacks present. 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 exception if authentication data is not valid.
    // You must have either UID or CredToken
    throw new WSSecurityPropagationException("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 log in 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 are 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 callback 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 the WSPincipal objects that are 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 a 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 the 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)
{
    // Retrieves 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 lookup 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 the 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 the shared state 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 shared state 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 (<i>LDAP UID</i>)
Windows	user (<i>Windows username</i>)
UNIX	user (<i>UNIX username</i>)

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 array list 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 that is 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 that is 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 array list

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 cache key needs to include location A so that the Subject that is 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 that is 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 > Secure administration, applications, and infrastructure > Java Authentication and Authorization Service > System logins > RMI_INBOUND**
 - b. Under Additional Properties, click **JAAS login modules > New** to add your login module to the `RMI_INBOUND` configuration.
 - c. Return to the JAAS login modules panel for `RMI_INBOUND`.
 - d. Click **Set order** to change the order that the login modules are loaded so that WebSphere Application Server loads your custom login module first. Use the **Move Up** or **Move Down** buttons to arrange the order of the login modules.
 - 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” on page 300 topic shows a custom login module that creates a `java.util.Hashtable` hashtable that is based on the specified `NameCallback` callback. The `java.util.Hashtable` hashtable is added to the `sharedState` `java.util.Map` map so that the WebSphere Application Server login modules can locate the information in the hashtable.

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. 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 (CSlv2) identity assertion. When enabled, the server 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 CSlv2 identity assertion, trust is established between 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 1.4 and higher compliant application servers. 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 CSlv2 identity assertion, see “Identity assertion” on page 216.

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 cannot be determined at the inbound server to do inbound mapping. For more information, see “Configuring outbound mapping to a different target realm” on page 303.

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 callback 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” on page 294. In other cases, such as identity assertion, you might receive a user name in the NameCallback callback of the inbound login configuration that enables you to map the identity.

Example: Custom login module for inbound mapping:

This sample shows a custom login module that creates a java.util.Hashtable hashtable that is based on the specified NameCallback callback. The java.util.Hashtable hashtable is added to the sharedState java.util.Map 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 579.)
        _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 579.)

// 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)
{
    // The authentication data is not valid. 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 log in for you. When passwords are presented, be very careful not
    // to validate the password because this is the initial authentication.

return true;
}

// You can map this uid to something else and set the identitySwitched
// boolean. If the identity is changed, clear the following propagated
// attributes 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.
        WSCREDENTIALIAL_UNIQUEID, uniqueid);
    hashtable.put(com.ibm.wsspi.security.token.AttributeNameConstants.
        WSCREDENTIALIAL_SECURITYNAME, securityName);
    hashtable.put(com.ibm.wsspi.security.token.AttributeNameConstants.
        WSCREDENTIALIAL_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 has
    // 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
    // WSCREDENTIALIAL_UNIQUEID returned, by default.
    hashtable.put(com.ibm.wsspi.security.token.AttributeNameConstants.
        WSCREDENTIALIAL_CACHE_KEY, "myCustomAttribute" + uniqueid);
    // Adds the hashtable to the shared state of the Subject.
    _sharedState.put(com.ibm.wsspi.security.token.AttributeNameConstants.
        WSCREDENTIALIAL_PROPERTIES_KEY, hashtable);
}
else if (requiresLogin == false)
{
    // For more information on this section, see
    // "Security attribute propagation" on page 191.
    // 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 579.)
}

// 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 topic details alternatives for enabling one server to send outbound requests to a target server in a different realm.

This outbound request is rejected to protect against a rogue server reading potentially sensitive information if successfully impersonating the home of the object. Select one of the following alternative procedures so that one server can send outbound requests to a target server in a different realm. When you are finished with a procedure on the administrative console, click **Apply**.

- 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. Complete the following steps in the administrative console:
 1. Click **Security > Secure administration, applications, and infrastructure**.
 2. Under RMI/IIOP security, click **CSlv2 outbound authentication**.
 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 operating system. 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 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 304.
 2. Use the code sample in “Example: Using the WSLogin configuration to create a basic authentication subject” on page 304 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 > Secure administration, applications, and infrastructure**.
 - b. Under RMI/IIOP security, click **CSlv2 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 579.

The “Example: Sample login configuration for RMI_OUTBOUND” on page 305 shows a custom login module that determines whether the realm names match. In this example, the realm names do not match so the WSLoginmodule 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 > Secure administration, applications, and infrastructure**.

2) Under Java Authentication and Authorization Service, click **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.

5) 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 > Secure administration, applications, and infrastructure**.

2. Under Java Authentication and Authorization Service, click **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.

The following changes are made to the security.xml file:

```
<entries xmi:id="JAASConfigurationEntry_2" alias="WSLogin">
  <loginModules xmi:id="JAASLoginModule_2"
    moduleName="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 log in 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 code is an alternative that validates the user ID and
// password specified against the target realm. The code performs 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 not valid. If false is
// returned, a WSLoginFailedException exception is created. 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();
    }
};

// Starts 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 579.)
    }

    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 579.)

        // 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 application programming interface (API) documentation for
        // more information.
        csiv2PerformPolicy = (CSIv2PerformPolicy) ((WSProtocolPolicyCallback)callbacks[0]).
            getProtocolPolicy();

        // Checks if the realms do not match. If they do not match, then log in 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 579.)

    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 WSSLoginFailedException
                {
                    // 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 WSSLoginFailedException (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 WSSLoginFailedException (e.getMessage(), e);
                    }

                    return null;
                }
            });
        }
        catch (PrivilegedActionException e)
        {
            throw new WSSLoginFailedException (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;
}
```

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. For example, this property accepts any valid Web address.

```
-Dcom.ibm.CORBA.ConfigURL=file:profile_root/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.

V6.0.x Use the following properties to configure the SAS and CSiv2 authentication protocols:

- “Security Authentication Service authentication protocol client settings” on page 312

Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

Common authentication protocol settings for a client configuration

You can use settings in the `sas.client.props` file to configure Security Authentication Service (SAS) and Common Secure Interoperability Version 2 (CSiv2) clients.

V6.0.x Use the following settings in the `app_server_root/properties/sas.client.props` file to configure SAS and CSiv2 clients.

Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

com.ibm.CORBA.securityEnabled

Use to determine if security is enabled for the client process.

Setting	Value
Data Type	Boolean
Default	True
Valid values	True or false

com.ibm.CSI.protocol

Use to determine which authentication protocols are active.

The client can configure protocols of `ibm`, `csiv2` or both as active. The only possible values for an authentication protocol are `ibm`, `csiv2` and `both`. Do not use `sas` for the value of an authentication protocol. This restriction applies to both client and server configurations. The following list provides information about using each of these protocol options:

ibm Use this authentication protocol option when you are communicating with WebSphere Application Server Version 4.x or earlier servers.

V6.0.x csiv2

Use this authentication protocol option when you are communicating with WebSphere Application Server Version 5 or later servers because the SAS interceptors are not loaded and running for each method request.

both Use this authentication protocol option for interoperability between WebSphere Application Server Version 4.x or earlier servers and WebSphere Application Server Version 5 or later servers. Typically, specifying both provides greater interoperability with other servers.

Setting	Value
Data type	String
Default	Both
Valid values	<code>ibm</code> , <code>csiv2</code> , <code>both</code>

com.ibm.CORBA.authenticationTarget

Use to determine the type of authentication mechanism for sending security information from the client to the server.

If basic authentication is specified, the user ID and password are sent to the server. Using the Secure Sockets Layer (SSL) transport with this type of authentication is recommended; otherwise, the password is not encrypted. The target server must support the specified authentication target.

If you specify Lightweight Third Party Authentication (LTPA), then LTPA must be the mechanism configured at the server for a method request to proceed securely.

Setting	Value
Data type	String
Default	<code>BasicAuth</code>
Valid values	<code>BasicAuth</code> , <code>LTPA</code>

com.ibm.CORBA.validateBasicAuth

Use to determine if the user ID and password get validated immediately after the login data is entered when the `authenticationTarget` property is set to `BasicAuth`.

In previous releases, `BasicAuth` logins validated only with the initial method request. During the first request, the user ID and password are sent to the server. This request is the first time that the client can notice an error, if the user ID or password is incorrect. The `validateBasicAuth` method is specified and the validation of the user ID and password occurs immediately to the security server.

For performance reasons, you might want to disable this property if you do not want to verify the user ID and password immediately. If the client program can wait, it is better to have the initial method request flow to the user ID and password. However, program logic might not be this simple because of error handling considerations.

Setting	Value
Data type	Boolean
Default	True
Valid values	True, False

com.ibm.CORBA.authenticationRetryEnabled

Use to specify that a failed login attempt is retried. This property determines if a retry occurs for other errors, such as stateful sessions that are not found on a server or validation failures at the server because of an expiring credential.

The minor code in the exception that is returned to a client determines which errors are retried. The number of retry attempts is dependent upon the `com.ibm.CORBA.authenticationRetryCount` property.

Setting	Value
Data type	Boolean
Default	True
Valid values	True, False

com.ibm.CORBA.authenticationRetryCount

Use to specify the number of retries that occur until either a successful authentication occurs or the maximum retry value is reached.

When the maximum retry value is reached, the authentication exception is returned to the client.

Setting	Value
Data type	Integer
Default	3
Range	1-10

com.ibm.CORBA.loginSource

Use to specify how the request interceptor attempts to log in if it does not find an invocation credential already set.

This property is valid only if message layer authentication occurs. If only transport layer authentication occurs, this property is ignored. When specifying properties, the following two additional properties must be defined:

- `com.ibm.CORBA.loginUserid`
- `com.ibm.CORBA.loginPassword`

When performing a programmatic login, it is not necessary to specify none as the login source. The request fails if a credential is set as the invocation credential during a method request.

Setting	Value
Data type	String
Default	Prompt
Valid values	Prompt, key file, stdin, none, properties

com.ibm.CORBA.loginUserid

Use to specify the user ID when a properties login is configured and message layer authentication occurs.

This property is valid only when `com.ibm.CORBA.loginSource=properties`. Also set the `com.ibm.CORBA.loginPassword` property.

Setting	Value
Data type	String
Range	Any string that is appropriate for a user ID in the configured user registry of the server.

com.ibm.CORBA.loginPassword

Use to specify the password when a properties login is configured and message layer authentication occurs.

This property is valid only when `com.ibm.CORBA.loginSource=properties`. Also set the `com.ibm.CORBA.loginUserid` property.

Setting	Value
Data type	String
Range	Any string that is appropriate for a password in the configured user registry of the server.

com.ibm.CORBA.keyFileName

Use to specify the key file that is used to log in.

A key file is a file that contains a list of realm, user ID, and password combinations that a client uses to log into multiple realms. The realm that is used is the one found in the interoperable object reference (IOR) for the current method request. The value of this property is used when the `com.ibm.CORBA.loginSource=key` file is used.

Setting	Value
Data type	String
Default	C:/WebSphere/AppServer/properties/wssserver.key
Range	Any fully qualified path and file name of a WebSphere Application Server key file.

com.ibm.CORBA.loginTimeout

Use to specify the length of time that the login prompt stays available before it is considered a failed login.

Setting	Value
Data type	Integer
Units	Seconds
Default	300 (5 minute intervals)
Range	0 - 600 (10 minute intervals)

com.ibm.CORBA.securityEnabled

Use to determine if security is enabled for the client process.

Setting	Value
Data type	Boolean
Default	True
Range	True, False

Related tasks

“Configuring RMI over IIOP” on page 286

Complete the following steps to configure Common Secure Interoperability Version 2 (CSIV2) and Security Authentication Service (SAS).

Related reference

“Common Secure Interoperability Version 2 and Security Authentication Service client configuration” on page 308

A secure Java client requires configuration properties to determine how to perform security with a server.

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.

Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

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

Java Authentication and Authorization Service

The standard Java 2 security application programming interface (API) helps enforce access control based on the location of the code source or the author or packager of the code that signed the jar file. 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. JAAS 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. JAAS 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” on page 64 for more information.

The following sections cover the JAAS implementation and programming model:

- “Login configuration for Java Authentication and Authorization Service” on page 555
- “Programmatic login” on page 343
- “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*. 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 runtime automatically determines whether the policy grants the required permission to a specific principal only. If so, the operation is supported if the subject that is 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 the `java.security.PrivilegedAction` or `java.security.PrivilegedExceptionAction` method. 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 Enterprise JavaBeans (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 that is specified in the subject when invoking the EJB method within the `Subject.doAs` action block.

Letting the `Subject.doAs` action overwrite the `run-as` identity setting is an ideal way to integrate the JAAS programming model with the J2EE run-time environment. However, JAAS introduced an issue into the Software Development Kit (SDK), Java Technology Edition Versions 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` action 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;
}
});

```

In the previous code example, the Subject object is associated with the context of the current thread. 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.

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 that is specified by the run-as setting and not by the subject identity.

- The Subject object that is generated by the WSLoginModuleImpl instance and the WSClietLoginModuleImpl 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 that represents the caller security identity. The return type differs from the getCallerPrincipal method of the java.security.Principal EJBContext interface.

- The Subject object that is generated by the Java 2 Connector (J2C) DefaultPrincipalMapping module contains a resource principal and a PasswordCredentials list. The resource principal represents the RunAs identity.

For more information, see J2EE connector security.

Using the Java Authentication and Authorization Service programming model for Web authentication

WebSphere Application Server supports the Java 2 Platform, Enterprise Edition (J2EE) declarative security model. You can define the authentication and access control policy using the J2EE deployment descriptor. You can further stack custom login modules to customize the WebSphere Application Server authentication mechanism.

A custom login module can perform principal and credential mapping, custom security token and custom credential-processing, and error-handling among other possibilities. Typically, you do not need to use application code to perform authentication function. Use the programming techniques that are described in this section if you have to perform authentication function in application code. For example, if you have applications that programmed to the SSOAuthenticator helper function, you can use the following programming interface. The SSOAuthenticator helper function was deprecated starting with WebSphere Application Server Version 4.0. Use declarative security as a rule; use the techniques that are described in this section as a last resort.

When the Lightweight Third-Party Authentication (LTPA) mechanism single sign-on (SSO) option is enabled, the Web client login session is tracked by an LTPA SSO token cookie after successful login. At logout, this token is deleted to terminate the login session, but the server-side subject is not deleted. When you use the declarative security model, the WebSphere Application Server Web container performs client authentication and login session management automatically. You can perform authentication in application code by setting a login page without a J2EE security constraint and by directing client requests to your login page first. Your login page can use the Java Authentication and Authorization Service (JAAS) programming model to perform authentication. To enable WebSphere Application Server Web login modules to generate SSO cookies, use the following steps.

1. Select the `wsMapDefaultInboundLoginModule` login module and click **Custom properties**. There are two login modules defined in your login configuration: `ltpaLoginModule` and `wsMapDefaultInboundLoginModule`.
2. Create a new system login JAAS configuration. To access the panel, click **Security > Secure administration, applications, and infrastructure**. Under Java Authentication and Authorization Service, click **System logins**.
3. Manually clone the `WEB_INBOUND` login configuration, and give it a new alias. To clone the login configuration, click **New**, enter a name for the configuration, click **Apply**, then click **JAAS login modules** under Additional properties. Click **New** and configure the JAAS login module. For more information, see “Login module settings for Java Authentication and Authorization Service” on page 565. WebSphere Application Server Web container uses the `WEB_INBOUND` login configuration to authenticate Web clients. Changing the `WEB_INBOUND` login configuration affects all Web applications in the cell. You should create your own login configuration by cloning the contents of the `WEB_INBOUND` login configuration.
4. Select the `wsMapDefaultInboundLoginModule` login module and click **Custom properties**. There are two login modules defined in your login configuration: `ltpaLoginModule` and `wsMapDefaultInboundLoginModule`.
5. Add a login property name `cookie` with a value of **true**. The two login modules are enabled to generate LTPA SSO cookies. Do not add the cookie login option to the original `WEB_INBOUND` login configuration.
6. Stack your custom LoginModule(s) in the new login configuration (optional).
7. Use your login page for programmatic login by perform a `JAAS LoginContext.login` using your newly defined login configuration. After a successful login, either the `ltpaLoginModule` or the

`wsMapDefaultInboundLoginModule` generates an LTPA SSO cookie upon a successful authentication. Exactly which `LoginModule` generates the SSO cookie depends on many factors, including system authentication configuration and runtime condition (which is beyond the scope of this section).

8. Call the modified `WSSubject.setRunAsSubject` method to add the subject to the authentication cache. The subject must be a WebSphere Application Server JAAS subject created by `LoginModule`. Adding the subject to the authentication cache recreates a subject from SSO token.
9. Use your programmatic logout page to revoke SSO cookies by invoking the `revokeSSOCookies` method from the `WSecurityHelper` class. The term cookies is used because WebSphere Application Server Release 5.1.1 (and later) release supports a new LTPA SSO token with a different encryption algorithm, but can be configured to generate the original LTPA SSO token for backward compatibility. Note that the subject is still in the authentication cache and only the SSO cookies are revoked.

Use the following code sample to perform authentication:

Suppose you wrote a `LoginServlet.java`:

```
import com.ibm.wsspi.security.auth.callback.WSCallbackHandlerFactory;
import com.ibm.websphere.security.auth.WSSubject;

public Object login(HttpServletRequest req, HttpServletResponse res)
throws ServletException {

    PrintWriter out = null;
    try {
        out = res.getWriter();
        res.setContentType("text/html");
    } catch (java.io.IOException e){
        // Error handling
    }

    Subject subject = null;
    try {
        LoginContext lc = new LoginContext("system.Your_login_configuration",
        WSCallbackHandlerFactory.getInstance().getCallbackHandler(
        userid, null, password, req, res, null));
        lc.login();
        subject = lc.getSubject();
        WSSubject.setRunAsSubject(subject);
    } catch(Exception e) {
        // catch all possible exceptions if you want or handle them separately
        out.println("Exception in LoginContext login + Exception = " +
        e.getMessage());
        throw new ServletException(e.getMessage());
    }
}
```

The following is sample code to revoke the SSO cookies upon a programming logout:

The `LogoutServlet.java`:

```
public void logout(HttpServletRequest req, HttpServletResponse res,
Object retCreds) throws ServletException {
    PrintWriter out =null;
    try {
        out = res.getWriter();
        res.setContentType("text/html");
    } catch (java.io.IOException e){
        // Error Handling
    }
    try {
        WSSecurityHelper.revokeSSOCookies(req, res);
    } catch(Exception e) {
        // catch all possible exceptions if you want or handle them separately
    }
}
```

```
    out.println("JAASLogoutServlet: logout Exception = " + e.getMessage());  
    throw new ServletException(e);  
  }  
}
```

For more information on JAAS authentication, refer to [Developing programmatic logins with the Java Authentication and Authorization Service](#). For more information on the `AuthenLoginModule` login module, refer to [Example: Customizing a server-side Java Authentication and Authorization Service authentication and login configuration](#).

Chapter 6. Authorizing access to resources

WebSphere Application Server provides many different methods for authorizing accessing resources. For example, you can assign roles to users and configure a built-in or external authorization provider.

You can create an application, an Enterprise JavaBeans (EJB) module, or a Web module and secure them using assembly tools.

To authorize user or group access to resources, read the following articles:

1. Secure you application during assembly and deployment. For more information on how to create a secure application using an assembly tool, such as the IBM Rational Application Developer, see “Securing applications during assembly and deployment” on page 903.
For general information about the tools that WebSphere Application Server supports, see Assembly tools and Assembling applications.
2. Authorize access to Java 2 Platform, Enterprise Edition (J2EE) resources. WebSphere Application Server supports authorization that is based on the Java Authorization Contract for Containers (JACC) specification in addition to the default authorization. When security is enabled in WebSphere Application Server, the default authorization is used unless a JACC provider is specified. For more information, see “Authorization providers” on page 329.
3. Authorize access to administrative resources. You can assign users and groups to predefined administrative roles such as the monitor, configurator, operator, administrator, and iscadmins roles. These roles determine which tasks a user can perform in the administrative console. For more information, see “Authorizing access to administrative roles” on page 372.

After authorizing access to resources, configure the Application Server for secure communication. For more information, see Chapter 7, “Securing communications,” on page 381.

Authorization technology

Authorization information determines whether a user or group has the necessary privileges to access resources.

WebSphere Application Server supports many authorization technologies including the following:

- Authorization involving the Web container and Java 2 Platform, Enterprise Edition (J2EE) technology
- Authorization involving an enterprise bean application and J2EE technology
- Authorization involving Web services and J2EE technology
- Java Message Service (JMS)
- Java Authorization Contract for Containers (JACC)

WebSphere Application Server supports both a default authorization provider, which was supported in previous releases, and an authorization provider that is based on the Java Authorization Contract for Containers (JACC) specification. The JACC-based authorization provider enables third-party security providers to handle the J2EE authorization. For more information, see “JACC support in WebSphere Application Server” on page 330.

- Java Authentication and Authorization Service (JAAS)
For more information, see “Java Authentication and Authorization Service” on page 312.
- Java 2 security
For more information, see “Java 2 security” on page 64.
- Naming and administrative authorization
- Pluggable authorization

WebSphere Application Server supports an authorization infrastructure that enables you to plug in an external authorization provider. For more information, see “Enabling an external JACC provider” on page 354.

Administrative roles 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 roles

A number of administrative roles are defined to provide the degrees of authority that are needed to perform certain WebSphere Application Server administrative functions from either the administrative console or the system management scripting interface called wsadmin. The authorization policy is only enforced when administrative security is enabled. The following table describes the administrative roles:

Table 10. Administrative roles that are available through the administrative console and wsadmin

Role	Description
Monitor	<p>An individual or group that uses the monitor role has the least amount of privileges. A monitor can complete the following tasks:</p> <ul style="list-style-type: none"> • View the WebSphere Application Server configuration. • View the current state of the Application Server.
Configurator	<p>An individual or group that uses the configurator role has the monitor privilege plus the ability to change the WebSphere Application Server configuration. The configurator can perform all the day-to-day configuration tasks. For example, a configurator can complete the following tasks:</p> <ul style="list-style-type: none"> • Create a resource. • Map an application server. • Install and uninstall an application. • Deploy an application. • Assign users and groups-to-role mapping for applications. • Set up Java 2 security permissions for applications. • Customize the Common Secure Interoperability Version 2 (CSIv2), Security Authentication Service (SAS), and Secure Sockets Layer (SSL) configurations. <p>Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.</p>
Operator	<p>An individual or group that uses the operator role has monitor privileges plus ability to change the runtime state. For example, an operator can complete the following tasks:</p> <ul style="list-style-type: none"> • Stop and start the server. • Monitor the server status in the administrative console.

Table 10. Administrative roles that are available through the administrative console and wsadmin (continued)

Role	Description
Administrator	<p>An individual or group that uses the administrator role has the operator and configurator privileges plus additional privileges that are granted solely to the administrator role. For example, an administrator can complete the following tasks:</p> <ul style="list-style-type: none"> • Modify the server user ID and password. • Configure authentication and authorization mechanisms. • Enable or disable administrative security. <p>Note: In previous releases of WebSphere Application Server, the Enable administrative security option is known as the Enable global security option.</p> <ul style="list-style-type: none"> • Enforce Java 2 security using the Use Java 2 security to restrict application access to local resources option. • Change the Lightweight Third Party Authentication (LTPA) password and generate keys. • Create, update, or delete users in the federated repositories configuration. • Create, update, or delete groups in the federated repositories configuration. <p>Note: An administrator cannot map users and groups to the administrator roles.</p>
AdminSecurityManager	<p>Only users who are granted this role can map users to administrative roles. Also, when fine-grained administrative security is used, only users who are granted this role can manage authorization groups. See “Administrative roles” on page 326 for more information.</p>

Table 11. Additional administrative role that is available through the administrative console

Role	Description
iscadmins	<p>This role is only available for administrative console users and not for wsadmin users. Users who are granted this role have administrator privileges for managing users and groups in the federated repositories. For example, a user of the iscadmins role can complete the following tasks:</p> <ul style="list-style-type: none"> • Create, update, or delete users in the federated repositories configuration. • Create, update, or delete groups in the federated repositories configuration.

Table 12. Additional administrative role that is available through wsadmin

Role	Description
Deployer	<p>This role is only available for wsadmin users and not for administrative console users. Users who are granted this role can perform both configuration actions and run-time operations on applications.</p>

When administrative security is enabled, the administrative subsystem role-based access control is enforced. The administrative subsystem includes the security server, the administrative console, the wsadmin scripting tool, and all the Java Management Extensions (JMX) MBeans. When administrative 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.

When you are changing registries (for example, from a federated repository to LDAP), make sure you remove the information that pertains to the previously configured registry for console users and console groups.

When administrative 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 runtime 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 administrative security
- Enforce Java 2 security using the **Use Java 2 security to restrict application access to local resources** option.
- Change the LTPA password or generate keys
- Assign users and groups to administrative roles

Primary administrative user name

The Version 6.1 release of WebSphere Application Server requires an administrative user, distinguished from the server user identity, to improve auditability of administrative actions. The user name specifies a user with administrative privileges that is defined in the local operating system.

Server user identity

The Version 6.1 release of WebSphere Application Server distinguishes the server identity from the administrative user identity to improve auditability. The server user identity is used for authenticating server-to-server communications.

Internal server ID

The internal server ID enables the automatic generation of the user identity for server-to-server authentication. Automatic generation of the server identity supports improved auditability for cells only for Version 6.1 or later nodes. In the Version 6.1 release of WebSphere Application Server, you can save the internally-generated server ID because the Security WebSphere Common Configuration Model (WCCM) model contains a new tag, internalServerId. You do not need to specify a server user ID and a password during security configuration except in a mixed-cell environment. An internally-generated server ID adds a further level of protection to the server environment because the server password is not exposed as it is in releases prior to Version 6.1. However, to maintain backwards compatibility, you must specify the server user ID if you use earlier versions of WebSphere Application Server.

When enabling security, you can assign one or more users and groups to naming 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*.

Special subject

In addition to mapping users or groups, you can map a *special-subject* to the administrative roles. A special-subject is a generalization of a particular class of users. The AllAuthenticated special subject means that the access check of the administrative role ensures that the user making the request is at least authenticated. The Everyone special subject means that anyone, authenticated or not, can perform the action as if security is not enabled.

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. These functions affect the content of the WebSphere Application Server name space. Generally, you have two ways in which client programs result in CosNaming calls. The first is through the Java Naming and Directory Interface (JNDI) call. The second is with common object request broker architecture (CORBA) clients invoking CosNaming methods directly.

Four security roles are introduced :

- CosNamingRead
- CosNamingWrite
- CosNamingCreate
- CosNamingDelete

The roles have authority levels from low to high:

CosNamingRead

You can query 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

You can perform write operations such as JNDI bind, rebind, or unbind, and CosNamingRead operations. As a default policy, Subjects are not assigned this role.

CosNamingCreate

You can create new objects in the name space through such operations as JNDI createSubcontext and CosNamingWrite operations. As a default policy, Subjects are not assigned this role.

CosNamingDelete

You can destroy objects in the name space, for example using the JNDI destroySubcontext method and CosNamingCreate operations. As a default policy, Subjects are not assigned this role.

Additionally, a Server special-subject is assigned to all of the four CosNaming roles by default. The Server special-subject provides a WebSphere Application Server process, which runs under the server identity, to access all the CosNaming operations. The Server special-subject does not display and cannot be modified through the administrative console or other administrative tools.

Special configuration is not required to enable the server identity as specified when enabling administrative security for administrative use because the server identity is automatically mapped to the administrator 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 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 administrative security is enabled. When administrative 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.

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.

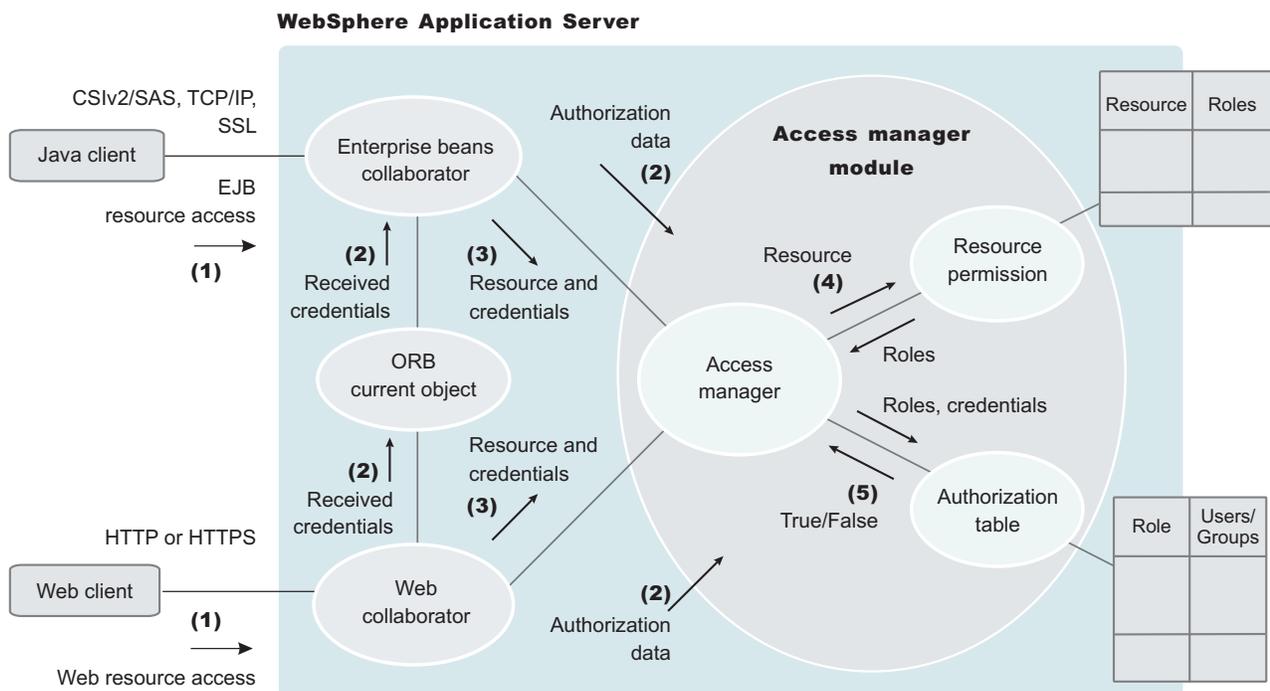
Although the ability exists to greatly restrict access to the name space by changing the default policy, unexpected org.omg.CORBA.NO_PERMISSION exceptions can occur at runtime. Typically, J2EE applications access the name space and the identity they use is that of the user that authenticated to WebSphere Application Server when accessing the J2EE application. Unless the J2EE application provider clearly communicates the expected naming roles, use caution when changing the default naming authorization policy.

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 that is used during authorization.

Authentication



Web resource access from a Web client is handled by a Web collaborator. The Enterprise JavaBeans (EJB) resource access from a Java client, whether an enterprise bean 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 object. The resource and the received credentials are presented to the WSAccessManager access manager to check whether access is permitted to the client for accessing the requested resource.

The access manager module contains two main modules:

- The resource permission module helps determine the required roles for a given resource. This module uses a resource-to-roles mapping table that is built by the security runtime during application startup. To build the resource-to-role mapping table, the security runtime reads the deployment descriptor of the enterprise beans or the Web module (ejb-jar.xml file or web.xml file)

- The 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 runtime during application startup.

To build the authorization table, the security runtime reads the `ibm-application-bnd.xml` application binding file.

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 Platform, 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 that are 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.

Two *special subjects* are not defined by the J2EE model: `AllAuthenticatedUsers` and `Everyone`. A special subject is a product-defined entity that is independent of the user registry. This entity is used to generically represent a class of users or groups in the registry.

- The `AllAuthenticatedUsers` subject 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. All of this is done independent of the user registry.
- `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. All of this is done independent of the user registry.

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 that are 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 instead of 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 runtime, 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 run 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 beans, 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>

Administrative roles

The Java 2 Platform, Enterprise Edition (J2EE) role-based authorization concept is extended to protect the WebSphere Application Server administrative subsystem.

A number of administrative roles are defined to provide degrees of authority that are needed to perform certain administrative functions from either the Web-based administrative console or the system management scripting interface. The authorization policy is only enforced when administrative security is enabled. The following table describes the administrative roles:

Administrative roles

Role	Description
Monitor	An individual or group that uses the monitor role has the least amount of privileges. A monitor can complete the following tasks: <ul style="list-style-type: none"> • View the WebSphere Application Server configuration. • View the current state of the Application Server.
Configurator	An individual or group that uses the configurator role has the monitor privilege plus the ability to change the WebSphere Application Server configuration. The configurator can perform all the day-to-day configuration tasks. For example, a configurator can complete the following tasks: <ul style="list-style-type: none"> • Create a resource. • Map an application server. • Install and uninstall an application. • Deploy an application. • Assign users and groups-to-role mapping for applications. • Set up Java 2 security permissions for applications. • Customize the Common Secure Interoperability Version 2 (CSIv2), Security Authentication Service (SAS), and Secure Sockets Layer (SSL) configurations. <p>Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.</p>

Administrative roles

Operator	An individual or group that uses the operator role has monitor privileges plus ability to change the runtime state. For example, an operator can complete the following tasks: <ul style="list-style-type: none">• Stop and start the server.• Monitor the server status in the administrative console.
Administrator	An individual or group that uses the administrator role has the operator and configurator privileges plus additional privileges that are granted solely to the administrator role. For example, an administrator can complete the following tasks: <ul style="list-style-type: none">• Modify the server user ID and password.• Configure authentication and authorization mechanisms.• Enable or disable administrative security.• Enable or disable Java 2 security.• Change the Lightweight Third Party Authentication (LTPA) password and generate keys.• Create, update, or delete users in the federated repositories configuration.• Create, update, or delete groups in the federated repositories configuration. Note: An administrator cannot map users and groups to the administrator roles.
iscadmins	This role is only available for administrative console users, not for wsadmin users. Users who are granted this role have administrator privileges for managing users and groups in the federated repositories. For example, a user of the iscadmins role can complete the following tasks: <ul style="list-style-type: none">• Create, update, or delete users in the federated repositories configuration.• Create, update, or delete groups in the federated repositories configuration.
Deployer	This role is only available for wsadmin users, not for administrative console users. Users granted this role can perform both configuration actions and runtime operations on applications. See the “Deployer role” section for more details.
AdminSecurityManager	This role is only available for wsadmin users, not for administrative console users. When using wsadmin, users granted this role can map users to administrative roles. Also, when fine grained admin security is used, users granted this role can manage authorization groups. See the “AdminSecurityManager role” on page 328 section for more details.

The server ID that is specified and the administrative ID, if specified, when enabling administrative security is automatically mapped to the administrator role.

Users and groups can be added or removed from the administrative roles from the WebSphere Application Server administrative console at any time. A best practice is to map a group or groups, rather than specific users, to administrative roles because it is more flexible and easier to administer.

In addition to mapping user or groups, a special-subject can also be mapped to the administrative roles. A special-subject subject is a generalization of a particular class of users. The AllAuthenticated special subject means that the access check of the administrative role ensures that the user making the request is at least authenticated. The Everyone special subject means that anyone, authenticated or not, can perform the action, as if security was not enabled.

Deployer role

A user that is granted a deployer role can perform all of the configuration and runtime operations on an application. A deployer role can be subsets of both configurator and operator roles. However, a user granted a deployer role cannot configure or operate any other resources (server, node).

When fine-grained administrative security is used, only a user granted a deployer role to an application can configure and operate that application.

Cell level configurators can configure applications (install, edit, deploy, and uninstall). Cell level operators can also operate (start and stop) applications. However, a user granted a deployer role at cell level can also perform configuration and operation on all applications.

The following table lists the capabilities of the deployer role when fine-grained administrative security is used:

Operation	Required Roles (Any one)
Install application	Cell-configurator, target-deployer
Uninstall application	Cell-configurator, application-deployer
List application	Cell-monitor, application-monitor
Edit, update and redeploy application	Cell-configurator, application-deployer
Export application	Cell-monitor, application-monitor
Start-stop application	Cell-operator, application-deployer

Where:

Cell-configurator

is the configurator role at cell level.

Application-deployer

is the deployer role for the application that is being managed.

Target-deployer

is the deployer role for all servers or clusters for which an application is targeted. If you have a target-deployer role, you can install a new application on the target. However, to edit or update the installed application, you must be included in the authorization group of the installed application-deployer.

The target-deployer can not explicitly start or stop a new application. However, when a target-deployer starts a server on a target, all of the applications that have their auto-start attribute set to yes are started when the server starts.

It is recommended that the application-deployer set this attribute to true if the application-deployer does not want the application to be started by the target-deployer.

AdminSecurityManager role

The AdminSecurityManager role separates administrative security administration from other application administration.

By default, serverId and adminID, if specified, are assigned to this role in the cell level authorization table. This role implies a monitor role. However, an administrator role does not imply the AdminSecurityManager role.

When fine-grained admin security is used, only a user granted this role at cell level can manage administrative authorization groups. However, a user granted this role for each administrative authorization group can map users to administrative roles for those groups. The following lists the capabilities of the AdminSecurityManager role at different levels (cell and administrative authorization group):

Action	Who can perform
Map users to administrative roles for cell level	Only the AdminSecurityManager of the cell

Action	Who can perform
Map users to administrative roles for an authorization group	Only the AdminSecurityManager of that authorization group or the AdminSecurityManager of the cell
Manage authorization groups (create, delete, add resource to an authorization group, or remove resource from an authorization group or list)	Only the AdminSecurityManager of the cell

Related tasks

“Assigning users to naming roles” on page 376

Use this task to assign users to naming roles by using the administrative console.

Enterprise bean component security

An Enterprise JavaBeans (EJB) module consists of one or more beans. You can use development tools such as Rational Application 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 access mark the set **excluded**. You can give everyone access to a set of enterprise beans methods by clearing those methods. You can run enterprise beans as a different identity, using the runAs identity, before invoking other enterprise beans.

Authorization providers

WebSphere Application Server supports authorization that is 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.

When security is enabled in 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 set up for WebSphere Application Server to use, all of the enterprise beans and Web authorization decisions are 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 Enterprise JavaBeans (EJB) components are protected and authorized per the J2EE specification. The administrative components are internal to WebSphere Application Server and are protected by the role-based authorizer. The administrative components include the administrative console, MBeans, and other components such as naming and security. For more information on administrative security, see “Role-based authorization” on page 324.

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, and function the same. Choose a JACC provider only when you want to work with an external security provider such as Tivoli Access Manager. In this instance, the security provider must support the JACC specification and be set up to work with 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 support in WebSphere Application Server

WebSphere Application Server supports the Java Authorization Contract for Containers (JACC) specification, which enables third-party security providers to handle the Java 2 Platform, Enterprise Edition (J2EE) authorization.

The JACC 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 store the policy information in their repository during application deployment. The providers then use this information to make authorization decisions when called by the container.

JACC access decisions

When security is enabled and an enterprise bean or Web resource is accessed, the Enterprise JavaBeans (EJB) container or Web container calls the security runtime 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 Authorization 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 that is implemented by the provider to make the access decision.

The following sections describe how the provider is called for both the enterprise bean 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. Creates the `EJBMethodPermission` object using the bean name, method name, interface name, and the method signature.
2. Creates the context ID and sets it on the thread by using the `PolicyContext.setContextID(contextID)` method.
3. Registers the required policy context handlers, including the Subject policy context handler.
4. Creates the `ProtectionDomain` object with principal in the Subject. If no principal exists, 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` object.

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) file is accessed, the security runtime delegates the authorization decision to the JACC provider by using the following process:

1. A `WebResourcePermission` object is created to see if the URI is cleared. If the provider honors the Everyone subject it is also selected here.

- a. The `WebResourcePermission` object is constructed with the `urlPattern` and the HTTP method accessed.
 - b. A `ProtectionDomain` object with a null principal name is created.
 - c. The JACC provider `Policy.implies` method is called with the permission and the protection domain. If the URI access is cleared or given access to the `Everyone` subject, the provider permits access (return `true`) in the `implies` method. Access is then granted without further checks.
2. If access is not granted in the previous step, a `WebUserDataPermission` object is created and used to see if the Uniform Resource Identifier (URI) is precluded, excluded or must be redirected using the HTTPS protocol.
 - a. The `WebUserDataPermission` object is constructed with the `urlPattern` accessed, 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. A `ProtectionDomain` object with a null principal name is created.
 - c. The JACC provider `Policy.implies` method is called with the permission and the protection domain. If the request is using the HTTPS protocol and the `implies` method returns `false`, the HTTP 403 error is returned to imply excluded and precluded permission. In this case 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, LTPA token), 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 is granted access privileges to the URI.
 - a. As in Step 1, the `WebResourcePermission` object is created. The `ProtectionDomain` object 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 `implies` method is called using the `Permission` object and the `ProtectionDomain` object created previously. If the user is granted permission to access the resource, the `implies` method returns `true`. If the user is not granted access, the `implies` method returns `false`.

Even if the order listed previously is changed later (for example, to improve performance) the end result is 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 `WSCredential` credential. 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 used to make the access decision.

If the provider adds the required information to the Subject, WebSphere Application Server can use the information to make the access decision. The provider might add the information by using the Trust Association Interface feature or by plugging login modules into the Application Server.

The security attribute propagation section contains additional documentation on how to add the WebSphere Application Server required information to the Subject. For more information, see "Propagating security attributes among application servers" on page 267.

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 started as appropriate. The other existing modules in the application are not impacted, and the application itself is not stopped and then restarted.

When using the default authorization engine, any security policies are modified in the Web modules and the application is stopped and then restarted. When using the Java Authorization 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 it supports the dynamic updates by configuring the **Supports dynamic module updates** option in the JACC configuration model (see “Authorizing access to J2EE resources using Tivoli Access Manager” on page 350 for more information). This option can be enabled or disabled using the administrative console or by scripting. It is expected that most providers store the policy information in their external repository, which makes it possible for them to support these dynamic updates. This option should be 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 334.

Initialization of the JACC provider

If a Java Authorization Contract for Containers (JACC) provider requires initialization during server startup, for example, to enable the client code to communicate to the server code, the provider can implement the `com.ibm.wsspi.security.authorization.InitializeJACCProvider` interface. See “Interfaces that support JACC” on page 366 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 is used only if the provider requires initialization during server startup.

Mixed node environment and JACC

Authorization using Java Authorization Contract for Containers (JACC) is a new feature in WebSphere Application Server Version 6.0.x. 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 must contain Version 6.0.x and later nodes only. This restriction implies that a mixed node environment containing a set of Version 5.x nodes in a Version 6.0.x or later cell is not supported.

JACC providers

The Java Authorization Contract for Containers (JACC) is a new specification that is introduced in Java 2 Platform, Enterprise Edition (J2EE) Version 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.

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, which are all required by the JACC specification.

The JACC specification does not specify how to handle the authorization table information between the container and the provider. It is the responsibility of the provider to provide some management facilities to handle this information. The container is not required to provide the authorization table information in the binding file to the provider.

WebSphere Application Server provides the RoleConfigurationFactory and the RoleConfiguration role configuration interfaces 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 that support JACC” on page 366 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. 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 Application Server Network Deployment (ND) package.

The JACC provider is not the default authorization. You must configure WebSphere Application Server to use the JACC provider.

JACC policy context handlers

WebSphere Application Server supports all of the policy context handlers that are required by the Java Authorization 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 arguments of each EJB method.

If the provider supports and requires this context handler, select 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 option are effective after the servers are restarted. By default this option is disabled. Disable this option when using Tivoli Access Manager as the JACC provider, because 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 propagated to the JACC provider using the JACC specification APIs.

WebSphere Application Server makes the contextID unique by using the href:cellName/appName/moduleName string as the contextID format for the modules. The href part of the string indicates that a hierarchical name is passed as the context ID.

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 context ID for the modules in the application contain base1 (not cell1) as the cell name because the application is initially installed in base1.

The `appName` part of the string in the context ID represents the application name containing the module. The `moduleName` refers to the name of the module.

As an example, the context ID for the module `Increment.jar` file in an application named `DefaultApplication` that is installed in `cell1` is the `href:cell1/DefaultApplication/Increment.jar` file.

JACC policy propagation

When an application is installed or deployed in WebSphere Application Server, the security policy information in the application is propagated to the provider when the configuration is saved. The context ID for the application is saved in its `application.xml` file, that is used for propagating the policy to the Java Authorization Contract for Containers (JACC) provider, and also for access decisions for Java 2 Platform, Enterprise Edition (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 implemented the `RoleConfiguration` interface, the security policy information that is propagated to the policy provider also contains the authorization table information. See “Interfaces that support JACC” on page 366 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.

After an application is installed, it can be updated without 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 is 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 the security information is not propagated to the provider during application updates, you can set the `com.ibm.websphere.security.jacc.propagateonappupdate` Java virtual machine (JVM) property to `false` in the deployment manager, in a Network Deployment environment, or the unmanaged base application server. If this property is set to `false`, 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 enterprise archive (EAR) file is used to replace the existing one, and the provider is refreshed with the entire application security policy information.

As mentioned earlier, the security policy information is propagated to the JACC provider during the save operation. The `SystemOut.log` file indicates the success or failure of the propagation to the provider. Check the log file after the installation to ensure that the propagation had no problems. If the propagation had any problems, access to the application fails when Tivoli Access Manager is used as the JACC provider.

If the security policy information for the application is successfully propagated to the provider, the audit statements with the message key `SECJ0415I` appear. However, if there was a problem propagating the security policy information to the provider (for example: network problems, JACC provider is not available), the `SystemOut.log` files contain the error message with the message keys `SECJ0396E` during install or `SECJ0398E` during modification. The installation of the application is not stopped due to a failure to propagate the security policy to the JACC provider. Also, in the case of failure, no exception or error messages appear during the save operation. When the problem causing this failure is fixed, run the **propagatePolicyToJaccProvider** tool to propagate the security policy information to the provider without

reinstalling the application. For more information, see “Propagating security policy of installed applications to a JACC provider using wsadmin scripting” on page 646.

JACC registration of the provider implementation classes

The JACC specification states that providers can plug in their provider using the `javax.security.jacc.policy.provider` and the `javax.security.jacc.PolicyConfigurationFactory.provider` system properties.

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 `PolicyConfigurationFactory` implementation.

Although both system properties are supported in WebSphere Application Server, it is highly recommended that you use the configuration model that is provided. You can set these values using either the JACC configuration panel (see “Authorizing access to J2EE resources using Tivoli Access Manager” on page 350 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.

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.

Roles	Methods		
	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 Server 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 that is 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 Application Server 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.

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 JACC provider of Tivoli Access Manager . This policy information contains permission-to-role mappings and role-to-principal and role-to-group mapping information.
2. The JACC provider of Tivoli Access Manager 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 that is granted to this object is attached as an extended attribute.

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.

Tivoli Access Manager 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 are 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 application programming interfaces (API).

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 that support JACC” on page 366 for more information about these interfaces.

Tivoli Access Manager client configuration

To configure the Tivoli Access Manager client, you can use either the administrative console or wsadmin scripting. You can access the administrative console panels for the Tivoli Access Manager client configuration by clicking **Security > Secure administration, applications, and infrastructure > External authorization providers**. Under Related Items, click **External JACC provider**. The Tivoli client must be set up to use the Tivoli Access Manager JACC Provider.

For more information about how to configure the Tivoli Access Manager client, see “Tivoli Access Manager JACC provider configuration” on page 357.

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.

Note: **V6.0.x** SWAM is deprecated in WebSphere Application Server Version 6.1 and will be removed in a future release.

Tivoli Access Manager security for WebSphere Application Server

WebSphere Application Server provides embedded IBM Tivoli Access Manager client technology to secure your WebSphere Application Server-managed resources.

The benefits of using Tivoli Access Manager that are described here are only applicable when Tivoli Access Manager client code is used with the Tivoli Access Manager 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 administrative console for Tivoli Access Manager Web Portal Manager
- No requirements for coding or deployment changes to applications
- Easy management of users, groups, and roles using the WebSphere Application Server administrative console

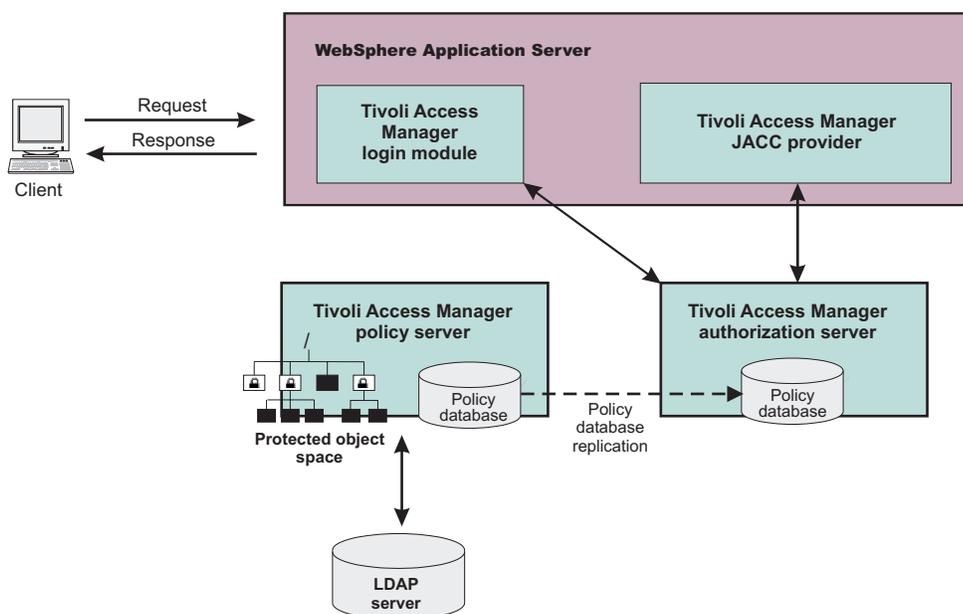
WebSphere Application Server supports the Java Authorization Contract for Containers (JACC) specification. JACC details the contract requirements for J2EE containers and authorization providers. With this contract, authorization providers can perform the access decisions for resources in J2EE Version 1.4 application servers such as WebSphere Application Server. The Tivoli Access Manager security utility that is embedded within WebSphere Application Server is JACC-compliant and is used to:

- Add security policy information when applications are deployed
- Authorize access to WebSphere Application Server-secured resources.

When applications are deployed, the embedded Tivoli Access Manager 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 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 that is 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 the information that is necessary to make deployment decisions, by reviewing the product documentation. The following guides are available at the IBM Tivoli Access Manager for e-business information center:

- *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.

To access this guide in the IBM Tivoli Access Manager for e-business information center, click **Access Manager for e-business > Base Information > Base Installation Guide**.

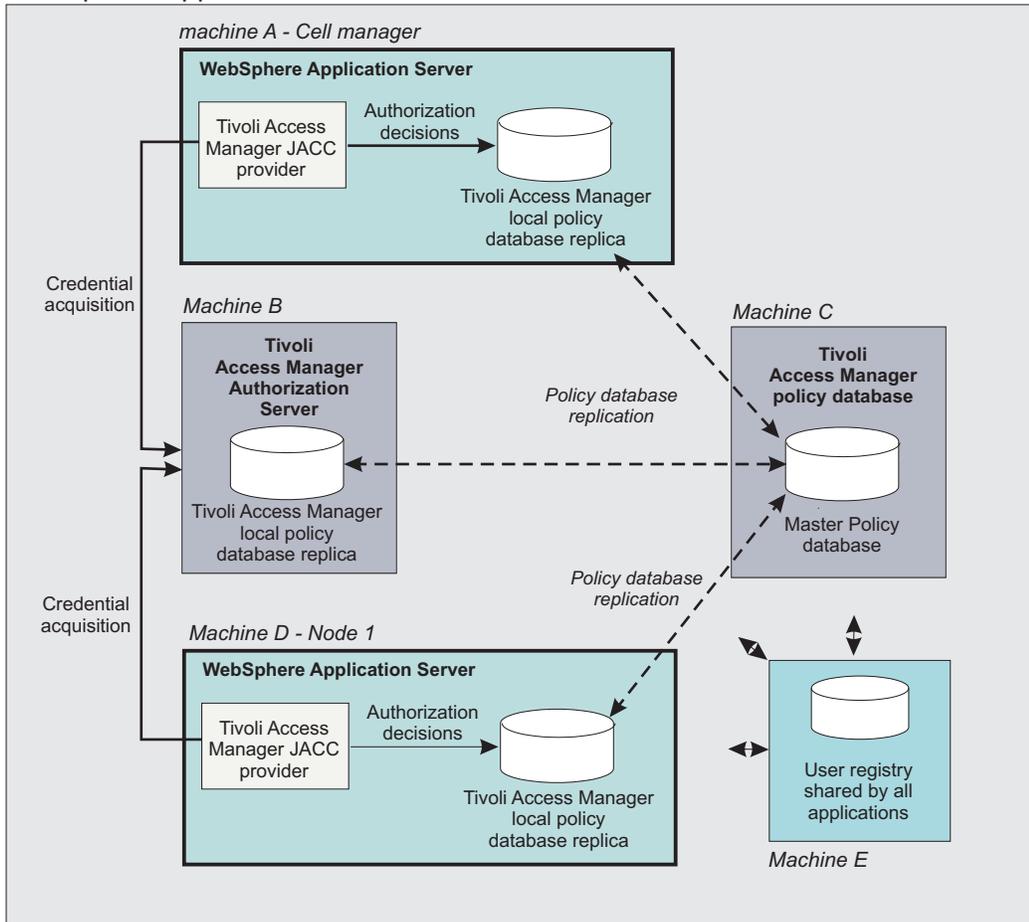
- *IBM Tivoli Access Manager Base Administration Guide*

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.

To access this guide in the IBM Tivoli Access Manager for e-business information center, click **Access Manager for e-business > Base Information > Base Administration Guide**.

Tivoli Access Manager provides centralized administration of multiple servers.

WebSphere Application Server Cell



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. The master policy database is installed as part of the Tivoli Access Manager installation. Having policy database replicas on each participating WebSphere Application Server node optimizes performance when making authorization decisions and provides failover capability.

Although the authorization server can also be installed on the same system as WebSphere Application Server, 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.

It is possible to have separate WebSphere Application Server profiles on the same host that is configured for different Tivoli Access Manager servers. Such an architecture requires that the profiles are configured for separate Java Runtime Environments (JRE) and therefore you need multiple JREs installed on the same host.

Delegations

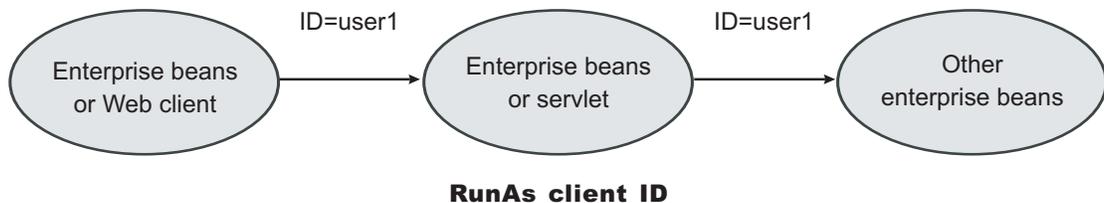
Delegation is a process security identity propagation from a caller to a called object. As per the Java 2 Platform, Enterprise Edition (J2EE) specification, a servlet and enterprise beans can propagate either the client or remote user identity when invoking enterprise beans, or they can use another specified identity as indicated in the corresponding deployment descriptor.

The extension supports enterprise bean propagation to the server ID when invoking other entity beans.

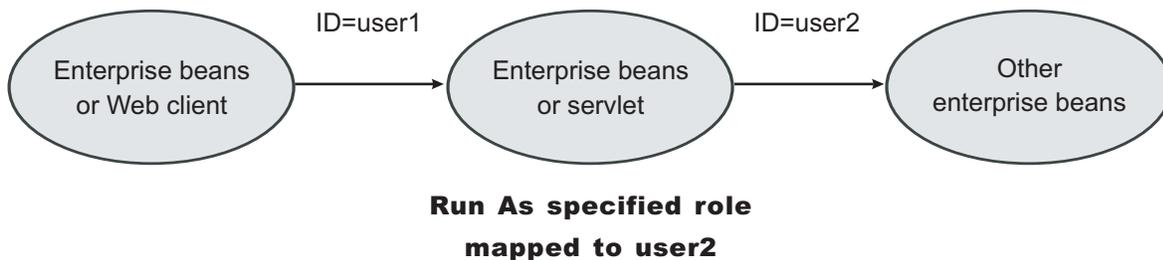
Three types of delegations are possible:

- Delegate (RunAs) client identity
- Delegate (RunAs) specified identity
- Delegate (RunAs) system identity

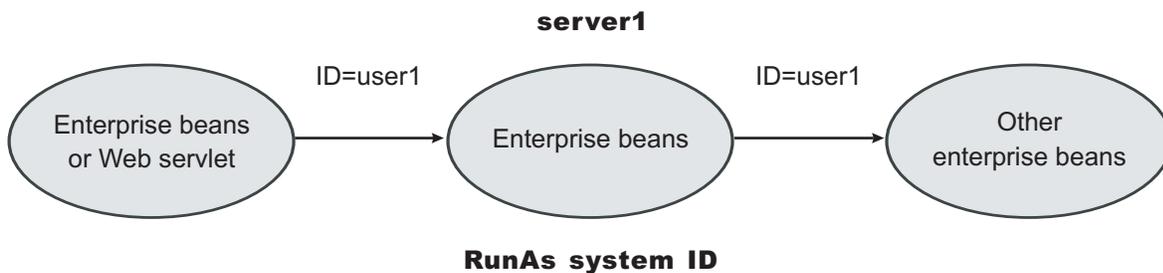
Delegate (RunAs) client identity



Delegate (RunAs) specified identity



Delegate (RunAs) system identity



Note: The RunAs system identity delegation only works when server ID and password are used. When the internalServerId feature is used, it does not work because runAs with system identity is not supported. You must specify RunAs roles. When internalServerID is used, use the RunAsSpecified

with a user ID and password that is mapped to the administrator role. See “Administrative roles and naming service authorization” on page 320 for more information about internalServerId.

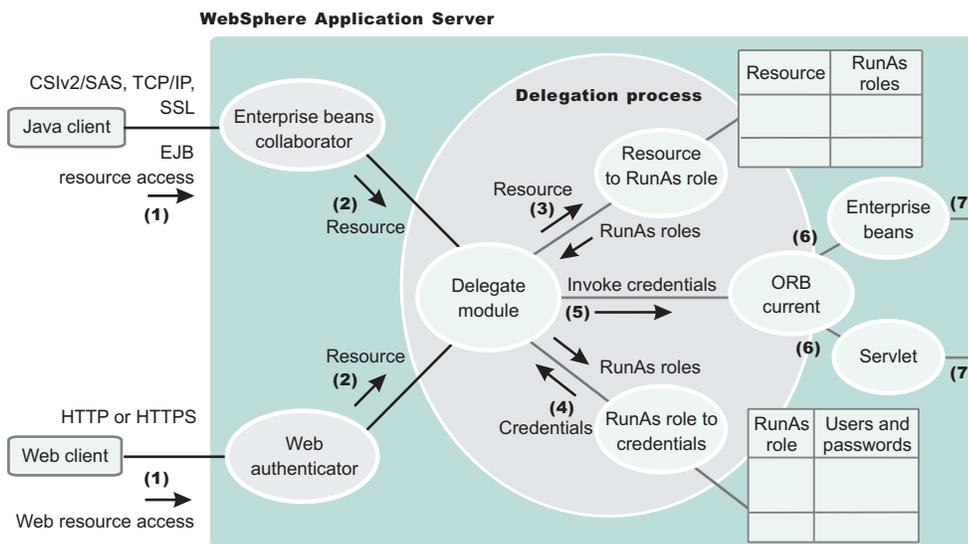
The EJB specification only supports delegation (RunAs) at the Enterprise JavaBeans (EJB) level. But an extension allows EJB method-level RunAs specification. With an EJB method level, the RunAs specification you can specify a different RunAs role for different methods within the same enterprise beans.

The RunAs specification is detailed in the deployment descriptor, which is the `ejb-jar.xml` file in the EJB module and the `web.xml` file in the Web module. The extension to the RunAs specification is included in the `ibm-ejb-jar-ext.xml` file.

An IBM-specific binding file is available for each application that contains a mapping from the RunAs role to the user. This file is specified in the `ibm-application-bnd.xml` file.

These specifications are read by the runtime during application startup. The following figure illustrates the delegation mechanism, as implemented in the WebSphere Application Server security model.

Delegation



Delegation Process

Two tables 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 the figure.

The resulting credential is stored in the Object Request Broker (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).

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. The work of the programmer can be broadly classified 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 that is 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 administrative 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 1.4 Specification. JAAS is a collection of strategic authentication application programming interfaces (API) that replace the Common Object Request Broker Architecture (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 running 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 551.

- For the pure Java client application or the client container application, make sure that the host name and the port number of the target Java Naming and Directory Interface (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 programmatic logins for Java Authentication and Authorization Service” on page 552 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 525 to the application, “Using PolicyTool to edit policy files” on page 512 and “Configuring the `was.policy` file” on page 520 sections for details. For more details of which APIs are protected by Java 2 Security permissions, check the IBM Developer Kit, Java Technology Edition; JAAS and the WebSphere Application Server public APIs documentation for more details. The following list contains the APIs that are 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 JAAS Version 1.0, `javax.security.auth.Subject.getSubject` does not return the Subject associated with the running thread inside a `java.security.AccessController.doPrivileged` code block. This can present an inconsistent behavior that is problematic and causes an undesirable effort to work around. The `com.ibm.websphere.security.auth.WSSubject` API provides a workaround to associate the Subject to the running thread. The `com.ibm.websphere.security.auth.WSSubject` API extends the JAAS model to J2EE resources for authorization checks. The Subject that is associated with the running thread 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.
- Administrative console support for defining new JAAS login configuration: You can configure JAAS login configuration in the administrative console and store it in the WebSphere Application Server configuration API. Applications can define new JAAS login configuration in the administrative console and the data is persisted in the configuration repository that is stored with the WebSphere Application Server configuration API. However, WebSphere Application Server still supports the default JAAS login configuration format that is provided by the JAAS default implementation. If duplication login configurations are defined in both the WebSphere Application Server configuration API and the plain text file format, the login configuration in the WebSphere Application Server configuration API takes precedence. Advantages to define the login configuration in the WebSphere Application Server configuration API include:
 - Defining the JAAS login configuration using the administrative console.
 - Managing the JAAS login configuration centrally.
- JAAS login configurations for WebSphere Application Server: WebSphere Application Server provides JAAS login configurations for applications to perform programmatic authentication to the WebSphere Application Server security runtime. These JAAS login configurations for WebSphere Application Server perform authentication to the configured authentication mechanism, Simple WebSphere Authentication Mechanism (SWAM) or Lightweight Third-Party Authentication (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 the WebSphere Application Server security runtime to perform authorization checks on J2EE role-based protected resources.

Note: SWAM is deprecated in WebSphere Application Server Version 6.1 and will be removed in a future release.

Here are the JAAS login configurations that are 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 that is based on a user ID and password, or a token to the WebSphere Application Server security runtime. However, this configuration does not support the CallbackHandler handler that is specified in the client container deployment descriptor.
- *ClientContainer JAAS login configuration:* This JAAS login configuration recognizes the CallbackHandler handler that is specified in the client container deployment descriptor. The login module of this login configuration uses the CallbackHandler handler in the client container deployment descriptor if one is specified, even if the application code specified one CallbackHandler handler in the login context. This is for client container application.
- The Subjects that are authenticated with the previously mentioned JAAS login configurations contain a `com.ibm.websphere.security.auth.WSPincipal` principal and a `com.ibm.websphere.security.auth.WSCredential` credential. If the authenticated Subject is passed to the `com.ibm.websphere.security.auth.WSSubject.doAs` method or the other `doAs` methods, the WebSphere Application Server security runtime can perform authorization checks on J2EE resources, based on the Subject `com.ibm.websphere.security.auth.WSCredential` credential.
- **Customer-defined JAAS login configurations:** You can define other JAAS login configurations. See “Configuring programmatic logins for Java Authentication and Authorization Service” on page 552 for details. Use these login configurations to perform programmatic authentication to the custom authentication mechanism. However, the subjects from these customer-defined JAAS login configurations might not be used by the WebSphere Application Server security runtime 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` exception 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` class. This exception has a `getCause` method with which you can pull out the exception that was wrapped after issuing the previous command.

You are not always guaranteed to get a `WSLoginFailedException` exception, but most of the exceptions that are generated from the user registry display here. The following example illustrates a `LoginContext.login` API with the associated catch block. Cast the `WSLoginFailedException` exception to `com.ibm.websphere.security.auth.WSLoginFailedException` class if you want to issue the `getCause` API.

The following `determineCause` example 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; go back and 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 exception is useful because it shows the user what happened. You can issue the following API to obtain the root cause exception:

```

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

```

When you have the exception, you can run it through the previous `determineCause` example 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 > Secure administration, applications, and infrastructure > 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 is 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();

    // maybe 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 Pages (JSP) files, servlet, or other Java 2 Platform, Enterprise Edition (J2EE) resources. See “Example: Programmatic logins” on page 551 for more information about Object Request Broker (ORB) security initialization requirements in a pure Java client.

Note: The `WSCallbackHandlerImpl` callback handler is different depending on whether you use WebSphere Application Server security or Web services security. It is located in the `sas.jar` file for security, and in the `was-wssecurity.jar` file for Web services security.

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 a user through user interface login prompts. The `com.ibm.websphere.security.auth.callback.WSGUICallbackHandlerImpl` callback handler 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 data 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();

// maybe 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 files, and so on. The user interface login prompt blocks the server for user input. This behavior is not good for a server process.

Stdin prompt programmatic login

WebSphere Application Server also provides a stdin implementation of the `javax.security.auth.callback.CallbackHandler` interface. The callback handler, `com.ibm.websphere.security.auth.callback.WSStdinCallbackHandlerImpl`, prompts and collects authentication data from a user through the stdin prompt.

```

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 determines how authentication data is collected
// in this case, the authentication data is collected by stdin prompt
// and passed 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();

// maybe 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
}

```

Attention: 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 good for a server process.

Authorizing access to J2EE resources using Tivoli Access Manager

The Java Authorization Contract for Containers (JACC) defines a contract between Java 2 Platform, Enterprise Edition (J2EE) containers and authorization providers. You can use the default authorization or an external JACC authorization provider. When security is enabled in WebSphere Application Server, the default authorization is used unless a JACC provider is specified.

JACC enables any third-party authorization providers to plug into a J2EE application server (such as WebSphere Application Server) to make the authorization decisions when a J2EE resource is accessed. By default, WebSphere Application Server implements the JACC provider by using Tivoli Access Manager as the external authorization provider.

Read the following articles for more detailed information about JACC before you attempt to configure WebSphere Application Server to use a JACC provider:

- “JACC support in WebSphere Application Server” on page 330
- “JACC providers” on page 332
- “Tivoli Access Manager integration as the JACC provider” on page 336

Using the default authorization provider

You can extend the capabilities of WebSphere Application Server by plugging in your own authorization provider. You can use the default authorization or an external JACC authorization provider.

For an explanation of the administrative console panels that support these capabilities, see:

- Use the default authorization provider. It is recommended that you do not modify any settings on the authorization provider panels if you use the **Default authorization** option. For more information, see “External authorization provider settings” on page 351.
- Use an external authorization provider. If you use the **External authorization using a JACC provider** option, the external providers must be based on the Java™ Authorization Contract for Containers (JACC) specification to handle the Java 2 Platform, Enterprise Edition (J2EE) authorization. By default, WebSphere Application Server enables you to configure the Tivoli Access Manager Java Authorization Contract for Containers (JACC) provider as the default external JACC provider. For more information, see “External Java Authorization Contract for Containers provider settings” on page 351 and “Tivoli Access Manager JACC provider settings” on page 357.

External authorization provider settings

Use this page to enable a Java Authorization Contract for Containers (JACC) provider for authorization decisions.

To view this administrative console page, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Click **External authorization providers**.

The application server provides a default authorization engine that performs all of the authorization decisions. In addition, the 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 the 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 use with the application server, do not enable **External authorization using a JACC provider**.

Default authorization:

Use this option all the time unless you want an external security provider such as the Tivoli Access Manager to perform the authorization decision for J2EE applications that are based on the JACC specification.

Default: Enabled

External JACC provider: Use this link to configure the 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 that are 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.

External Java Authorization Contract for Containers provider settings

Use this page to configure the 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.

Use these settings when you have set up an external security provider that supports the JACC specification to work with the application server. The configuration 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, these settings will be ignored.

To view this administrative console page, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Click **External 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 the application server. Use the Tivoli Access

Manager properties link under Additional properties, and 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 that is 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 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. For information on enabling the JACC provider using this field, see the "Enabling the JACC provider for Tivoli Access Manager" article in the information center.

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 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 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. For information on enabling the JACC provider using this field, see the "Enabling the JACC provider for Tivoli Access Manager" article in the information center.

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 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 that are 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. For information on enabling the JACC provider using this field, see the "Enabling the JACC provider for Tivoli Access Manager" article in the information center.

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` handler 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 that are required by the provider.

These properties are propagated to the provider during the startup 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.

The Tivoli Access Manager implementation does not require that you enter any properties in this link.

Tivoli Access Manager properties:

Specifies properties that are 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.

Enabling an external JACC provider

Use this topic to enable an external JACC provider using the administrative console.

The Java Authorization 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.

1. From the WebSphere Application Server administrative console, click **Security > Secure administration, applications, and infrastructure > External authorization providers**.
2. Under Related items, click **External JACC provider**.
3. The fields are set for Tivoli Access Manager by default. If you do not plan to use Tivoli Access Manager as the JACC provider, replace these fields with the details for your own external JACC provider.
4. If any custom properties are required by the JACC provider, click **Custom properties** under Additional properties and enter the properties. When using the Tivoli Access Manager, use the **Tivoli Access Manager properties** link instead of the Custom properties link. For more information, see “Configuring the JACC provider for Tivoli Access Manager using the administrative console” on page 355.
5. On the External authorization providers panel, select the **External authorization using a JACC provider** option and click **OK**.
6. Complete the remaining steps to enable security. If you are using Tivoli Access Manager, you must select LDAP as the user registry and use the same LDAP server. For more information on configuring LDAP registries, see “Configuring Lightweight Directory Access Protocol user registries” on page 93.
7. In a multinode environment, stop and start the deployment manager configuration.

Issue the following commands:

```
profile_root/bin/stopManager.bat  
-username user_name  
-password password
```

```
profile_root/bin/startManager.bat
```

8. Restart all servers to make these changes effective.

Configuring the JACC provider for Tivoli Access Manager using the administrative console

Use this task to configure Tivoli Access Manager as the Java Authorization Contract for Containers (JACC) provider using the administrative console.

Prior to completing the following steps, verify that you have previously created a security administrative user. For more information, see “Creating the security administrative user” on page 356.

The following configuration is performed on the management server. When you click either **Apply** or **OK**, configuration information is checked for consistency, saved, and applied if successful.

To configure Tivoli Access Manager as the JACC provider using the administrative console, complete the following steps:

1. Start the WebSphere Application Server administrative console by clicking `http://yourhost.domain:port_number/ibm/console` after starting 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 ID is typically the server user ID that is specified when you configure the user registry.
2. Click **Security > Secure administration, applications, and infrastructure > External 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:

Enable embedded Tivoli Access Manager

Select this option to enable 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 port 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. 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. 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 `policy_server:port` form. 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 `auth_server:port:priority` form. The authorization server communication port is set at the time of the Tivoli Access Manager configuration, and the default is 7136. 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 Application Server, for example, `o=ibm, 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 is not 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 Application Server security administrator ID (for example, `cn=wasadmin, o=organization, c=country`). The ID name must match the Server user ID on the Lightweight Directory Access Protocol (LDAP) User Registry panel in the administrative console. To access the LDAP User Registry panel, click **Security > Secure administration, applications, and infrastructure**. Under **User account repository**, choose **Standalone LDAP registry** as the available realm definition. Then click **Configure**.

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:

- Validates the configuration parameters.
- Configures 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.

Creating the security administrative user:

Enabling security requires the creation of a WebSphere Application Server administrative user. Use the Tivoli Access Manager command-line `pdadmin` utility to create the Tivoli Access Manager administrative user for WebSphere Application Server. This utility is available on the policy server host machine.

Follow these steps 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. For more information, see “Tivoli Access Manager JACC provider configuration.”

Tivoli Access Manager JACC provider configuration:

You can configure the Java Authorization Contract for Containers (JACC) provider for Tivoli Access Manager to deliver authentication and authorization protection for your applications or for authentication only. Most deployments that use the JACC provider for Tivoli Access Manager to configure Tivoli Access Manager 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 `com.tivoli.pd.as.amwas.DisableAddAuthorizationTableEntry=true` property to the `amwas.amjacc.template.properties` file. The file is located in the `profile_root/config/cells/cell_name` directory.

After this property is set, perform the tasks for setting Tivoli Access Manager Security, as documented.

You can configure the JACC provider for Tivoli Access Manager using either the WebSphere Application Server administrative console or the **wsadmin** command-line utility.

- For details on configuring the JACC provider for Tivoli Access Manager using the administrative console, refer to “Configuring the JACC provider for Tivoli Access Manager using the administrative console” on page 355.
- 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” on page 648.

The JACC configuration files for Tivoli Access Manager that are common across multiple WebSphere Application Server profiles are created by default under the `java/jre` directory. When you install WebSphere Application Server, you are given permissions to read and write to the files in this directory.

This situation is not ideal because configuration of the JACC provider for Tivoli Access Manager fails in these situations. To avoid this situation, you can add the following property to the `profile_root/config/cells/cell_name/amwas.amjacc.template.properties` file: `com.tivoli.pd.as.jacc.CommonFileLocation=new location` where `new location` is a fully qualified directory name.

This property applies read and write permissions to the `java/jre` directory.

The **wsadmin** command is available to reconfigure the Java Authorization Contract for Containers (JACC) Tivoli Access Manager interface:

```
$AdminTask reconfigureTAM -interactive
```

This command effectively prompts you through the process of unconfiguring the interface and then reconfiguring it.

Tivoli Access Manager JACC provider settings:

Use this page to configure the Java Authorization Contract for Container (JACC) provider for Tivoli Access Manager.

Note: When a third-party authorization such as Tivoli Access Manager or SAF for z/OS is used, the information in the administrative console panel might not represent the data in the provider. Also, any changes to the panel might not be reflected in the provider automatically. Follow the provider's instructions to propagate any changes made to the provider.

To view the JACC provider settings for Tivoli Access Manager, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Authentication, click **External 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

Note: If you want to disable Tivoli Access Manager as the JACC provider, clear this option and also select **Default authorization**.

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 re-configuring 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.

The 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 you specify a range of ports, 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
9900:9999
```

Note: Each of the servants might need to open up a listener port.

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 the 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 is set at the time of Tivoli Access Manager configuration. The default is 7136. You can specify more than one authorization server by entering each server on a new line. Configuring more than one authorization server provides for failover. 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 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 that is 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 the 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 application server users and groups.

Specification of the Tivoli Access Manager domain is required because 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 fully distinguished name of the security administrator ID for the application server. For example, *cn=wasadmin,o=organization,c=country*

JACC provider configuration properties for Tivoli Access Manager:

The JACC provider configuration properties detailed below may require configuration.

The Java property files are created in the *profile_root/etc/tam* directory.

Two properties files might require configuration:

- `amwas.node_name_server_name.amjacc.properties` contains properties that are used by the JACC provider of Tivoli Access Manager.
- `amwas.node_name_server_name.pdjlog.properties` contains logging properties that are created from the `amwas.pdjlog.template.properties` file for the specific node and server combination at the time of configuration.

Use `amwas.node_name_server_name.amjacc.properties` file 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 `profile_root/etc/tam/amwas.node_name_server_name.amjacc.properties` file.

The `profile_root` directory is the value of the `profilePath` parameter at profile creation time.

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 do not need to change this property, although the opportunity exists to implement your own cache, if necessary.

Define static roles

```
com.tivoli.pd.as.cache.StaticRoleCache.Roles=Administrator,Operator,Monitor,Deployer
```

Defines the administration roles for WebSphere Application Server.

Tip: Enhance Application performance by adding the static roles: **CosNamingRead**, **CosNamingWrite**, **CosNamingCreate**, **CosNamingDelete**. These roles support 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 `profile_root/etc/tam/amwas.node_name_server_name.amjacc.properties` file.

The `profile_root` directory is the value of the `profilePath` parameter at profile creation time.

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 do not need to change this property, although the opportunity exists to implement your own cache, if 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 that is 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 that is returned from a unique Lightweight Directory Access Protocol 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 *profile_root/etc/tam/amwas.node_name_server_name.amjacc.properties* file.

The *profile_root* directory is the value of the *profilePath* parameter when the profile is created.

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:

Although it is very unlikely that you will need to change these properties, use this file to reference supported properties within the role-based policy framework.

The role-based policy framework parameters are located in the Java Authorization Contract for Containers (JACC) configuration file and in the authorization configuration file. These parameters 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 *amwas.node_server.authztable.properties* configuration file is generated from the authorization table. The configuration file is generated from the JACC provider, *amwas.node_name_server_name.amjacc.properties*. Both files are stored on the WebSphere Application Server *profile_root/etc/tam* directory. It is unlikely that you need to change these properties, but these properties 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) and 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 that is specified by the `com.tivoli.pd.as.rbpf.AMAction` property. The permission set in the `com.tivoli.pd.as.rbpf.AMAction` property 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 that is specified in the posroot property to separate other products in the protected object space. Embedded Tivoli Access Manager objects are found in the /WebAppServer/deployedResources 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. Make changes in the template properties file before any configuration actions are performed. Properties that are changed after configuration will cause access decisions to fail.

System-dependent configuration properties:

Do not change these system-dependent configuration properties. These properties are included in this article for reference only.

These properties are in the *app_server_root/etc/amwas.node_name_server_name.amjacc.properties* file.

The *profile_root* variable is the value of the profilePath parameter when the profile is created.

The supported arguments include:

com.tivoli.pd.as.rbpf.AmasSession.CfgURL=file:/\$WAS_HOME/profiles/profile_name/etc/tam/amwas.node_server.pdperm.properties

This entry is generated by the Java Authorization Contract for Containers (JACC) provider configuration. This argument specifies the location of the file that contains information about the JACC provider of Tivoli Access Manager. Do not change this entry or the properties in the *amwas.node_server.pdperm.properties* file.

com.tivoli.pd.as.rbpf.AmasSession.LoggingURL=file:/\$WAS_HOME/profiles/profile_name/etc/tam/amwas.node_server.pdlog.properties

This entry contains the location of the logging configuration file for the JACC provider of Tivoli Access Manager. The referenced file is generated by the JACC provider of Tivoli Access Manager configuration. Do not change this entry.

Administering security users and roles with Tivoli Access Manager

Use these steps to manage user-to-role mappings and user-to-group mappings for applications.

User-to-role mapping and user-to-group mapping for the JACC provider of Tivoli Access Manager are performed using the WebSphere Application Server administrative console.

1. Click **Applications > Enterprise applications > application_name**.
2. Under Additional properties, click **Security role to user/group mapping**. The user and groups management screen is displayed.
3. Select the role that 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. For more information, see *csec_role_based_sec.dita* and *tsec_use_TAM_groups.dita*.

Configuring Tivoli Access Manager groups

Use these steps to configure the WebSphere Application Server administrative console to add objects of the accessGroup class to the list of object classes that represent user registry groups.

You can use the WebSphere Application Server administrative console to specify security policies for applications that run in the WebSphere Application Server environment. You can also use the WebSphere Application Server administrative console to specify security policies for other Web resources, based on the entities that are stored in the user registry.

Tivoli Access Manager adds the accessGroup object class to the registry. Tivoli Access Manager administrators can use the pdadmin utility, which is 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 > Secure administration, applications, and infrastructure**.
2. Under User account repository, click the **Available realm definitions** drop-down list, select **Standalone LDAP registry**, and click **Configure**.
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 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 looks like the following example:

```
groupOfNames:member;groupOfUniqueNames:uniqueMember;
accessGroup:member
```

6. Stop and restart WebSphere Application Server.

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 in the IBM Tivoli Access Manager for e-business information center.
2. Re-configure the Java Authorization Contract for Containers (JACC) provider using the \$AdminTask reconfigureTAM interactive **wsadmin** command. Enter all new and existing options.

Logging Tivoli Access Manager security

Use this topic to enable the trace specification to indicate tracing at the required level.

The Java Authorization Contract for Containers (JACC) for Tivoli Access Manager provider messages are logged to the configured trace output location, and messages are written to standard out SystemOut.log file. When trace is enabled, all logging, both trace and messaging, is sent to the trace.log file.

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 JACC provider for Tivoli Access Manager, set the following line to *true*:
`amwas.node_server.pdjlog.properties:baseGroup.AMWASWebTraceLogger.isLogging=true`
2. Enable tracing for the JACC provider of Tivoli Access Manager 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** option.
 - d. Click **Apply**.
 - e. Click **Troubleshooting > Logs and Trace > server_name**.
 - f. Click **Change Log Detail Levels**.
 - g. Click **Components**. Tracing for all components can be enabled using the **com.tivoli.pd.as.*** command. Tracing for separate components can be enabled using the following commands:
 - **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

For more information, see `utrb_loglevel.dita`.
 - h. Click **Apply**.

The trace specification now indicates that tracing is enabled at the required level. Save the configuration and restart the server for the changes to take effect.

Tivoli Access Manager loggers:

The Java Authorization Contract for Containers (JACC) for Tivoli Access Manager provider messages are logged to the configured trace output location, and messages are written to standard out `SystemOut.log` file. When trace is enabled, all logging, both trace and messaging, is sent to the `trace.log` file.

The JACC provider for Tivoli Access Manager uses the JLog logging framework as does the Java runtime environment for Tivoli Access Manager. You can enable tracing and messaging selectively for specific JACC provider for Tivoli Access Manager components.

Tracing and message logging for the JACC provider for Tivoli Access Manager are configured in the `amwas.node_server.pdjlog.properties` properties file, which is located in the `profile_root/etc/tam` directory. This file contains logging properties from the `amwas.pdjlog.template.properties` template file for the specific node and server combination at the time of JACC provider for Tivoli Access Manager configuration.

The contents of this file let the user control:

- Whether tracing is enabled or disabled for the JACC provider of Tivoli Access Manager components.
- Whether message logging is enabled or disabled for the JACC provider of Tivoli Access Manager components.

The `amwas.node_server.pdjlog.properties` file defines several loggers, each of which is associated with one JACC provider of Tivoli Access Manager component. These loggers include:

Logger Name	Description
AmasRBPFTTraceLogger AmasRBPFTMessageLogger	Logs messages and trace for the role-based policy framework. This underlying framework is used by embedded Tivoli Access Manager to make access decisions.
AmasCacheTraceLogger AmasCacheMessageLogger	Logs messages and trace for the policy caches that are used by the role-based policy framework.
AMWASWebTraceLogger AMWASWebMessageLogger	Logs messages and trace for the WebSphere Application Server authorization plug-in.
AMWASConfigTraceLogger AMWASConfigMessageLogger	Logs messages and trace for the configuration actions of the JACC provider for Tivoli Access Manager .
JACCTraceLogger JACCMessageLogger	Logs messages and trace for the JACC provider activity of Tivoli Access Manager .

Note: Tracing can have a significant impact on system performance. Enable tracing only 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 trace.log file.

For each logger, the `amwas.node_server.pdolog.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.

The `amwas.node_server.pdolog.properties` file defines the parent loggers `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 JACC provider for Tivoli Access Manager is enabled, the `isLogging` attribute is set to `true` for the `MessageLogger` and set to `false` for the `TraceLogger` logger. Message logging is enabled for all components and tracing is disabled for all components, by default.

To turn on tracing for a JACC provider component, see [Logging Tivoli Access Manager security](#) .

Interfaces that support JACC

WebSphere Application Server provides the `RoleConfigurationFactory` and the `RoleConfiguration` interfaces, which are similar to `PolicyConfigurationFactory` and `PolicyConfiguration` interfaces so the information that is stored in the bindings file can be propagated to the provider during installation. The implementation of these interfaces is optional.

RoleConfiguration interface

Use the `RoleConfiguration` interface to propagate the authorization information to the provider. This interface is similar to the `PolicyConfiguration` interface that is 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 installation. 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 does not 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 does not 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 does not 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 does not 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 interface

The RoleConfigurationFactory interface is similar to the PolicyConfigurationFactory interface that is introduced by JACC, and is used to obtain RoleConfiguration objects based on the contextID IDs.

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 does not 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 does not 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
 * cannot be obtained.
 */
public abstract com.ibm.ws.security.policy.RoleConfiguration
    getRoleConfiguration(String contextID, boolean remove)
    throws RoleConfigurationException

```

InitializeJACCProvider provider

When implemented by the provider, this interface is called by every process where the JACC provider can be used for authorization. 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 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 the interface, 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()
```

Enabling the JACC provider for Tivoli Access Manager

The Java Authorization Contract for Container (JACC) provider for Tivoli Access Manager is configured by default. Use this topic to enable the JACC provider for Tivoli Access Manager.

Restriction: Do not perform this task if you are configuring the 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

Field	Value
Requires the EJB arguments policy context handler for access decisions	false
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 > Secure administration, applications, and infrastructure > External authorization providers**.
2. Select the **External authorization using a JACC provider** option, then click **Apply**.
3. Under Related Items, click **External JACC provider**. The JACC provider settings for Tivoli Access Manager are displayed.
4. Verify that the correct settings are present to work with your Tivoli Access Manager configuration. For more information, see “External Java Authorization Contract for Containers provider settings” on page 351.
5. Under Additional properties, click **Tivoli Access Manager properties**.
6. Click the **Enable embedded Tivoli Access Manager** option and verify that the correct Tivoli Access Manager server and WebSphere Application Server settings exist. For more information, see “Tivoli Access Manager JACC provider settings” on page 357.
7. Click **OK**.
8. Save the settings by clicking **Save** at the top of the page.
9. Log out of the WebSphere Application Server administrative console.
10. Restart 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.

Enabling embedded Tivoli Access Manager

Embedded Tivoli Access Manager is not enabled by default, and you need to configure it 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 user registry contains the user and group information for both Tivoli Access Manager and WebSphere Application Server.
- Tivoli Access Manager server exists and is configured to use the user registry. For details on the installation and configuration of Tivoli Access Manager, refer to the IBM Tivoli Access Manager for e-business information center.

Note: WebSphere Application Server contains an embedded client for Tivoli Access Manager. To use Tivoli Access Manager, you must also configure the Tivoli Access Manager server.

However, the server version must be the same version or later as the client version. For information on the supported version of Tivoli Access Manager, see WebSphere Application Server - Supported Prerequisites.

- WebSphere Application Server is installed either in a single server model or as WebSphere Application Server Network Deployment.
- When administrative security is configured with a Federal Information Processing Standard (FIPS) provider, the Tivoli Access Manager server must be configured for FIPS as well

Complete the following steps to enable embedded Tivoli Access Manager security:

1. Create the security administrative user.

For more information, see the *Securing applications and their environment* PDF.

2. Configure the Java Authorization Contract for Containers (JACC) provider for Tivoli Access Manager .
For more information, see the *Securing applications and their environment* PDF.

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 the *Securing applications and their environment* PDF.

4. Enable the JACC provider for Tivoli Access Manager.

For more information, see the *Securing applications and their environment* PDF.

Disabling embedded Tivoli Access Manager client

To unconfigure Tivoli Access Manager Security in WebSphere Application Server, you can use either the wsadmin command-line utility or the WebSphere Application Server administrative console.

- For details on unconfiguring the embedded Tivoli Access Manager client using the WebSphere Application Server administrative console, refer to “Disabling embedded Tivoli Access Manager client using the administrative console.”
- For details on unconfiguring the embedded Tivoli Access Manager client using the wsadmin command line utility, refer to “Disabling embedded Tivoli Access Manager client using wsadmin” on page 649.

Disabling embedded Tivoli Access Manager client using the administrative console

To unconfigure the JACC provider for Tivoli Access Manager, you can use the WebSphere Application Server administrative console.

1. Click **Security > Secure administration, applications, and infrastructure > External authorization providers**.
2. Under Related items, click **External JACC provider**.
3. Under Additional properties, click **Tivoli Access Manager Properties**. The configuration screen for the JACC provider for Tivoli Access Manager is displayed.
4. Clear 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.
5. Click **OK**.
6. **Optional:** If you want security enabled without Tivoli Access Manager re-enable administrative security.
7. 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 JACC provider for Tivoli Access Manager 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 files. This utility effectively removes the JACC provider for Tivoli Access Manager.

1. Back up the security.xml and wsjaas.conf files.
2. Enter the following command as one continuous line.

```
app_server_root/java/jre/bin/java
-classpath "app_server_root /$WAS_HOME/plugin/com.ibm.ws.runtime_1.0.0.jar"
com.tivoli.pd.as.jacc.cfg.CleanSecXML
fully_qualified_path/security.xml fully_qualified_path/wsjaas.conf
```

Authorizing access to administrative roles

You can assign users and groups to administrative roles to identify users who can perform WebSphere Application Server administrative functions.

Administrative roles enable you to control access to WebSphere Application Server administrative functions. Refer to the descriptions of these roles in `rsec_adminroles.dita`.

Before you assign users to administrative roles, you must set up your user registry. For information on the supported registry types, see “Selecting a registry or repository” on page 87.

The following steps are needed to assign users to administrative roles.

In the administrative console, click **Users and Groups**. Click either **Administrative User Roles** or **Administrative Group Roles**.

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 **Special subject** menu, highlight **Administrator**, and click **OK**. If no validation error occurs, the specified group or special subject is displayed 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, click **Show filter function** on the User Roles or Group Roles panel. In the **Search term(s)** box, type a value, then click **Go**. For example, `user*` displays only users with the user prefix.
6. After the modifications are complete, click **Save** to save the mappings.
7. Restart the application server for changes to take effect.

After 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.

Administrative user roles settings and CORBA naming service user settings

Use the Administrative User Roles page to give users specific authority to administer application servers through 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, complete either of the following steps:

- Click **Security > Secure administration, applications, and infrastructure > Administrative User Roles**.
- Click **Users and Groups > Administrative User Roles**.

To view the CORBA naming service groups administrative console page, click **Environment > Naming > CORBA Naming Service Groups**.

Note: When a third-party authorization such as Tivoli Access Manager or Service Access Facility (SAF) for z/OS is used, the information in this panel might not represent the data in the provider. Also, any changes to this panel might not be reflected in the provider automatically. Follow the provider's instructions to propagate any changes made here to the provider.

User (Administrative user roles)

Specifies users.

The users that are entered must exist in the configured active user registry.

Data type: String

User (CORBA naming service users)

Specifies CORBA naming service users.

The users that are entered must exist in the configured active user registry.

Data type: String

Role (Administrative user roles)

Specifies user roles.

The following administrative roles provide different degrees of authority that are needed to perform certain application server administrative functions:

Administrator

The administrator role has operator permissions, configurator permissions, and the permission that is required to access sensitive data including server password, Lightweight Third Party Authentication (LTPA) password and keys, and so on.

Operator

The operator role has monitor permissions and can change the run-time state. For example, the operator can start or stop services.

Configurator

The configurator role has monitor permissions and can change the WebSphere Application Server configuration.

Monitor

The monitor role has the least permissions. This role primarily confines the user to viewing the application server configuration and current state.

adminsecuritymanager

The adminsecuritymanager role has privileges for managing users and groups from within the administrative console and determines who has access to modify users and groups using administrative role mapping. Only the adminsecuritymanager role can map users and groups to administrative roles, and by default, AdminId is granted to the adminsecuritymanager.

iscadmins

The iscadmins role has administrator privileges for managing users and groups from within the administrative console only.

Note: To manage users and groups, click **Users and Groups** in the console navigation tree. Click either **Manage Users** or **Manage Groups**.

Data type: String
Range: Administrator, Operator, Configurator, Monitor, and iscadmins

Note: Other arbitrary administrative roles might also be visible in the administrative console collection table. Other contributors to the console might create these additional roles, which can be used for applications that are deployed to the console.

Role (CORBA naming service users)

Specifies naming service user roles.

A number of naming roles are defined to provide degrees of authority that are needed to perform certain application server 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 roles now have authority levels from low to high:

CosNamingRead

You can query the application server name space by using, for example, the Java Naming and Directory Interface (JNDI) lookup method. The EVERYONE special-subject is the default policy for this role.

CosNamingWrite

You can perform write operations such as JNDI bind, rebind, or unbind, plus CosNamingRead operations.

CosNamingCreate

You can create new objects in the name space through operations such as JNDI createSubcontext and CosNamingWrite operations.

CosNamingDelete

You can destroy objects in the name space, for example using the JNDI destroySubcontext method and CosNamingCreate operations.

Data type:

String

Range:

CosNamingRead, CosNamingWrite, CosNamingCreate and CosNamingDelete

Login status (Administrative user roles)

Species whether the user is active or inactive.

Administrative group roles and CORBA naming service groups

Use the Administrative Group Roles page to give groups specific authority to administer application servers through tools such as the administrative console or wsadmin scripting. The authority requirements are only effective when administrative security is enabled. Use the Common Object Request Broker Architecture (CORBA) naming service groups page to manage CORBA Naming Service groups settings.

To view the Console Groups administrative console page, complete either of the following steps:

- Click **Security > Secure administration, applications, and infrastructure > Administrative Group Roles**.
- Click **Users and Groups > Administrative Group Roles**.

To view the CORBA naming service groups administrative console page, click **Environment > Naming > CORBA Naming Service Groups**.

Group (CORBA naming service groups)

Identifies CORBA naming service groups.

In previous releases of WebSphere Application Server, there were two default groups: ALL AUTHENTICATED and EVERYONE. However, EVERYONE is now the only default group, and it provides CosNamingRead privileges only.

Data type:

String

Range:

EVERYONE

Role (CORBA naming service groups)

Identifies naming service group roles.

A number of naming roles are defined to provide the degrees of authority that are needed to perform certain application server 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 roles have authority levels from low to high:

Cos Naming Read

You can query the application server name space using, for example, the Java Naming and Directory Interface (JNDI) lookup method. The EVERYONE special-subject is the default policy for this role.

Cos Naming Write

You can perform write operations such as JNDI bind, rebind, or unbind, and CosNamingRead operations. The ALL_AUTHENTICATED special-subject is the default policy for this role.

Cos Naming Create

You can create new objects in the name space through operations such as JNDI createSubcontext and CosNamingWrite operations. The ALL_AUTHENTICATED special-subject is the default policy for this role.

Cos Naming Delete

You can destroy objects in the name space, for example using the JNDI destroySubcontext method and CosNamingCreate operations. The ALL_AUTHENTICATED special-subject is the default policy for this role.

Data type:

String

Range:

CosNamingRead, CosNamingWrite, CosNamingCreate, and CosNamingDelete

Group (Administrative group roles)

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

Role (Administrative group roles)

Specifies user roles.

The following administrative roles provide different degrees of authority needed to perform certain application server administrative functions:

Administrator

The administrator role has operator permissions, configurator permissions, and the permission that is required to access sensitive data, including server password, Lightweight Third Party Authentication (LTPA) password and keys, and so on.

Operator

The operator role has monitor permissions and can change the run-time state. For example, the operator can start or stop services.

Configurator

The configurator role has monitor permissions and can change the application server configuration.

Monitor

The monitor role has the least permissions. This role primarily confines the user to viewing the application server configuration and current state.

iscadmins

The iscadmins role has administrator privileges for managing users and groups from within the administrative console only.

Note: To manage users and groups, click **Users and Groups** in the console navigation tree. Click either **Manage Users** or **Manage Groups**.

Data type:	String
Range:	Administrator, Operator, Configurator, Monitor, and iscadmins

Note: Other arbitrary administrative roles might also be visible in the administrative console collection table. Other contributors to the console might create these additional roles, which can be used for applications that are deployed to the console.

Assigning users to naming roles

Use this task to assign users to naming roles by using the administrative console.

The following steps are needed to assign users to naming roles. In the administrative console, click **Environment > Naming**, and click **CORBA Naming Service Users** or **CORBA Naming Service Groups**.

1. Click **Add** on the CORBA Naming Service Users or the 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 **EVERYONE**. 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 runtime.

Propagating administrative role changes to Tivoli Access Manager

These steps provide an example of how to migrate the admin-authz.xml file.

Additions and changes to console users and groups are not automatically added to the Tivoli Access Manager object space after the Java Authorization Contract for Containers (JACC) provider for Tivoli Access Manager is configured. Changes to console users and groups are saved in the admin-authz.xml

file and this file must be migrated before any changes take effect. The JACC provider for Tivoli Access Manager includes the **migrateEAR** migration utility 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 JACC provider for Tivoli Access Manager is configured. The utility does not need to run for changes and additions to console user and groups made prior to the configuration of the JACC provider for Tivoli Access Manager because the changes made to the `admin-authz.xml` and `naming-authz.xml` files are automatically migrated at configuration time. Furthermore, the migration tool does not need to run before deploying standard Java 2 Platform, Enterprise Edition (J2EE) applications; J2EE application policy deployment is also performed automatically.

For example, if you wanted to migrate the `admin-authz.xml` file, perform the following steps:

1. Set up the environment.

Before running the **migrateEAR** utility, set up the environment by running the `setupCmdLine.bat` or `setupCmdLine.sh` file that is located in the `app_server_root/bin` directory.

Make sure that the `WAS_HOME` environment variable is set to the WebSphere Application Server installation directory.

2. Change to the `app_server_root/bin` directory where the **migrateEAR** utility is located.

3. Run the **migrateEAR** utility to migrate the data contained in the `admin-authz.xml` file. Use the parameter descriptions that are listed in “The **migrateEAR** utility for Tivoli Access Manager.”

A status message is displayed when the migration completes. Output of the utility is logged to the `pdwas_migrate.log` file, which is created in 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.

4. WebSphere Application Server does not require a restart for the changes to take effect.

The **migrateEAR** utility for Tivoli Access Manager

The **migrateEAR** utility migrates changes made to console users and groups in the `admin-authz.xml` and `naming-authz.xml` files into the Tivoli Access Manager object space.

Syntax

```
migrateEAR
-j fully_qualified_filename
-c pdPerm.properties_file_location
-a Tivoli_Access_Manager_administrator_ID
-p Tivoli_Access_Manager_administrator_password
-w WebSphere_Application_Server_administrator_user_name
-d user_registry_domain_suffix
[-r root_objectspace_name]
[-t ssl_timeout]
```

Parameters

-a *Tivoli_Access_Manager_administrator_ID*

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 *PdPerm.properties_file_location*

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:



```
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*

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

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 *fully_qualified_pathname*

The fully qualified path and file name of the Java 2 Platform, Enterprise Edition application archive file `.admin-authz.xml` or the roles definitions file `naming-authz.xml` that is used for a naming operation authorization. Optionally, this path can also be a directory of an expanded enterprise application. For example, when WebSphere Application Server is installed in the default location, the path to the data files to migrate includes:

▶ Solaris ▶ Linux ▶ HP-UX

```
file:/opt/IBM/WebSphere/AppServer/profiles/profile_name/config/cells  
/cell_name/admin-authz.xml
```

▶ AIX

```
file:/usr/IBM/WebSphere/AppServer/profiles/profile_name/config/cells  
/cell_name/admin-authz.xml
```

▶ Windows

```
"C:/Program Files/IBM/WebSphere/AppServer/profiles/profile_name/config/cells  
/cell_name/admin-authz.xml"
```

-p *Tivoli_Access_Manager_administrator_password*

The password for the Tivoli Access Manager administrative user. The administrative user must have the privileges that are 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`.

When this parameter is not specified, the user is prompted to supply the password for the administrative user name.

-r *root_objectspace_name*

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.

The default value for the root object space is `WebAppServer`.

Set the Tivoli Access Manager root object space name by modifying the `amwas.amjacc.template.properties` file prior to configuring the Java Authorization Contract for Containers (JACC) provider for Tivoli Access Manager for the first time. Use this option if the default object space value is not used in the configuration of the Tivoli Access Manager JACC provider for Tivoli Access Manager.

Do not change the Tivoli Access Manager object space name after the Tivoli Access Manager JACC provider is configured.

-t *ssl_timeout*

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 the policy server before the default connection times out.

The default is 60 minutes. The minimum value 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.

If you are not familiar with the administration of this value, you can safely use the default value.

-w *WebSphere_Application_Server_administrator_user_name*

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 the security administrative user” on page 356. 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

You can overwrite the action group name. The default name is `WebAppServer`. 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 or enterprise archive files to Tivoli Access Manager for WebSphere Application Server. The script calls `com.tivoli.pdwas.migrate.Migrate` the Java class.

Before invoking the script you must run the **setupCmdLine.bat** or the **setupCmdLine.sh** commands. 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 that contains the native language support libraries that are provided with the JACC provider for Tivoli Access Manager. These libraries are located in a subdirectory under the JACC provider for Tivoli Access Manager 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.

 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.

To enable a new user access to the administrative group in WebSphere Application Server, it is recommended that the user be added to the pdwas-admin group after JACC has been enabled. You can enter the administrative primary ID (adminID) in the group. This is required when the serverID is not the same as the adminID.

The following is an example of this command:

```
pdadmin> group modify pdwas-admin add adminID
```

Return codes

The utility can return the following exit status codes:

- 0** The command completed successfully.
- 1** The command failed.

Chapter 7. Securing communications

WebSphere Application Server provides several methods to secure communication between a server and a client.

The following topics are covered in this section:

- Secure communications using Secure Sockets Layer
- Creating an SSL configuration
- Creating a keystore configuration
- Creating a self-signed certificate
- Creating a certificate authority request
- Extracting a signer certificate from a personal certificate
- Retrieving signers from a remote SSL port
- Adding a signer certificate to a keystore
- Exchanging signer certificates in a keystore
- Configuring certificate expiration monitoring
- Key management for cryptographic uses
- Creating a key set configuration
- Creating a key set group configuration

Secure communications using Secure Sockets Layer

The Secure Sockets Layer (SSL) protocol provides transport layer security including authenticity, data signing, and data encryption to ensure a secure connection between a client and server that uses WebSphere Application Server. The foundation technology for SSL is *public key cryptography*, which guarantees that when an entity encrypts data using its private key, only entities with the corresponding public key can decrypt that data.

WebSphere Application Server uses Java Secure Sockets Extension (JSSE) as the SSL implementation for secure connections. JSSE is part of the Java 2 Standard Edition (J2SE) specification and is included in the IBM implementation of the Java Runtime Extension (JRE). JSSE handles the handshake negotiation and protection capabilities that are provided by SSL to ensure secure connectivity exists across most protocols. JSSE relies on X.509 certificate-based asymmetric key pairs for secure connection protection and some data encryption. Key pairs effectively encrypt session-based secret keys that encrypt larger blocks of data. The SSL implementation manages the X.509 certificates.

Managing X.509 certificates

Secure communications for WebSphere Application Server require digitally-signed X.509 certificates. The contents of an X.509 certificate, such as its distinguished name and expiration, are either signed by a certificate authority (CA) or are self-signed. When a trusted CA signs an X.509 certificate, WebSphere Application Server identifies the certificate and freely distributes it. A certificate must be signed by a CA because the certificate represents the identity of an entity to the general public. Server-side ports that accept connections from the general public must use CA-signed certificates. Most clients or browsers already have the signer certificate that can validate the X.509 certificate so signer exchange is not necessary for a successful connection.

You can trust the identity of a self-signed X.509 certificate only when a peer in a controlled environment, such as internal network communications, accepts the signer certificate. To complete a trusted handshake,

you must first send a copy of the entity certificate to every peer that connects to the entity. Self-signed certificates are less expensive than CA-signed certificates because they do not require signer exchange for a secure connection.

CA and self-signed X.509 certificates reside in Java *keystores*. JSSE provides a reference to the keystore in which a certificate resides. You can select from many types of keystores, including Java Cryptographic Extension (JCE)-based and hardware-based keystores. Typically, each JSSE configuration has two Java keystore references: a keystore and a *truststore*. The keystore reference represents a Java keystore object that holds personal certificates. The truststore reference represents a Java keystore object that holds signer certificates.

A personal certificate without a private key is an X.509 certificate that represents the entity that owns it during a handshake. Personal certificates contain both public and private keys. A signer certificate is an X.509 certificate that represents a peer entity or itself. Signer certificates contain just the public key and verify the signature of the identity that is received during a peer-to-peer handshake.

For more information, see “Extracting a signer certificate from a personal certificate” on page 470

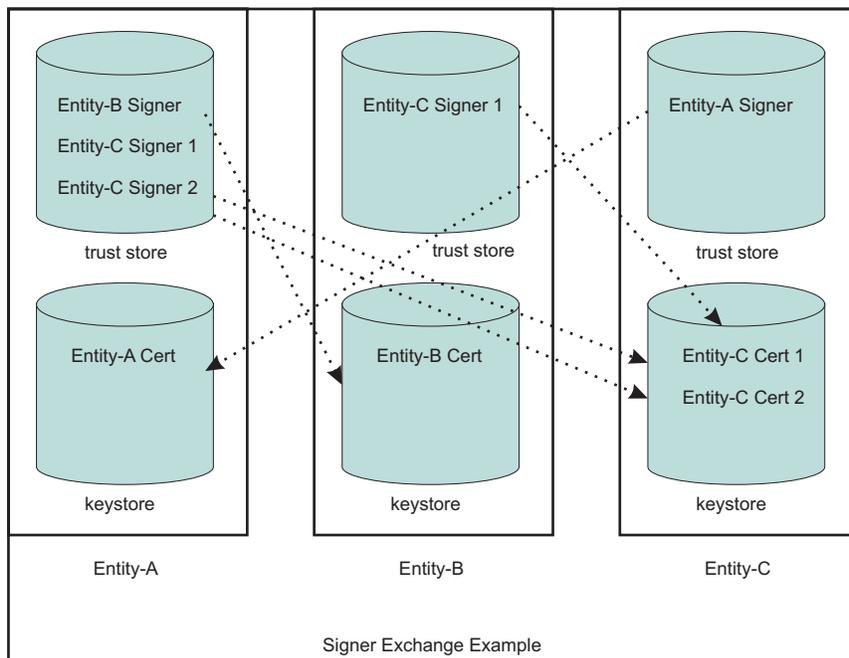
For more information about keystores, see `csec_sslkeystoreconfs.dita`.

Signer exchange

When you configure an SSL connection, you can exchange signers to establish trust in a *personal certificate* for a specific entity. Signer exchange enables you to extract the X.509 certificate from the peer keystore and add it into the truststore of another entity so that the two peer entities can connect. The signer certificate also can originate from a CA as a root signer certificate or an intermediate signer certificate. You can also extract a *signer certificate* directly from a *self-signed certificate*, which is the X.509 certificate with the public key.

Figure 1 illustrates a hypothetical keystore and truststore configuration. An SSL configuration determines which entities can connect to other entities, and the peer connections that are trusted by an SSL handshake. If you do not have the necessary signer certificate, the handshake fails because the peer cannot be trusted.

Figure 3. Signer exchange



In this example, the truststore for Entity A contains three signers. Entity A can connect to any peer as long as one of the three signers validates its personal certificate. For example, Entity A can connect to Entity B or Entity C because the signers can trust both signed personal certificates. The truststore for Entity-B contains one signer. Entity B is able to connect to Entity C only, and only when the peer endpoint is using certificate Entity-C Cert 1 as its identity. The ports that use the other personal certificate for Entity C are not trusted by Entity B. Entity C can connect to Entity A only.

In the example, the self-signed configuration seems to represent a one-to-one relationship between the signer and the certificate. However, when a CA signs a certificate, it typically signs many at a time. The advantage of using a single CA signer is that it can validate personal certificates that are generated by the CA for use by peers. However, if the signer is a public CA, you must be aware that the signed certificates might have been generated for another company other than your target entity. For your internal communications, private CAs and self-signed certificates are preferable to public CAs because they enable you to isolate the connections that you want to occur and prevent those that you do not want to occur.

SSL configurations

An SSL configuration comprises a set of configuration attributes that you can associate with an *endpoint* or set of endpoints in the WebSphere Application Server topology. The SSL configuration enables you to create an SSLContext object, which is the fundamental JSSE object that the server uses to obtain SSL socket factories. You can manage the following configuration attributes:

- An alias for the SSLContext object
- A handshake protocol version
- A keystore reference
- A truststore reference
- A key manager
- One or more trust managers
- A security level selection of a cipher suite grouping or a specific cipher suite list
- A certificate alias choice for client and server connections

To understand the specifics of each SSL configuration attribute, see “Secure Sockets Layer configurations” on page 387.

Selecting SSL configurations

In previous releases of WebSphere Application Server, you can reference an SSL configuration only by selecting the SSL configuration alias directly. Each secure endpoint was denoted by an alias attribute that references a valid SSL configuration within a repertoire of SSL configurations. When you made a single configuration change, you had to re-configure many alias references across the various processes. Although the current release still supports direct selection, this approach is no longer recommended.

The current release provides improved capabilities for managing SSL configurations and more flexibility when you select SSL configurations. In this release, you can select from the following approaches:

Programmatic selection

You can set an SSL configuration on the running thread prior to an outbound connection. WebSphere Application Server ensures that most system protocols, including Internet Inter-ORB Protocol (IIOP), Java Message Service (JMS), Hyper Text Transfer Protocol (HTTP), and Lightweight Directory Access Protocol (LDAP), accept the configuration. See “Example: Programmatically specifying an outbound SSL configuration using JSSEHelper API” on page 435

Dynamic selection

You can associate an SSL configuration dynamically with a specific target host, port, or outbound protocol by using a predefined selection criteria. When it establishes the connection, WebSphere Application Server checks to see if the target host and port match a predefined criteria that includes the domain portion of the host. Additionally, you can predefine the protocol for a specific outbound SSL configuration and certificate alias selection. See “Dynamic outbound selection of Secure Sockets Layer configurations” on page 395 for more information.

Direct selection

You can select an SSL configuration by using a specific alias, as in past releases. This method of selection is maintained for backwards compatibility because many applications and processes rely on alias references.

Scope selection

You can associate an SSL configuration and its certificate alias, which is located in the keystore associated with that SSL configuration, with a WebSphere Application Server management scope. This approach is recommended to manage SSL configurations centrally. You can manage endpoints more efficiently because they are located in one topology view of the cell. The inheritance relationship between scopes reduces the number of SSL configuration assignments that you must set.

Each time you associate an SSL configuration with a cell scope, the node scope within the cell automatically inherits the configuration properties. However, when you assign an SSL configuration to a node, the node configuration overrides the configuration that the node inherits from the cell. Similarly, all of the application servers for a node automatically inherit the SSL configuration for that node unless you override these assignments. Unless you override a specific configuration, the topology relies on the rules of inheritance from the cell level down to the endpoint level for each application server.

The topology view displays an inbound tree and outbound tree. You can make different SSL configuration selections for each side of the SSL connection based on what that server connects to as an outbound connection and what the server connects to as an inbound connection. See “Central management of Secure Sockets Layer configurations” on page 396 for more information.

The runtime uses an order of precedence for determining which SSL configuration to choose because you have many ways to select SSL configurations. Consider the following order of precedence when you select a configuration approach:

1. Programmatic selection. If an application sets an SSL configuration on the running thread using the `com.ibm.websphere.ssl.JSSEHelper` application programming interface (API), the SSL configuration is guaranteed the highest precedence.

2. Dynamic selection criteria for outbound host and port or protocol.
3. Direct selection.
4. Scope selection. Scope inheritance guarantees that the endpoint that you select is associated with an SSL configuration and is inherited by every scope beneath it that does not override this selection.

Default self-signed certificate configuration

By default, WebSphere Application Server creates a unique self-signed certificate for each node. WebSphere Application Server no longer relies on the default or dummy certificate that is shipped with the product. The `key.p12` default keystore and the `trust.p12` truststore are stored in the configuration repository within the node directory.

All of the nodes put their signer certificates in this common truststore (`trust.p12`). Additionally, after you federate a node, the default SSL configuration is automatically modified to point to the common truststore, which is located in the cell directory. The node can now communicate with all other servers in the cell.

All default SSL configurations contain a keystore with the name suffix `DefaultKeyStore` and a truststore with the name suffix `DefaultTrustStore`. These default suffixes instruct the WebSphere Application Server runtime to add the signer of the personal certificate to the common truststore. If a keystore name does not end with `DefaultKeyStore`, the keystore signer certificates are not added to the common truststore when you federate the server. You can change the default SSL configuration, but you must ensure that the correct trust is established for administrative connections, among others.

For more information, see “Default self-signed certificate configuration” on page 402 and “Web server plug-in default configuration” on page 410.

Certificate expiration monitoring

Certificate monitoring ensures that the self-signed certificate for each node is not allowed to expire. The certificate expiration monitoring function issues a warning before certificates and signers are set to expire. Those certificates and signers that are located in keystores managed by the WebSphere Application Server configuration can be monitored. You can configure the expiration monitor to automatically replace a self-signed certificate with a new self-signed certificate that is based upon the same data that is used for the initial creation. The monitor also can automatically replace old signers with the signers from the new self-signed certificates in keystores that are managed by WebSphere Application Server. The existing signer exchanges that occurred by the runtime during federation and by administration are preserved. For more information, see “Certificate expiration monitoring” on page 409.

WebSphere Application Server clients: signer-exchange requirements

A new self-signed certificate is generated for each node during its initial startup. To ensure trust, clients must be given these generated signers to establish a connection. Several enhancements in the current release make this process simpler. You can gain access to the signer certificates of various nodes to which the client must connect with any one of the following options (for more information, see “Secure installation for client signer retrieval” on page 404):

- A signer exchange prompt enables you to import signer certificates that are not yet present in the truststores during a connection to a server. By default, this prompt is enabled for administrative connections and can be enabled for any client SSL configuration. When this prompt is enabled, any connection that is made to a server where the signer is not already present offers the signer of the server along with the certificate information and a Secure Hash Algorithm (SHA) digest of the certificate for verification. The user is given a choice whether to accept these credentials. If the credentials are accepted, the signer is added to the truststore of the client until the signer is explicitly removed. The signer exchange prompt does not occur again when connecting to the same server unless the personal certificate changes.

Attention: It is unsafe to trust a signer exchange prompt without verifying the SHA digest. An unverified prompt can originate from a browser when a certificate is not trusted.

- You can run a `retrieveSigners` administrative script from a client prior to making connections to servers. To download signers, no administrative authority is required. To upload signers, you must have Administrator role authority. The script downloads all of the signers from a specified server truststore into the specified client truststore and can be called to download only a specific alias from a truststore. You can also call the script to upload signers to server truststores. When you select the `CellDefaultTrustStore` truststore as the specified server truststore and common truststore for a cell, all of the signers for that cell are downloaded to the specified client truststore, which is typically `ClientDefaultTrustStore`. For more information, see `rxml_retrievesigners.dita`.
- You can physically distribute to clients the `trust.p12` common truststore that is located in the cell directory of the configuration repository. When doing this distribution, however, you must ensure that the correct password has been specified in the `ssl.client.props` client SSL configuration file. The default password for this truststore is `WebAS`. Change the default password prior to distribution. Physical distribution is not as effective as the previous options. When changes are made to the personal certificates on the server, automated exchange can fail.

Dynamic SSL configuration changes

The SSL runtime for WebSphere Application Server maintains listeners for most SSL connections. A change to the SSL configuration causes the inbound connection listeners to create a new `SSLContext` object. Existing connections continue to use the current `SSLContext` object. Outbound connections automatically use the new configuration properties when they are attempted.

Make dynamic changes to the SSL configuration during off-peak hours to reduce the possibility of timing-related problems and to prevent the possibility of the server starting again. If you enable the runtime to accept dynamic changes, then change the SSL configuration and save the `security.xml` file. Your changes take effect when the new `security.xml` file reaches each node.

Note: If configuration changes cause SSL handshake failures, administrative connectivity failures also can occur, which can lead to outages. In this case, you must re-configure the SSL connections then perform manual node synchronization to correct the problem. You must carefully complete any dynamic changes. It is highly recommended that you perform changes to SSL configurations on a test environment prior to making the same changes to a production system. For more information, see “Dynamic configuration updates” on page 411.

Built-in certificate management

Certificate management that is comparable to `iKeyMan` functionality is now integrated into the keystore management panels of the administrative console. Use built-in certificate management to manage personal certificates, certificate requests, and signer certificates that are located in keystores. Additionally, you can remotely manage keystores. For example, you can manage a file-based keystore that is located outside the configuration repository on any node from the deployment manager. You also can remotely manage hardware cryptographic keystores from the deployment manager.

With built-in certificate management, you can replace a self-signed certificate along with all of the signer certificates scattered across many truststores and retrieve a signer from a remote port by connecting to the remote SSL host and port and intercepting the signer during the handshake. The certificate is first validated according to the certificate SHA digest, then the administrator must accept the validated certificate before it can be placed into a truststore.

When you make a certificate request, you can send it to a certificate authority (CA). When the certificate is returned, you can accept it within the administrative console. For more information, see “Certificate management” on page 413.

Tip: Although iKeyMan functionality still ships with WebSphere Application Server, configure keystores from the administrative console using the built-in certificate management functionality. iKeyMan is still an option when it is not convenient to use the administrative console. For more information, see `csec_sslkeymancertman.dita`.

AdminTask configuration management

The SSL configuration management panels in the administrative console rely primarily on administrative tasks, which are maintained and enhanced to support the administrative console function. You can use `wsadmin` commands from a Java console prompt to automate the management of keystores, SSL configurations, certificates, and so on.

Secure Sockets Layer configurations

Secure Sockets Layer (SSL) configurations contain attributes that enable you to control the behavior of both client and server SSL endpoints. You can identify SSL configurations by their user-assigned names or aliases, and assign them to specific management scopes. The scope that an SSL configuration inherits depends upon whether you create it using a cell, node, server, or endpoint link in the configuration topology.

When you create an SSL configuration, you can set the following SSL connection attributes:

- Keystore
- Default client certificate for outbound connections
- Default server certificate for inbound connections
- Truststore
- Key manager for selecting a certificate
- Trust manager or managers for establishing trust during the handshake
- Handshaking protocol
- Ciphers for negotiating the handshake
- Client authentication support and requirements

You can manage an SSL configuration using any of the following methods:

- Central management selection
- Direct reference selection
- Dynamic outbound connection selection
- Programmatic selection

You can view an SSL configuration in the topology at the point where it was created and at all of the scopes below that point. If you want the entire cell to view an SSL configuration, you must create the configuration at the cell level in the topology. Using the administrative console, you can manage all of the SSL configurations for WebSphere Application Server. From the administrative console, click **Security > SSL certificates and key management > Manage endpoint security configurations > Inbound | Outbound > SSL_configuration**. Using the `ssl.client.props` properties file, you can manage client SSL configurations. The `ssl.client.props` file is located in the `${USER_INSTALL_ROOT}/properties` directory for each profile. For more information about configuring this file, see the “`ssl.client.props` client configuration file” on page 445.

SSL configuration in the security.xml file

In the `security.xml` file, you can define specific attributes to configure an SSL configuration repertoire entry at a specific management scope. The scope determines the point at which other levels in the cell topology can see the configuration, as shown in the following example:

```

<repertoire xmi:id="SSLConfig_1" alias="NodeDefaultSSLSettings"
managementScope="ManagementScope_1" type="JSSE">
<setting xmi:id="SecureSocketLayer_1" clientAuthentication="false"
clientAuthenticationSupported="false" securityLevel="HIGH" enabledCiphers=""
jsseProvider="IBMJSSE2" sslProtocol="SSL_TLS" keyStore="KeyStore_1"
trustStore="KeyStore_2" trustManager="TrustManager_1" keyManager="KeyManager_1"
clientKeyAlias="default" serverKeyAlias="default"/>
</repertoire>

```

The SSL configuration attributes that display in the previous code are described in Table 1.

Table 13. security.xml Attributes

security.xml attribute	Description	Default	Associated SSL property
xmi:id	The xmi:id attribute represents the unique identifier for this XML entry and determines how the SSL configuration is linked to other XML objects, such as SSLConfigGroup. This system-defined value must be unique.	The administrative configuration service defines the default value.	None. This value is used only for XML associations.
alias	The alias attribute defines the name of the SSL configuration. Direct selection uses the alias attribute and the node is not prefixed to the alias. Rather, the management scope takes care of ensuring that the name is unique within the scope.		com.ibm.ssl.alias
managementScope	The managementScope attribute defines the management scope for the SSL configuration and determines the visibility of the SSL configuration at runtime.		The managementScope attribute is not mapped to an SSL property. However, it confirms whether or not the SSL configuration is associated with a process.
type	The type attribute defines the Java Secure Socket Extension (JSSE) or System Secure Sockets Layer (SSSL) configuration option. JSSE is the SSL configuration type for most secure communications within WebSphere Application Server.	The default is JSSE.	com.ibm.ssl.sslType
clientAuthentication	The clientAuthentication attribute determines whether SSL client authentication is required.	The default is false.	com.ibm.ssl.clientAuthentication
clientAuthenticationSupported	The clientAuthenticationSupported attribute determines whether SSL client authentication is supported. The client does not have to supply a client certificate if it does not have a client certificate. Attention: When you set the clientAuthentication attribute to true, you override the value that is set for the clientAuthenticationSupported attribute.	The default is false.	com.ibm.ssl.client.AuthenticationSupported

Table 13. security.xml Attributes (continued)

security.xml attribute	Description	Default	Associated SSL property
securityLevel	The securityLevel attribute determines the cipher suite group. Valid values include HIGH (128-bit ciphers), MEDIUM (40-bit ciphers), LOW (for all ciphers without encryption), and CUSTOM (if the cipher suite group is customized. When you set the enabledCiphers attribute with a specific list of ciphers, the system ignores this attribute.	The default is HIGH.	com.ibm.ssl.securityLevel
enabledCiphers	You can set the enabledCiphers attribute to specify a unique list of cipher suites. Separate each cipher suite in the list with a space.	The default is the securityLevel attribute for cipher suite selection.	com.ibm.ssl.enabledCipherSuites
jsseProvider	The jsseProvider attribute defines a specific JSSE provider.	The default is IBMJSSE2.	com.ibm.ssl.contextProvider
sslProtocol	The sslProtocol attribute defines the SSL handshake protocol. Valid options include: SSLv2 (client-side only), SSLv3, SSL, SSL_TLS, TLSv1, and TLS values. The SSL option includes SSLv2 and SSLv3 values. The TLS option includes the TLSv1 value. SSL_TLS, which is the most interoperable protocol, includes all these values and defaults to a Transport Layer Security (TLS) handshake.	The default is SSL_TLS.	com.ibm.ssl.protocol
keyStore	The keyStore attribute defines the keystore and attributes of the keyStore instance that the SSL configuration uses for key selection.		For more information, see Keystore configurations.
trustStore	The trustStore attribute defines the key store that the SSL configuration uses for certificate signing verification.		A trustStore is a logical JSSE term. It signifies a key store that contains signer certificates. Signer certificates validate certificates that are sent to WebSphere Application Server during an SSL handshake.
keyManager	The keyManager attribute defines the key manager that WebSphere Application Server uses to select keys from a key store. A JSSE key manager controls the javax.net.ssl.X509KeyManager interface. A custom key manager controls the javax.net.ssl.X509KeyManager and the com.ibm.wsspi.ssl.KeyManagerExtendedInfo interfaces. The com.ibm.wsspi.ssl.KeyManagerExtendedInfo interface provides more information from WebSphere Application Server.	The default is IbmX509.	com.ibm.ssl.keyManager defines a well-known key manager and accepts the algorithm and algorithm provider formats, for example IbmX509 and IbmX509 IBMJSSE2. com.ibm.ssl.customKeyManager defines a custom key manager and takes precedence over the other keyManager properties. This class must implement javax.net.ssl.X509KeyManager and can implement com.ibm.wsspi.ssl.KeyManagerExtendedInfo. For more information, see csec_sslx509certIDkeyman.dita

Table 13. *security.xml* Attributes (continued)

security.xml attribute	Description	Default	Associated SSL property
trustManager	The trustManager determines which trust manager or list of trust managers to use for determining whether to trust the peer side of the connection. A JSSE trust manager implements the javax.net.ssl.X509TrustManager interface. A custom trust manager might also implement com.ibm.wsspi.ssl.TrustManagerExtendedInfo interface to get more information from the WebSphere Application Server environment.	The default is IbmX509. You can specify the IbmPKIX trust manager for certificate revocation list (CRL) verification when the certificate contains a CRL distribution point.	com.ibm.ssl.trustManager defines a well-known trust manager, which is required for most handshake situations. com.ibm.ssl.trustManager performs certificate expiration checking and signature validation. You can define com.ibm.ssl.customTrustManagers with additional custom trust managers that are called during an SSL handshake. Separate additional trust managers with the vertical bar () character. For more information, see csec_sslx509certtrustdecisions.dita

Trust manager control of X.509 certificate trust decisions

The role of the trust manager is to validate the Secure Sockets Layer (SSL) certificate that is sent by the peer, which includes verifying the signature and checking the expiration date of the certificate. A Java Secure Socket Extension (JSSE) trust manager determines if the remote peer can be trusted during an SSL handshake.

WebSphere Application Server has the ability to call multiple trust managers during an SSL connection. The default trust manager does the standard certificate validation; custom trust manager plug-ins run customized validation such as host name verification. For more information, see “Example: Developing a custom trust manager for custom SSL trust decisions” on page 426

When a trust manager is configured in a server-side SSL configuration, the server calls the isClientTrusted method. When a trust manager is configured in a client-side SSL configuration, the client calls the isServerTrusted method. The peer certificate chain is passed to these methods. If the trust manager chooses not to trust the peer information, it might produce an exception to force a handshake failure.

Optionally, WebSphere Application Server provides the com.ibm.wsspi.ssl.TrustManagerExtendedInfo interface so that additional information can be passed to the trust manager. For more information, see the com.ibm.wsspi.ssl.TrustManagerExtendedInfo interface.

Default IbmX509 trust manager

The default IbmX509 trust manager, which is used in the following code sample, establishes trust by performing standard certificate validation.

```
<trustManagers xmi:id="TrustManager_1132357815717" name="IbmX509" provider="IBMJSSE2"
algorithm="IbmX509" managementScope="ManagementScope_1132357815717"/>
```

The trust manager provides a signer certificate to verify the peer certificate that is sent during the handshake. The signers who are added to the truststore for the SSL configuration must be trustworthy. If you do not trust the signers or do not want to allow others to connect to your servers, consider removing default root certificates from certificate authorities (CA). You might also remove any self-signed certificates if you cannot verify their origination.

Default IbmPKIX trust manager

You can use the default IbmPKIX trust manager to replace the IbmX509 trust manager, which is shown in the following code sample:

```
<trustManagers xmi:id="TrustManager_1132357815719" name="IbmPKIX" provider="IBMJSSE2"
algorithm="IbmPKIX" trustManagerClass="" managementScope="ManagementScope_1132357815717">
<additionalTrustManagerAttrs xmi:id="DescriptiveProperty_1132357815717"
```

```

name="com.ibm.security.enableCRLDP" value="true" type="boolean"/>
<additionalTrustManagerAttrs xmi:id="DescriptiveProperty_1132357815718"
name="com.ibm.jsse2.checkRevocation" value="true" type="boolean"/>
</trustManagers>

```

In addition to its role of standard certificate verification, the IbmPKIX trust manager checks for certificates that contain certificate revocation list (CRL) distribution points. This process is known as *extended CRL checking*. When you select a trust manager, its associated properties are automatically set as Java System properties so that the IBM CertPath and IBMJSSE2 providers are aware that CRL checking is enabled.

Custom trust manager

You can define a custom trust manager to perform additional trust checking, which is based upon the needs of the environment. For example, in one environment, you might enable connections from the same Transmission Control Protocol (TCP) subnet only. The `com.ibm.wsspi.ssl.TrustManagerExtendedInfo` interface provides extended information about the connection that is not provided by the standard Java Secure Sockets Extension (JSSE) `javax.net.ssl.X509TrustManager` interface. The configured `trustManagerClass` attribute determines which class is instantiated by the runtime, as shown in the following code sample:

```

<trustManagers xmi:id="TrustManager_1132357815718" name="CustomTrustManager"
trustManagerClass="com.ibm.ws.ssl.core.CustomTrustManager"
managementScope="ManagementScope_1132357815717"/>

```

The `trustManagerClass` attribute must implement the `javax.net.ssl.X509TrustManager` interface and, optionally, can implement the `com.ibm.wsspi.ssl.TrustManagerExtendedInfo` interface.

Disabling the default trust manager

In some cases, you might not want to perform the standard certificate verification that is provided by the IbmX509 and IbmPKIX default trust managers. For example, you might be working with an internal automated test infrastructure that is not concerned with SSL client or server authentication, integrity, or confidentiality. The following sample code shows a basic custom trust manager such as `com.ibm.ws.ssl.core.CustomTrustManager` whose property is set to `true`.

```
com.ibm.ssl.skipDefaultTrustManagerWhenCustomDefined=true
```

You can set this property in the global properties at the top of the `ssl.client.props` file for clients or in the `security.xml` custom properties file for servers. You must configure a custom trust manager when you disable the default trust manager to prevent the server from calling the default trust manager even though it is configured. Disabling the default trust manager is not a common practice. Be sure to test the system with the disabled default trust manager in a test environment first. For more information on setting up a custom trust manager, see “Creating a custom trust manager configuration” on page 422

Key manager control of X.509 certificate identities

The role of a Java Secure Socket Extension (JSSE) key manager is to retrieve the certificate that is used to identify the client or server during a Secure Sockets Layer (SSL) handshake.

WebSphere Application Server provides a default key manager that can select a certificate from a keystore when you define the following SSL configuration properties:

com.ibm.ssl.keyStoreClientAlias

Defines the alias that is chosen from the keystore for the client side of a connection. This alias must be present in the keystore.

com.ibm.ssl.keyStoreServerAlias

Defines the alias that is chosen from the keystore for the server side of a connection. This alias must be present in the keystore.

These two properties are set automatically when you use the administrative console because the default key manager is already configured.

With WebSphere Application Server, you can configure only one key manager at a time for a given SSL configuration. If you want custom certificate selection logic on the client side, you must write a new custom key manager. The custom key manager could provide function that prompts the user to choose a certificate dynamically. Also, you can implement an extended interface so that a key manager can provide information during connection time. For more information on the extended interface, see the `com.ibm.wsspi.ssl.KeyManagerExtendedInfo` interface. For more information on custom key manager development, see `rsec_ssldevcustomkeymgr.dita`.

Default IbmX509 key manager

The default IbmX509 key manager chooses a certificate to serve as the identity for an SSL handshake. The key manager is called to enable client authentication on either side of the SSL handshake; frequently on the server-side, and less frequently on the client side according to client and server requirements. If a keystore is not configured on the client-side and SSL client authentication is enabled, the key manager cannot select a certificate to send to the server. Therefore, the handshake fails.

The following sample code shows the key manager configuration in the `security.xml` file for an IbmX509 key manager.

```
<keyManagers xmi:id="KeyManager_1" name="IbmX509"
provider="IBMJSSE2" algorithm="IbmX509" keyManagerClass=""
managementScope="ManagementScope_1"/>
```

You do not specify the `keyManagerClass` class because the key manager is provided by the IBMJSSE2 provider. However, you can specify whether the key manager is a custom class implementation, in which case you must specify the `keyManager` class, or an algorithm name that WebSphere Application Server can start from the Java security provider framework.

Custom key manager

The following sample code shows the key manager configuration in the `security.xml` file for a custom class.

```
<keyManagers xmi:id="KeyManager_2" name="CustomKeyManager"
keyManagerClass="com.ibm.ws.ssl.core.CustomKeyManager"
managementScope="ManagementScope_1"/>
```

The custom class must implement the `javax.net.ssl.X509KeyManager` interface and, optionally, implement the `com.ibm.wsspi.ssl.KeyManagerExtendedInfo` interface to retrieve additional WebSphere Application Server information. This interface replaces the function of the default key manager because you can configure only one key manager at a time. Therefore, the custom key manager has sole responsibility for selecting the alias to use from the configured keystore. The benefit of a custom key manager is its ability, on the client side, to prompt for an alias. This process enables the user to decide which certificate to use in situations where the user knows the client certificate identity. For more information, see `tsec_sslcreatecuskeymgr.dita`.

Keystore configurations

Use keystore configurations to define how the runtime for WebSphere Application Server loads and manages keystore types for Secure Sockets Layer (SSL) configurations.

By default, the `java.security.Security.getAlgorithms("KeyStore")` attribute does not display a predefined list of keystore types in the administrative console. Instead, WebSphere Application Server retrieves all of the `KeyStore` types that can be referenced by the `java.security.KeyStore` object, including hardware cryptographic, z/OS platform, i5/OS platform, IBM Java Cryptography Extension (IBMJCE), and Java-based content management system (CMS)-provider keystores. If you specify a keystore provider in

the java.security file or add it to the provider list programmatically, WebSphere Application Server also retrieves custom keystores. The retrieval list depends upon the java.security configuration for that platform and process.

IBMJCE file-based keystores (JCEKS, JKS, and PKCS12)

A typical IBMJCE file-based keystore configuration is shown in the following sample code:

```
<keyStores xmi:id="KeyStore_1" name="NodeDefaultKeyStore"
password="{xor}349dkckdd=" provider="IBMJCE"
location="{USER_INSTALL_ROOT}/config/cells/myhostNode01Cell
/nodes/myhostNode01/key.p12" type="PKCS12" fileBased="true"
hostList="" initializeAtStartup="true" readOnly="false"
managementScope="ManagementScope_1"/>
```

For more information about default keystore configurations, see csec_ssldefselfsigncertconf.dita.

Table 1 describes the attributes that are used in the sample code.

Table 14. keystore configurations

Attribute name	Default	Description
xmi:id	Varies	A value that issued to reference the keystore from another area in the configuration, for example, from an SSL configuration. Make this value unique within the security.xml file.
name	For Java Secure Socket Extension (JSSE) keystore: NodeDefaultKeyStore. For JSSE truststore: NodeDefaultTrustStore.	A name that is used to identify the keystore by sight. The name can determine if the keystore is a default keystore based upon whether the name ends with DefaultKeyStore or DefaultTrustStore.
password	The default keystore password is WebAS. It is recommended that this be changed as soon as possible. See "Updating default key store passwords using scripting" on page 656 for more information.	The password that is used to access the keystore name is also the default that is used to store keys within the keystore.
provider	The default provider is IBMJCE.	The Java provider that implements the type attribute (for example, PKCS12 type). The provider can be left unspecified and the first provider that implements the keystore type specified is used.
location	The default varies, but typically references a key.p12 file or a trust.p12 file in the node or cell directories of the configuration repository. These files are PKCS12 type keystores.	The keystore location reference. If the keystore is file-based, the location can reference any path in the file system of the node where the keystore is located. However, if the location is outside of the configuration repository, and you want to manage the keystore remotely from the administrative console or from the wsadmin utility, then specify the hostList attribute that contains the host name of the node where it resides.

Table 14. keystore configurations (continued)

Attribute name	Default	Description
type	The default Java crypto device keystore type is PKCS12.	This type specifies the keystore. Valid types can be those returned by the <code>java.security.Security.getAlgorithms("KeyStore")</code> attribute. These types include the following keystore types, and availability depends on the process and platform <code>java.security</code> configuration: <ul style="list-style-type: none"> • JKS • JCEKS • PKCS12 • PKCS11 (Java crypto device) • CMSKS • IBMi5OSKeyStore • JCERACFKS • JCE4758KS (z/OS crypto device)
fileBased	The default is true.	This option is required for default keystores. It indicates a file-system keystore so you can use a <code>FileInputStream</code> or <code>FileOutputStream</code> for loading and storing the keystore.
hostList	The <code>hostList</code> attribute is used to specify a remote hostname so that the keystore can be remotely managed. There are no remotely managed keystores by default. All default keystores are managed locally in the configuration repository and synchronized out to each of the nodes.	The option manages a keystore remotely. You can set the host name of a valid node for a keystore. When you use either the administrative console or the <code>wsadmin</code> utility to manage certificates for this keystore, an MBean call is made to the node where the keystore exists for the approved operation. You can specify multiple hosts, although synchronization of the keystore operations are not guaranteed. For example, one of the hosts that is listed might be down when a specific operation is performed. Therefore, use multiple hosts in this list.
initializeAtStartup	The default is true.	This option informs the runtime to initialize the keystore during startup. This option can be important for hardware cryptographic device acceleration.
readOnly	The default is false.	This option informs the configuration that you cannot write to this keystore. That is, certain update operations on the keystore cannot be attempted and are not allowed. An example of a read-only keystore type is JCERACFKS on the z/OS platform. This type is read-only from the WebSphere certificate management standpoint, but you can also update it using the keystore management facility for RACF.
managementScope	The default scope is the node scope for a base Application Server environment and the cell scope for a Network Deployment environment.	This option references a particular management scope in which you can see this keystore. For example, if a hardware cryptographic device is physically located on a specific node, then create the keystore from a link to that node in the topology view under Security > Security Communications > SSL configurations . You can also use management scope to isolate a keystore reference. In some cases, you might need to allow only a specific application server to reference the keystore; the management scope is for that specific server.

CMS keystores

You can set some provider-specific attributes in CMS keystores.

If the CMSKS provider supports the `createStashFileForCMS` attribute, and you set the attribute to `true` for CMSKS keystores, WebSphere Application Server creates an `.sth` file in the keystore location that is referenced by the attribute. The `.sth` extension is appended to the keystore name. For example, if the CMSKS keystore is available for a plug-in configuration and you set `createStashFileForCMS` to `true`, the stash file that is represented in the following sample code is created in the `${USER_INSTALL_ROOT}\profiles\AppSrv01/config/cells/myhostCell01/nodes/myhostNode01/servers/webserver1/plugin-key.sth` path.

```
<keyStores xmi:id="KeyStore_1132071489571" name="CMSKeyStore"
password="{xor}HRYNFAtRbxEw0zpvbhw6MzM=" provider="IBMCMSProvider"
location="${USER_INSTALL_ROOT}\profiles\AppSrv01/config/cells/myhostCell01
/nodes/myhostNode01/servers/webserver1/plugin-key.kdb" type="CMSKS"
fileBased="true" createStashFileForCMS="true"
managementScope="ManagementScope_1132071489569"/>
```

When you create a CMS keystore, the CMS provider is `IBMi50SJSSEProvider`, and the CMS type is `IBMi50SKeyStore`, as shown in the following sample code:

```
<keyStores xmi:id="KeyStore_1132071489571" name="CMSKeyStore"
password="{xor}HRYNFAtRbxEw0zpvbhw6MzM=" provider="IBMi50SJSSEProvider"
location="${USER_INSTALL_ROOT}\profiles\AppSrv01/config/cells/myhostCell01
/nodes/myhostNode01/servers/webserver1/plugin-key.kdb" type="IBMi50SKeyStore"
fileBased="true" createStashFileForCMS="true"
managementScope="ManagementScope_1132071489569"/>
```

Hardware cryptographic keystores

For cryptographic device configuration, see “Key management for cryptographic uses” on page 486 and “Configuring a hardware cryptographic keystore” on page 453

You can add a slot either as the custom property, `com.ibm.ssl.keyStoreSlot`, or as the configuration attribute, `slot="0"`. The custom property is read before the attribute for backwards compatibility.

In certain environments, you can use the hardware cryptographic card for hardware acceleration. Either set the `useForAcceleration` attribute to `true` or set the `com.ibm.ssl.keyStoreUseForAcceleration` custom property. When you set the attribute to `true`, you are not required to configure a password. However, you cannot use the device to store keys. For more information on configuring a hardware cryptographic keystore, see [Configuring hardware cryptographic keystore configurations](#).

Default key store passwords

When WebSphere Application Server starts for the first time as a standalone application server or as a network deployment, each server creates a key store and trust store for the default SSL configuration. When WebSphere Application Server creates these files, they use a default password (WebAS). Because the default password is well known, it is important that you change the password to protect the security of the key store files and the SSL configuration.

You can change the keystore passwords of:

- A single keystore or multiple key stores using a command task
- Multiple keystores using a command task
- A single keystore, using the administrative console

Dynamic outbound selection of Secure Sockets Layer configurations

WebSphere Application Server provides dynamic outbound selection that enables you to choose a specific Secure Sockets Layer (SSL) configuration and certificate alias for each outbound protocol, target host,

target port, or any combination of these attributes. You can specify the dynamic selection information for outbound connections from a pure client or from a server that is acting as a client.

Before the SSL runtime for WebSphere Application Server starts an outbound connection, the runtime attempts to match the outbound protocol, target host, and target port attributes with the dynamic outbound selection information that is associated with an SSL configuration and certificate alias in the configuration.

The runtime caches both selection misses and selection hits, so the impact on performance can be minimal. However, a relationship exists between the amount of dynamic outbound selection information and its impact on the initial connection performance.

Target information during outbound connections

The dynamic outbound selection configurations are only effective when the outbound protocol uses the JSSEHelper application programming interface (API) when you select an SSL configuration with a specified connectionInfo hash map. This hash map must contain the following properties:

com.ibm.ssl.direction

The value for outbound connections is OUTBOUND.

com.ibm.ssl.remoteHost

The format should match what the protocol provides. Typically this is the canonical Domain Name Space (DNS), but it also could be the IP address.

com.ibm.ssl.remotePort

The port is target port.

com.ibm.ssl.endPointName

The value for an outbound connection must be one of the following protocol strings:

- IIOP
- HTTP
- SIP
- LDAP
- ADMIN_SOAP
- BUS_TO_BUS
- BUS_CLIENT
- BUS_TO_WEBSPHERE_MQ

Central management of Secure Sockets Layer configurations

By default, Secure Sockets Layer (SSL) configurations for servers are managed from a central location in the topology view in the administrative console. You can associate an SSL configuration and certificate alias with a management scope. This method is the most efficient method to manipulate and modify configurations when the server topology changes.

In prior releases, SSL configurations are managed in the `server.xml` file for each process. You have to edit individual `server.xml` documents to modify individual SSL configuration aliases in the configuration topology. In this release of WebSphere Application Server, management control of SSL configurations offers more flexibility and options. You can make coarse-grained changes using the cell-scope and fine-grained changes using a particular endpoint name, as defined in the `serverindex.xml` file for a specific application server process.

Because SSL configuration associations manifest inheritance behaviors, you can simplify the number of associations by referencing only the highest level management scope that needs a unique configuration. Obviously, the security environment influences issues such as SSL configuration uniqueness, and SSL configuration and certificate alias placement in the topology.

To configure the inbound and outbound topologies, which must be done separately in the administrative console, click **Security > SSL certificates and key management > Manage endpoint security configurations > Inbound | Outbound**.

The topology view provides the scoping mechanism. The SSL configuration inherits its visibility, which is its display in the topology, at the scope where you created the configuration and at all the scopes beneath this parent scope. When you create an SSL configuration at a specific node, the configuration can be seen by that node agent and by every application server that is part of that node. Any application server or node that is not part of this particular node cannot see this SSL configuration. You can configure different certificate aliases and SSL configurations for inbound versus outbound connections.

Default centrally-managed SSL configuration

The default management scope is the node scope. When a node is federated into a cell, the default SSL configurations for the node are maintained, as shown in the following sample code for the `sslConfigGroups` and management scopes attributes:

```
<sslConfigGroups xmi:id="SSLConfigGroup_1" name="myhostNode01"
direction="inbound" certificateAlias="default" sslConfig="SSLConfig_1"
managementScope="ManagementScope_1"/>
<sslConfigGroups xmi:id="SSLConfigGroup_2" name="myhostNode01"
direction="outbound" certificateAlias="default" sslConfig="SSLConfig_1"
managementScope="ManagementScope_1"/>

<managementScopes xmi:id="ManagementScope_1"
scopeName="(cell):myhostNode01Cell:(node):myhostNode01" scopeType="node"/>
```

The SSL configuration `xmi:id "SSLConfig_1"` is also federated and applicable:

```
<repertoire xmi:id="SSLConfig_1" alias="NodeDefaultSSLSettings"
managementScope="ManagementScope_1">
<setting xmi:id="SecureSocketLayer_1" clientAuthentication="true"
securityLevel="HIGH" enabledCiphers="" jsseProvider="IBMJSSE2"
sslProtocol="SSL_TLS" keyStore="KeyStore_1" trustStore="KeyStore_2"
trustManager="TrustManager_1" keyManager="KeyManager_1"/>
</repertoire>
```

The keystores that are associated with the `SSLConfig_1` SSL configuration are also federated, and `key.p12` is located in the node directory of the configuration repository:

```
<keyStores xmi:id="KeyStore_1" name="NodeDefaultKeyStore"
password="{xor}HRYNFAtRbxEw0zpvbhw6MzM=" provider="IBMJCE"
location="{USER_INSTALL_ROOT}/config/cells/myhostNode01Cell/nodes
/myhostNode01/key.p12" type="PKCS12" fileBased="true" hostList=""
initializeAtStartup="true" managementScope="ManagementScope_1"/>
<keyStores xmi:id="KeyStore_2" name="NodeDefaultTrustStore"
password="{xor}HRYNFAtRbxEw0zpvbhw6MzM=" provider="IBMJCE"
location="{USER_INSTALL_ROOT}/config/cells/myhostNode01Cell
/nodes/myhostNode01/trust.p12" type="PKCS12" fileBased="true"
hostList="" initializeAtStartup="true" managementScope="ManagementScope_1"/>
```

Secure Sockets Layer node, application server, and cluster isolation

Secure Sockets Layer (SSL) enables you to ensure that any client that attempts to connect to a server during the handshake first performs server authentication. You can isolate communications between servers that must not communicate with each other over secure ports using SSL configurations at the node, application server, and cluster scopes.

Before you attempt to isolate communications controlled by WebSphere Application Server, you must have a good understanding of the deployment topology and application environment. To isolate a node, application server, or cluster, you must be able to control the signers that are contained in the trust stores that are associated with the SSL configuration. When the client does not contain the server signer, it cannot establish a connection to the server. If you use self-signed certificates, the server that created the

personal certificate controls the signer, although you do have to manage the self-signed certificates. If you obtain certificates from a certificate authority (CA), you must obtain multiple CA signers because all of the servers can connect to each other if they share the same signer.

Authenticating only the server-side of a connection is not adequate protection when you need to isolate a server. Any client can obtain a signer certificate for the server and add it to its trust store. SSL client authentication must also be enabled between servers so that the server can control its connections by deciding which client certificates it can trust. For more information, see `tsec_sslclientauthinbound.dita`, which applies as well to enabling SSL client authentication at the cell level.

Isolation also requires that you use centrally managed SSL configurations for all or most endpoints in the cell. Centrally managed configurations can be scoped, unlike direct or end point configuration selection, and they enable you to create SSL configurations, key stores, and trust stores at a particular scope. Because of the inheritance hierarchy of WebSphere Application Server cells, if you select only the properties that you need for an SSL configuration, only these properties are defined at your selected scope or lower. For example, if you configure at the node scope, your configuration applies to the application server and individual end point scopes below the node scope. For more information, see `tsec_sslassocconfigscope.dita`, `tsec_sslselconfigdirect.dita`, and `tsec_sslassocconfigout.dita`

When you configure the key stores, which contain cryptographic keys, you must work at the same scope at which you define the SSL configuration and not at a higher scope. For example, if you create a key store that contains a certificate whose host name is part of the distinguished name (DN), then store that keystore in the node directory of the configuration repository. If you decide to create a certificate for the application server, then store that keystore on the application server in the application server directory.

When you configure the trust stores, which control trust decisions on the server, you must consider how much you want to isolate the application servers. You cannot isolate the application servers from the node agents or the deployment manager. However, you can configure the SOAP connector end points with the same personal certificate or to share trust. Naming persistence requires IIOP connections when they pass through the deployment manager. Because application servers always connect to the node agents when the server starts, the IIOP protocol requires that WebSphere Application Server establish trust between the application servers and the node agents.

Establishing node SSL isolation

By default, WebSphere Application Server uses a single self-signed certificate for each node so you can isolate nodes easily. A common trust store, which is located in the cell directory of the configuration repository, contains all of the signers for each node that is federated into the cell. After federation, each cell process trusts all of the other cell processes because every SSL configuration references the common trust store.

You can modify the default configuration so that each node has its own trust store, and every application server on the node trusts only the node agent that uses the same personal certificate. You must also add the signer to the node trust store so that WebSphere Application Server can establish trust with the deployment manager. To isolate the node, ensure that the following conditions are met:

- The deployment manager must initiate connections to any process
- The node agent must initiate connections to the deployment manager and its own application servers
- The application servers must initiate connections to the applications servers on the same node, to its own node agent, and the deployment manager

Figure 1 shows Node Agent A contains a `key.p12` keystore and a `trust.p12` trust store at the node level of the configuration repository for node A.

Figure 4.

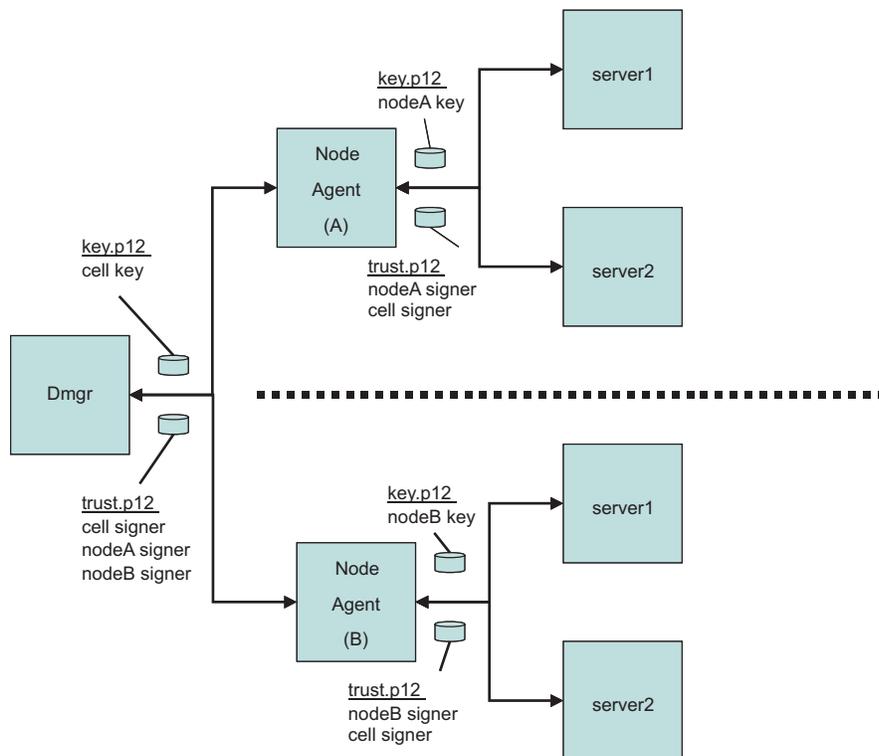


Figure 1: SSL Node Isolation

When you associate an SSL configuration with this keystore and truststore, you break the link with the cell-scoped trust store. To isolate the node completely, repeat this process for each node in the cell. WebSphere Application Server SSL configurations override the cell scope and use the node scope instead so that each process at this scope uses the SSL configuration and certificate alias that you selected at this scope. You establish proper administrative trust by ensuring that nodeA signer is in the common trust store and the cell signer is in the nodeA trust store. The same logic applies to node B as well. For more information, see `tsec_sslassocconfigscope.dita`.

Establishing application server SSL isolation

Isolating application server processes from one another is more challenging than isolating nodes. You must consider the following application design and topology conditions:

- An application server process might need to communicate with the node agent and deployment manager
- Isolating application server processes from each other might disable single sign-on capabilities for horizontal propagation

If you configure outbound SSL configurations dynamically, you can accommodate these conditions. When you define a specific outbound protocol, target host, and port for each different SSL configuration, you can override the scoped configuration.

Figure 2 shows how you might isolate an application server completely, although in practice this approach would be more complicated.

Figure 5.

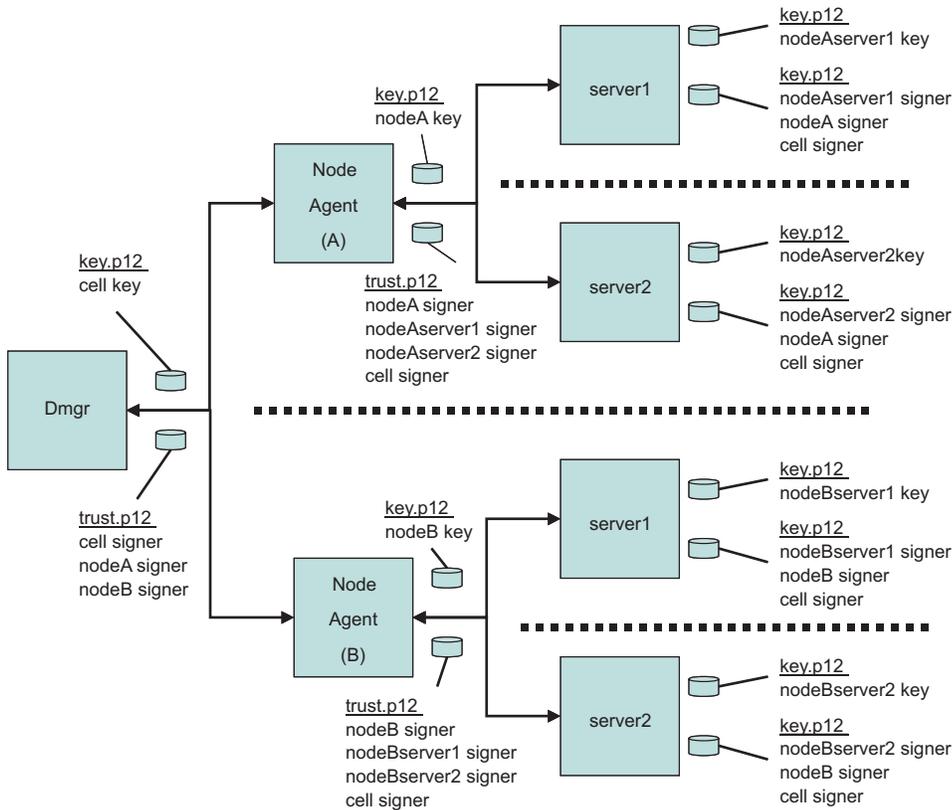


Figure 2: SSL Application Server Isolation

The dynamic configuration enables server1 on Node A to communicate with server 1 on Node B only over IIOp. The dynamic outbound rule is `IIOp,nodeBhostname,*`. For more information, see `tsec_sslssoconfigout.dita`

Establishing cluster SSL isolation

You can configure application servers into clusters instead of scoping them centrally at the node or dynamically at the server to establish cluster SSL isolation. While clustered servers can communicate with each other, application servers outside of the cluster cannot communicate with the cluster, thus isolating the clustered servers. For example, you might need to separate applications from different departments while maintaining a basic level of trust among the clustered servers. Using the dynamic outbound SSL configuration method described for servers above, you can easily extend the isolated cluster as needed.

Figure 3 shows a sample cluster configuration where cluster 1 contains a `key.p12` with its own self-signed certificate, and a `trust.p12` that is located in the `config/cells/<cellname>/clusters/<clustername>` directory.

Figure 6.

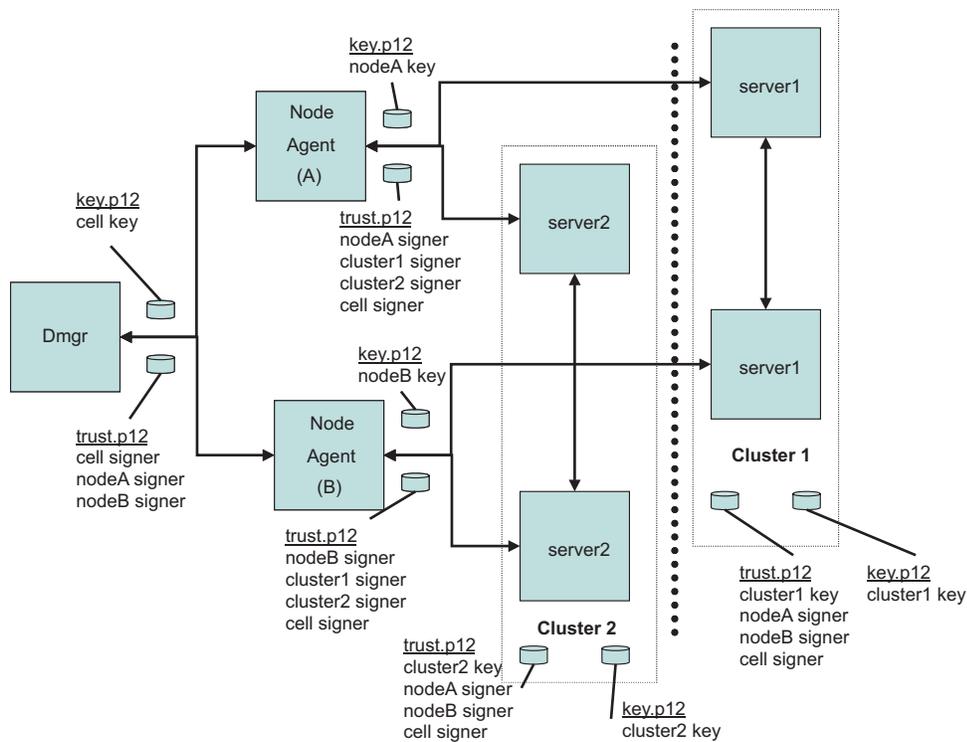


Figure 3: SSL Cluster Isolation

In the example, cluster1 might contain web applications, and cluster2 might contain EJB applications. Considering the various protocols, you decide to enable IIOp traffic between the two clusters. Your task is to define a dynamic outbound SSL configuration at the cluster1 scope with the following properties:

```
IIOP,nodeAhostname,9403|IIOP,nodeAhostname,9404|IIOP,nodeBhostname,9403|IIOP,nodeBhostname,9404
```

You must create another SSL configuration at the cluster1 scope that contains a new trust.p12 file with the cluster2 signer. Consequently, outbound IIOp requests go either to nodeAhostname ports 9403 and 9404 or to nodeBhostname ports 9403 and 9404. The IIOp SSL port numbers on these two application server processes in cluster2 identify the ports.

As you review Figure 3, notice the following features of the cluster isolation configuration:

- The trust.p12 for cluster1 contains signers that allow communications with itself (cluster1 signer), between both node agents (nodeAsigner and nodeBsigner), and with the deployment manager (cell signer).
- The trust.p12 for cluster2 contains signers that allow communications with itself (cluster2 signer), between both node agents (nodeAsigner and nodeBsigner), and with the deployment manager (cell signer).
- Node agent A and Node agent B can communicate with themselves, the deployment manager, and both clusters.

For more information, see tsec_sslssoconfigout.dita.

Although this article presents an overview of isolation methods from an SSL perspective, you must also ensure that non-SSL ports are closed or applications require the confidentiality constraint in the deployment descriptor. For example, you can set the CSiv2 inbound transport panel to require SSL and disable the channel ports that are not secure from the server ports configuration.

Also, you must enable SSL client authentication for SSL to enforce the isolation requirements on both sides of a connection. Without mutual SSL client authentication, a client can easily obtain a signer for the server programmatically and thus bypass the goal of isolation. With SSL client authentication, the server would require the client's signer for the connection to succeed. For HTTP/S protocol, the client is typically a browser, a Web Service, or a URL connection. For the IIOP/S protocol, the client is typically another application server or a Java client. WebSphere Application Server must know the clients to determine if SSL client authentication enablement is possible. Any applications that are available through a public protocol must not enable SSL client authentication because the client may fail to obtain a certificate to authenticate to the server.

Note: It is beyond the scope of this article to describe all of the factors you must consider to achieve complete isolation.

Default self-signed certificate configuration

When a WebSphere Application Server process starts for the first time, the Secure Sockets Layer (SSL) runtime initializes the default keystores and truststores that are specified in the SSL configuration.

Default keystore and truststore properties

WebSphere Application Server creates the key.p12 default keystore file and the trust.p12 default truststore file during profile creation. A default, self-signed certificate is also created in the key.p12 file at this time. The signer or public key is extracted from the key.p12 file and added to the trust.p12 file. If the files do not exist during process startup, they are recreated during startup.

You can easily identify keystore and truststore defaults because of their suffixes: DefaultKeyStore and DefaultTrustStore. Also, in the SSL configuration, you must set the fileBased attribute to true so that the runtime uses the default keystores and truststores only.

On a base application server, default key and truststores are stored in the node directory of the configuration repository. For example, the default key.p12 and trust.p12 stores are created with the AppSrv01 profile name, the myhostNode01Cell name, and the myhostNode01 node name. The key and truststores are located in the following directories:

```
C:\WebSphere\AppServer\profiles\AppSrv01\config\cells\myhostNode01Cell
\nodes\myhostNode01\key.p12
C:\WebSphere\AppServer\profiles\AppSrv01\config\cells\myhostNode01Cell
\nodes\myhostNode01\trust.p12
```

The default password is WebAS for all default keystores generated by WebSphere Application Server. Change the default password after the initial configuration for a more secure environment.

Default self-signed certificate

The default self-signed certificate is created during profile creation for both the server and client for that profile.

You can recreate the certificates with different information simply by deleting the *.p12 files in /config and /etc. Change the four properties below to the values you want the certificates to contain, then restart the processes. This causes the server certificate in /config and the client certificate in /etc to differ.

If you want to set up SSL client authentication between the client and server, you must perform a signer exchange. The certificate properties below that exist in the ssl.client.props file do not exist in the server configuration. However, you can use these values in the server configuration by adding them as custom security properties in the administrative console. The certificate properties appear in the ssl.client.props file, but do not appear in the server configuration. However, you can modify these values in the server configuration by adding them as custom properties in the administrative console.

Click **Security > Secure administration, applications, and infrastructure > Custom properties** to change the following properties:

```
com.ibm.ssl.defaultCertReqAlias=default_alias
com.ibm.ssl.defaultCertReqSubjectDN=cn=${hostname},o=IBM,c=US
com.ibm.ssl.defaultCertReqDays=365
com.ibm.ssl.defaultCertReqKeySize=1024
```

If a default_alias value already exists, the runtime appends `_#`, where the number sign (#) is a number that increments until it is unique in the keystore. `${hostname}` is a variable that is resolved to the host name where it was originally created. The default expiration date of self-signed certificates is one year from their creation date.

The runtime monitors the expiration dates of self-signed certificates using the Certificate Expiration Monitor. These self-signed certificates are automatically replaced along with the signer certificates when they are within the expiration threshold, which is typically 30 days before expiration. You can increase the default key size beyond 1024 bits only when the Java Runtime Environment policy files are unrestricted, that is, not exported. For more information, see `csec_sslcertmonitoring.dita`.

Default keystore and truststore configurations for new Base Application Server processes

The following sample code shows the default SSL configuration for a base application server. References to the default keystores and truststores files are highlighted.

```
<repertoire xmi:id="SSLConfig_1" alias="NodeDefaultSSLSettings"
managementScope="ManagementScope_1">
<setting xmi:id="SecureSocketLayer_1" clientAuthentication="false"
securityLevel="HIGH" enabledCiphers="" jsseProvider="IBMJSSE2" sslProtocol="SSL_TLS"
keyStore="KeyStore_1" trustStore="KeyStore_2" trustManager="TrustManager_1"
keyManager="KeyManager_1"/>
</repertoire>
```

Default keystore

In the following sample code, the keystore object that represents the default keystore is similar to the XML object.

```
<keyStores xmi:id="KeyStore_1" name="NodeDefaultKeyStore"
password="{xor}349dkckdd=" provider="IBMJCE" location="{USER_INSTALL_ROOT}/config
/cells/myhostNode01Cell/nodes/myhostNode01/key.p12" type="PKCS12" fileBased="true"
hostList="" initializeAtStartup="true" managementScope="ManagementScope_1"/>
```

The `NodeDefaultKeyStore` keystore contains the personal certificate that represents the identity of the secure endpoint. Any keystore reference can use the `{USER_INSTALL_ROOT}` variable, which is expanded by the runtime. The PKCS12 default keystore type is in the most interoperable format, which means that it can be imported into most browsers. The `myhostNode01Cell` password is encoded.

The management scope determines which server runtime loads the keystore configuration into memory, as shown in the following code sample:

```
<managementScopes xmi:id="ManagementScope_1" scopeName="
(cell):myhostNode01Cell:(node):myhostNode01" scopeType="node"/>
```

Any configuration objects that are stored in the `security.xml` file whose management scopes are outside the current process scope are not loaded in the current process. Instead, the management scope is loaded by servers that are contained within the `myhostNode01` node. Any application server that is on the specific node can view the keystore configuration.

When you list the contents of the `key.p12` file to show the self-signed certificate, note that the common name (CN) of the distinguished name (DN) is the host name of the resident machine. This listing enables

you to verify the host name by its URL connections. Additionally, you can verify the host name from a custom trust manager. For more information, see csec_sslx509certrustdecisions.dita.

Contents of default keystore

The following sample code shows the contents of the default key.p12 file in a keytool list:

```
keytool -list -keystore c:\WebSphere\AppServer\profile\AppSrv01\profiles\config
\cells\myhostNode01Cell\nodes\myhostNode01\key.p12 -storepass myhostNode01Cell
-storetype PKCS12 -v
```

```
Alias name: default
Entry type: keyEntry
Owner: CN=myhost.austin.ibm.com, O=IBM, C=US
Issuer: CN=myhost.austin.ibm.com, O=IBM, C=US
Valid from: 10/18/05 4:06 PM until: 10/18/06 4:06 PM
Certificate fingerprint:
    SHA1: 33:6E:9E:10:65:04:CE:7A:6C:C3:B1:79:8B:9A:05:49:AC:E5:67:F3
```

The default alias name and the keyEntry entry type indicate that the private key is stored with the public key, which represents a complete personal certificate. The certificate is owned by CN=myhost.austin.ibm.com, O=IBM, C=US and it is issued by the same entity, which is self-signed. By default, the certificate is valid for one year from the date of creation.

Additionally, in some signer-exchange situations, the certificate fingerprint ensures that the sent certificate has not been modified. The fingerprint, which is a hash algorithm output for the certificate, is displayed by the WebSphere Application Server runtime during an automated signer exchange on the client side. The client fingerprint must match the fingerprint that is displayed on the server. The runtime typically uses the SHA1 hash algorithm to generate certificate fingerprints.

Default truststore

In the following sample code, the keystore object represents the default trust.p12 truststore. The truststore contains signer certificates that are necessary for making trust decisions:

```
<keyStores xmi:id="KeyStore_2" name="NodeDefaultTrustStore"
password="{xor}349dkckdd=" provider="IBMJCE" location="{USER_INSTALL_ROOT}
/config/cells/myhostNode01Cell/nodes/myhostNode01/trust.p12" type="PKCS12"
fileBased="true" hostList="" initializeAtStartup="true" managementScope="ManagementScope_1"/>
```

Contents of default truststore

The following sample code shows the contents of the default trust.p12 truststore in a keytool listing. For the sample self-signed certificate, the default_signer alias name and the trustedCertEntry entry type indicate that the certificate is the public key. The private key is not stored in this truststore.

```
keytool -list -keystore c:\WebSphere\AppServer\profile\AppSrv01\profiles\config
\cells\myhostNode01Cell\nodes\myhostNode01\trust.p12 -storepass myhostNode01Cell
-storetype PKCS12 -v
```

```
Alias name: default_signer
Entry type: trustedCertEntry
```

```
Owner: CN=myhost.austin.ibm.com, O=IBM, C=US
Issuer: CN=myhost.austin.ibm.com, O=IBM, C=US
Valid from: 10/18/05 4:06 PM until: 10/18/06 4:06 PM
Certificate fingerprint:
    SHA1: 33:6E:9E:10:65:04:CE:7A:6C:C3:B1:79:8B:9A:05:49:AC:E5:67:F3
```

Secure installation for client signer retrieval

Each profile in the WebSphere Application Server environment contains a unique self-signed certificate that was created when the profile was created. This certificate replaces the default dummy certificate that

ships with WebSphere Application Server in releases prior to version 6.1. When a profile is federated to a deployment manager, the signer for that self-signed certificate is added to the common truststore for the cell.

By default, clients do not trust servers from different profiles in the WebSphere Application Server environment. That is, they do not contain the signer for these servers. There are some things that you can do to assist in establishing this trust:

1. Enable the signer exchange prompt to except the signer during the connection attempt.
2. Run the **retrieveSigners** utility to download the signers from that system prior to making the connection.
3. Copy the trust.p12 file from the `/config/cells/<cell_name>/nodes/<node_name>` directory of the server profile to the `/etc` directory of the client. Update the SSL configuration to reflect the new file name and password, if they are different. Copying the file provides the client with a trust.p12 that contains all signers from servers in that cell. Also, you might need to perform this step for back-level clients that are still using the `DummyClientTrustFile.jks` file. In this case, you might need to change the `sas.client.props` or `soap.client.props` file to reflect the new truststore, truststore password, and truststore type (PKCS12).

For clients to perform an in-band signer exchange, you must specify the `ssl.client.props` file as a `com.ibm.SSL.ConfigURL` property in the SSL configuration. For managed clients, this is done automatically. Signers are designated either as in-band during the connection or out-of-band during runtime. You must also set the `com.ibm.ssl.enableSignerExchangePrompt` attribute to `true`.

Tip: You can configure a certificate expiration monitor to replace server certificates that are about to expire. For more information about how clients can retrieve the new signer from the configuration, see `csec_sslcertmonitoring.dita`.

Using the signer exchange prompt to retrieve signers from a client

When the client does not already have a signer to connect to a process, you can enable the signer exchange prompt if you want to be able to accept the signer during the connection attempt. The prompt displays once for each unique certificate and for each node. After the signer for the node is added, the signer remains in the client truststore. The following sample code shows the signer exchange prompt retrieving a signer from a client:

```
C:\WASX_e0540.11\AppServer\profiles\AppSrv01\bin\serverStatus -all
ADMU0116I: Tool information is being logged in file
           C:\WASX_e0540.11\AppServer\profiles\AppSrv01\logs\serverStatus.log
ADMU0128I: Starting tool with the AppSrv01 profile
ADMU0503I: Retrieving server status for all servers
ADMU0505I: Servers found in configuration:
ADMU0506I: Server name: dmgr
```

```
*** SSL SIGNER EXCHANGE PROMPT ***
SSL signer from target host 192.168.1.5 is not found in truststore
C:\WebSphere\AppServer\profiles\AppSrv01\etc\trust.p12.
```

Here is the signer information (verify the digest value matches what is displayed at the server):

```
Subject DN:   CN=myhost.austin.ibm.com, O=IBM, C=US
Issuer DN:   CN=myhost.austin.ibm.com, O=IBM, C=US
Serial number: 1128544457
Expires:     Thu Oct 05 15:34:17 CDT 2006
SHA-1 Digest: 91:A1:A9:2D:F2:7D:70:0F:04:06:73:A3:B4:A4:9C:56:9D:A8:A3:BA
MD5 Digest:  88:72:C5:88:00:1C:A7:FA:D6:EB:04:88:AC:A1:C9:13
```

Add signer to the truststore now? (y/n) y

A retry of the request may need to occur.

```
ADMU0508I: The Deployment Manager "dmgr" is STARTED
```

To automate this process, see `rxml_retrievesigners.dita`.

When a prompt occurs to accept the signer, a socket timeout can occur and the connection might be broken. For this reason, the message `A retry of the request may need to occur.` displays after answering the prompt. The message informs the user to resubmit the request. This problem should not happen frequently, and it might be more prevalent for some protocols than others.

Verify the displayed SHA-1 digest, which is the signature of the certificate that is sent by the server. If you look at the certificate on the server, verify that the same SHA-1 digest displays.

You can disable the prompt when you do not want it to display by running the **retrieveSigners** utility to retrieve all of the signers for a particular cell. You can download or upload the signers from any remote keystore to any local keystore by referencing a common truststore with this client script. For more information, see `csec_ssldefselfsigncertconf.dita`.

Using the retrieveSigners utility to download signers for a client

You can run the **retrieveSigners** utility to retrieve all of the signers from the remote keystore for a specified client keystore. The truststore contains the signers that enable the client to connect to its processes. The **retrieveSigners** utility can point to any keystore in the target configuration, within the scope of the target process, and can download the signers (certificate entries only) to any client keystore in the `ssl.client.props` file.

Obtaining signers for clients and servers from a previous release

When a client from a release prior to version 6.1 connects to the current release, the client must obtain signers for a successful handshake. Clients using previous releases of WebSphere Application Server cannot obtain signers as easily as in the current release. You can copy the deployment manager *common* truststore to your back-level client or server, and then re-configure the SSL configuration to directly reference that truststore. This common truststore of type PKCS12 is located in the `/config/cells/<cell_name>/nodes/<node_name>` directory in the configuration repository and has a default password of WebAS.

To collect all of the signers for the cell in a single `trust.p12` keystore file, complete following steps:

1. Copy the `trust.p12` keystore file on the server and replicate it on the client. The client references the file directly from the `sas.client.props` and `soap.client.props` files that specify the SSL properties for previous releases.
2. Change the client-side keystore password so that it matches the default cell name that is associated with the copied keystore.
3. Change the default keystore type for the `trust.p12` file to PKCS12 in the client configuration.

The following two code samples show you a before and an after view of the changes to make.

Default SSL configuration of `sas.client.props` for a previous release

```
com.ibm.ssl.protocol=SSL
com.ibm.ssl.keyStore=file:///
C\:/SERV1_601_0208/AppServer/profiles/AppSrv01/etc/DummyClientKeyFile.jks
com.ibm.ssl.keyStorePassword={xor}CDo9Hgw\=
com.ibm.ssl.keyStoreType=JKS
com.ibm.ssl.trustStore=file:///
C\:/SERV1_601_0208/AppServer/profiles/AppSrv01/etc/DummyClientTrustFile.jks
com.ibm.ssl.trustStorePassword={xor}CDo9Hgw\=
com.ibm.ssl.trustStoreType=JKS
```

SSL configuration changes that are required to common truststore file in the `/etc` directory of the client

```

com.ibm.ssl.protocol=SSL
com.ibm.ssl.keyStore=file:///
C:/SERV1_601_0208/AppServer/profiles/AppSrv01/etc/DummyClientKeyFile.jks
com.ibm.ssl.keyStorePassword={xor}CDo9Hgw\=
com.ibm.ssl.keyStoreType=JKS
com.ibm.ssl.trustStore=file:///
C:/SERV1_601_0208/AppServer/profiles/AppSrv01/etc/trust.p12
com.ibm.ssl.trustStorePassword=myhostNode01Cell
com.ibm.ssl.trustStoreType=PKCS12

```

You can also make these changes in the `soap.client.props` file and specify the `key.p12` file in place of the `DummyClientKeyFile.jks` file. However, you must also change the `keyStorePassword` and `keyStoreType` values to match those in the default `key.p12` file.

In releases of WebSphere Application Server prior to version 6.1, you must edit the SSL configuration on the server to replace the common truststore. The `trust.p12` file, which is used by the server, also must contain the default dummy certificate signer for connections among servers at previous release levels. You might need to manually extract the default certificate from the `DummyServerKeyFile.jks` file and then import the certificate into the `trust.p12` file that you added to the configuration.

retrieveSigners command: The **retrieveSigners** command creates a new client self-signed certificate, keystore, and SSL configuration in the `ssl.client.props` file. Using this command you can optionally extract the signer to a file.

For more information about where to run this command, see the [Using command tools](#) article.

Syntax

The command syntax is as follows:

```
retrieveSigners <remoteKeyStoreName> <localKeyStoreName> [options]
```

where the `<remoteKeyStoreName>` and `<localKeyStoreName>` parameters are required. The optional parameters include the following:

```

[-remoteAlias aliasFromRemoteStore]
[-localAlias storeAsAlias]
[-listRemoteKeyStoreNames] [-listLocalKeyStoreNames]
[-autoAcceptBootstrapSigner] [-uploadSigners] [-host host]
[-port port] [-conntype RMI|SOAP] [-user user]
[-password password]
[-trace] [-logfile filename]
[-replaceLog] [-quiet] [-help]

```

Parameters

The following parameters are available for the **retrieveSigners** command:

-remoteKeyStoreName

The name of a truststore that is located in the server configuration from which to retrieve the signers. This will typically be the `CellDefaultTrustStore` file for an managed environment or the `NodeDefaultTrustStore` file for an unmanaged environment.

-localKeyStoreName

The name of the truststore that is located in the `ssl.client.props` file for the profile to which the retrieved signers is added. This will typically be the `ClientDefaultTrustStore` file for either a managed or unmanaged environment.

-remoteAlias <aliasFromRemoteStore>

Specifies one alias from the remote truststore that you want to retrieve. Otherwise, all signers from the remote truststore will be retrieved.

-localAlias <storeAsAlias>

Determines the name of the alias stored in the local truststore. This option is only valid if you specify the `-remoteAlias` option. If you do not specify the `-localAlias` option, the alias name from the remote truststore will be used, if possible. If an alias clash occurs, the alias name will be used and it will have an incremented number appended to the end of it until it finds a unique alias.

-listRemoteKeyStoreNames

Sends a remote request to the server to list all keystores that you can specify for the `remoteKeyStoreName` parameter. Use this command when you are unsure of the name of the remote truststore that you want to download the signers from.

-listLocalKeyStoreNames

Lists the keystores located in the `ssl.client.props` file that you can specify for the `localKeyStoreName` parameter. This truststore will receive the signers from the server. Use this parameter when you are unsure of the name of the local truststore that you want to retrieve the signers into. The default name of the truststore is `ClientDefaultTrustStore` and is located in the `ssl.client.props` file.

-autoAcceptBootstrapSigner

Automatically adds a signer in order to make a secure connection to the server. The purpose of the option is to allow automation of the command so that you do not need to accept the signer. After the signer is added to the local truststore, a SHA hash will print so that you can verify the certificate.

-uploadSigners

Converts the signer download into a signer upload. The signers from the `localKeyStoreName` parameter will be sent to the `remoteKeyStoreName` parameter instead.

-host <host>

Specifies the target host from which the signers will be retrieved.

-port <port>

Specifies the target administrative port to which to connect. You must specify the port based on the `-conntype` parameter. If the `conntype` is `SOAP`, the default port is 8879. This can vary for different servers. If the `conntype` is `RMI`, the default port is 2809. This can vary for different servers.

-conntype <RMISoap>

Determines the administrative connector type that is used for the MBean call to retrieve the signers.

-user <user>

When global security is enabled, you can specify this option to supply the user name that will be authenticated for the MBean operation. This must be an identity with administrator authority. If you do not specify this parameter when global security is enabled, you will be prompted for credentials by default.

-password <password>

When global security is enabled, you can specify this option to supply the password that will be authenticated for the MBean operation. The password goes along with the `-user` parameter.

-trace

When specified, this enables tracing of the trace specification necessary to debug this component. By default, the trace will appear in the `profiles/profile_name/log/retrieveSigners.log` file.

-logfile <filename>

Overrides the default trace file. By default, the trace will appear in the `profiles/profile_name/log/retrieveSigners.log` file.

-replacelog

Causes the existing trace file to be replaced when the command is executed.

-quite

Suppresses most messages from printing out on the console.

-help

Prints a usage statement.

-? Prints a usage statement.

Usage scenario

The following examples demonstrate correct syntax:

- The following example lists remote and local keystores:

```
retrieveSigners.bat -listRemoteKeyStoreNames -listLocalKeyStoreNames -conntype RMI -port 2809 [Windows]
```

```
retrieveSigners.sh -listRemoteKeyStoreNames -listLocalKeyStoreNames -conntype RMI -port 2809 [Unix]
```

Example output

```
CWPKI0306I: The following remote keystores exist on the specified server:  
CMSKeyStore, NodeLTPAKeys, NodeDefaultTrustStore, NodeDefaultKeyStore
```

```
CWPKI0307I: The following local keystores exist on the client:  
ClientDefaultKeyStore, ClientDefaultTrustStore
```

- The following example retrieves all signers from NodeDefaultTrustStore:

```
retrieveSigners.bat NodeDefault TrustStore ClientDefaultTrustStore -autoAcceptBootstrapSigner  
-conntype RMI -port 2809 [Windows]
```

```
retrieveSigners.sh NodeDefault TrustStore ClientDefaultTrustStore -autoAcceptBootstrapSigner  
-conntype RMI -port 2809 [Unix]
```

Example output

```
CWPKI0308I: Adding signer alias "CN=BIRKT40.austin.ibm.com, O=IBM, C=US" to  
local keystore "ClientDefaultTrustStore" with the following SHA  
digest: 40:20:CF:BE:B4:B2:9C:F0:96:4D:EE:E5:14:92:9E:37:8D:51:A5:47
```

Certificate expiration monitoring

The certificate expiration monitor administrative task cycles through all the keystores that are configured in the security.xml file and reports on any certificates that expire within a specified threshold, which is typically within 30 days.

The default self-signed certificate on each node expires 365 days after creation. You can modify the certificate validity period by changing the default value for the `com.ibm.ssl.defaultCertReqDays=365` property in the `ssl.client.props` global property area for clients. You can also specify this property as a security custom property on the administrative console. Click **Security > Secure administration, applications, and infrastructure > Custom properties**.

You can configure this monitor task to run according to a particular schedule. The schedule produces the next start date that persists in the configuration and, when the date is reached, WebSphere Application Server starts the monitor to check all of the keystores for certificates that meet the expiration threshold. You can start the task manually to run at any time.

The following security.xml configuration object specifies when the monitor task starts, determines the certificate expiration threshold, and indicates whether you are notified in an e-mail using Simple Mail Transfer Protocol (SMTP) or in a message log.

```
<wsCertificateExpirationMonitor xmi:id="WSCertificateExpirationMonitor_1"  
name="Certificate Expiration Monitor" daysBeforeNotification="30"  
isEnabled="true" autoReplace="true" deleteOld="true"  
wsNotification="WSNotification_1" wsSchedule="WSSchedule_2"  
nextStartDate="1134358204849"/>
```

The expiration monitor automatically replaces only self-signed certificates that meet the expiration threshold criteria. To replace all of the signers from the old certificate with the signer that belongs to the new certificate in all the keystores in the configuration for that cell, set the `autoReplace` attribute to `true`. When the `deleteOld` attribute is `true`, the old personal certificate and old signers also are deleted from the keystores. The `isEnabled` attribute determines whether the expiration monitor task runs based upon the `nextStartDate` attribute that is derived from the schedule. The `nextStartDate` attribute is derived from the schedule in milliseconds since 1970, and is identical to the `System.currentTimeMillis()`. If the `nextStartDate`

has already passed when an expiration monitor process begins, and the expiration monitor is enabled, the task is started, but a new nextStartDate value is established based on the schedule.

The following sample code shows the frequency attribute as the number of days between each run.

```
<wsSchedules xmi:id="WSSchedule_2" name="ExpirationMonitorSchedule"
frequency="30" dayOfWeek="1" hour="21" minute="30"/>
```

The dayOfWeek attribute adjusts the schedule to run on a specified day of the week, which is always the same day regardless of whether the frequency is set to 30 or 31 days. Based on 24-hour clock, the hour and minute attributes determine when the expiration monitor is started on the specified day.

The following sample code shows the notification configuration, which notifies you after the expiration monitor runs.

```
<>wsNotifications xmi:id="WSNotification_1" name="MessageLog" logToSystemOut="true" emailList=""/
```

For expiration monitor notifications, you can select message log, e-mail using SMTP server, or both methods of notification. When you configure the e-mail option, use the format user@domain@smtpserver. If you do not specify an SMTP server, WebSphere Application Server defaults to the same domain as the e-mail address. For example, if you configure joeuser@ibm.com, WebSphere Application Server attempts to call smtp-server.ibm.com. To specify multiple e-mail addresses using scripting, you must add a pipe (|) character between entries. When you specify the logToSystemOut attribute, the expiration monitor results are sent to the message log for the environment, which is typically the SystemOut.log file.

Web server plug-in default configuration

When you create a new Web server definition, WebSphere Application Server associates the Web server plug-in with a Certificate Management Services (CMS) keystore for a specific node. The keystore contains all of the signers for the current cell with the self-signed certificate, which belongs to the node. The plug-in can communicate securely to WebSphere Application Server, even when the plug-in is configured with Secure Sockets Layer (SSL) client authentication enabled.

When you set the Web server definition to webserver1 on node myhostNode01, WebSphere Application Server creates the keystore configuration. The keystore is scoped to the webserver1 server, which makes it visible to this server only. Other processes cannot use this keystore definition.

The following sample code from the security.xml file shows the configuration entries for the Web server plug-in.

```
<keyStores xmi:id="KeyStore_1132357815719" name="CMSKeyStore"
password="{xor}HRYNFAtRbxEw0zpvbhw6MzM=" provider="IBMCMSProvider"
Location="C:\WASX_e0540.11\AppServer\profiles\AppSrv01/config/cells
/myhostCell01/nodes/myhostNode01/servers/webserver1/plugin-key.kdb"
type="CMSKS" fileBased="true" createStashFileForCMS="true"
managementScope="ManagementScope_1132357815718"/>
<managementScopes xmi:id="ManagementScope_1132357815718" scopeName="
(cell):myhostCell01:(node):myhostNode01:(Server):webserver1" scopeType="server"/>
```

The following sample code shows how the CMS keystore and stash file are generated in the security.xml file.

```
C:\WebSphere\AppServer\profiles\Dmgr01\config\cells\myhostCell01\nodes
\myhostNode01\servers\webserver1\plugin-key.kdb
C:\WebSphere\AppServer\profiles\Dmgr01\config\cells\myhostCell01
\nodes\myhostNode01\servers\webserver1\plugin-key.sth
```

The default password for the keystore is WebAS. You can change the default keystore password by using either the administrative console or the appropriate **AdminTask** command. The following sample code shows the **AdminTask** command that you can use to create this CMS keystore.

```
$AdminTask createCMSKeyStore /config/cells/myhostCell01/nodes/myhostNode01
/servers/webserver1/plugin-key.kdb myhost.austin.ibm.com
```

Note the following characteristics of the previous example:

- You can create only one CMSKeyStore entry for each management scope. If a CMS keystore already exists for scope (cell):myhostCell01:(node):myhostNode01:(server):webserver1, then you cannot create another CMSKeyStore entry
- The Uniform Resource Identifier (URI) for the keystore name is /config/cells/myhostCell01/nodes/myhostNode01/servers/webserver1/plugin-key.kdb
- The host name in the plug-in location is myhost.austin.ibm.com. WebSphere Application Server uses this name to create a self-signed certificate, if a self-signed certificate does not already exist for that particular node. If a self-signed certificate already exists for the node, then the certificate is put into the CMS keystore and all the signers from the cell are added, by default.

When additional nodes are federated, the signers for these nodes are not automatically added to each Web server for the CMS keystore. For the Web server plug-in to be able to communicate with a newly federated node, you must manually exchange signers with the CMSKeyStore keystore. Use the administrative console keystore certificate management function to exchange signers. For more information, see “Extracting a signer certificate from a personal certificate” on page 470.

Dynamic configuration updates

During the Secure Sockets Layer (SSL) runtime, dynamic configuration updates affect both inbound and outbound SSL endpoints. For inbound SSL endpoints, the changes that are implemented by the SSL channel are only affected by dynamic changes. For outbound SSL endpoints, all outbound connections inherit the new configuration changes.

In this release, dynamic update functionality provides you with greater flexibility and efficiency. You can change SSL configurations without restarting WebSphere Application Server for the changes to take effect.

To make dynamic changes, in the administrative console click **Security > SSL certificates and key management**, then select the **Dynamically update the runtime when SSL configuration changes occur** check box. You must save your changes and then synchronize the security.xml file with remote systems. A remote system must be able to confirm that dynamicallyUpdateSSLConfig=true is in the security.xml file.

The SSL runtime reloads the modified SSL configuration and creates a new SSLEngine for the modified connections that are associated with inbound endpoints. New outbound connections use the new configuration while existing connections continue to use the old SSLEngine object and are not affected.

Tip: Make dynamic changes to the SSL configuration during off-peak hours. Synchronization delays can negatively affect connections when you update SSL configurations during peak hours.

You can turn on and off the dynamicallyUpdateSSLConfig attribute in the security.xml file to ensure successful updates by doing the following actions:

1. Set dynamicallyUpdateSSLConfig=On.
2. Save the updated configuration.
3. Synchronize the security.xml file with remote systems.
4. Set the dynamicallyUpdateSSLConfig attribute to Off.

You must verify that all of the nodes receive the changes before turning off the dynamicallyUpdateSSLConfig attribute. Test the changes in a test environment before updating the production environment.

Tip: Some SSL changes, especially administrative SSL changes, can cause server outages if you fail to test them first. When a change prevents trust between two endpoints, the endpoints cannot communicate with each other. Additionally, if administrative SSL connection updates cause system

outages, you might need to disable the nodes after you make corrective changes using the deployment manager. From the command line, you can manually synchronize the server to retrieve the new SSL changes, then restart the nodes.

Restriction: In this release, the Object Request Broker (ORB) inbound SSL socket factory and the Admin SOAP inbound SSL socket factory are not affected by dynamic configuration changes. SSL socket factories cannot be bound to a port again without affecting the existing connections, with the exception of the SSL channel. You must restart the server to make a change to the SSL configuration for these two inbound protocols. Outbound configurations can still be changed dynamically. These changes are affected because the sockets are created during each new outbound connection.

Certificate management using iKeyman

Starting in WebSphere Application Server Version 6.1, you can manage your certificates from the administrative console. When using versions of WebSphere Application Server prior to Version 6.1, use iKeyman for certificate management. iKeyman is a key management utility.

WebSphere Application Server certificate management requires that you define the keystores in your WebSphere Application Server configuration. With iKeyman, you need access to the keystore file only. You can read a keystore file with personal certificates and signers that is created in iKeyman can be read into the WebSphere Application Server configuration by using the **createKeyStore** command.

The majority of certificate management functions are the same between WebSphere Application Server and iKeyman, especially for personal certificates and signer certificates. However, certificate requests are special. The underlying behavior is different in the two certificate management schemes. Because of the different behavior, when a certificate request is generated from iKeyman, the process must be completed in iKeyman. For example, a certificate that is generated by a certificate request that originated in iKeyman must be received in iKeyman as well.

The same is true for WebSphere Application Server. For example, when a certificate is generated from a certificate request that originated in WebSphere Application Server, the certificate must be received in WebSphere Application Server.

You can perform the following certificate operations using iKeyman:

Types of certificates	Functions	Description
Personal certificates	Create a self-signed certificate	Creates a self-signed certificate and stores it in a keystore.
	List personal certificates	Lists all the personal certificates in a keystore.
	Get information about a personal certificate	Gets information about a personal certificate.
	Delete a personal certificate	Deletes a personal certificate from a keystore.
	Import a certificate	Imports a certificate from a keystore to a keystore.
	Export a certificate	Exports a certificate from a keystore to another keystore.
	Extract a certificate	Extracts the signer part of a personal certificate to a file.
Signer certificates	Receive a certificate	Reads a certificate that comes from a certificate authority (CA) into a keystore.
	Add a signer certificate	Adds a signer certificate from a file to a keystore.
	List signer certificates	Lists all the signer certificates in a keystore.

Types of certificates	Functions	Description
	Get information about a signer certificate	Gets information about a signer certificate.
	Delete a signer certificate	Deletes a signer certificate from a keystore.
	Extract a signer certificate	Extracts a signer certificate from a keystore, and stores the certificate in a file.
Certificate requests	Create a certificate request	Creates a certificate request that can be sent to a CA.
	List certificate requests	Lists the certificate requests in a keystore.
	Get information about a certificate request	Gets information about a certificate request.
	Delete a certificate request	Deletes a certificate request from a keystore.
	Extract a certificate request	Extracts a certificate request to a file.

Certificate management

You can manage certificate operations that involve personal certificates, signer certificates, and personal certificate requests on the administrative console.

Types of certificates

WebSphere Application Server uses the certificates that reside in keystores to establish trust for a Secure Sockets Layer (SSL) connection. Click **Security > SSL certificate and key management > Manage endpoint security configurations > Inbound | Outbound > *SSL_configuration_name* > Key stores and certificates**, then select an existing or create a new keystore. After selecting a keystore, and depending on the type of certificate you need, choose one of the following types of certificates under Related Items:

- Personal certificate
- Signer certificate
- Personal certificate request

The following table describes the certificate operations that you can perform on the administrative console:

Types of certificates	Functions	Description
Personal certificates	Create a self-signed certificate	Creates a self-signed certificate and stores it in a keystore.
	List personal certificates	Lists all the personal certificates in a keystore.
	Get information about a personal certificate	Gets information about a personal certificate.
	Delete a personal certificate	Deletes a personal certificate from a keystore.
	Import a certificate	Imports a certificate from a keystore to a keystore.
	Export a certificate	Exports a certificate from a keystore to another keystore.
	Extract a certificate	Extracts the signer part of a personal certificate to a file.
	Exchange signer certificates	Exchange signer part of a personal certificate between key store.
	Receive a certificate	Reads a certificate that comes from a certificate authority (CA) into a keystore.

Types of certificates	Functions	Description
Signer certificates	Replace a certificate	Replaces all occurrences of a personal certificate alias in the WebSphere Application Server configuration with another certificate. Also, replaces all occurrences of the personal certificates signer with the new personal certificate signer.
	Add a signer certificate	Adds a signer certificate from a file to a keystore.
	List signer certificates	Lists all the signer certificates in a keystore.
	Get information about a signer certificate	Gets information about a signer certificate.
	Delete a signer certificate	Deletes a signer certificate from a keystore.
	Extract a signer certificate	Extracts a signer certificate from a keystore, and stores the certificate in a file.
Certificate requests	Retrieve a signer from a port	Retrieves a signer certificate from a port, and stores it in a key store.
	Create a certificate request	Creates a certificate request that can be sent to a CA.
	List certificate requests	Lists the certificate requests in a keystore.
	Get information about a certificate request	Gets information about a certificate request.
	Delete a certificate request	Deletes a certificate request from a keystore.
	Extract a certificate request	Extracts a certificate request to a file.

Personal certificates

The following table lists the operations that you can perform on personal certificates, the AdminTask object that you can use to perform that operation, and how to navigate to the certificate on the console:

Function	AdminTask object	Administrative console
Create a self-signed certificate	createSelfSignedCertificate	Security > Secure Communications > Key store and certificates > key store > Create a Self-Signed Certificate
List personal certificates	listPersonalCertificates	Security > Secure Communications > Key store and certificates > key store > personal certificates
Get information about a personal certificate	getPersonalCertificate	Security > Secure Communications > Key store and certificates > key store > personal certificates > alias
Delete a personal certificate	deletePersonalCertificate	Security > Secure Communications > Key store and certificates > key store > personal certificates > delete
Import a certificate	importCertificate	Security > Secure Communications > Key store and certificates > key store > personal certificates > import
Export a certificate	exportCertificate	Security > Secure Communications > Key store and certificates > key store > personal certificates > export
Extract a certificate	extractCertificate	Security > Secure Communications > Key store and certificates > key store > personal certificates > extract
Exchange signer certificates	exchangeSignerCertificates	Security > Secure Communications > Key store and certificates > Exchange signers

Signer certificates

The following table lists the operations that you can perform with signer certificates, the AdminTask object that you can use to perform the operation, and how to navigate to the certificate on the console:

Function	AdminTask object	Administrative console
Add a signer certificate	addSignerCertificate	Security > Secure communications > Key store and certificates > key store > signer certificates > Add
List signer certificates	listSignerCertificates	Security > Secure communications > Key store and certificates > key store > signer certificates
Get information about a signer certificate	getSignerCertificate	Security > Secure communications > Key store and certificates > key store > signer certificates > alias
Delete a signer certificate	deleteSignerCertificate	Security > Secure communications > Key store and certificates > key store > signer certificate > delete
Extract a signer certificate to a file	extractSignerCertificate	Security > Secure communications > Key store and certificates > key store > signer certificates > extract
Retrieve a signer certificate from a port	retrieveSignerFromPort	Security > Secure communications > Key store and certificates > key store > signer certificates > retrieve from port

Personal certificate requests

The following table lists the operations that you can perform on personal certificate requests, the AdminTask object that you can use to perform that operation, and how to navigate to the certificate request on the console:

Function	AdminTask object	Administrative console
Create a personal certificate request	createCertificateRequest	Security > Secure communications > Key store and certificates > key store > Personal certificate Requests > Add
List personal certificate requests	listCertificateRequests	Security > Secure communications > Key store and certificates > key store > Personal certificate requests
Get information about a personal certificate request	getCertificateRequest	Security > Secure communications > Key store and certificates > key store > Personal certificate requests > alias
Delete a personal certificate request	deleteCertificateRequest	Security > Secure communications > Key store and certificates > key store > Personal certificate requests > delete
Extract a personal certificate request to a file	extractCertificateRequest	Security > Secure communications > Key store and certificates > key store > Personal certificate requests > Extract

Creating a Secure Sockets Layer configuration

Secure Sockets Layer (SSL) configurations contain the attributes that you need to control the behavior of client and server SSL endpoints. You create SSL configurations with unique names within specific management scopes on the inbound and outbound tree in the configuration topology. This task shows you how to define SSL configurations, including quality of protection and trust and key manager settings.

You must decide at which scope you need to define an SSL configuration, for instance, the cell, node group, node, server, cluster, or endpoint scope, from the least specific to the most specific scope. When you define an SSL configuration at the node scope, for example, only those processes within that node can load the SSL configuration; however, any processes at the endpoint in the cell can use an SSL configuration at the cell scope, which is higher in the topology.

You must also decide which scope to associate with the new SSL configuration, according to the processes that the configuration affects. For example, an SSL configuration for a hardware cryptographic device might require a keystore that is available only on a specific node, or you might need an SSL configuration for a connection to a particular SSL host and port. For more information, see “Dynamic outbound selection of Secure Sockets Layer configurations” on page 395.

Complete the following steps in the administrative console:

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations**.
2. Select an SSL configuration link on either the Inbound or Outbound tree, depending on the process you are configuring.
 - If the scope is already associated with a configuration and alias, the SSL configuration alias and certificate alias are noted in parentheses.
 - If the parenthetical information is not included, then the scope is not associated. Instead, the scope inherits the configuration properties of the first scope above it that is associated with an SSL configuration and certificate alias.

The cell scope must be associated with an SSL configuration because it is at the top of the topology and represents the default SSL configuration for the inbound or outbound connection.

3. Click **SSL configurations** under Related Items. You can view and select any of the SSL configurations that are configured at this scope. You can also view and select these configuration at every scope that is lower on the topology.
4. Click **New** to display the SSL configuration panel. You cannot select links under Additional Properties until you type a configuration name and click **Apply**.
5. Type an SSL configuration name. This field is required. The configuration name is the SSL configuration alias. Make the alias name unique within the list of SSL configuration aliases that are already created at the selected scope. The new SSL configuration uses this alias for other configuration tasks.
6. Select a truststore name from the drop-down list. A truststore name refers to a specific truststore that holds signer certificates that validate the trust of certificates sent by remote connections during an SSL handshake. If there is no truststore in the list, see “Creating a keystore configuration” on page 452 to create a new truststore, which is a keystore whose role is to establish trust during the connection.
7. Select a keystore name from the drop-down list. A keystore contains the personal certificates that represent a signer identity and the private key that WebSphere Application Server uses to encrypt and sign data.
 - If you change the keystore name, click **Get certificate aliases** to refresh the list of certificates from which you can choose a default alias. WebSphere Application Server uses a server alias for inbound connections and a client alias for outbound connections.
 - If there is no keystore in the list, see “Creating a keystore configuration” on page 452 to create a new keystore.
8. Choose a default server certificate alias for inbound connections. Select the default only when you have not specified an SSL configuration alias elsewhere and have not selected a certificate alias. A centrally managed SSL configuration tree can override the default alias. For more information, see “Central management of Secure Sockets Layer configurations” on page 396.
9. Choose a default client certificate alias for outbound connections. Select the default only when the server SSL configuration specifies an SSL client authentication.

10. Review the identified management scope for the SSL configuration. Make the management scope in this field identical to the link you selected in Step 2. If you want to change the scope, you must click a different link in the topology tree and continue at Step 3.
11. Click **Apply** if you intend to configure Additional Properties. If not, go to Step 24.
12. Click **Quality of protection (QoP) settings** under Additional Properties. QoP settings define the strength of the SSL encryption, the integrity of the signer, and the authenticity of the certificate.
13. Select a client authentication setting to establish an SSL configuration for inbound connections and for clients to send their certificates, if appropriate.
 - If you select **None**, the server does not request that a client send a certificate during the handshake.
 - If you select **Supported**, the server requests that a client send a certificate. However, if the client does not have a certificate, the handshake might still succeed.
 - If you select **Required**, the server requests that a client send a certificate. However, if the client does not have a certificate, the handshake fails.

Important: The signer certificate that represents the client must be in the truststore that you select for the SSL configuration. By default, servers within the same cell trust each other because they use the common truststore, `trust.p12`, that is located in the cell directory of the configuration repository. However, if you use keystores and truststores that you create, perform a signer exchange before you select either **Supported** or **Required**.

14. Select a protocol for the SSL handshake.
 - The default protocol, `SSL_TLS`, supports client protocols `TLSv1`, `SSLv3`, and `SSLv2`.
 - The `TLSv1` protocol supports `TLS` and `TLSv1`. The SSL server connection must support this protocol for the handshake to proceed.
 - The `SSLv3` protocol supports `SSL` and `SSLv3`. The SSL server connection must support this protocol for the handshake to proceed.

Important: Do not use the `SSLv2` protocol for the SSL server connection. Use it only when necessary on the client side.

15. Select one of the following options:
 - A predefined Java Secure Socket Extension (JSSE) provider. The `IBMJSSE2` provider is recommended for use on all platforms which support it. It is required for use by the channel framework SSL channel. When Federal Information Processing Standard (FIPS) is enabled, `IBMJSSE2` is used in combination with the `IBMJCEFIPS` crypto provider.
 - A custom JSSE provider. Type a provider name in the **Custom provider** field.
16. Select from among the following cipher suite groups:
 - **Strong:** WebSphere Application Server can perform 128-bit confidentiality algorithms for encryption and support integrity signing algorithms. However, a strong cipher suite can affect the performance of the connection.
 - **Medium:** WebSphere Application Server can perform 40-bit encryption algorithms for encryption and support integrity signing algorithms.
 - **Weak:** WebSphere Application Server can support integrity signing algorithms but not to perform encryption. Select this option with care because passwords and other sensitive information that cross the network are visible to an Internet Protocol (IP) sniffer.
 - **Custom:** you can select specific ciphers. Any time you change the ciphers that are listed from a specific cipher suite group, the group name changes to `Custom`.
17. Click **Update selected ciphers** to view a list of the available ciphers for each cipher strength.
18. Click **OK** to return to the new SSL configuration panel.
19. Click **Trust and key managers** under Additional Properties.
20. Select a default trust manager for the primary SSL handshake trust decision.

- Choose **lbmPKIX** when you require certificate revocation list (CRL) checking using CRL distribution points in the certificates.
 - Choose **lbmX509** when you do not require CRL checking but do need increased performance. You can configure a custom trust manager to perform CRL checking, if necessary.
21. Define a custom trust manager, if appropriate. You can define a custom trust manager that runs with the default trust manager you select. The custom trust manager must implement the JSSE `javax.net.ssl.X509TrustManager` interface and, optionally, the `com.ibm.wsspi.ssl.TrustManagerExtendedInfo` interface to obtain product-specific information.
 - a. Click **Security > SSL certificate and key management > Manage endpoint security configurations > SSL_configuration > Trust and key managers > Trust managers > New**.
 - b. Type a unique trust manager name.
 - c. Select the **Custom** option.
 - d. Type a class name.
 - e. Click **OK**. When you return to the Trust and key managers panel, the new custom trust manager displays in the **Additional ordered trust managers** field. Use the left and right list boxes to add and remove custom trust managers.
 22. Select a key manager for the SSL configuration. By default, `lbmX509` is the only key manager unless you create a custom key manager.

Important: If you choose to implement your own key manager, you can affect the alias selection behavior because the key manager is responsible for selecting the certificate alias from the keystore. The custom key manager might not interpret the SSL configuration as the WebSphere Application Server key manager `lbmX509` does. To define a custom key manager, click **Security > Secure communications > SSL configurations > SSL_configuration > Trust and key managers > Key managers > New**.

23. Click **OK** to save the trust and key manager settings and return to the new SSL configuration panel.
24. Click **Save** to save the new SSL configuration.

Important: You can override the default trust manager when you configure at least one custom trust manager and set the `com.ibm.ssl.skipDefaultTrustManagerWhenCustomDefined` property to `true`. Click **Custom Property** on the SSL configuration panel. However, if you change the default, you leave all the trust decisions to the custom trust manager, which is not recommended for production environments. In test environments, use a dummy trust manager to avoid certificate validation. Remember that these environment are not secure.

In this release of WebSphere Application Server, you can associate SSL configurations with protocols using one of the following methods:

- Set the SSL configuration on the thread programmatically
- Associate the SSL configuration with an outbound protocol, and target host and port. For more information, see “Associating a Secure Sockets Layer configuration dynamically with an outbound protocol and remote secure endpoint” on page 434
- Associate the SSL configuration directly using the alias. For more information, see `tsec_sslselconfigdirect.dita`
- Manage the SSL configurations centrally by associating them with SSL configuration groups or zones that are scoped for endpoints. For more information, see “Associating Secure Sockets Layer configurations centrally with inbound and outbound scopes” on page 437

SSL certificate and key management

Use this page to configure security for Secure Socket Layer (SSL) and key management, certificates, and notifications. The SSL protocol provides secure communications between remote server processes or

endpoints. SSL security can be used for establishing communications inbound to and outbound from an endpoint. To establish secure communications, a certificate and an SSL configuration must be specified for the endpoint.

To view this administrative console page, click **Security > SSL certificate and key management**.

Configuration settings

Specifies the following administrative console tasks:

- Manage endpoint security configurations
- Manage certificate expiration

Use Federal Information Processing Standard (FIPS) algorithms

Specifies the Federal Information Processing Standard (FIPS)-compliant Java cryptography engine is enabled.

- Does not affect the SSL cryptography that is performed by the application server for z/OS System Secure Sockets Layer (SSSL).
- Does not change the JSSE provider if this cell includes any Application Server versions before the application server for z/OS Version 6.0.x.

When you select the **Use the Federal Information Processing Standard (FIPS)** option, the Lightweight Third Party Authentication (LTPA) implementation uses IBMJCEFIPS. IBMJCEFIPS supports the Federal Information Processing Standard (FIPS)-approved cryptographic algorithms for Data Encryption Standard (DES), Triple DES, and Advanced Encryption Standard (AES). Although the LTPA keys are backwards compatible with prior releases of the application server, the LTPA token is not compatible with prior releases. In prior releases, the application server did not generate the LTPA token using a FIPS-approved algorithm.

The IBMJSSE2 JSSE provider does not perform cryptographic functions directly, and therefore does not need to be FIPS-approved. Instead, the IBMJSSE2 JSSE provider uses the JCE framework for cryptographic functions and uses IBMJCEFIPS when FIPS mode is enabled.

Important:  The IBMJSSEFIPS provider is not supported on the HP-UX platform. However, the IBMJSSE2 provider, which uses IBMJCEFIPS, is supported on the HP-UX platform.

Default: Disabled

Dynamically update the runtime when SSL configuration changes occur

Specifies that all of the SSL-related attributes that change must be read from the configuration dynamically after they have been saved, then reused for new connections. To avoid customer impact, it is recommended that changes to production servers be made during off-peak periods.

Default: Disabled

When this option is selected, the configuration is updated each time you configure an SSL communication.

SSL configurations for selected scopes

Use this page to display Secure Socket Layer (SSL) configurations for selected scopes, such as a cell, node, server, or cluster. From this page you can navigate to configuration panels for the following: SSL configurations, dynamic inbound and outbound endpoint SSL configurations, key stores, key sets, key set groups, key managers, and trust managers.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > ssl_configuration**.

Name

Specifies the SSL configuration scope, which is derived from the selected object in the hierarchy.

Data type: Text

Direction

Specifies the direction for which the SSLConfig applies. Inbound refers to any listener port. Outbound refers to outbound end point connections.

Data type: Text

Inherited SSL configuration name

Specifies the name of the SSL configuration that is inherited from a higher level scope

This field displays for server and nodegroups within the object hierarchy.

This field does not display for the top-level node.

Data type: Text

Inherited certificate alias

Specifies the certificate alias that is inherited from a higher-level scope.

This field displays for server and node groups within the object hierarchy.

This field does not display for the top-level node.

Data type: Text

Override inherited values

Specifies the SSL configuration to be used for this scope and any lower scopes that have not already designated an SSL configuration.

This field displays for server and node groups within the object hierarchy.

This field does not display for the top-level node.

Default: Disabled

SSL configuration

Specifies the SSL configuration that is used by requests at this scope.

Data type: Text

Update certificate alias list

Specifies the certificate aliases contained in the key store for this SSL configuration can be selected from the **Certificate alias in key store** list. You must update the certificate list after choosing a different SSL configuration alias. If you do not update the list, you will save a certificate alias that is not contained in the SSL configuration.

Manage certificates

Specifies to open the keystore panel for the key store in this SSL configuration, which enables you to manage personal certificates, signers, and certificate requests.

Certificate alias in key store

Specifies the certificate to use in the key store.

If you select **None**, the Java Secure Sockets Extension (JSSE) key manager determines which certificate is used. If multiple certificates exist in the key store, the key manager might not consistently select the same certificate.

Data type: Text

SSL configurations collection

Use this page to define a list of Secure Sockets Layer (SSL) configurations.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > ssl_configuration**. Under Related items, click **SSL configurations**.

Button	Resulting action
New	The Java Secure Socket Extension (JSSE) repertoire is for Java-based SSL communications. You can define a new JSSE configuration that can be used to create an SSLContext, URLStreamHandler, SSLSocketFactory, SSLServerSocketFactory, and so on, using the com.ibm.websphere.ssl.JSSEHelper API.
Delete	Deletes an existing JSSE configuration (administrator only). Be careful that any references to the SSL configuration have been removed prior to deleting this SSL configuration.

Name

Specifies the unique name of the SSL configuration in the management scope.

SSL configuration settings

Use this page to define Secure Sockets Layer (SSL) configuration properties.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > SSL_configuration_name**. Under Related items click **SSL configurations > New**.

Name

Specifies the unique name of the SSL configuration within the management scope in which it resides. For ways to programmatically access the properties that are configured for this SSL configuration, see the com.ibm.websphere.ssl.JSSEHelper application programming interface (API).

Data type: Text

Trust store name

Specifies a reference to a specific trust store used by Java Secure Sockets Extension (JSSE). The trust store holds signer certificates that validate the trust of certificates sent by remote connections during an SSL handshake.

Data type: Text
Default: *selected trust store*

Key store name

Specifies a reference to a specific key store. The key store holds personal certificates that represent the identity of one side of a connection. The public key of this personal certificate is sent to the other side of the connection to establish trust during the handshake. The remote side of the connection needs the root certificate authority (CA) certificate or self-signed public key (signer) to be in the trust store to validate this personal certificate.

Data type: Text
Default: *selected key store*

Get certificate aliases

Queries the keystore for the aliases of all the personal certificates in the keystore from which to choose.

Default server certificate alias

Specifies the certificate alias used as the identity for this SSL configuration if one has not been specified elsewhere.

If you select **None**, the Java Secure Sockets Extension (JSSE) key manager determines which certificate is used. If multiple certificates exist in the key store, the key manager might not consistently select the same certificate.

Data type: Text

Default client certificate alias

Specifies the description for a client certificate alias.

If you select **None**, the Java Secure Sockets Extension (JSSE) key manager determines which certificate is used. If multiple certificates exist in the key store, the key manager might not consistently select the same certificate.

Data type: Text

Management scope

Specifies the scope where this SSL configuration is visible. For example, if you choose a specific node, then the configuration is visible only on that node and on any servers that are part of that node.

Data type: Text

Creating a custom trust manager configuration

You can create a custom trust manager configuration at any management scope and associate the new trust manager with a Secure Sockets Layer (SSL) configuration.

You must develop, package, and locate a Java Archive JAR file for a custom key manager in the `was.install.root/lib/ext` directory on WebSphere Application Server. For more information, see “Example: Developing a custom trust manager for custom SSL trust decisions” on page 426.

Complete the following steps in the administrative console:

1. Decide whether you want to create the custom trust manager at the cell scope or below the cell scope at the node, server, or cluster, for example.

Important: When you create a custom trust manager at a level below the cell scope, you can associate it only with a Secure Sockets Layer (SSL) configuration at the same scope or higher. An SSL configuration at a scope lower than the trust manager does not see the trust manager configuration.

- To create a custom trust manager at the cell scope, click **Security > SSL certificate and key management > Trust managers**. Every SSL configuration in the cell can select the trust manager at the cell scope.
 - To create a custom trust manager at a scope below the cell level, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > ssl_configuration > Trust managers**.
2. Click **New** to create a new custom trust manager.
 3. Type a unique trust manager name.
 4. Select the **Custom** implementation setting. The custom setting enables you to define a Java class with an implementation of the javax.net.ssl.X509TrustManager Java interface and, optionally, the com.ibm.wsspi.ssl.TrustManagerExtendedInfo WebSphere Application Server interface.

Note: The standard implementation setting applies only when the trust manager is already defined in the Java security provider list as a provider and an algorithm, which is not the case for a custom trust manager.

5. Type a class name, for example, com.ibm.test.CustomTrustManager.
6. Select one of the following actions:
 - Click **Apply**, then click **Custom properties** under Additional Properties to add custom properties to the new custom trust manager. When you are finished adding custom properties, click **OK** and **Save**, then go to the next step.
 - Click **OK** and **Save**, then go to the next step.
7. Click **SSL certificate and key management** in the page navigation at the top of the panel.
8. Select one of the following actions:
 - Click **SSL configurations** under Related Items for a cell-scoped SSL configuration.
 - Click **Manage endpoint security configurations** to select an SSL configuration at a lower scope.
9. Click the link for the existing SSL configuration that you want to associate with the new custom trust manager. You can create a new SSL configuration instead of associating the custom trust manager with an existing configuration. For more information, see “Creating a Secure Sockets Layer configuration” on page 415.
10. Click **Trust and Key managers** under Additional Properties. If the new custom trust manager is not listed in the **Additional ordered trust managers** list, verify that you selected an SSL configuration scope that is at the same level or below the scope that you selected in Step 8.
11. Click **Add**. This action adds the new trust manager to the list of custom trust managers.
12. Click **OK** and **Save**.

You have created a custom trust manager configuration that references a JAR file in the install directory of WebSphere Application Server and associates it with an SSL configuration during the connection handshake.

You can create a custom trust manager for a pure client. For more information, see “Commands for the TrustManagerCommands group of the AdminTask object” on page 803.

Trust and key managers settings

Use this page to specify trust and key managers for the selected SSL configuration.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > Inbound | Outbound > *SSL_configuration_name***. Under **Related items** click **SSL configurations > *SSL_configuration_name* | New**. Under **Additional Properties** click **Trust and key managers**.

Attention: The application server checks the default trust managers first before checking the additional ordered trust managers in descending order.

Default trust manager:

Specifies the default trust manager, which is typically the IbmX509 trust manager by the IBMJSSE2 provider. The other default trust manager is IbmPKIX, which can be selected when certificate revocation checks must be made using the X509Certificate CRL distribution list. The IbmPKIX trust manager does not perform as well as the IbmX509 trust manager.

Data type: Text
Default: ibmX509TrustManager

Additional ordered trust managers:

Specifies additional trust managers that are used in the order shown for this SSL configuration.

Add:

Specifies to add the selection to the **Additional ordered trust managers** right-hand list.

Remove:

Specifies to remove the selection from the **Additional ordered trust managers** right-hand list.

Key manager:

Specifies the key manager that runs for this SSL configuration.

Data type: Text
Default: IbmX509KeyManager

Trust managers collection

Use this page to define the implementation settings for the trust manager. A trust manager is a class that is invoked during an Secure Sockets Layer (SSL) handshake to make trust decisions about the remote end point. A default trust manager is used to validate the signature and expiration of the certificate. Custom trust managers can be plugged in to perform an extended certificate and host name check.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > *ssl_configuration*** . Under **Related items**, click **Trust managers**.

Button	Resulting action
New	Adds a new trust manager that can be selected by an SSL configuration. A trust manager is invoked during an SSL handshake and can decide whether the handshake should be accepted based on the information it knows about the remote certificate and host.
Delete	Deletes an existing trust manager. Make sure the trust manager is not referenced by any SSL configuration before you delete it.

Name:

Specifies the name of the trust manager. This name is used as a selection in the SSL configuration panel.

Class name:

Specifies a class that implements the javax.net.ssl.X509TrustManager interface. Optionally, the class can implement the com.ibm.wsspi.ssl.TrustMangerExtendedInfo interface to get extended information about the connection. The class can use the information to verify the host name and so on.

Algorithm:

Specifies the algorithm name of the trust manager that is implemented by the selected provider.

Trust managers settings

This page enables you to view and set definitions for trust manager implementation settings. A trust manager is a class that gets invoked during a Secure Sockets Layer (SSL) handshake to make trust decisions about the remote end point. A default trust manager is used to validate the signature and expiration of the certificate. Custom trust managers can be plugged in to perform an extended certificate and hostname check.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > ssl_configuration** . Under Related items click **Trust managers > New** .

Name:

Specifies the name of the trust manager.

Data type:	Text
Default:	ibmX509TrustManager

Standard:

Specifies that the trust manager selection is available from a Java provider that is installed in the java.security file. This provider might be shipped by the Java Secure Sockets Extension (JSSE) or might be a custom provider that implements the javax.net.ssl.X509TrustManager interface.

Default:	Enabled
-----------------	---------

Provider:

Specifies the provider name that has an implementation of the javax.net.ssl.X509TrustManager interface. This provider is typically set to IBMJSSE2.

Enabled when **Standard** is selected.

Default	IBMJCE
----------------	--------

Algorithm:

Specifies the algorithm name of the trust manager implemented by the selected provider.

Enabled when **Standard** is selected.

Default ibmX509 or IbmPKIX
Range ibmX509, IbmPKIX

Custom:

Specifies that the trust manager selection is based on a custom implementation class that implements the javax.net.ssl.X509TrustManager interface and optionally the com.ibm.wsspi.ssl.TrustManagerExctendedInfo interface to obtain additional connection information that is not otherwise available.

Default: Disabled

Class name:

Specifies a class that implements the javax.net.ssl.X509TrustManager interface. Optionally, the class can implement the com.ibm.wsspi.ssl.TrustMangerExtendedInfo interface to get extended information about the connection. The class can use the information to verify the host name and so on.

Enabled when **Custom** is selected.

Data type: Text

Example: Developing a custom trust manager for custom SSL trust decisions

The following example is of a sample custom trust manager. The custom trust manager makes no trust decisions but instead uses the information in the X.509 certificate that it references to make decisions.

After you build and package the custom trust manager, configure it either from the ssl.client.props file for a pure client or the SSLConfiguration TrustManager link in the administrative console. See “Trust manager control of X.509 certificate trust decisions” on page 390 for more information about trust managers.

Note: This example should only be used as a sample, and is not supported.

```
import java.security.cert.X509Certificate;
import javax.net.ssl.*;
import com.ibm.wsspi.ssl.TrustManagerExctendedInfo;

public final class CustomTrustManager implements X509TrustManager,
TrustManagerExctendedInfo
{
    private static ThreadLocal threadLocStorage = new ThreadLocal();
    private java.util.Properties sslConfig = null;
    private java.util.Properties props = null;

    public CustomTrustManager()
    {
    }

    /**
     * Method called by WebSphere Application Server run time to set the target
     * host information and potentially other connection info in the future.
     * This needs to be set on ThreadLocal since the same trust manager can be
     * used by multiple connections.
     *
     * @param java.util.Map - Contains information about the connection.
     */
    public void setExtendedInfo(java.util.Map info)
    {
```

```

        threadLocStorage.set(info);
    }

    /**
     * Method called internally to retrieve information about the connection.
     *
     * @return java.util.Map - Contains information about the connection.
     */
    private java.util.Map getExtendedInfo()
    {
        return (java.util.Map) threadLocStorage.get();
    }

    /**
     * Method called by WebSphere Application Server run time to set the custom
     * properties.
     *
     * @param java.util.Properties - custom props
     */
    public void setCustomProperties(java.util.Properties customProps)
    {
        props = customProps;
    }

    /**
     * Method called internally to the custom properties set in the Trust Manager
     * configuration.
     *
     * @return java.util.Properties - information set in the configuration.
     */
    private java.util.Properties getCustomProperties()
    {
        return props;
    }

    /**
     * Method called by WebSphere Application Server runtime to set the SSL
     * configuration properties being used for this connection.
     *
     * @param java.util.Properties - contains a property for the SSL configuration.
     */
    public void setSSLConfig(java.util.Properties config)
    {
        sslConfig = config;
    }

    /**
     * Method called by TrustManager to get access to the SSL configuration for
     * this connection.
     *
     * @return java.util.Properties
     */
    public java.util.Properties getSSLConfig ()
    {
        return sslConfig;
    }

    /**
     * Method called on the server-side for establishing trust with a client.
     * See API documentation for javax.net.ssl.X509TrustManager.
     */
    public void checkClientTrusted(X509Certificate[] chain, String authType)
        throws java.security.cert.CertificateException
    {
        for (int j=0; j<chain.length; j++)
        {
            System.out.println("Client certificate information:");
        }
    }

```

```

        System.out.println(" Subject DN: " + chain[j].getSubjectDN());
        System.out.println(" Issuer DN: " + chain[j].getIssuerDN());
        System.out.println(" Serial number: " + chain[j].getSerialNumber());
        System.out.println("");
    }
}

/**
 * Method called on the client-side for establishing trust with a server.
 * See API documentation for javax.net.ssl.X509TrustManager.
 */
public void checkServerTrusted(X509Certificate[] chain, String authType)
    throws java.security.cert.CertificateException
{
    for (int j=0; j<chain.length; j++)
    {
        System.out.println("Server certificate information:");
        System.out.println(" Subject DN: " + chain[j].getSubjectDN());
        System.out.println(" Issuer DN: " + chain[j].getIssuerDN());
        System.out.println(" Serial number: " + chain[j].getSerialNumber());
        System.out.println("");
    }
}

/**
 * Return an array of certificate authority certificates which are trusted
 * for authenticating peers. You can return null here since the IbmX509
 * or IbmPKIX will provide a default set of issuers.
 *
 * See API documentation for javax.net.ssl.X509TrustManager.
 */
public X509Certificate[] getAcceptedIssuers()
{
    return null;
}
}

```

Related concepts

“Trust manager control of X.509 certificate trust decisions” on page 390

The role of the trust manager is to validate the Secure Sockets Layer (SSL) certificate that is sent by the peer, which includes verifying the signature and checking the expiration date of the certificate. A Java Secure Socket Extension (JSSE) trust manager determines if the remote peer can be trusted during an SSL handshake.

Creating a custom key manager

You can create a custom key manager configuration at any management scope and associate the new key manager with a Secure Sockets Layer (SSL) configuration.

You must develop, package, and locate a Java Archive (.JAR) file for a custom key manager in the `was.install.root/lib/ext` directory on WebSphere Application Server. For more information, see `rsec_ssldevcustomkeymgr.dita`.

Complete the following steps in the administrative console:

1. Decide whether you want to create the custom key manager at the cell scope or below the cell scope at the node, server, or cluster, for example.

Important: When you create a custom key manager at a level below the cell scope, you can associate it only with a Secure Sockets Layer (SSL) configuration at the same scope or higher. An SSL configuration at a scope lower than the key manager does not see the key manager configuration.

- To create a custom key manager at the cell scope, click **Security > SSL certificate and key management > Key managers**. Every SSL configuration in the cell can select the key manager at the cell scope.
 - To create a custom key manager at a scope below the cell level, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > SSL_configuration > Key managers**.
2. Click **New** to create a new key manager.
 3. Type a unique key manager name.
 4. Select the **Custom** implementation setting. With the custom setting, you can define a Java class that has an implementation on the Java interface `javax.net.ssl.X509KeyManager` and, optionally, the `com.ibm.wsspi.ssl.KeyManagerExtendedInfo` WebSphere Application Server interface. The standard implementation setting applies only when the key manager is already defined in the Java security provider list as a provider and an algorithm, which is not the case for a custom key manager. The typical standard key manager is `algorithm = IbmX509`, `provider = IBMJSSE2`.
 5. Type a class name. For example, `com.ibm.test.CustomKeyManager`.
 6. Select one of the following actions:
 - Click **Apply**, then click **Custom properties** under Additional Properties to add custom properties to the new custom key manager. When you are finished adding custom properties, click **OK** and **Save**, then go to the next step.
 - Click **OK** and **Save**, then go to the next step.
 7. Click **SSL certificate and key management** in the page navigation at the top of the panel.
 8. Select one of the following actions:
 - Click **SSL configurations** under Related Items for a cell-scoped SSL configuration.
 - Click **Manage endpoint security configurations** to select an SSL configuration at a lower scope.
 9. Click the link for the existing SSL configuration that you want to associate with the new custom key manager. You can create a new SSL configuration instead of associating the custom key manager with an existing configuration. For more information, see “Creating a Secure Sockets Layer configuration” on page 415.
 10. Click **Trust and Key managers** under Additional Properties.
 11. Select the new custom key manager in the **Key manager** drop-down list. If the new custom key manager is not listed, verify that you selected an SSL configuration scope that is at the same level or below the scope that you selected in Step 8.
 12. Click **OK** and **Save**.

You have created a custom key manager configuration that references a JAR file in the installation directory of WebSphere Application Server and associates the custom configuration with an SSL configuration during the connection handshake.

You can create a custom key manager for a pure client. For more information, see “Commands for the `keyManagerCommands` group of the `AdminTask` object” on page 807.

Example: Developing a custom key manager for custom Secure Sockets Layer key selection

The following example is of a sample custom key manager. This simple key manager provides the alias if it is set, and defers to the custom key manager if it is not set. Additional code can be added, for example, to prompt for a key that is given to the issuers that are provided.

After you build and package a custom key manager, you can configure it from either the `ssl.client.props` file for a pure client or by using the `SSLConfiguration KeyManager` link in the administrative console. See “Key manager control of X.509 certificate identities” on page 391 for more information about key managers.

Because only one key manager can be configured at a time for any given Secure Sockets Layer (SSL) configuration, the certificate selections on the server side might not work as they would when the default IbmX509 key manager is specified. When a custom key manager is configured, it is up to the owner of that key manager to ensure that the selection of the alias from the SSL configuration supplied is set properly when `chooseClientAlias` or `chooseServerAlias` are called. Look for the `com.ibm.ssl.keyStoreClientAlias` and `com.ibm.ssl.keyStoreServerAlias` SSL properties.

Note: This example should only be used as a sample, and is not supported.

```
package com.ibm.test;

import java.security.cert.X509Certificate;
import com.ibm.wsspi.ssl.KeyManagerExtendedInfo;

public final class CustomKeyManager
    implements javax.net.ssl.X509KeyManager, com.ibm.wsspi.ssl.KeyManagerExtendedInfo
{
    private java.util.Properties props = null;
    private java.security.KeyStore ks = null;
    private javax.net.ssl.X509KeyManager km = null;
    private java.util.Properties sslConfig = null;
    private String clientAlias = null;
    private String serverAlias = null;
    private int clientslotnum = 0;
    private int serverslotnum = 0;

    public CustomKeyManager()
    {
    }

    /**
     * Method called by WebSphere Application Server runtime to set the custom
     * properties.
     *
     * @param java.util.Properties - custom props
     */
    public void setCustomProperties(java.util.Properties customProps)
    {
        props = customProps;
    }

    private java.util.Properties getCustomProperties()
    {
        return props;
    }

    /**
     * Method called by WebSphere Application Server runtime to set the SSL
     * configuration properties being used for this connection.
     *
     * @param java.util.Properties - contains a property for the SSL configuration.
     */
    public void setSSLConfig(java.util.Properties config)
    {
        sslConfig = config;
    }

    private java.util.Properties getSSLConfig()
    {
        return sslConfig;
    }
}
```

```

}

/**
 * Method called by WebSphere Application Server runtime to set the default
 * X509KeyManager created by the IbmX509 KeyManagerFactory using the KeyStore
 * information present in this SSL configuration. This allows some delegation
 * to the default IbmX509 KeyManager to occur.
 *
 * @param javax.net.ssl.KeyManager defaultX509KeyManager - default key manager for IbmX509
 */
public void setDefaultX509KeyManager(javax.net.ssl.X509KeyManager defaultX509KeyManager)
{
    km = defaultX509KeyManager;
}

public javax.net.ssl.X509KeyManager getDefaultX509KeyManager()
{
    return km;
}

/**
 * Method called by WebSphere Application Server runtime to set the SSL
 * KeyStore used for this connection.
 *
 * @param java.security.KeyStore - the KeyStore currently configured
 */
public void setKeyStore(java.security.KeyStore keyStore)
{
    ks = keyStore;
}

public java.security.KeyStore getKeyStore()
{
    return ks;
}

/**
 * Method called by custom code to set the server alias.
 *
 * @param String - the server alias to use
 */
public void setKeyStoreServerAlias(String alias)
{
    serverAlias = alias;
}

private String getKeyStoreServerAlias()
{
    return serverAlias;
}

/**
 * Method called by custom code to set the client alias.
 *
 * @param String - the client alias to use
 */
public void setKeyStoreClientAlias(String alias)
{
    clientAlias = alias;
}

```

```

private String getKeyStoreClientAlias()
{
    return clientAlias;
}

/**
 * Method called by custom code to set the client alias and slot (if necessary).
 *
 * @param String - the client alias to use
 * @param int - the slot to use (for hardware)
 */
public void setClientAlias(String alias, int slotnum) throws Exception
{
    if ( !ks.containsAlias(alias))
    {
        throw new IllegalArgumentException ( "Client alias " + alias + "
        not found in keystore." );
    }
    this.clientAlias = alias;
    this.clientslotnum = slotnum;
}

/**
 * Method called by custom code to set the server alias and slot (if necessary).
 *
 * @param String - the server alias to use
 * @param int - the slot to use (for hardware)
 */
public void setServerAlias(String alias, int slotnum) throws Exception
{
    if ( ! ks.containsAlias(alias))
    {
        throw new IllegalArgumentException ( "Server alias " + alias + "
        not found in keystore." );
    }
    this.serverAlias = alias;
    this.serverslotnum = slotnum;
}

/**
 * Method called by JSSE runtime to when an alias is needed for a client
 * connection where a client certificate is required.
 *
 * @param String keyType
 * @param Principal[] issuers
 * @param java.net.Socket socket (not always present)
 */
public String chooseClientAlias(String[] keyType, java.security.Principal[] issuers, java.net.Socket socket)
{
    if (clientAlias != null && !clientAlias.equals(""))
    {
        String[] list = km.getClientAliases(keyType[0], issuers);
        String aliases = "";

        if (list != null)
        {
            boolean found=false;
            for (int i=0; i<list.length; i++)

```

```

        {
            aliases += list[i] + " ";
            if (clientAlias.equalsIgnoreCase(list[i]))
                found=true;
        }

        if (found)
        {
            return clientAlias;
        }
    }
}

// client alias not found, let the default key manager choose.
String[] keyArray = new String [] {keyType[0]};
String alias = km.chooseClientAlias(keyArray, issuers, null);
return alias.toLowerCase();
}

/**
 * Method called by JSSE runtime to when an alias is needed for a server
 * connection to provide the server identity.
 *
 * @param String[] keyType
 * @param Principal[] issuers
 * @param java.net.Socket socket (not always present)
 */
public String chooseServerAlias(String keyType, java.security.Principal[]
issuers, java.net.Socket socket)
{
    if (serverAlias != null && !serverAlias.equals(""))
    {
        // get the list of aliases in the keystore from the default key manager
        String[] list = km.getServerAliases(keyType, issuers);
        String aliases = "";

        if (list != null)
        {
            boolean found=false;
            for (int i=0; i<list.length; i++)
            {
                aliases += list[i] + " ";
                if (serverAlias.equalsIgnoreCase(list[i]))
                    found = true;
            }

            if (found)
            {
                return serverAlias;
            }
        }
    }

    // specified alias not found, let the default key manager choose.
    String alias = km.chooseServerAlias(keyType, issuers, null);
    return alias.toLowerCase();
}

public String[] getClientAliases(String keyType, java.security.Principal[] issuers)

```

```

    {
        return km.getClientAliases(keyType, issuers);
    }

    public String[] getServerAliases(String keyType, java.security.Principal[] issuers)
    {
        return km.getServerAliases(keyType, issuers);
    }

    public java.security.PrivateKey getPrivateKey(String s)
    {
        return km.getPrivateKey(s);
    }

    public java.security.cert.X509Certificate[] getCertificateChain(String s)
    {
        return km.getCertificateChain(s);
    }

    public javax.net.ssl.X509KeyManager getX509KeyManager()
    {
        return km;
    }
}

```

Associating a Secure Sockets Layer configuration dynamically with an outbound protocol and remote secure endpoint

After you create a Secure Sockets Layer (SSL) configuration, you must associate a secure outbound management scope with the new configuration. In this release, you can associate one SSL configuration with one remote secure endpoint and a different SSL configuration to another remote secure endpoint. Both endpoints can use the same outbound protocol, if appropriate. This task describes how to create the association dynamically.

Dynamic outbound selection requires that you provide only the outbound protocol name, the target host, and the target port so that WebSphere Application Server can make a connection between the SSL configuration and the outbound protocol or remote secure endpoint. The dynamic outbound selection method takes precedence over other selection methods, such as central management and direct selection, but is second to the programmatic method, that is, setting an SSL configuration on the running thread. For more information about the selection types and precedence rules, see “Secure communications using Secure Sockets Layer” on page 381.

Complete the following steps in the administrative console:

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations > Outbound**.
2. Select the management scope that you want to associate with an SSL configuration on the topology tree.
3. Under Related Items, click **Dynamic outbound endpoint SSL configurations**. The default dynamic outbound configuration name, the target protocol, host, and port connection information, and the SSL configuration name display.
4. Click **New** to create a new dynamic outbound configuration.
5. Type a dynamic outbound configuration name. Use a name that is descriptive of the purpose of the dynamic selection configuration.
6. Optionally, type a dynamic selection configuration description.

7. Type the connection information that you want to associate with the configuration that is displayed in the SSL configuration drop-down list. The connection information must be in the format *protocol name, target host, target port*. You can substitute an asterisk (*) for any value, as in the following examples:

- *,*,443
- *,www.ibm.com,443
- HTTP,.austin.ibm.com,*

where 443 is a port, www.ibm.com is a host, HTTP is a protocol, and .austin.ibm.com is a target host. You can add multiple connections, but each additional connection can affect outbound performance.

8. Click **Add** to add the new connection to the set of SSL configuration connections. To remove a connection, select it and click **Remove**.
9. Select an SSL configuration from the list.
10. Click **Get certificate aliases** to refresh the certificate aliases that are contained in the associated key store.
11. Choose a certificate alias from the list.
12. Click **OK** and **Save**.

WebSphere Application Server is ready to connect one or more SSL configurations to one or more remote secure endpoints.

You can return to the outbound tree and select another management scope to associate with the same or a new outbound configuration.

Example: Programmatically specifying an outbound SSL configuration using JSSEHelper API

WebSphere Application Server provides a way to specify programmatically which Secure Sockets Layer (SSL) configurations to use prior to making an outbound connection. The `com.ibm.websphere.ssl.JSSEHelper` interface provides a complete set of application programming interfaces (APIs) for handling SSL configurations.

You must perform the following steps for your application when using the JSSEHelper API to establish an SSL properties object on the thread for use by the runtime. Some of these APIs have Java 2 Security permission requirements. See the JSSEHelper API documentation for more information about the permissions required by your application.

1. Obtain an instance of the JSSEHelper API by typing the following command:

```
com.ibm.websphere.ssl.JSSEHelper jsseHelper
= com.ibm.websphere.ssl.JSSEHelper.getInstance();
```

2. Obtain SSL properties from the WebSphere Application Server configuration or use those provided by your application. You can obtain these properties in several ways:

- By direction selection of an alias name, within the same management scope or higher as in the following example:

```
try
{
    String alias = "NodeAServer1SSLSettings";
    // As specified in the WebSphere SSL configuration
    Properties sslProps = jsseHelper.getProperties(alias);
}
catch (com.ibm.websphere.ssl.SSLException e)
{
    e.printStackTrace(); // handle exception
}
```

- By using the `getProperties` API for programmatic, direction, dynamic outbound, or management scope selection (based on precedence rules and inheritance). The SSL runtime uses the `getProperties` API to determine which SSL configuration to use for a particular protocol. This

decision is based on both the input (sslAlias and connectionInfo) and the management scope from which the property is called. The getProperties API makes decisions in the following order:

- a. The API checks the thread to see if properties already exist.
- b. The API checks for a dynamic outbound configuration that matches the ENDPOINT_NAME, REMOTE_HOST, and or REMOTE_PORT.
- c. The API checks to see if the optional sslAlias property is specified. You can configure any protocol as direct or centrally managed. When a protocol is configured as direct, the sslAlias parameter is null. When a protocol is configured as centrally managed, the sslAlias parameter is also null.
- d. If no selection has been made, the API chooses the dynamic outbound configuration based on the management scope it was called from. If the dynamic outbound configuration is not defined in the same scope, it then searches the hierarchy to locate one.

The last choice is the cell-scoped SSL configuration (in Network Deployment) or the node-scoped SSL configuration (in Base Application Server). The com.ibm.websphere.ssl.SSLConfigChangeListener parameter is notified when the SSL configuration that is chosen by a call to the getProperties API changes. The protocol can then call the API again to obtain the new properties as in the following example:

```
try
{
    String sslAlias = null; // The sslAlias is not specified directly at this time.
    String host = "myhost.austin.ibm.com"; // the target host
    String port = "443"; // the target port

    HashMap connectionInfo = new HashMap();
    connectionInfo.put(JSSEHelper.CONNECTION_INFO_DIRECTION,
        JSSEHelper.DIRECTION_OUTBOUND);
    connectionInfo.put(JSSEHelper.CONNECTION_INFO_REMOTE_HOST, host);
    connectionInfo.put(JSSEHelper.CONNECTION_INFO_REMOTE_PORT,
        Integer.toString(port));
    connectionInfo.put(JSSEHelper.CONNECTION_INFO_ENDPOINT_NAME,
        JSSEHelper.ENDPOINT_IIOp);

    java.util.Properties props = jsseHelper.getProperties(sslAlias,
        connectionInfo, null);
}
catch (com.ibm.websphere.ssl.SSLException e)
{
    e.printStackTrace(); // handle exception
}
```

- By creating your own SSL properties and then passing them to the runtime, as in the following example:

```
try
{
    // This is the recommended "minimum" set of SSL properties. The trustStore can
    // be the same as the keyStore.
    Properties sslProps = new Properties();
    sslProps.setProperty("com.ibm.ssl.trustStore", "some value");
    sslProps.setProperty("com.ibm.ssl.trustStorePassword", "some value");
    sslProps.setProperty("com.ibm.ssl.trustStoreType", "some value");
    sslProps.setProperty("com.ibm.ssl.keyStore", "some value");
    sslProps.setProperty("com.ibm.ssl.keyStorePassword", "some value");
    sslProps.setProperty("com.ibm.ssl.keyStoreType", "some value");

    jsseHelper.setSSLPropertiesOnThread(sslProps);
}
catch (com.ibm.websphere.ssl.SSLException e)
{
    e.printStackTrace(); // handle exception
}
```

3. Use the `JSSEHelper.setSSLPropertiesOnThread(props)` API to set the `Properties` object on the thread so that the runtime picks it up and uses the same `JSSEHelper.getProperties` API. You can also obtain properties from the thread after they are set with the `jsseHelper.getSSLPropertiesOnThread()` API, as in the following example:

```
try
{
    Properties sslProps = jsseHelper.getProperties(null,
        connectionInfo, null); jsseHelper.setSSLPropertiesOnThread(sslProps);
}
catch (com.ibm.websphere.ssl.SSLException e)
{
    e.printStackTrace(); // handle exception
}
```

4. When the connection is completed, you must clear the SSL properties from the thread by passing the `null` value to the `setPropertiesOnThread` API, as in the following example:

```
try
{
    jsseHelper.setSSLPropertiesOnThread(null);
}
catch (com.ibm.websphere.ssl.SSLException e)
{
    e.printStackTrace(); // handle exception
}
```

Select the approach that best fits your connection situation when you specify programmatically which Secure Sockets Layer (SSL) configurations to use prior to making an outbound connection.

Associating Secure Sockets Layer configurations centrally with inbound and outbound scopes

After you create a Secure Sockets Layer (SSL) configuration, you must associate a secure inbound or outbound management scope with the new configuration. You can manage the association centrally so that you can easily make changes that affect all the scopes that are lower on the topology and that are associated with the configuration. Beginning with WebSphere Application Server version 6.1, the recommended and the default configuration method is centrally managed SSL configurations.

You can simplify the number of associations that you need to make for an SSL configuration by associating the configuration with the highest level management scope requiring a unique configuration. SSL configuration associations manifest inheritance behaviors. Because of the inheritance behaviors, all of the scopes that are lower on the topology inherit this SSL configuration. For example, an association you make at the cell level affects nodes, servers, clusters, and endpoints. For more information, see “Central management of Secure Sockets Layer configurations” on page 396.

A precedence rule determines which SSL configuration association is used at a particular scope. The highest precedence is given to endpoints on the topology. If you establish an association at the endpoint, this association overrides any prior association that you made higher up on the management scope topology.

Complete the following steps in the administrative console:

1. Click **Security > SSL certificate and key management**.
2. Select the **Dynamically update the runtime when SSL configuration changes** check box if you want changes that you make to an existing SSL configuration to occur dynamically. All outbound SSL communications honor the dynamic SSL changes. Protocols that do not use the channel frameworks SSL channel for inbound communications, including Object Request Broker (ORB) and administrative SOAP protocols, do not honor dynamic updates. For more information, see “Dynamic configuration updates” on page 411.
3. Click **Manage endpoint security configurations**.

4. Select either the inbound or the outbound tree. After finishing the selected tree, you can return to this step to repeat the following steps for the other tree.
5. Click the link for the selected cell, node, node group, server, cluster, or endpoint on the topology tree. If the scope already has an associated SSL configuration and alias, these objects display in parentheses immediately following the scope name, for example: Node01(NodeDefaultSSLSettings,default). If the deployment manager has federated a node, the node scope SSL configuration overrides the cell scope configuration above it in the topology.
6. Decide whether to override the inherited values that display in the read-only fields. Read-only fields include the management scope name, the direction, and the inherited SSL configuration name and certificate alias.
 - If you are satisfied with these values, do not override them.
 - If you want to override the inherited values, select the **Override inherited values** check box.
7. Select an SSL configuration from the list.
8. Click **Update certificate alias list**. The certificate alias list comes from the key store that is referenced by the new SSL configuration.
9. Click **Manage certificates** if you want to manage the personal certificates that are contained in the key store that is referenced in the SSL configuration.
10. Click **Update certificate alias list** to refresh the list of aliases.
11. Select a certificate alias in the key store to represent the identity of the endpoint.
12. Click **OK** to save your changes.
13. Click **Manage endpoint security configurations and trust zones** to return to the topology tree.
14. Configure the opposite direction on the topology tree using the steps in this task. You can also select additional scopes to associate with the SSL configuration, as needed.

Each SSL configuration at the selected scope and at scopes beneath it on the topology tree have the same SSL configuration properties. The following SSL configuration methods override the centrally managed configurations that you associate in the tree view:

- Direct selection at the endpoint
- Dynamic outbound SSL configuration associations
- Programmatic specifications

At any management scope, you can configure the following objects: dynamic outbound endpoint SSL configurations, key stores, key sets, key set groups, key managers, and trust managers. Like SSL configurations, these objects are scoped automatically so that they are not visible higher up in the tree nor are they loaded during runtime by processes that are higher up in the tree.

Selecting an SSL configuration alias directly from an endpoint configuration

You can associate a secure outbound endpoint with a new Secure Sockets Layer (SSL) configuration directly. If you are migrating from a release prior to version 6.1, WebSphere Application Server still supports configurations that were selected directly at an endpoint. Direct selection always overrides centrally managed configurations and preserves migrated configurations.

Select an SSL configuration alias directly at the following endpoints:

- **Security > Secure administration, applications, and infrastructure > RMI/IOP security > CSiv2 outbound transport**
- **Security > Secure administration, applications, and infrastructure > RMI/IOP security > CSiv2 inbound transport**
- **System administration > Deployment manager > Transport Chain > WCInboundAdminSecure > SSL inbound channel (SSL_1)**
- **System administration > Deployment manager > Administration Services > JMX connectors > SOAPConnector > Custom Properties > sslConfig**

- **System administration > Node agents > nodeagent > Administration Services > JMX connectors > SOAPConnector > Custom Properties > sslConfig**
- **Servers > Application servers > server1 > Messaging engine inbound transports > InboundSecureMessaging > SSL inbound channel (SIB_SSL_JFAP)**
- **Servers > Application servers > server1 > WebSphere MQ link inbound transports > InboundSecureMQLink > SSL inbound channel (SIB_SSL_MQFAP)**
- **Servers > Application servers > server1 > SIP Container Settings > SIP container transport chains > SIPInboundDefaultSecure > SSL inbound channel (SSL_5)**
- **Servers > Application servers > server1 > Web Container Settings > Web container transport chains > WCInboundAdminSecure > SSL inbound channel (SSL_1)**
- **Servers > Application servers > server1 > Web Container Settings > Web container transport chains > WCInboundDefaultSecure > SSL inbound channel (SSL_2)**

Attention: Keep in mind that central management of SSL configurations can be a more efficient strategy because multiple configurations can be contained within a single SSLConfigGroup. If you need to convert configuration references that are already directly managed to centrally managed configurations, modify each endpoint individually. For more information on specific wsadmin commands, see “Commands for the SSLConfigGroupCommands group of the AdminTask object” on page 811.

Complete the following steps in the administrative console:

Note: These steps provide an example to follow when you directly select any of the endpoints listed above.

1. Click **Security > Secure administration, applications, and infrastructure > RMI/IIOP security > CSlv2 outbound transport**.
2. Click **Use specific SSL alias**. When you identify a specific SSL alias, you override the centrally managed scope associations.
3. Select an SSL configuration alias from the drop-down list.
4. Click **OK**.
5. Repeat these steps for additional protocols or endpoints, if desired.

By associating the endpoint directly, you have overridden a centrally managed SSL configuration.

If you decide to use management scopes instead of endpoints to associate an SSL configuration, follow the steps above, but click **Centrally managed** instead of **Use specific SSL alias**, then click **Manage endpoint security configurations**. The console is redirected to **Security > SSL certificate and key management > Manage endpoint security configurations**.

Enabling Secure Sockets Layer client authentication for a specific inbound endpoint

When you establish a Secure Sockets Layer (SSL) configuration, you can enable client authentication for a specific inbound endpoint.

The endpoint configuration must already exist in the SSL topology.

Complete the following steps in the administrative console:

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations > Inbound > SSL_configuration**. If you want to enable SSL client authentication for all processes, define an SSL configuration for the new endpoint at the node or cell level so that it is visible to all processes on the same node or on the entire cell. For more information, see “Creating a Secure Sockets Layer configuration” on page 415.

2. Select **Override inherited values**. The SSL configuration is used for the current scope and any lower scopes that have not already designated an SSL configuration. This field displays for server and node groups within the object hierarchy and does not display for the top-level node or cell.
3. Select an SSL configuration from the drop-down list.
4. Click **Update certificate alias list**.
5. Select a **Certificate alias** from the drop-down list.
6. Click **OK** to save the configuration.

You can repeat the previous steps for each endpoint that uses the same SSL configuration to enable client authentication for the inbound endpoints.

CSlv2 Protocol Exception:

The Common Secure Interoperability Version 2 (CSlv2) secure endpoints, used for Remote Method Invocation over the Internet Inter-ORB Protocol (RMI/IIOP) security, cannot override inherited values. While the rest of the SSL properties are effective for CSlv2 when they are selected at the centrally-managed Secure Communications panel, the client authentication selection is controlled by the CSlv2 protocol configuration.

To enable SSL client certificate authentication for the CSlv2 protocol, you must use the CSlv2 inbound and outbound authentication panels. For SSL client authentication to occur between two servers, you must enable (support or require) SSL client certificate authentication for both the inbound and the outbound policies.

WebSphere Application Server can either request (support) clients to provide signer certificates for the SSL handshake, or the server can require clients to provide a valid signer certificates for the SSL handshake, which is a more secure method. However, when the server requires certificates, the server must obtain a signer for each client that connects to the server, which involves more server-side management.

The client certificate should not be used for the identity when it is used from server-to-server. However, when a pure client sends the client certificate it is used for the identity unless a message level identity is specified, such as a user ID or a password.

Do the following to enable client certificate authentication for the CSlv2 protocol for server-to-server:

1. **Click Security > Secure administration, applications, and infrastructure.**
2. Expand the **RMI/IIOP** security section.
3. Click **CSlv2 inbound authentication**.
4. Under Client authentication, select either **supported** or **required**. When you select required, only one SSL port is opened (CSV2_SSL_MUTUALAUTH_LISTENER_ADDRESS). When you select supported, two SSL ports are opened (both CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS and CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS).
If there are two ports, the client can select either based on the security configuration policy of the port.
5. Click **OK** to save.
6. If you want server-to-server SSL client authentication, then complete the remaining steps. If you don't complete the remaining steps, only pure clients are enabled to send client certificates.
7. Expand the **RMI/IIOP** security section.
8. Click **CSlv2 outbound authentication**.
9. Under Client authentication, select either **supported** or **required**.

The SSL configuration for the inbound secure endpoints for which you enable SSL client certificate authentication must have the signer certificate from any client that attempts to open a connection to that inbound secure endpoint. You must collect those signers and then add them to the trust store associated with the inbound secure endpoints SSL configuration.

Manage endpoint security configurations

Use this page to select a Secure Socket Layer (SSL) configuration from the Local Topology hierarchy, which includes cells, nodes, node groups, servers, and clusters.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations**.

Local topology:

The Local topology represents the hierarchy of nodes, node groups, clusters, servers, and end points within the cell that comprise a centralized SSL configuration.

The topology acts as a hierarchical tree in terms of inheritance. For example, if an SSL configuration has been associated with a specific node, then all servers within that node will inherit that SSL configuration selection, provided the servers are not associated with an SSL configuration at the server scope. Centralized management of SSL is the default configuration; however, it can be overridden at various locations to directly select a specific SSL alias as in previous releases for backwards compatibility.

Scope	Description
Inbound/Outbound	Specifies the topology tree in terms of connection direction. For example, the inbound tree represents all server endpoints that receive connections at the various servers within the cell. The outbound tree represents the client side of connections from the various servers within the cell.
Nodes	Specifies the nodes that are part of the cell. The list of nodes is updated anytime a node gets federated into the cell.
Servers	Specifies the servers that are part of a specific node. You can enable a specific server to have an SSL configuration associated with it so that resources within the same server can use the associated SSL configuration.
Clusters	Specifies the clusters that are part of the cell. When an SSL configuration is associated with a cluster, all servers within the cluster will use the same SSL configuration unless specified at a lower level in the topology.
Nodegroups	Specifies the node groups that are part of the cell. When an SSL configuration is associated with a node group, all nodes within that node group may use the same SSL configuration unless one is specified at a lower scope in the topology or the specific end point has chosen a direct alias reference.
Secure port and transport	Specifies an endpoint name to associate with an SSL configuration when more specific SSL settings are needed at this level. You could select an alias directly at the endpoint panel; however, when you use Secure port and transport , you can maintain more centralized control of the SSL configuration and make changes more easily.

Dynamic inbound and outbound endpoint SSL configurations collection

Use this page to manage dynamic endpoint Secure Sockets Layer (SSL) configurations, which represent associations between Secure Socket Layer (SSL) configurations and their target protocol, host, and port.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > ssl_configuration** . Under Related items, click **Dynamic [inbound | outbound] endpoint SSL configurations**.

When an outbound connection is attempted, this association is checked ahead of the SSL configuration scope association. Based on the target protocol,host,port, the outbound SSL configuration used can be different from the default specified in the SSL scope configuration.

Button	Resulting action
New	Adds a new dynamic outbound selection criteria. The outbound connection selects an SSL configuration based upon connection information, including DNS host name and domain, port, and protocol type. When an outbound connection is being made, the dynamic outbound selection criteria are queried for a match, and if found the SSL configuration associated is used.
Delete	Deletes an existing dynamic outbound endpoint SSL configuration.

Name:

Specifies the unique name of the dynamic endpoint configuration.

Connection information:

Specifies the set of target protocol, host, port for the outbound request in the form *protocol,host,port*.

SSL Configuration:

Specifies the SSL configuration that is used by requests at this scope when a match occurs for the given selection criteria.

Dynamic outbound endpoint SSL configuration settings

Use this page to set properties for dynamic outbound endpoint SSL configurations, which represent associations between SSL configurations and their target protocol, host, and port.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > ssl_configuration** . Under Related items, click **Dynamic [inbound | outbound] endpoint SSL configurations > New**.

When an outbound connection is attempted, this association is checked ahead of the Secure Sockets Layer (SSL) configuration scope association. This means based on the target protocol,host,port, the outbound SSL configuration used can be different than the default specified in the SSL scope configuration.

Name:

Specifies the unique name of the dynamic endpoint configuration.

Data type: Text

Description:

Specifies text that describes the purpose of this dynamic selection criteria.

Data type: Text

Add connection information:

Specifies select information in the form protocol,host,port for the outbound connection. Multiple selection criteria can be entered. An asterisk (*) can be used to mean all protocols, hosts, or ports. You can use an * for any field.

Data type: Text

An example of selection criteria is *,www.ibm.com,*, which means that any time the target host is www.ibm.com, you must use the SSL configuration specified here. Another example selection criteria is IIOP,*,*, which means that any outbound IIOP request uses the SSL configuration that is specified in the SSL configuration field. When there is a conflict between two selection criteria, the application server uses the first match. The list of valid protocols you can use include: IIOP, HTTP, JMS, LDAP, SIP, ADMIN_SOAP, or ADMIN_IIOP.

Add:

Specifies to add the selected information from the **Add select information** menu to the right-hand list.

Remove:

Specifies to remove the selection from the right-hand list.

SSL Configuration:

Specifies the SSL configuration to be used by requests at this scope when a match occurs for the given selection criteria.

Data type: Text

Get certificate alias:

When selected, the keystore within the selected SSL configuration is queried for a list of personal certificates from which to choose.

Certificate alias:

Specifies the certificate alias that is used as the identity for the connection.

If you select **None**, the Java Secure Sockets Extension (JSSE) key manager determines which certificate is used. If multiple certificates exist in the keystore, the key manager might not consistently select the same certificate.

Data type: Text
Default: (none)

Quality of protection (QoP) settings

Use this page to specify security level, ciphers, and mutual authentication settings for the Secure Socket Layer (SSL) configuration.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound *SSL_configuration_name*}**. Under **Related Items**, click **SSL configurations > {*SSL_configuration_name* | New}**. Under **Additional Properties**, click **Quality of protection (QoP) settings**.

Client authentication

Specifies the whether SSL client authentication should be requested if the SSL connection is used for the server side of the connection.

If None is selected, the server does not request that a client certificate be sent during the handshake. If Supported is selected, the server requests that a client certificate be sent. If the client does not have a certificate, the handshake might still succeed. If Required is selected, the server requests that a client certificate be sent. If the client does not have a certificate, the handshake fails.

Data type: Text
Default: None

Protocol

Specifies the Secure Sockets Layer (SSL) handshake protocol. This protocol is typically SSL_TLS, which supports all handshake protocols except for SSLv2 on the server side. When United States Federal Information Processing standard (FIPS) option is enabled, Transport Layer Security (TLS) is automatically used regardless of this setting.

Data type: text
Default: SSL_TLS

Predefined JSSE provider

Specifies one of the predefined Java Secure Sockets Extension (JSSE) providers. The IBMJSSE2 provider is recommended for use on all platforms which support it. It is required for use by the channel framework SSL channel. When Federal Information Processing Standard (FIPS) is enabled, IBMJSSE2 is used in combination with the IBMJCEFIPS crypto provider.

Default: Enabled

Select provider

Specifies a package that implements a subset of the cryptography aspects for the Java security application programming interface (API). This value is a JSSE provider name that is listed in the java.security file. Note that cipher suites and protocol values depend upon the provider.

Data type: Text
Default: IBMJSSE2

Custom JSSE provider

Specifies that a custom JSSE provider should be used.

Default: Disabled

Custom provider

Specifies a package that implements a subset of the cryptography aspects for the Java security application programming interface (API). This value is a Java Secure Sockets Extension (JSSE) provider name that is listed in the java.security file. Note that cipher suites and protocol values depend upon the provider.

Data type: Text

Cipher suite groups

Specifies the various cipher suite groups that can be chosen depending upon your security needs. The stronger the cipher suite strength, the better the security; however, this can result in performance consequences.

Data type: Text
Default: Strong

Update selected ciphers

When selected, the cipher suites that are contained within the selected Cipher suite group are added to the list of **Selected ciphers**. Any change to this list changes the Cipher suite group to custom.

Selected ciphers

Specifies the ciphers that are effective when the configuration is saved. These ciphers are used to negotiate with the remote side of the connection during the handshake. A common cipher needs to be selected or the handshake fails.

Data type: Text

Add

Specifies to add the selected cipher to the **Selected ciphers** list.

Remove

Specifies to remove the selected cipher from the **Selected ciphers** list.

ssl.client.props client configuration file

Use the `ssl.client.props` file to configure Secure Sockets Layer (SSL) for clients. In previous releases of WebSphere Application Server, SSL properties were specified in the `sas.client.props` or `soap.client.props` files or as system properties. By consolidating the configurations, WebSphere Application Server enables you to manage security in a manner that is comparable to server-side configuration management. You can configure the `ssl.client.props` file with multiple SSL configurations.

Setting up the SSL configuration for clients

Client runtimes are dependent on the WebSphere Application Server `ssl.client.props` configurations.

Use the `setupCmdLine.bat` or `setupCmdLine.sh` script on the command line to specify the `com.ibm.SSL.ConfigURL` system property.

```
-Dcom.ibm.SSL.ConfigURL=file:C:\WebSphere\AppServer\profiles\default  
\properties\ssl.client.props
```

The `com.ibm.SSL.ConfigURL` property references a file URL that points to the `ssl.client.props` file. You can reference the `CLIENTSSL` variable on the command line of any script that uses the `setupCmdLine.bat` or `setupCmdLine.sh` file.

When you specify the `com.ibm.SSL.ConfigURL` system property, the SSL configuration is available to all protocols that use SSL. SSL configurations, which are referenced in the `ssl.client.props` file, also have aliases that you can reference. In the following sample code from the `sas.client.props` file, all of the SSL properties are replaced with a property that points to an SSL configuration in the `ssl.client.props` file:

```
com.ibm.ssl.alias=default
```

The following sample code shows a property in the `soap.client.props` file that is similar to the `com.ibm.SSL.ConfigURL` property. This property references a different SSL configuration on the client side:

```
com.ibm.ssl.alias=ADMIN_SOAP
```

In the `ssl.client.props` file, you can change the administrative SSL configuration to avoid modifying the `soap.client.props` file.

Tip: In WebSphere Application Server Version 6.1, support for SSL properties is still specified in the `sas.client.props` and `soap.client.props` files. However, consider moving the SSL configurations to the `ssl.client.props` file, because this file is the new configuration model for client SSL.

Properties of the `ssl.client.props` file

This section describes the default `ssl.client.props` file properties in detail, by sections within the file.

Global properties

Global SSL properties are process-specific properties that include Federal Information Processing Standard (FIPS) enablement, the default SSL alias, the `user.root` property for specifying the root location of the key and truststore paths, and so on.

Property	Default	Description
<code>com.ibm.ssl.defaultAlias</code>	DefaultSSLSettings	Specifies the default alias that is used whenever an alias is not specified by the protocol that calls the JSSEHelper API to retrieve an SSL configuration. This property is the final arbiter on the client side for determining which SSL configuration to use.
<code>com.ibm.ssl.validationEnabled</code>	false	When set to true, this property validates each SSL configuration as it is loaded. Use this property for debug purposes only, to avoid unnecessary performance overhead during production.
<code>com.ibm.ssl.performURLHostNameVerification</code>	false	When set to true, this property enforces URL host name verification. When HTTP URL connections are made to target servers, the common name (CN) from the server certificate must match the target host name. Without a match, the host name verifier rejects the connection. The default value of false omits this check. As a global property, it sets the default host name verifier. Any <code>javax.net.ssl.HttpURLConnection</code> object can choose to enable host name verification for that specific instance by calling the <code>setHostnameVerifier</code> method with its own <code>HostnameVerifier</code> instance.
<code>com.ibm.security.useFIPS</code>	false	When set to true, FIPS-compliant algorithms are used for SSL and other Java Cryptography Extension (JCE)-specific applications. This property is typically not enabled unless the property is required by the operating environment.
<code>user.root</code>	C:\WebSphere\AppServer\profiles\default	This property can be used by key and truststore location properties as a single property for specifying the root path to the key and truststores. Typically, this property is the profile root. However, you can modify this property to any root directory on the local machine that has the proper read and potentially write authority to that directory.
profile_root of the profile	C:\WebSphere\AppServer\profiles\default	Key and truststore location properties can use this property as a single property for specifying the root path to the key and truststores. You can modify this property to any root directory on the local machine that has the proper read (and potentially write) authority to that directory.

Certificate creation properties

Use certificate creation properties to specify the default self-signed certificate values for the major attributes of a certificate. You can define the distinguished name (DN), expiration date, key size, and alias that are stored in the keystore.

Property	Default	Description
com.ibm.ssl.defaultCertReqAlias	default_alias	This property specifies the default alias to use to reference the self-signed certificate that is created in the keystore. If the alias already exists with that name, the default alias is appended with <code>_#</code> , where the number sign (#) is an integer that starts with 1 and increments until it finds a unique alias.
com.ibm.ssl.defaultCertReqSubjectDN	cn=\${hostname}, o=IBM,c=US	This property uses the property distinguished name (DN) that you set for the certificate when it is created. The <code>\${hostname}</code> variable is expanded to the host name on which it resides. You can use correctly formed DNs as specified by the X.509 certificate.
com.ibm.ssl.defaultCertReqDays	365	This property specifies the validity period for the certificate and can be as small as 1 day and as large as the maximum number of days that a certificate can be set, which is approximately 20 years.
com.ibm.ssl.defaultCertReqKeySize	1024	This property is the default key size. The valid values depend upon the Java Virtual Machine (JVM) security policy files that are installed. By default, the product JVMs ship with the export policy file that limits the key size to 1024. To get a large key size such as 2048, you can download the restricted policy files from the Web site.

SSL configuration properties

Use the SSL configuration properties section to set multiple SSL configurations. For a new SSL configuration specification, set the `com.ibm.ssl.alias` property because the parser starts a new SSL configuration with this alias name. The SSL configuration is referenced by using the alias property from another file, such as `sas.client.props` or `soap.client.props`, through the default alias property. The properties that are specified in the following table enable you to create a `javax.net.ssl.SSLContext`, among other SSL objects.

Property	Default	Description
com.ibm.ssl.alias	DefaultSSLSettings	This property is the name of this SSL configuration and must be the first property for an SSL configuration because it references the SSL configuration. If you change the name of this property after it is referenced elsewhere in the configuration, the runtime defaults to the <code>com.ibm.ssl.defaultAlias</code> property whenever the reference is not found. The error trust file is null or key file is null might display when you start an application using an SSL reference that is no longer valid.
com.ibm.ssl.protocol	SSL_TLS	This property is the SSL handshake protocol that is used for this SSL configuration. This property attempts Transport Layer Security (TLS) first, but accepts any remote handshake protocol, including SSLv3 and TLS. Valid values for this property include SSLv2 (client side only), SSLv3, SSL, TLS, TLSv1, and SSL_TLS.

Property	Default	Description
com.ibm.ssl.securityLevel	HIGH	This property specifies the cipher group that is used for the SSL handshake. The typical selection is HIGH, which specifies 128-bit or higher ciphers. The MEDIUM selection provides 40-bit ciphers. The LOW selection provides ciphers that do not perform encryption, but do perform signing for data integrity. If you specify your own cipher list selection, uncomment the property com.ibm.ssl.enabledCipherSuites.
com.ibm.ssl.trustManager	IbmX509	This property specifies the default trust manager that you must use to validate the certificate sent by the target server. This trust manager does not perform certificate revocation list (CRL) checking. You can choose to change this to IbmPKIX for CRL checking using CRL distribution lists in the certificate, which is a standard way to perform CRL checking. When you want to perform custom CRL checking, you must implement a custom trust manager and specify the trust manager in the com.ibm.ssl.customTrustManagers property. The IbmPKIX option might affect performance because this option requires IBM CertPath for trust validation. Use IbmX509 unless CRL checking is necessary.
com.ibm.ssl.keyManager	IbmX509	This property specifies the default key manager to use for choosing the client alias from the specified keystore. This key manager uses the com.ibm.ssl.keyStoreClientAlias property to specify the keystore alias. If this property is not specified, the choice is delegated to Java Secure Socket Extension (JSSE). JSSE typically chooses the first alias that it finds.
com.ibm.ssl.contextProvider	IBMJSSE2	This property is used to choose the JSSE provider for the SSL context creation. It is recommended that you default to IBMJSSE2 when you use a Java virtual machine (JVM). The client plug-in can use the SunJSSE provider when using a Sun JVM.
com.ibm.ssl.enableSignerExchangePrompt		This property determines whether to display the signer exchange prompt when a signer is not present in the client truststore. The prompt displays information about the remote certificate so that WebSphere Application Server can decide whether or not to trust the signer. It is very important to validate the certificate signature (hash). This signature is the only reliable information that can guarantee that the certificate has not been modified from the original server certificate. For automated scenarios, disable this property to avoid SSL handshake exceptions. Run the retrieveSigners.bat file or the retrieveSigners.sh script, which sets up the SSL signer exchange, to download the signers from the server prior to running the client.
com.ibm.ssl.keyStoreClientAlias	default	This property is used to reference an alias from the specified keystore when the target does not request client authentication. When WebSphere Application Server creates a self-signed certificate for the SSL configuration, this property determines the alias and overrides the global com.ibm.ssl.defaultCertReqAlias property.

Property	Default	Description
com.ibm.ssl.customTrustManagers	Commented out by default	This property enables you to specify one or more custom trust managers, which are separated by commas. These trust managers can be in the form of <i>algorithm provider</i> or <i>classname</i> . For example, <code>IbmX509IIBMJSSE2</code> is in the <i>algorithm provider</i> format, and the <code>com.acme.myCustomTrustManager</code> interface is in the <i>classname</i> format. The class must implement the <code>javax.net.ssl.X509TrustManager</code> interface. Optionally, the class can implement the <code>com.ibm.wsspi.ssl.TrustManagerExtendedInfo</code> interface. These trust managers run in addition to the default trust manager that is specified by the <code>com.ibm.ssl.trustManager</code> interface. These trust managers do not replace the default trust manager.
com.ibm.ssl.customKeyManager	Commented out by default	This property enables you to have one, and only one, custom key manager. The key manager replaces the default key manager that is specified in the <code>com.ibm.ssl.keyManager</code> property. The form of the key manager is <i>algorithm provider</i> or <i>classname</i> . See the format examples for the <code>com.ibm.ssl.customTrustManagers</code> property. The class must implement the <code>javax.net.ssl.X509KeyManager</code> interface. Optionally, the class can implement the <code>com.ibm.wsspi.ssl.KeyManagerExtendedInfo</code> interface. This key manager is responsible for alias selection.
com.ibm.ssl.dynamicSelectionInfo	Commented out by default	This property enables dynamic association with the SSL configuration. The syntax for a dynamic association is <i>outbound_protocol, target_host, or target_port</i> . For multiple specifications, use the vertical bar () as the delimiter. You can replace any of these values with an asterisk (*) to indicate a wildcard value. Valid <i>outbound_protocol</i> values include: <code>IIOP</code> , <code>HTTP</code> , <code>LDAP</code> , <code>SIP</code> , <code>BUS_CLIENT</code> , <code>BUS_TO_WEBSPIHERE_MQ</code> , <code>BUS_TO_BUS</code> , and <code>ADMIN_SOAP</code> . When you want the dynamic selection criteria to choose the SSL configuration, uncomment the default property, and add the connection information. For example, add the following on one line <pre>com.ibm.ssl.dynamicSelectionInfo=HTTP, .ibm.com,443 HTTP,.ibm.com,9443 .ADMIN_IIOP, and ADMIN_SOAP represent administrative uses of these protocols for the Remote Method Invocation (RMI) and SOAP connectors, respectively. In the SSL configuration alias AdminSOAPSSLSettings, you can set the value to ADMIN_SOAP,*,*, which indicates that any outbound administrative SOAP connection can use this SSL configuration. Dynamic selection takes precedence over direct alias selection in the soap.client.props file.</pre>
com.ibm.ssl.enabledCipherSuites	Commented out by default	This property enables you to specify a custom cipher suite list and override the group selection in the <code>com.ibm.ssl.securityLevel</code> property. The valid list of ciphers varies according to the provider and JVM policy files that are applied. For cipher suites, use a space as the delimiter.

Property	Default	Description
com.ibm.ssl.keyStoreName	ClientDefaultKeyStore	This property references a keystore configuration name. If you have not already defined the keystore, the rest of the keystore properties must follow this property. After you define the keystore, you can specify this property to reference the previously specified keystore configuration. New keystore configurations in the <code>ssl.client.props</code> file have a unique name.
com.ibm.ssl.trustStoreName	ClientDefaultTrustStore	This property references a truststore configuration name. If you have not already defined the truststore, the rest of the truststore properties must follow this property. After you define the truststore, you can specify this property to reference the previously specified truststore configuration. New truststore configurations in the <code>ssl.client.props</code> file should have a unique name.

Keystore configurations

SSL configurations reference keystore configurations whose purpose is to identify the location of certificates. Certificates represent the identity of clients that use the SSL configuration. You can specify keystore configurations with other SSL configuration properties. However, it is recommended that you specify the keystore configurations in this section of the `ssl.client.props` file after the `com.ibm.ssl.keyStoreName` property identifies the start of a new keystore configuration. After you fully define the keystore configuration, the `com.ibm.ssl.keyStoreName` property can reference the keystore configuration at any other point in the file.

Property	Default	Description
com.ibm.ssl.keyStoreName	ClientDefaultKeyStore	This property specifies the name of the keystore as it is referenced by the runtime. Other SSL configurations can reference this name further down in the <code>ssl.client.props</code> file to avoid duplication.
com.ibm.ssl.keyStore	<code>\${user.root}/etc/key.p12</code>	This property specifies the location of the keystore in the required format of the <code>com.ibm.ssl.keyStoreType</code> property. Typically, this property references a keystore file name. However, for cryptographic token types, this property references a Dynamic Link Library (DLL) file.
com.ibm.ssl.keyStorePassword	WebAS	This property is the default password, which is the cell name for the profile when it is created. The password is typically encoded using an <code>{xor}</code> algorithm. You can use <code>iKeyman</code> to change the password in the keystore, then change this reference. If you do not know the password and if the certificate is created for you, change the password in this property, then delete the keystore from the location where it resides. Restart the client to recreate the keystore by using the new password, but only if the keystore name ends with <code>DefaultKeyStore</code> and if the <code>fileBased</code> property is <code>true</code> . Delete both the keystore and truststore at the same time so that a proper signer exchange can occur when both are recreated together.
com.ibm.ssl.keyStoreType	PKCS12	This property is the keystore type. Use the default, PKCS12, because of its interoperability with other applications. You can specify this property as any valid keystore type that is supported by the JVM on the provider list.

Property	Default	Description
com.ibm.ssl.keyStoreProvider	IBMJCE	The IBM Java Cryptography Extension property is the keystore provider for the keystore type. The provider is typically IBMJCE or IBMPKCS11Impl for cryptographic devices.
com.ibm.ssl.keyStoreFileBased	true	This property indicates to the runtime that the keystore is file-based, meaning it is located on the file system.
com.ibm.ssl.keyStoreReadOnly	false	This property indicates to the runtime whether the keystore can be modified during the runtime.

Truststore Configurations

SSL configurations reference truststore configurations, whose purpose is to contain the signer certificates for servers that are trusted by this client. You can specify these properties with other SSL configuration properties. However, it is recommended that you specify truststore configurations in this section of the `ssl.client.props` file after the `com.ibm.ssl.trustStoreName` property has identified the start of a new truststore configuration. After you fully define the truststore configuration, the `com.ibm.ssl.trustStoreName` property can reference the configuration at any other point in the file.

A truststore is a keystore that JSSE uses for trust evaluation. A truststore contains the signers that WebSphere Application Server requires before it can trust the remote connection during the handshake. If you configure the `com.ibm.ssl.trustStoreName=ClientDefaultKeyStore` property, you can reference the keystore as the truststore. Further configuration is not required for the truststore because all of the signers that are generated through signer exchanges are imported into the keystore where they are called by the runtime.

Property	Default	Description
com.ibm.ssl.trustStoreName	ClientDefaultTrustStore	This property specifies the name of the truststore as it is referenced by the runtime. Other SSL configurations can reference further down in the <code>ssl.client.props</code> file to avoid duplication.
com.ibm.ssl.trustStore	<code>\${user.root}/etc/trust.p12</code>	This property specifies the location of the truststore in the format that is required by the truststore type that is referenced by the <code>com.ibm.ssl.trustStoreType</code> property. Typically, this property references a truststore file name. However, for cryptographic token types, this property references a DLL file.
com.ibm.ssl.trustStorePassword	WebAS	This property specifies the default password, which is the cell name for the profile when it is created. The password is typically encoded using an {xor} algorithm. You can use <code>iKeyman</code> to change the password in the keystore, then change the reference in this property. If you do not know the password and if the certificate was created for you, change the password in this property, then delete the truststore from the location where it resides. Restart the client to recreate the truststore by using the new password, but only if the keystore name ends with <code>DefaultTrustStore</code> and the <code>fileBased</code> property is <code>true</code> . It is recommended that you delete the keystore and the truststore at the same time so that a proper signer exchange can occur when both are recreated together.

Property	Default	Description
com.ibm.ssl.trustStoreType	PKCS12	This property is the truststore type. Use the default PKCS12 type because of its interoperability with other applications. You can specify this property as any valid truststore type that is supported by the JVM functionality on the provider list.
com.ibm.ssl.trustStoreProvider	IBMJCE	This property is the truststore provider for the truststore type. The provider is typically IBMJCE or IBMPKCS11Impl for cryptographic devices.
com.ibm.ssl.trustStoreFileBased	true	This property indicates to the runtime that the truststore is file-based, meaning it is located on the file system.
com.ibm.ssl.trustStoreReadOnly	false	This property indicates to the runtime whether the truststore can be modified during the runtime.

Creating a keystore configuration

A Secure Sockets Layer (SSL) configuration references keystore configurations during WebSphere Application Server runtime. Whether a keystore file was created by another keystore tool or saved from a previous configuration, the file must be part of a keystore configuration object. You can create a keystore configuration for the existing keystore object.

A keystore must already exist.

Complete the following steps in the administrative console:

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > SSL_configuration > Key stores and certificates > New**.
2. Type a name in the **Name** field. This name uniquely identifies the keystore in the configuration.
3. Type the location of the keystore file in the **Path** field. The location can be a file name or a file URL to an existing keystore file.
4. Type the keystore password in the **Password** field. This password is for the keystore file that you specified in the **Path** field.
5. Type the keystore password again in the **Confirm Password** field to confirm the password.
6. Select a keystore type from the list. The type that you select is for the keystore file that you specified in the **Path** field.
7. Select any of the following optional selections:
 - The **Read only selection** creates a keystore configuration object but does not create a keystore file. If this option is selected, the keystore file that you specified in the **Path** field must already exist.
 - The **Initialize at startup selection** initializes the keystore during runtime.
8. Click **Apply** and **Save**.

You have created a keystore configuration object for the keystore file that you specified. This keystore can now be used in an SSL configuration.

You can create additional keystore configurations, as needed.

Changing a keystore password

You can change the WebSphere Application Server password from the default password value, WebAS, in the administrative console or at a Java command prompt. Because the default password is well known, it is important that you change the password after you install WebSphere Application Server to protect the security of the keystore files and the Secure Sockets Layer (SSL) configuration.

When WebSphere Application Server starts for the first time as a standalone application server or as a Network Deployment Server, each server creates a keystore and truststore for the default SSL configuration. WebSphere Application Server creates these files and assigns the default password, WebAS.

Complete the following steps in the administrative console:

1. Click **Security > SSL certificate and key management > Key Stores and Certificates**.
2. Select a keystore.
3. Type the new keystore password in the **Change Password** field.
4. Type the keystore password again in the **Confirm Password** field to confirm the password.
5. Select **Apply**.

WebSphere Application Server and its keystores are protected by a unique password.

Use the new password the next time you start the WebSphere Application Server.

Configuring a hardware cryptographic keystore

You can create a hardware cryptographic keystore that WebSphere Application Server can use to provide cryptographic token support in the server configuration.

Complete the following steps in the administrative console:

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > SSL_configuration > Key stores and certificates**.
2. Click **New**.
3. Type a name to identify the keystore. This name is used to enable hardware cryptography in the Web services security configuration.
4. Type the path for the hardware device-specific configuration file. The configuration file is a text file that contains entries in the following format: *attribute = value*. The valid values for attribute and value are described in detail in the Software Developer Kit, Java Technology Edition documentation. The two mandatory attributes are name and library, as shown in the following sample code:

```
name = FooAccelerator
library = /opt/foo/lib/libpkcs11.so
slotListIndex = 0
```
5. Type a password if the token login is required. Operations that use keys on the token require a secure login. This field is optional if the keystore is used as a cryptographic accelerator. In this case, you need to select **Enable pure acceleration for hardware cryptographic operations**.
6. Select the **PKCS11** type.
7. Select **Read only**.
8. Click **OK** and **Save**.

WebSphere Application Server can now provide cryptographic token support in the server configuration.

You can refer to this keystore in any server Secure Sockets Layer (SSL) configuration to achieve the following results:

- Cryptographic acceleration because the cryptographic hardware device has no persistent key storage
- Secure cryptographic hardware because a cryptographic token generates and securely stores the private key that WebSphere Application Server uses for SSL key exchange.

You can also refer to this keystore in the Web services security default bindings configuration to achieve similar results.

Managing keystore configurations remotely

You can manage keystores remotely in a Network Deployment environment on separate machines. A node server can hold the configuration for a keystore, while the actual keystore resides on another system. After you set up a remotely managed configuration, you can perform all of the certificate and keystore operations for the keystore on the remote machine from the server that contains the keystore remote configuration.

Key stores can be remotely managed only in network deployed environments.

Complete the following steps in the administrative console:

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > ssl_configuration > Key stores and certificates**.
2. Click **New**.
3. Type a name in the **Name** field. This name uniquely identifies the keystore in the configuration.
4. Type the location of the keystore file in the **Path** field. The location can be a file name or a file Uniform Resource Locator (URL) to an existing keystore file.
5. Type the keystore password in the **Password** field. This password is for the keystore file that you specified in the **Path** field.
6. Type the keystore password again in the **Confirm Password** field to confirm the password.
7. Select a keystore type from the list. The type you select is for the keystore file that you specified in the **Path** field.
8. Select the **Remotely managed** check box, and then fill in one or more hosts names of the systems where the keystore file is to be located. If you provide multiple host names, separate the host names with a pipe (|).
9. Select any of the following optional selections:
 - The **Read only selection** creates a keystore configuration object but does not create a keystore file. If this option is selected, the keystore file that you specified in the **Path** field must already exist.
 - The **Initialize at startup selection** initializes the keystore during run time.
10. Select **Apply** and **Save**.

A keystore configuration object is created on the server from where the command was run. The keystore file for the configuration will be created on each system that you specified in the host list.

Now, you can perform all certificate management operations on the keystore from the system where the keystore configuration resides. For example, you can perform certificate management operations, such as: creating a self-signed certificate, extracting a certificate, or extracting a signer certificate.

To manage a self-signed certificates by using the wsadmin tool, use the PersonalCertificateCommands group commands of the AdminTask object. For more information, see “Commands for the PersonalCertificateCommands group of the AdminTask object” on page 862.

Key stores and certificates collection

Use this page to manage key store types, including cryptography, Resource Access Control Facility (RACF) , Certificate Management Services (CMS), Java, and all trust store types.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > Inbound | Outbound > SSL_configuration_name**. Under Related items, click **Key stores and certificates**.

Button	Resulting action
New	Adds a new key store object that can be referenced by Secure Sockets Layer (SSL) configurations or KeySets. The KeyStore management scope is based on the part of the topology tree from which it was created.
Delete	Deletes an existing key store. The key store should not be referenced by any other parts of the configuration before you delete it.
Exchange signers	Refers to exchanging signers in a key store. You can select two key stores, along with personal certificates from a selected key store, then add them as a signer to another selected key store.

Name

Specifies the unique name that is used to identify the key store. This name is typically scoped by the ManagementScope scopeName and based upon the location of the key store. The name must be unique within the existing key store collection.

This is a user-defined name.

Path

Specifies the location of the key store file in the format needed by the key store type. This file can be a card-specific configuration file for cryptographic devices or a filename or file URL for file-based key stores. It can be a safkeyring URL for RACF keyrings.

Key store settings

Use this page to create all keystore types, including cryptographic, Resource Access Control Facility (RACF), Certificate Management Services (CMS), Java, and all truststore types.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound SSL_configuration_name}**. Under Related Items, click **Key stores and certificates**. Click either **New** or an existing keystore.

Links to Personal certificates, Signer certificates, and Personal certificate requests enable you to manage certificates in a manner similar to iKeyman capabilities. A keystore can be file-based, such as CMS or Java keystore types, or it can be remotely managed.

Note: Any changes made to this panel are permanent.

Name

Specifies the unique name to identify the keystore. The keystore is typically scoped by the ManagementScope scopeName based on the location of the keystore. The name must be unique within the existing keystore collection.

Data type: Text

Path

Specifies the location of the keystore file in the format needed by the keystore type. This file can be a dynamic link library (DLL) for cryptographic devices or a filename or file URL for file-based keystores. It can be a safkeyring URL for RACF keyrings.

Data type: Text

Enable cryptographic operations on hardware device

Specifies whether a hardware cryptographic device is used for cryptographic operations only. Operations that require a login are not supported when using this option.

Default: Disabled

Password [new keystore] | Change password [existing keystore]

Specifies the password used to protect the keystore. For the default keystore (names ending in DefaultKeyStore or DefaultTrustStore), the password is the Cell name. This default password must be changed.

This field can be edited.

Data type: Text

Confirm password

Specifies confirmation of the password to open the keystore file or device.

Data type: Text

Type

Specifies the implementation for keystore management. This value defines the tool that operates on this keystore type.

The list of options is returned by `java.security.Security.getAlgorithms("KeyStore")`. Some options might be filtered and some might be added based on the `java.security` configuration.

Data type: Text
Default: PKCS12

Read only

Specifies whether the keystore can be written to or not. If the keystore cannot be written to, certain operations cannot be performed, such as creating or importing certificates.

Default: Disabled

Initialize at startup

Specifies whether the keystore needs to be initialized before it can be used for cryptographic operations. If enabled, the keystore is initialized at server startup.

Default: Disabled

Key managers collection

Use this page to define the implementation settings for key managers. A key manager is invoked during a Secure Sockets Layer (SSL) handshake to determine which certificate alias is used. The default key manager (`WSX509KeyManager`) performs alias selection. If more advanced function is desired, define a custom key manager and select it on the **Manage endpoint security configurations** panel.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > Inbound | Outbound > *SSL_configuration_name***. Under Related items, click **Key managers**.

Button	Resulting action
New	Adds a new key manager that can be selected by an SSL configuration. A key manager is invoked during an SSL handshake to select a specific certificate alias to use from a key store.

Button	Resulting action
Delete	Deletes an existing key manager. The key manager should not be referenced by any SSL configuration before you can delete it.

Name

Specifies the name of the key manager, which you can select on the SSL configuration panel.

Class name

Specifies the name of the key manager implementation class. This class implements javax.net.ssl.X509KeyManager interface and, optionally, the com.ibm.wsspi.ssl.KeyManagerExtendedInfo interface.

Algorithm

Specifies the algorithm name of the key manager that is implemented by the selected provider.

Key managers settings

Use this page to define key managers implementation settings. A key manager gets invoked during an Secure Sockets Layer (SSL) handshake to determine the certificate alias to be used. The default key manager (WSX509KeyManager) performs alias selection. If more advanced function is desired, a custom key manager can be specified here and selected in the SSL configuration.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > Inbound | Outbound > *SSL_configuration_name***. Under Related items, click **Key managers > New**.

Name

Specifies the name of the key manager, which you can select on the SSL configuration panel.

Data type: Text

Standard

Specifies the key manager selection that is available from a Java provider that is installed in the java.security file. This provider might be shipped by Java Secure Sockets Extension (JSSE) or be a custom provider that implements an X509KeyManager interface.

Default: Enabled

Provider

Specifies the provider name that has an implementation of an X509KeyManager interface. This provider is typically set to IBMJSSE2.

Data type: Text

Default: IBMJCE

Algorithm

Specifies the algorithm name of the trust manager implemented by the selected provider.

Data type: Text

Default: IbmX509

Custom

Specifies that the key manager selection is based on a custom implementation class that implements the `javax.net.ssl.X509KeyManager` interface and optionally the `com.ibm.wsspi.ssl.KeyManagerExtendedInfo` interface to obtain additional connection information not otherwise available.

Default: Disabled

Class name

Specifies the name of the key manager implementation class.

Data type: Text

Creating a self-signed certificate

You can create a self-signed certificate. WebSphere Application Server uses the certificate at runtime during the handshake protocol. Self-signed certificates are located in the default keystore.

You must create a keystore before you can create a self-signed certificate.

Complete the following steps in the administrative console:

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > ssl_configuration > Key stores and certificates > [keystore]**.
2. From Additional Properties, click **Personal certificates**.
3. Click **Create a self-signed certificate**.
4. Type a certificate alias name. The alias identifies the certificate request in the keystore.
5. Type a common name (CN) value. This value is the CN value in the certificate distinguished name (DN).
6. Type an organization value. This value is the O value in the certificate DN.
7. You can configure one or more of the following optional values:
 - a. **Optional:** Select a key size value. The default key size value is 1024 bits.
 - b. **Optional:** Type an organizational unit value. This organizational unit value is the OU value in the certificate DN.
 - c. **Optional:** Type a locality value. This locality value is the L value in the certificate DN.
 - d. **Optional:** Type a state or providence value. This value is the ST value in the certificate DN.
 - e. **Optional:** Type a zip code value. This zip code value is the POSTALCODE value in the certificate DN.
 - f. **Optional:** Select a country value from the list. This country value is the C= value in the certificate request DN.
8. Click **Apply**.

You have created a self-signed certificate that resides in the keystore. The SSL configuration for the WebSphere Application Server runtime uses this certificate for SSL communication. Extract the signer of the self-signed certificate to add the signer to another keystore.

To create a self-signed certificate by using the `wsadmin` tool, use the `createSelfSignedCertificate` command of the `AdminTask` object. For more information, see “Commands for the `PersonalCertificateCommands` group of the `AdminTask` object” on page 862.

Replacing an existing self-signed certificate

Occasionally, you need to replace an existing or expired self-signed certificate with a new certificate. Certificates are referenced in the runtime configuration by the Secure Sockets Layer (SSL) Configuration

object and the Dynamic SSL Configuration Selection object. You can replace a certificate with a new certificate alias reference or with a new signer certificate.

The current certificate and the certificate replacement must exist in the same keystore before you can replace a certificate.

Complete the following steps in the administrative console:

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > ssl_configuration > Key stores and certificates > [keystore]**.
2. Under Additional Properties, click **Personal certificates**.
3. Select a personal certificate. The alias list must include at least two certificates that reside in the keystore.
4. Click **Replace**.
5. Select a replacement certificate alias from the list.
6. You can delete one of the following types of certificates:
 - Select **Delete old certificate** to delete the existing certificate.
 - Select **Delete old signers** to delete the existing signer certificates.
7. Click **Apply**.

Your results depend on what you selected:

- If you selected **Delete old certificate**, the new certificate alias replaces all of the references to the certificate alias in the configuration.
- If you selected **Delete old signers**, the new signer certificate replaces all of the occurrences of the old signer certificates.
- If the new certificate alias replaces the existing alias, the WebSphere Application Server runtime checks to make sure that:
 - All of the SSL Configurations objects reference the certificate
 - The Dynamic SSL Configuration Selections objects and the SSL Configuration group objects reference the certificate.
- If you selected **Delete old signers**, the existing signer certificates are replaced.
- If you selected **Delete old certificate**, the existing certificate are deleted.

To replace a self-signed certificate by using the wsadmin tool, use the **replaceCertificate** command of the AdminTask object. For more information, see `rxml_atpersonalcert.dita`.

Creating a certificate authority request

To ensure Secure Sockets Layer (SSL) communication, servers require a personal certificate that is either self-signed or signed by a certificate authority (CA). You must first create a personal certificate request to obtain a certificate that is signed by a CA.

The keystore that contains a personal certificate request must already exist.

Complete the following steps in the administrative console:

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > ssl_configuration > Key stores and certificates > keystore**.
2. Click **Personal certificate requests > New**.
3. Type the full path of the certificate request file. The certificate request is created in this location.
4. Type an alias name in the **Key label** field. The alias identifies the certificate request in the keystore.
5. Type a common name (CN) value. This value is the CN value in the certificate distinguished name (DN).

6. Type an organization value. This value is the O value in the certificate DN.
7. You can configure one or more of the following optional values:
 - a. **Optional:** Select a key size value. The default key size value is 1024 bits.
 - b. **Optional:** Type an organizational unit value. This organizational unit value is the OU value in the certificate DN.
 - c. **Optional:** Type a locality value. This locality value is the L value in the certificate DN.
 - d. **Optional:** Type a state or province value. This value is the ST value in the certificate DN.
 - e. **Optional:** Type a zip code value. The zip code value is the POSTALCODE value in the certificate DN.
 - f. **Optional:** Select a country value from the list. This country value is the C= value in the certificate request DN.
8. Click **Apply**.

The certificate request is created in the specified file location in the keystore. The request functions as a temporary placeholder for the signed certificate until you manually receive the certificate in the keystore.

To create a certificate request using the wsadmin tool, use the **createCertificateRequest** command of the AdminTask object. For more information, see “Commands for the PersonalCertificateCommands group of the AdminTask object” on page 862.

Note: Key store tools (such as iKeyman and keyTool) cannot receive signed certificates that are generated by certificate requests from WebSphere Application Server. Similarly, WebSphere Application Server cannot accept certificates that are generated by certificate requests from other keystore utilities.

Now you can receive the CA-signed certificate into the keystore to complete the process of generating a signed certificate for your server.

Certificate request settings

Use this page to verify the properties of a personal certificate request.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > Inbound | Outbound > *ssl_configuration***. Under Related items, click **Key stores and certificates > *key store***. Under Additional Properties, click **Personal certificate requests > *certificate request***.

Key label

Specifies the certificate alias name for the signer in the key store, which is specified in the SSL configuration.

Key size

Specifies the size of the keys that are generated.

Requested by

Specifies the Subject distinguished name (DN) that represents the identity of the certificate request.

Fingerprint (SHA Digest)

Specifies the SHA hash of the personal certificate, which can be used to verify that the certificate has not been altered when it is used in a remote connection.

Signature algorithm

Specifies the algorithm used to sign the certificate.

Personal certificates collection

Use this page to manage personal certificates.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > ssl_configuration**. Under Related items, click **Key stores and certificates > key store**. Under Additional Properties, click **Personal certificates**.

The **Personal certificates** page lists all Personal certificates in the selected key store. You can do most certificate management operations in this panel, including creating a new self-signed certificate, deleting a certificate, receiving one generated from a CA, replacing a certificate (simultaneous delete and create, replacing references across all key stores), extracting the signer, and importing or exporting a personal certificate.

Personal certificate requests are temporary place holders for certificates that will be signed by a certificate authority (CA).

The Key store collection must contain at least two key store files. You must select one file in order to replace, extract, or export a key store,

Button	Resulting action
Create a self-signed certificate	Enables the application server to create a new self-signed certificate.
Delete	Specifies to delete a certificate from the key store. Be careful that the certificate alias is not referenced elsewhere in the Secure Sockets Layer configuration.
Receive a certificate from a certificate authority	Enables the application server to receive a certificate authority (CA)-generated certificate from a file to complete a certificate request.
Replace	Replaces a self-signed certificate with another self-signed certificate that contains the same information, but with a new expiration period. The signer from the old certificate that is contained in any managed key store in the cell is replaced by the signer from the new certificate.
Extract	Extracts a certificate from the key store that will be added to another key store as a trusted certificate (signer).
Import	Imports a certificate, including the private key, from a key store file.
Export	Exports a certificate, including the private key, to a specified key store file.

Alias

Specifies the alias by which the personal certificate is referenced in the key store.

When you select an alias, the View Certificate panel opens.

Issued by

Specifies the distinguished name of the entity by which the certificate was issued. This name is the same as the issued-to distinguished name when the personal certificate is self-signed.

Issued to

Specifies the distinguished name of the entity to which the certificate was issued.

Serial number

Specifies the certificate serial number that is generated by the issuer of the certificate.

Expiration

Specifies the expiration date of the signer certificate for validation purposes.

Personal certificates settings

Use this page to create new personal certificates.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > Inbound | Outbound > *SSL_configuration_name***. Under Related items, click **Key stores and certificates > *key store***. Under Additional Properties, click **Personal certificates > Create a self-signed certificate**.

This same help file is available when you create a new certificate or modify an existing certificate. The fields in this file are marked according to when they show on the administrative console.

Alias

Specifies the alias for the personal certificate in the key store.

This field displays when you create a new certificate. This field is read-only when you view an existing certificate.

Data type: Text

Version

Specifies the version of the personal certificate. Valid versions include X509 V3, X509 V2, or X509 V1. It is recommended to use X509 V3 certificates.

This field is read-only when you create or view a certificate.

Data type: Text
Default: X509 V3
Range:

Key size

Specifies the key size of the private key that is used by the personal certificate.

This field displays when you create or view a certificate.

Data type: Integer
Default: 1024

Common name

Specifies the common name portion of the distinguished name (DN). It is recommended that this name be the host name of the machine on which the certificate resides. In some cases, the common name is used to login during Secure Socket Layer (SSL) certificate authentication; therefore, in some cases, this name might be used as a user ID for a local operating system registry.

This field displays when you create a new certificate, but does not display when you view an existing certificate.

Data type: Text

Serial number

Specifies the certificate serial number that is generated by the issuer of the certificate.

This field displays only when you view an existing certificate.

Validity period

Specifies the length in days during which the certificate is valid. The default is 365 days.

This field displays when you create or view a certificate.

Data type: Text

Organization

Specifies the organization portion of the distinguished name.

This field displays only when you create a new certificate.

Data type: Text

Organization unit

Specifies the organization unit portion of the distinguished name. This field is optional.

This field displays only when you create a new certificate.

Data type: Text

Locality

Specifies the locality portion of the distinguished name. This field is optional.

This field displays only when you create a new certificate.

Data type: Text

State/Province

Specifies the state portion of the distinguished name. This field is optional.

This field displays only when you create a new certificate.

Data type: Text

Zip code

Specifies the zip code portion of the distinguished name. This field is optional.

This field displays only when you create a new certificate.

Data type: Integer

Country or region

Specifies the country portion of the distinguished name.

This field displays only when you create a new certificate.

Data type: Text
Default: (none)

Refer to <http://www.iso.org/iso/en/prods-services/iso3166ma/02iso-3166-code-lists/list-en1.html> for a list of ISO 3166 country codes.

Validity period

Specifies the length, in days, when the certificate is valid. The default is 365 days.

This read-only field displays only when you view an existing certificate.

Issued to

Specifies the distinguished name of the entity to which the certificate was issued.

This read-only field displays only when you view an existing certificate.

Issued by

Specifies the distinguished name of the entity that issued the certificate. When the personal certificate is self-signed, this name is identical to the **Issued to** distinguished name.

This read-only field displays only when you view an existing certificate.

Fingerprint (SHA Digest)

Specifies the Secure Hash Algorithm (SHA hash) of the certificate, which can be used to verify the certificate's hash at another location, such as the client side of a connection.

This read-only field displays only when you view an existing certificate.

Signature algorithm

Specifies the algorithm used to sign the certificate.

This read-only field displays only when you view an existing certificate.

Personal certificate requests collection

Use this page to manage personal certificate requests. Personal certificate requests are temporary placeholders for certificates that will be signed by a certificate authority (CA).

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > ssl_configuration** . Under Related items, click **Key stores and certificates > key store** . Under Additional Properties, click **Personal certificate requests**.

A private key is generated during the certificate request generation, but only the certificate is sent to the CA. The CA generates a new certificate, signed by the CA. This can be added in the Personal Certificates panel.

Button	Resulting action
New	Creates a personal certificate request that can be given to a certificate authority to complete.
Delete	Deletes a personal certificate request.
Extract	Extracts a personal certificate request. Only one certificate request can be selected at a time.

Note: Any changes made to this panel are permanent.

Key label

Specifies the alias that represents the personal certificate request in the key store.

Requested by

Specifies the Subject distinguished name (DN) that represents the identity of the certificate request.

Personal certificate requests settings

Use this page to create a new certificate request that can be extracted and sent to a certificate authority (CA).

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > ssl_configuration** . Under Related items, click **Key stores and certificates > key store** . Under Additional Properties, click **Personal certificate requests > New**.

Personal certificate requests are temporary place holders for certificates that will be signed by a certificate authority (CA). The private key is generated during the certificate request generation, but only the certificate is sent to the CA. The CA generates a new certificate, signed by the CA.

Note: Any changes made to this panel are permanent.

File for certificate request

Specifies the fully qualified file name from which the certificate request is exported. This portion of the certificate request can be given to the certificate authority to generate the real certificate. After the real certificate is generated, you can perform an "Receive a certificate from a certificate authority" from the personal certificate collection view.

Data type: Text

Key label

Specifies the alias that represents the personal certificate request in the key store.

Data type: Text

Key size

Specifies the size of the keys that are generated.

Data type: Integer
Default: 1024

Common name

Specifies the name of the entity that the certificate represents. This common name can represent a person, company, or machine. For Web sites, the common name is frequently the DNS host name where the server resides.

Data type: Text

Organization

Specifies the organization portion of the distinguished name.

Data type: Text

Organizational unit

Specifies the organization unit portion of the distinguished name. This field is optional.

Data type: Text

Locality

Specifies the locality portion of the distinguished name. This field is optional.

Data type: Text

State/Province

Specifies the state portion of the distinguished name. This field is optional.

Data type: Text

Zip code

Specifies the zip code portion of the distinguished name. This field is optional.

Data type: Integer

Country or region

Specifies the country portion of the distinguished name.

Data type: Text

Default: US

Refer to <http://www.iso.org/iso/en/prods-services/iso3166ma/02iso-3166-code-lists/list-en1.html> for a list of ISO 3166 country codes.

Extract certificate request

Use this page to extract a certificate request and send it to a certificate authority (CA).

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > ssl_configuration** . Under Related items, click **Key stores and certificates > key store** . Under Additional Properties, click **Personal certificate requests > Extract**.

Key label

Specifies the alias that represents the personal certificate request in the key store.

File for certificate request

Specifies the filename where the extracted certificate request is placed.

Data type: Text

Receiving a certificate issued by a certificate authority

When a certificate authority (CA) receives a certificate request, it issues a new certificate that functions as a temporary placeholder for a CA-issued certificate. A keystore receives the certificate from the CA and generates a CA-signed personal certificate that WebSphere Application Server can use for Secure Sockets Layer (SSL) security.

The keystore must contain the certificate request that was created and sent to the CA. Also, the keystore must be able to access the certificate that is returned by the CA.

WebSphere Application Server can receive only those certificates that are generated by a WebSphere Application Server certificate request. It cannot receive certificates that are created with certificate requests from other keystore tools, such as **iKeyman** and **keyTool**.

Complete the following steps in the administrative console:

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > ssl_configuration > Key stores and certificates > [keystore]**.
2. Under Additional Properties, click **Personal certificates**.
3. Select a personal certificate.
4. Click **Receive a certificate from a certificate authority**.
5. Type the full path and name of the certificate file.
6. Select a data type from the list.
7. Click **Apply** and **Save**.

The keystore contains a new personal certificate that is issued by a CA. The original certificate request is changed to a personal certificate.

The SSL configuration is ready to use the new CA-signed personal certificate.

To receive a certificate by using the wsadmin tool, use the **receiveCertificate** command of the AdminTask object. For more information, see “Commands for the PersonalCertificateCommands group of the AdminTask object” on page 862.

Export certificate to a key file

Use this page to specify a personal certificate to export to a key file.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > ssl_configuration** . Under Related items, click **Key stores and certificates > key store** . Under Additional Properties, click **Personal certificates > Export**.

Certificate alias to export:

Displays the name of the certificate that you selected to export on the previous panel.

Data type: Text

Alias:

Specifies the alias that the personal certificate is referenced by in the key store.

Data type: Text

Key file name:

Specifies the key store file name into which the exported certificate is added. If the key store file name already exists, the exported certificate will be added. If the key store file name does not already exist, one will be created, and the exported certificate will be added.

Data type: Text

Type:

Specifies the type of key store file. The valid types are listed in the menu.

Data type: Text

Default: JKS

Key file password:

Specifies the password that is used to access the key store file.

Data type: Text

Confirm password:

Confirms the password entered in the previous field.

The key file password and the new alias must be specified in case a certificate already exists with the same name.

Data type: Text

Import certificate from a key file

Use this page to specify a personal certificate to import from a key file.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > Inbound | Outbound > SSL_configuration_name**. Under Related items, click **Key stores and certificates > key store**. Under Additional Properties, click **Personal certificates > Import certificates from a key file**.

Key file name:

Specifies the fully qualified path to keystore file that contains the certificate to import.

Data type: Text

Type:

Specifies the type of keystore file. The valid types are listed in the menu.

Data type: Text

Key file password:

Specifies the password that is used to access the keystore file.

Data type: Text

Certificate alias to import:

Specifies the certificate alias identified as the **Key file name** that you want to import into the current key store.

Data type: Text
Default: (none)

Imported certificate alias:

Specifies the new alias that you want the certificate to be named in the current key store.

Data type: Text

Receive certificate from CA

Use this page to import your personal certificate from the certificate authority (CA). The imported certificate replaces the temporary certificate associated with the public/private keys in the certificate request that is stored in the key store.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > ssl_configuration** . Under Related items click **Key stores and certificates > key store** . Under Additional Properties, click **Personal certificates > Receive certificate from CA**.

Certificate file name:

Specifies the filename that contains the certificate generated by the certificate authority (CA).

Data type: Text

Data type:

Specifies the format of the file that is either Base64 encoded ASCII data or Binary DER data.

Data type: Text

Default: Base64-encoded ASCII data

Replace a certificate

Use this page to specify two certificates: the first selected certificate is replaced by the second selected certificate. The replace function replaces all the old signer certificates in key stores that are managed throughout the cell with the new signer from the new certificate. The same level of trust that was established with the old certificate is maintained.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > Inbound | Outbound > ssl_configuration** . Under Related items, click **Key stores and certificates > key store** . Under Additional Properties click **Personal certificates > Replace certificate**.

Old certificate

Specifies the certificate that you want to replace.

Data type: Text

Replace with

Specifies the certificate that you want to replace the old certificate.

Data type: Text

Default: (none)

Delete old certificate after replacement

Specifies that you want to delete the old certificate and all associated signer certificates after the new certificate replaces it. If you do not replace the old personal certificate, it might be assigned a new alias name.

Default: Disabled

Delete old signers

Specifies that you want to delete the old signer certificates that are associated with the old certificate after the new signer certificates replace them. If you do not delete the old signer certificates, they might be assigned new alias names.

Default: Disabled

Extracting a signer certificate from a personal certificate

Personal certificates contain a private key and a public key. You can extract the public key, called the *signer certificate*, to a file, then import the certificate into another keystore. The client requires the signer portion of a personal certificate for Security Socket Layer (SSL) communication.

The keystore that contains a personal certificate must already exist.

Complete the following steps in the administrative console:

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > ssl_configuration > Key stores and certificates > keystore** .
2. Under Additional Properties, click **Personal certificates**.
3. Select a personal certificate.
4. Click **Extract**.
5. Type the full path for the certificate file name. The signer certificate is written to this certificate file.
6. Select a data type from the list.
7. Click **Apply**.

The signer portion of the personal certificate is stored in the file that is provided.

This signer can now be imported into other keystores.

To extract a signer certificate from a personal certificate using the wsadmin tool, use the **extractCertificate** command of the AdminTask object. For more information, see “Commands for the PersonalCertificateCommands group of the AdminTask object” on page 862.

Extract certificate

Use this page to extract the signer from the personal certificate. The certificate can be imported into a trust store for trust verification.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > ssl_configuration** . Under Related items, click **Key stores and certificates > key store** . Under Additional Properties, click **Personal certificates > Extract**.

Certificate alias to extract

Specifies the filename that contains the extracted certificate.

Data type: Text

Certificate file name

Specifies the fully qualified path where the certificate file will reside.

Data type: Text

Data type

Specifies the format of the file, which is either Base64-encoded ASCII data or Binary DER data.

Data type: Text
Default: Base64-encoded ASCII data

Extract signer certificate

Use this page to extract a signer certificate that can be added elsewhere.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > Key stores and certificates > key store > Signer Certificates > Extract signer certificate.**

File name

Specifies the fully qualified file name where the extracted signer certificate is placed.

Data type: Text

Data type

Specifies the format of the file, which is either Base64 encoded ASCII data or Binary DER data.

Data type: Text

Retrieving signers using the retrieveSigners utility at the client

The client requires the signer certificates from the server to be able to communicate with WebSphere Application Server. Use the **retrieveSigners** command to get the signer certificate from a server.

The retrieveSigners utility is located in one of the following directories, depending on your operating system:

- On Windows operating system: *profile_root\bin* . For example: C:\WebSphere\AppServer\profiles\AppSrv01\bin
- On UNIX operating systems: *../profile_root/bin*

In this release, a Java client that does not have access to a stdin console prompt should use the retrieveSigners utility to download the signers from the remote server key store when signers are needed for a Secure Sockets Layer (SSL) handshake. For example, you might interpret the client as failing to respond if an applet client or Java Web Start Client application cannot access the stdin signer exchange prompt. Thus, you must add the WebSphere Java method call **com.ibm.wsspi.ssl.RetrieveSignersHelper.callRetrieveSigners** to your client application to retrieve the signers and to avoid running the retrieveSigners utility manually.

Use the retrieveSigners utility for situations where you cannot verify whether or not the `com.ibm.ssl.enableSignerExchangePrompt=` property is enabled or disabled when the application makes a request. Set the `com.ibm.ssl.enableSignerExchangePrompt=` property to `false` in the `ssl.client.props` file if you cannot see the console.

Alternatively, you can manually create the server key in the client truststore.

Complete the following steps, as required:

1. Use the **retrieveSigners** command to get the signer certificate from a server. You can find details about the **retrieveSigners**. parameters in “Secure installation for client signer retrieval” on page 404.
2. If the client and server are on the same machine, you will need only the *remoteKeyStoreName* and *localKeyStoreName* parameters. The most typical key store to reference on a remote system is *CellDefaultTrustStore* on a network deployed environment and *NodeDefaultTrustStore* on an application server.
3. When retrieving signers from a remote server, add these required connection-related parameters: **-host** *host*, **-port** *port*, **-conntype** {*RMI* | *SOAP*}.
4. If the remote server has security enabled, also supply the **-user** *user* **-password** *password* parameters.
5. Use the **-autoAcceptBootstrapSigner** parameter if you want to enable automation of the signer retrieval. This parameter automatically adds to the server all the signers that are needed to make the connection.

After running, the command displays the SHI-1 digest of the signers added. The output looks similar to the following output:

```
C:\WebSphere\AppServer\profiles\AppSrv01\bin\retrieveSigners.bat
CellDefaultTrustStore ClientDefaultTrustStore

CWPKI0308I: Adding signer alias "default_signer" to local keystore
             "ClientDefaultTrustStore" with the following SHA digest:
```

See the following examples of how to call the `retrieveSigners.bat` file on the Windows operating system:

To retrieve signers on the same system, enter:

```
'profile_home'\bin\retrieveSigners.bat CellDefaultTrustStore ClientDefaultTrustStore
```

To retrieve signers on a remote system with a SOAP connection, enter:

```
'profile_home'\bin\retrieveSigners.bat CellDefaultTrustStore ClientDefaultTrustStore
-host myRemoteHost -port 8879 -conntype SOAP -autoAcceptBootstrapSigner
```

To retrieve signers on a remote system with an RMI connection, enter:

```
'profile_home'\bin\retrieveSigners.bat CellDefaultTrustStore ClientDefaultTrustStore
-host myRemoteHost -port 2809 -conntype RMI -autoAcceptBootstrapSigner
```

To retrieve signers on a remote system that has security enabled, enter:

```
'profile_home'\bin\retrieveSigners.bat CellDefaultTrustStore ClientDefaultTrustStore
-host myRemoteHost -port 8879 -conntype SOAP -user testuser -password testuserpwd
-autoAcceptBootstrapSigner
```

Changing the signer auto-exchange prompt at the client

For clients to communicate with WebSphere Application Server, clients must obtain a signer certificate from the server. Clients can use the **retrieveSigners** command to connect to a server to obtain the appropriate signer. A prompt displays that asks whether or not you want to add a signer to the truststore. If the Secure Sockets Layer (SSL) configuration uses an automated script that might hang, use the prompt to obtain the certificate.

The `com.ibm.ssl.enableSignerExchangePrompt` property in the `profile_home/properties/ssl.client.props` file controls the signer certificate prompt. By default, this property is set to `true`, meaning the prompt is enabled.

Complete the following steps to disable or enable the signer-exchange prompt at the client:

1. Open the `profile_home/properties/ssl.client.props` file using an editor.
2. Locate the section containing the SSL configuration information for the client that you are working with.

3. Change the value of the `com.ibm.ssl.enableSignerExchangePrompt` property to `false` if you do not want the signer-exchange prompt, or set it to `true` if you want to be prompted.
4. Save and close the file.

When the `com.ibm.ssl.enableSignerExchangePrompt` property is set to `false`, no prompt displays, so you can exchange signers while some administrative clients are running. When the `com.ibm.ssl.enableSignerExchangePrompt` property is set to `true`, a signer-exchange prompt displays, and you are asked to accept or reject the certificate. The prompt looks like the following example:

```
C:\WebSphere\AppServer\profiles\dmgr\bin>serverStatus -all
ADMU0116I: Tool information is being logged in file
          C:\WebSphere\AppServer\profiles\Dmgr\logs\serverStatus.log
ADMU0128I: Starting tool with the dmgr profile
ADMU0503I: Retrieving server status for all servers
ADMU0505I: Servers found in configuration:
ADMU0506I: Server name: dmgr

*** SSL SIGNER EXCHANGE PROMPT ***
SSL signer from target host 192.174.1.5 is not found in truststore
C:/WebSphere/AppServer/profiles/Dmgr/etc/trust.p12.
```

Verify that the digest value matches what is displayed at the server in the following signer information:

```
Subject DN:  CN=hostname.austin.ibm.com, O=IBM, C=US
Issuer DN:   CN=hostname.austin.ibm.com, O=IBM, C=US
Serial number: 1128544457
Expires:     Thu Oct 20 15:34:17 CDT 2006
SHA-1 Digest: 91:A1:A9:2D:F2:7D:70:0F:04:06:73:A3:B4:A4:9C:56:9D:A8:A3:BA
MD5 Digest:  88:72:C5:88:00:1C:A7:FA:D6:EB:04:88:AC:A1:C9:13
```

```
Add signer to the truststore now? (y/n) y
A retry of the request might need to occur.
ADMU0508I: The Application Server "server1" is STARTED.
```

Clients can instigate communications for various processes using signer certificates obtained from WebSphere Application Server.

Importing a signer certificate from a truststore to a z/OS keyring

You can import a signer certificate, which is also called a certificate authority (CA) certificate, from a truststore on a non-z/OS platform server to a z/OS keyring.

To import a certificate to a z/OS keyring, complete the following steps:

1. On the non-z/OS platform server, change to the `install_root/bin` directory and start the `iKeyman` utility, which is called `keyman.bat` (Windows) or `keyman.sh` (UNIX). The `install_root` variable refers to the installation path for WebSphere Application Server.
2. Within the `iKeyman` utility, open the server truststore. The default server truststore is called the `DummyServerTrustFile.jks` file. The file is located in the `[/USER_INSTALL_ROOT]/etc/` directory. The default password is `WebAS`.
3. Extract the signer certificate from the truststore using the **`ikeyman`** utility. Complete the following steps to extract the signer certificate:
 - a. Select **Signer certificates** from the menu.
 - b. Select **new websphere dummy server alias**.
 - c. Select **Extract**.
 - d. Select the correct data type. The `signer_certificate` can have either a Base64-encoded ASCII data type or a Binary DER data type.
 - e. Specify the fully qualified path and the file name of the certificate.
4. From an FTP prompt on the non-z/OS platform server, type `cd bin` to change to binary mode.

5. From an FTP prompt on the non-z/OS platform server, type `put 'signer_certificate' mvs.dataset`. The *signer_certificate* variable refers to the name of the signer certificate on the non-z/OS platform server. The *mvs.dataset* variable is the data set name to which the certificate was exported.

The RACDCERT CERTAUTH ADD command in the next step works with a Multiple Virtual Storage (MVS) data set only. You can either turn the certificate file into a binary MVS data set or use the `put` command with an Hierarchical File System (HFS) file, and then use the following command to copy the file into a MVS data set:

```
0GET file_name mvs.dataset
```
6. On the z/OS platform server, go to option 6 in the Interactive System Productivity Facility (ISPF) dialog panels and issue the following commands as a super user to add the signer certificate to the z/OS keyring:
 - a. Type `RACDCERT CERTAUTH ADD ('signer_certificate') WITHLABEL('Dummy Server CA') TRUST` The Dummy Server CA variable refers to the label name for the certificate authority (CA) certificate that you are importing from a non-z/OS platform server. The *keyring_name* variable refers to the name of the z/OS keyring that is used by the servers in the cell.
 - b. Type `RACDCERT ID(ASCR1) CONNECT(CERTAUTH LABEL('Dummy Server CA') RING(keyring_name))`
 - c. Type `RACDCERT ID(DMCR1) CONNECT(CERTAUTH LABEL('Dummy Server CA') RING(keyring_name))`
 - d. Type `RACDCERT ID(DMSR1) CONNECT(CERTAUTH LABEL('Dummy Server CA') RING(keyring_name))` In the previous commands, ASCR1, DMCR1, and DMSR1 are the RACF IDs under which the started tasks for the cell run in WebSphere Application Server for z/OS. The ASCR1 value is the RACF ID for the application server control region. The DMCR1 value is the RACF ID for the deployment manager control region. The DMSR1 value is the RACF ID for the deployment manager server region.

After completing these steps, the z/OS keyring contains the signer certificates that originated on the non-z/OS platform server.

To verify that the certificates were added, use option 6 on the ISPF dialog panel and type the following command:

```
RACDCERT ID(CBSYMSR1) LISTRING(keyring_name)
```

The CBSYMSR1 value is the RACF ID for the application server region.

Note: Although iKeyman is supported for WebSphere Application Server Version 6.1, customers are encouraged to use the administrative console to export signer certificates.

Exporting a signer certificate from WebSphere Application Server for z/OS to a truststore

You can export a signer certificate, which is also called a certificate authority (CA) certificate, from WebSphere Application Server for z/OS to a truststore.

WebSphere Application Server, WebSphere Application Server Network Deployment, or WebSphere Application Server - Express can use the certificate in the truststore.

To export the certificate to a truststore, complete the following steps:

1. Export the z/OS[®] signer certificate to a data set by issuing the following Resource Access Control Facility (RACF) command as a super user using Time Sharing Option (TSO) option 6:


```
RACDCERT CERTAUTH EXPORT(LABEL('signer_certificate')) DSN('mvs.dataset') FORMAT(CERTDER)
```

The *signer_certificate* variable is the RACF label name of the certificate that is used by the cell. The *signer_certificate* can have either a Base64-encoded ASCII data type or a Binary DER data type. The *mvs.dataset* variable is the data set name to which the certificate is exported. You do not need to pre-allocate this data set because it is created by RACF.

2. From a command line on the non-z/OS platform server, type `cd` and change to the following directory:

install_root/profiles/default/etc

3. From an FTP prompt on the non-z/OS platform server, type `cd bin` to change to binary mode.
4. From an FTP prompt on the non-z/OS platform server, type the following command:

```
get 'mvs.dataset' signer_certificate
```
5. On the non-z/OS platform server, change to the *install_root*/bin directory and start the **iKeyman** utility, which is called `ikeyman.bat` for Windows or `ikeyman.sh` for UNIX.
6. Within the iKeyman utility, open the server truststore. The default server truststore is called the `DummyServerTrustFile.jks` file. The file is located in the `${USER_INSTALL_ROOT}/etc/` directory. The default password is WebAS. It is recommended that you create a new key file and trust file if you plan to use the certificate in a production environment.
7. Add your exported signer certificate to the server truststore using the iKeyman utility. Complete the following steps to add your exported signer certificate:
 - a. Select **Signer certificates** from the menu.
 - b. Select the correct data type. The signer certificate can have either a Base64-encoded ASCII data type or a Binary DER data type.
 - c. Specify the fully qualified path and file name of the signer certificate.
8. Within the iKeyman utility, open the client truststore. The default client truststore is called the `DummyClientTrustFile.jks` file. The file is located in the `${USER_INSTALL_ROOT}/etc/` directory. The default password is WebAS. It is recommended that you create a new key file and trust file if you plan to use the certificate in a production environment.
9. Add your exported signer certificate to the client truststore using the iKeyman utility. Complete the following steps to add your exported signer certificate:
 - a. Select **Signer certificates** from the menu.
 - b. Select the correct data type. The signer certificate can have either a Base64-encoded ASCII data type or a Binary DER data type.
 - c. Specify the fully qualified path and file name of the signer certificate.
10. Restart the server process to use the new signer certificates.

After completing these steps, you can use the exported signer certificates with the WebSphere Application Server, WebSphere Application Server Network Deployment, or WebSphere Application Server - Express products.

Importing a signer certificate from a truststore to a z/OS keyring

You can import a signer certificate, which is also called a certificate authority (CA) certificate, from a truststore on a non-z/OS platform server to a z/OS keyring.

To import a certificate to a z/OS keyring, complete the following steps:

1. On the non-z/OS platform server, change to the *install_root*/bin directory and start the iKeyman utility, which is called `ikeyman.bat` (Windows) or `ikeyman.sh` (UNIX). The *install_root* variable refers to the installation path for WebSphere Application Server.
2. Within the iKeyman utility, open the server truststore. The default server truststore is called the `DummyServerTrustFile.jks` file. The file is located in the `[$USER_INSTALL_ROOT]/etc/` directory. The default password is WebAS.
3. Extract the signer certificate from the truststore using the **ikeyman** utility. Complete the following steps to extract the signer certificate:
 - a. Select **Signer certificates** from the menu.
 - b. Select **new websphere dummy server alias**.
 - c. Select **Extract**.
 - d. Select the correct data type. The `signer_certificate` can have either a Base64-encoded ASCII data type or a Binary DER data type.

- e. Specify the fully qualified path and the file name of the certificate.
4. From an FTP prompt on the non-z/OS platform server, type `cd bin` to change to binary mode.
5. From an FTP prompt on the non-z/OS platform server, type `put 'signer_certificate' mvs.dataset`. The *signer_certificate* variable refers to the name of the signer certificate on the non-z/OS platform server. The *mvs.dataset* variable is the data set name to which the certificate was exported.
The RACDCERT CERTAUTH ADD command in the next step works with a Multiple Virtual Storage (MVS) data set only. You can either turn the certificate file into a binary MVS data set or use the `put` command with an Hierarchical File System (HFS) file, and then use the following command to copy the file into a MVS data set:

```
OGGET file_name mvs.dataset
```
6. On the z/OS platform server, go to option 6 in the Interactive System Productivity Facility (ISPF) dialog panels and issue the following commands as a super user to add the signer certificate to the z/OS keyring:
 - a. Type `RACDCERT CERTAUTH ADD ('signer_certificate') WITHLABEL('Dummy Server CA') TRUST` The Dummy Server CA variable refers to the label name for the certificate authority (CA) certificate that you are importing from a non-z/OS platform server. The *keyring_name* variable refers to the name of the z/OS keyring that is used by the servers in the cell.
 - b. Type `RACDCERT ID(ASCR1) CONNECT(CERTAUTH LABEL('Dummy Server CA') RING(keyring_name)`
 - c. Type `RACDCERT ID(DMCR1) CONNECT(CERTAUTH LABEL('Dummy Server CA') RING(keyring_name)`
 - d. Type `RACDCERT ID(DMSR1) CONNECT(CERTAUTH LABEL('Dummy Server CA') RING(keyring_name)` In the previous commands, ASCR1, DMCR1, and DMSR1 are the RACF IDs under which the started tasks for the cell run in WebSphere Application Server for z/OS. The ASCR1 value is the RACF ID for the application server control region. The DMCR1 value is the RACF ID for the deployment manager control region. The DMSR1 value is the RACF ID for the deployment manager server region.

After completing these steps, the z/OS keyring contains the signer certificates that originated on the non-z/OS platform server.

To verify that the certificates were added, use option 6 on the ISPF dialog panel and type the following command:

```
RACDCERT ID(CBSYMSR1) LISTRING(keyring_name)
```

The CBSYMSR1 value is the RACF ID for the application server region.

Note: Although iKeyman is supported for WebSphere Application Server Version 6.1, customers are encouraged to use the administrative console to export signer certificates.

Exporting a signer certificate from WebSphere Application Server for z/OS to a truststore

You can export a signer certificate, which is also called a certificate authority (CA) certificate, from WebSphere Application Server for z/OS to a truststore.

WebSphere Application Server, WebSphere Application Server Network Deployment, or WebSphere Application Server - Express can use the certificate in the truststore.

To export the certificate to a truststore, complete the following steps:

1. Export the z/OS[®] signer certificate to a data set by issuing the following Resource Access Control Facility (RACF) command as a super user using Time Sharing Option (TSO) option 6:

```
RACDCERT CERTAUTH EXPORT(LABEL('signer_certificate')) DSN('mvs.dataset') FORMAT(CERTDER)
```

The *signer_certificate* variable is the RACF label name of the certificate that is used by the cell. The *signer_certificate* can have either a Base64-encoded ASCII data type or a Binary DER data type. The

mvs.dataset variable is the data set name to which the certificate is exported. You do not need to pre-allocate this data set because it is created by RACF.

2. From a command line on the non-z/OS platform server, type `cd` and change to the following directory:
`install_root/profiles/default/etc`
3. From an FTP prompt on the non-z/OS platform server, type `cd bin` to change to binary mode.
4. From an FTP prompt on the non-z/OS platform server, type the following command:
`get 'mvs.dataset' signer_certificate`
5. On the non-z/OS platform server, change to the `install_root/bin` directory and start the **iKeyman** utility, which is called `ikeyman.bat` for Windows or `ikeyman.sh` for UNIX.
6. Within the iKeyman utility, open the server truststore. The default server truststore is called the `DummyServerTrustFile.jks` file. The file is located in the `${USER_INSTALL_ROOT}/etc/` directory. The default password is `WebAS`. It is recommended that you create a new key file and trust file if you plan to use the certificate in a production environment.
7. Add your exported signer certificate to the server truststore using the iKeyman utility. Complete the following steps to add your exported signer certificate:
 - a. Select **Signer certificates** from the menu.
 - b. Select the correct data type. The signer certificate can have either a Base64-encoded ASCII data type or a Binary DER data type.
 - c. Specify the fully qualified path and file name of the signer certificate.
8. Within the iKeyman utility, open the client truststore. The default client truststore is called the `DummyClientTrustFile.jks` file. The file is located in the `${USER_INSTALL_ROOT}/etc/` directory. The default password is `WebAS`. It is recommended that you create a new key file and trust file if you plan to use the certificate in a production environment.
9. Add your exported signer certificate to the client truststore using the iKeyman utility. Complete the following steps to add your exported signer certificate:
 - a. Select **Signer certificates** from the menu.
 - b. Select the correct data type. The signer certificate can have either a Base64-encoded ASCII data type or a Binary DER data type.
 - c. Specify the fully qualified path and file name of the signer certificate.
10. Restart the server process to use the new signer certificates.

After completing these steps, you can use the exported signer certificates with the WebSphere Application Server, WebSphere Application Server Network Deployment, or WebSphere Application Server - Express products.

Retrieving signers from a remote SSL port

To perform Secure Sockets Layer (SSL) communication with a server, WebSphere Application Server must retrieve a signer certificate from a secure remote SSL port during the handshake. After the signer certificate is retrieved, you can add the signer certificate to a keystore.

The keystore that is to contain the signer certificate must already exist.

Complete the following steps in the administrative console:

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > Key stores and certificates > keystore > Signer certificates > Retrieve from port**.
2. Click **Retrieve from port**.
3. Type the host name of the machine on which the signer resides.

4. Type the port location on the host machine on which the signer resides. The port location is not limited to ports on WebSphere Application Server. The ports can include Lightweight Directory Access Protocol (LDAP) ports or ports on any server on which an SSL port is already configured, such as `SIB_ENDPOINT_SECURE_ADDRESS`.
5. Select an SSL configuration for the outbound connection from the list.
6. Type an alias name for the certificate.
7. Click **Retrieve signer information**. A message window displays information about the retrieved signer certificate, such as: the serial number, issued-to and issued-by identities, SHA hash, and expiration date.
8. Click **Apply**. This action indicates that you accept the credentials of the signer.

The signer certificate that is retrieved from the remote port is stored in the keystore.

An SSL configuration or client process that requires an SSL connection to the server can use the retrieved and approved signer certificate.

To retrieve a signer certificate from a port using the wsadmin tool, use the **retrieveSignerFromPort** command of the AdminTask object. For more information, see “Commands for the PersonalCertificateCommands group of the AdminTask object” on page 862.

Retrieve from port

Use this page to retrieve a signer certificate from a remote SSL port. The system connects to the specified remote SSL host and port and receives the signer during the handshake using a trust manager. The signer SHA hash displays for validation and, if approved by an administrator, is added to the currently selected trust store.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > ssl_configuration**. Under Related items, click **Key stores and certificates > key store > Signer certificates > Retrieve from port**.

Host

Specifies the host name to which you connect when attempting to retrieve the signer certificate from the Secure Sockets Layer (SSL) port.

Data type: Text

Port

Specifies the SSL port to which you connect when attempting to retrieve the signer certificate.

Data type: Text

SSL configuration for outbound connection

Specifies the SSL configuration that is used to connect to the previously specified SSL port. This configuration is also the SSL configuration that contains the signer after retrieval. This SSL configuration does not need to have the trusted certificate for the SSL port as it is retrieved during validation and presented here.

Data type: Text
Default: DefaultSSLConfig

Alias

Specifies the certificate alias name that you want to reference the signer in the key store, which is specified in the SSL configuration.

Data type: Text

Retrieved signer information

Specifies the signer certificate information if it is retrieved from the remote host and port.

Serial number

Specifies the certificate serial number that is generated by the issuer of the certificate.

Issued to

Specifies the distinguished name of the entity to which the certificate was issued.

Issued by

Specifies the distinguished name of the entity that issued the certificate. This name is the same as the issued-to distinguished name when the signer certificate is self-signed.

Fingerprint (SHA Digest)

Specifies the Secure Hash Algorithm (SHA hash) of the certificate, which can be used to verify the certificate's hash at another location, such as the client side of a connection.

Expiration

Specifies the expiration date of the retrieved signer certificate for validation purposes.

Adding a signer certificate to a keystore

Signer certificates establish the trust relationship in SSL communication. You can extract the signer part of a personal certificate from a keystore, and then you can add the signer certificate to other keystores.

The keystore that you want to add the signer certificate to must already exist.

Complete the following steps in the administrative console:

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations > Inbound | Outbound > SSL_configuration_name > Key stores and certificates.**
2. Select a keystore from the list of keystores.
3. Click **Add signers.**
4. Enter an alias for the signer certificate in the **Alias** field
5. Enter the full path to the signer certificate file in the **File name** field.
6. Select a data type from the list in the **Data type** field.
7. Click **Apply.**

When these steps are completed, the signer from the certificate file is stored in the keystore. You can see the signer in the keystore files list of signer certificates. Use the keystore to establish trust relationships for the SSL configurations.

To add a signer certificate to a keystore by using the wsadmin tool, use the **addSignerCertificate** command of the AdminTask object. For more information, see `rxml_atpersonalcert.dita`.

Add signer certificate

Use this page to add a signer certificate to the key store.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > ssl_configuration**. Under Related items click **Key stores and certificates > key store > Signer certificates > Add.**

Alias

Specifies the alias that the signer certificate is referenced by in the key store.

Data type: Text

File name

Specifies the filename where the encoded signer certificate is located.

Data type: Text

Data type

Specifies the format of the file, which is either Base64 encoded ASCII data or Binary DER data.

Data type: Text

Signer certificates collection

Use this page to manage signer certificates in key stores. Signer certificates are used by Java Secure Socket Extensions (JSSE) to validate certificates sent by the remote side of the connection during a Secure Sockets Layer (SSL) handshake. If a signer does not exist in the trust store that can validate the certificate sent, the handshake fails and generates an "unknown certificate" error.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > ssl_configuration** . Under Related items click **Key stores and certificates > key store > Signer certificates**.

Button	Resulting action
Add	Adds a new trusted (signer) certificate.
Delete	Deletes an existing signer certificate.
Extract	Extracts a signer certificate from a personal certificate to a file.
Retrieve from port	Makes a test connection to an SSL port and retrieves the signer from the server during the handshake. The information from the certificate will be displayed so you can decide whether to trust it based upon the MD5 and/or SHA hash.

Alias

Specifies the alias for this signer certificate in the key store.

Issued to

Specifies the distinguished name of the entity that requested the certificate.

Fingerprint (SHA digest)

Specifies the Secure Hash Algorithm (SHA hash) of the certificate. This can be used to verify the hash for the certificate at another location, such as the client side of a connection.

Expiration

Specifies the expiration date of the signer certificate for validation purposes.

Signer certificate settings

Use this page to verify the general properties of the selected signer certificate.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > ssl_configuration** . Under Related items, click **Key stores and certificates > key store > Signer certificates > signer certificate** .

Alias

Specifies the alias for this signer certificate in the key store.

Version

Specifies the version of the personal certificate. Valid versions include X509 V3, X509 V2, or X509 V1.

Key size

Specifies the key size of the public key used by the signer certificate.

Serial number

Specifies the certificate serial number that is generated by the issuer of the certificate.

Validity period

Specifies the begin and end dates of the certificate.

Issued to

Specifies the distinguished name of the entity that requested the certificate.

Issued by

Specifies the distinguished name of the entity that issued the certificate. This name is the same as the issued-to distinguished name when the signer certificate is self-signed.

Fingerprint (SHA Digest)

Specifies the Secure Hash Algorithm (SHA) hash of the certificate, which can be used to verify the hash for the certificate at another location such as the client side of a connection.

Signature algorithm

Specifies the algorithm that is used to sign the certificate.

Exchanging signer certificates

To establish trust relationships, you can exchange signer certificates between keystores. When you exchange signer certificates, you are extracting a personal certificate from one keystore and adding it to another keystore as a *signer certificate*.

To exchange signer certificates, there must be two keystores.

Complete the following steps in the administrative console:

1. Click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > ssl_configuration > Key stores and certificates**.
2. Select two keystores from the list of keystores.
3. Click **Exchange signers**.
4. Select any of the certificates in the first personal certificates list, and click **Add**. After adding, the signer part of the selected personal certificate appears in the other (second) keystore signers list.
5. Select any of the certificates in the second personal certificates list, and click **Add**. After adding, the signer part of the selected personal certificate appears in the other (first) keystore signers list.
6. **Optional:** If you need to remove any of the certificates from either of the signers lists, highlight one or more of the certificates, and click **Remove**.
7. Click **Apply** and **Save**.

The signer certificate appears in the list for each keystore.

The extracted signer certificate is available to both keystores during the connection handshake.

Key stores and certificates exchange signers

Use this page to extract a personal certificate from one keystore and add it to another keystore as a signer certificate.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > ssl_configuration**. Under Related items, click **Key stores and certificates > Exchange signers**.

Note: Any changes made to this panel are permanent.

[key store] personal certificates

Specifies the personal certificates that are currently stored in the specified key store.

Press and hold the **Ctrl** key to select more than one item from the list.

Data type: Text

[key store] signers

Specifies the trusted signer certificates that are currently stored in the specified key store and selected for the exchange.

Press and hold the **Ctrl** key to select more than one item from the list.

Data type: Text

Add

Specifies to extract the signer from the selected personal certificate in the key store list on the left and add it to the signers list of the key store on the right.

After the certificate is added, it no longer displays in the left-hand list. The personal certificate is still in the key store, but it is no longer selectable

Remove

Specifies to remove a selected signer from the signers list of the key store on the right. The removed certificate displays in the key store list on the left.

Configuring certificate expiration monitoring

When certificates expire, they can no longer be used by the system. WebSphere Application Server provides a utility to monitor certificates that are close to expiration or have already expired. You can schedule certificate monitoring, or you can request certificate monitoring on demand. You can also configure options for deleting expired certificates and for recreating certificates.

WebSphere Application Server notifies you when a certificate is about to expire. Complete the information required for notification messaging in “Notifications” on page 485.

Complete the following configuration steps in the administrative console:

1. Click **Security > SSL certificate and key management > Manage certificate expiration**.
2. Type a number for the number of days threshold in the **Expiration notification threshold** field. WebSphere Application Server issues an expiration warning *n* number of days before expiration.
3. Select or check one or more of the following options:

- **Expiration check notification.** Select the method from the list that you want to use to receive your notification.
 - **Automatically replace expiring self-signed certificates.** If you do not want to recreate the self-signed certificate, clear the check box.
 - **Delete expiring certificates and signers after replacement.** If you do not want to delete the expired certificates and signers, clear the check box.
 - **Enable checking.** If you do not want to have certificate monitoring enabled, clear the check box.
4. Enter the time of day when you want certificate monitoring to take place to schedule the running of the certificate expiration monitor.
 5. Select one of the following options:
 - **Check by calendar.** For **Weekday**, enter the day of week that you want to run the certificate expiration monitor. For **Repeat Interval**, specify the frequency to run the certificate monitor.
 - **Check by number of days.** Enter a number for how frequently the monitor runs, in number of days.
 6. Click **Apply**.

After completing the settings, a certificate expiration monitor object and a schedule are set up in the configuration. The certificate expiration monitor runs according to the configurations options that you configured.

You can generate reports that state which certificates have expired. The reports identify the notifications of certificate replacements and deletions. The report is sent according to the notification option that you specified.

Manage certificate expiration settings

Use this page to configure the certificate expiration monitor.

To view this administrative console page, click **Security > SSL certificate and key management > Manage certificate expiration**.

Attention: To see the changes to the Expiration checking fields, you must click **Apply**.

Expiration notification threshold

Specifies a threshold number of days during which the application warns specified individuals that a certificate is about to expire. For example, when the expiration monitor is run and the threshold is 30 days, if the current date is 30 days or less from the certificate expiration date, the certificate is flagged for notification. The application server can be configured to provide certification expiration notification through either e-mail or the message log file.

Data type:	Integer
Default:	30 days

Expiration check notification

Specifies the notification type (such as e-mail or System Out) when an expiration monitor runs.

Default:

Automatically replace expiring self-signed certificates

Specifies a new self-signed certificate be generated using the same certificate information if the expiration notification threshold is reached. The old certificate is replaced and uses the same alias. All old signers are managed by the key store configuration are also replaced. The system only replaces self-signed certificates.

Default: Enabled

Delete expiring certificates and signers after replacement

Specifies whether to completely remove old, self-signed certificates from the key store during a replace operation or leave them there under a renamed alias. If an old certificate is not deleted, the system renames the alias so that the new certificate can use the old alias, which might be referenced elsewhere in the configuration.

Default: Enabled

Enable checking

Specifies the certificate monitor is active and will run as scheduled.

Scheduled time of day to check for expired certificates

Specifies the scheduled time that the system checks for expired certificates.

You can type the scheduled time in hours and minutes, specify either A.M. or P.M., or 24-hour.

Data type Integer
Default: 0, 0
Range: 1–12, 0–59

Check by calendar

Indicates that you want to schedule a specific day of the week on which the expiration monitor runs. For example, it might run on Sunday.

Default: Disabled

Weekday

Specifies the day of the week on which the expiration monitor runs if **Check on a specific day** is selected.

Default: Sunday
Range: Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday

Repeat interval

Specifies the period of time between each schedule time to check for expired certificates or the interval between schedule checks.

Default: Daily
Range: Daily, Weekly

Check by number of days

Specifies that you want to schedule a specific number of days between each run of the expiration monitor. The day of the week on which this occurs is not counted. For example, if you set the interval to check for expired certificates every seven days, the expiration monitor runs on day eight.

Default: Disabled

Next start date

Specifies the date for the next scheduled check. This allows the deployment manager to be stopped and restarted without resetting the date.

Notifications

Use this page to specify the generic notification definitions that are used in certificate expiration monitors.

To view this administrative console page, complete the following steps:

1. Click **Security > SSL certificate and key management**.
2. Under Configuration settings, click **Manage certificate expiration**.
3. Under Related items, click **Notifications**.

Button	Resulting action
New	Adds a notification. The notification configures how the expiration monitor notifies the administrator of certificates that will expire within the specified threshold.
Delete	Deletes an existing notification.

Notification name

Specifies the notification name.

Message log

Specifies that this configuration intends to log certificate expiration information to the message log file.

List of e-mail addresses

Specifies the e-mail addresses that are sent notifications when certificates fall within the expiration threshold. You must specify the SMTP server for each e-mail address. If an e-mail address is not specified, by default the application server assumes that the SMTP server is "smtp-server." For example, if you type *name@domain*, the SMTP server will be smtp-server.domain.

Notifications settings

Use this page to set properties for new notifications used in certificate expiration monitors.

To view this administrative console page, complete the following steps:

1. Click **Security > SSL certificate and key management**.
2. Under Configuration settings, click **Manage certificate expiration**.
3. Under Related items, click **Notifications > New**.

Notification name

Specifies the name of the notification configuration.

Data type: Text

Message log

Specifies that this configuration will log certificate expiration information to a message log file.

Default: Disabled

Email sent to notification list

Specifies that this configuration intends to send certificate expiration information in an e-mail to the e-mail list.

Default: Disabled

Email address to add

Specifies the e-mail addresses that are sent notifications when certificates fall within the expiration threshold. You must specify the SMTP server for each e-mail address. If an e-mail address is not specified, by default the application server assumes that the SMTP server is "smtp-server." For example, if you type *name@domain*, the SMTP server will be smtp-server.domain.

Data type: Text (format as valid Internet mail address)

Add

Adds the e-mail address to the right-hand list.

Remove

Removes the e-mail address from the right-hand list.

Outgoing mail (SMTP) server

Specifies the SMTP server to be used with the e-mail address. If none is specified, the e-mail realm will be used.

Key management for cryptographic uses

WebSphere Application Server provides a framework for managing keys (secret keys or key pairs) that applications use to perform cryptographic operations on data. The key management framework provides an application programming interface (API) for retrieving these keys. Keys are managed in keystores so the keystore type can be supported by WebSphere Application Server, provided that the keystores can store the referenced key type. You can configure keys and scope keystores so that they are visible only to particular processes, nodes, clusters, and so on.

The key management infrastructure is based on two key configuration object types: key sets and key set groups. WebSphere Application Server uses a key set to manage instances of keys of the same type. You can configure a key set to generate a single key or a key pair, depending on the key or key pair generator class. A key set group manages one or more key sets and enables you to configure and generate different key types at the same time. For example, if your application needs both a secret key and key pair for cryptographic operations, you can configure two key sets, one for the key pair and one for the secret key that the key set group manages. The key set group controls the auto-generation characteristics of the keys, including the schedule. The framework can automatically generate keys on a scheduled basis, such as on a particular day of the week and time of day, so that key generation is done during off-peak hours.

Figure 1 shows an example of a key set group that is configured to manage two key sets: key set 1 and key set 2.

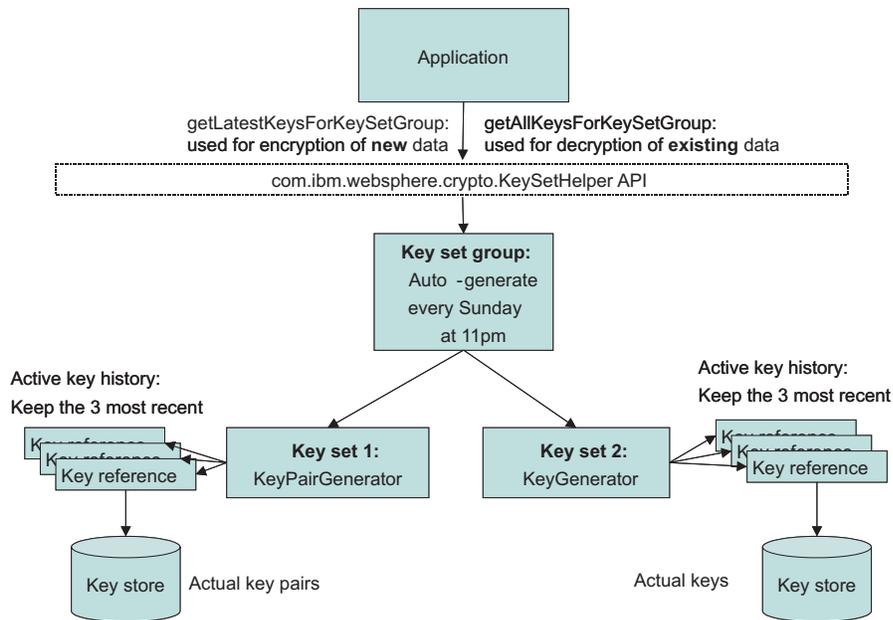


Figure 1: Key Management Infrastructure

Figure 7.

Key set 1 generates key pairs. Key set 2 generates secret keys. The application needs both types of keys for its cryptographic operations, signing and encryption, on data. The keys for each key set need to be generated in tandem. The application stores the key set group name with the encrypted data. The key set group generates a new set of keys every Sunday night at 11 P.M.. The application maintains key generation data for two weeks.

Creating a key set configuration

You can use key sets to manage multiple instances of cryptographic keys. WebSphere Application Server uses keys to encrypt or sign outbound data, and decrypt or verify inbound data during cryptographic operations.

You must have write-access to the keystore that will contain the keys after you generate them from a key set. However, if you want to generate keys outside of WebSphere Application Server, you can reference the keys from a read-only keystore that contains a secret key that you can access when you generate the keys. If you are creating a key pair using an X509Certificate and a PrivateKey object, see “Example: Developing a key or key pair generation class for automated key generation” on page 494.

Complete the following steps in the administrative console:

1. Decide whether you want to create the key set at the cell scope or below the cell scope at the node, server, or cluster, for example:
 - To create a key set at the cell scope, click **Security > SSL certificate and key management > Key sets**.

- To create a key set at a scope below the cell level, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > ssl_configuration > Key sets.**
2. Click **New** to create a new key set.
 3. Type a key set name. For example, CellmyKey.
 4. Type a key alias prefix name. For example, myKey. This field specifies the prefix for the key alias when the new key is generated and stored in the keystore. Following the prefix is the key reference version number, for example, 2, so that the full key alias name would be myKey_2. If the key reference already has a specified alias for a key that exists in the keystore, then WebSphere Application Server ignores this field.
 5. Type a key password. The key password protects the key in the keystore. This password is ignored by WebSphere Application Server if you already specified a password for the key alias reference. To check for a key reference password, click **Active key history** under Additional Properties. The key reference password protects keys that are generated by a key generator class.
 6. Type the password again to confirm it.
 7. **Optional:** Type the key generator class name. For example, com.ibm.ws.security.ltpa.LTPAKeyGenerator. The class name generates keys. If the class implements com.ibm.websphere.crypto.KeyGenerator, then a getKey method returns a java.security.Key object that is set in the keystore using the setKey method without a certificate chain. If the class implements com.ibm.websphere.crypto.KeyPairGenerator, then a getKeyPair method returns a com.ibm.websphere.crypto.KeyPair object that contains either a java.security.PublicKey and java.security.PrivateKey or a java.security.cert.Certificate and a java.security.PrivateKey object. The key generator class and the KeySetHelper API specify the details of the keys that are generated.
 8. **Optional:** Select **Delete key references that are beyond the maximum number of keys** if you do not want old keys saved in the keystore after WebSphere Application Server removes their references from the Active key history listing. The Active key history lists the keys that the KeySetHelper API is currently tracking. The number of keys in the list is equal to the number of keys that you specify in **Maximum number of keys referenced.**
 9. Type a numeric value for the maximum number of keys referenced. For example, if you type 3 and select **Delete key references that are beyond the maximum number of keys**, the fourth key version generation automatically triggers WebSphere Application Server to delete the first key version from the keystore. If you choose not to delete the old keys, they do not display in the Active key history list but instead remain in the keystore where you can remove them manually.
 10. Select a keystore from the drop-down list.
 - Select a JCEKS keystore if you are storing a secret key.
 - Select any keystore if you are storing a key pair with an X509Certificate and PrivateKey object.
 11. **Optional:** Select **Generates key pair** if your key generator class name implements the com.ibm.websphere.crypto.KeyPairGenerator interface instead of the com.ibm.websphere.crypto.KeyGenerator interface. This option designates that the key references a key pair instead of a single key. A key pair contains both a public key and a private key. The WebSphere Application Server run time determines whether or not key pairs are stored and loaded differently than single keys.
 12. **Optional:** Click **Apply** if you want to select **Active key history** under Additional Properties to add alias references or generate more keys.
 - a. Click **Active key history.**
 - b. Click **Add key alias reference** if you are not using the key generator class name to add key alias references to the keys that already exist in the keystore. Use this option to retrieve the keys from a read-only keystore without the key set generating them.
 - c. Type an alias reference.
 - d. Click **Generate key** if you want to generate a key using the class name that you defined in the key sets panel. Each new key increments numerically, for example, myAlias_2.

- e. Click **Apply**.
- 13. Click the key set name in the navigation path at the top of the panel.
- 14. Click **OK** and **Save**.

You have created a key set that you can manage using the **Active key history** link. You can generate keys manually to associate them with specified key sets.

After you generate new keys from a key set, you can access them programmatically using the `com.ibm.websphere.crypto.KeySetHelper` API. You must have Java 2 Security permissions, if enabled, to access keys in key sets. Specify the key set name within the fine-grained permissions, as in the following code sample: `WebSphereRuntimePermission "getKeySets.keySetName".` For more information, see “Example: Retrieving the generated keys from a key set group” on page 493. To generate multiple key types at the same time or to schedule the key generation on a specific schedule, see “Creating a key set group configuration” on page 492.

Active key history collection

Use this page to manage key alias references.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > SSL_configuration_name**. Under Related items, click **Key Sets > key set > Active key history**.

Button	Resulting action
Add key alias reference	Adds a reference to a key that already exists in a key store. If a key generation class is configured, the references are added automatically during generation and do not need to be added manually.
Delete	Deletes an existing key reference. This action does not delete the key in the keystore.
Generate key	Generates a key. The button is displayed only if a generator class name is specified for the key set, and the selected key store is editable.

Key alias reference

Specifies the name of the alias as it appears in the keystore.

Add key alias reference settings

Use this page to access key alias reference information.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > SSL_configuration_name**. Under Related items, click **Key Sets > key set > Active key history > Add key alias reference**.

Alias reference

Specifies the name of the alias as it appears in the key store.

Data type: Text

Password

Specifies the key password to get access to the key. This password is enforced by the keystore for that specific key. If the key does not have a password, this field can be left blank.

Data type: Text

Confirm password

Confirms the password entered in the previous field.

Data type: Text

Key sets collection

Use this page to manage key sets, which control a set of key instances of the same type for use in cryptographic operations. The keys can either be generated using a custom class or reference keys that already exist in a keystore.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > ssl_configuration** . Under Related items, click **Key sets**.

Button	Resulting action
New	Adds a new key set.
Delete	Deletes an existing key set. Make sure the key set is not referenced by a key set group before deleting it.

Key set name

Specifies the key set name that is used to select the key set from a key set group and from runtime application programming interfaces (API).

Key store

Specifies the key store that contains the keys for storage, retrieval, or both.

Key alias prefix name

Specifies the prefix for the key alias when a new key is generated and stored in a key store. The rest of the key alias comes from the key reference version number.

For example, if the alias prefix is mykey and the key reference version is 2, the keystore references the key using alias mykey_2. If the key reference already has a specified alias for a key already existing in the keystore, this field is ignored.

Key sets settings

Use this page to set the properties for a new key set.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > ssl_configuration** . Under Related items, click **Key sets > New**.

Key set name

Specifies the key set name that is used to select the key set from a key set group and from runtime application programming interfaces (API).

Data type: Text

Key alias prefix name

Specifies the prefix for the key alias when a new key is generated and stored in a keystore. The rest of the key alias comes from the key reference version number. For example, if the alias prefix is mykey and the key reference version is 2, the keystore references the key using alias mykey_2. If the key reference already has a specified alias for a key already existing in the keystore, this field is ignored.

Data type: Text

Key password

Specifies the password used to protect the key in the keystore. If a password is specified in the key reference as well, this password is ignored. This password is used for keys that get generated by a key generator class.

Data type: Text

Confirm password

Specifies the same password again to confirm it was entered correctly the first time.

Data type: Text

Key generator class name

Specifies the class name that generates keys. If the class implements `com.ibm.websphere.crypto.KeyGenerator`, then a `getKey()` method should return a `java.security.Key` object that is set in the key store using the `setKey` method without a certificate chain. The key store type associated with the key set must support storing keys without certificates, such as JCEKS.

Data type: Text

If the class implements `com.ibm.websphere.crypto.KeyPairGenerator`, then a `getKeyPair()` method should return a `com.ibm.websphere.crypto.KeyPair` object containing either a `java.security.PublicKey` and `java.security.PrivateKey`, or a `java.security.cert.Certificate[]` and a `java.security.PrivateKey`. The key generator class and the caller of the `KeySetHelper` API should know the details of the keys that are generated. This framework does not need to understand the key algorithms and lengths.

Delete key references that are beyond the maximum number of keys:

Specifies that the keys are deleted from the keystore at the same time that the key reference is deleted. The server deletes the older key references as the Maximum number of keys referenced value is exceeded.

Maximum number of keys referenced

Specifies the maximum number of key instances that are returned when keys from this key set are requested. The oldest key reference gets removed whenever a new key reference gets generated after the maximum has been reached.

Data type: Integer

Default: 3

Key store

Specifies the key store that contains the keys for storage, retrieval, or both.

Data type: Text

Generates key pair

Specifies that a key references a key pair instead of a key. The key pair contains both a public key and a private key.

Creating a key set group configuration

A key set group manages one or more key sets. WebSphere Application Server uses key set groups to automatically generate cryptographic keys or multiple synchronized key sets.

Complete the following steps in the administrative console:

1. Decide whether you want to create the key set group at the cell scope or below the cell scope at the node, server, or cluster, for example.
 - To create a key set group at the cell scope, click **Security > SSL certificate and key management > Key set groups**.
 - To create a key set group at a scope below the cell level, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > SSL_configuration > Key set groups**.
2. You can choose to generate a key for an existing key set group, delete an existing key set group, or create a new key set group.
 - To generate a key for an existing key set group, select a key set group from the list of existing key set groups, and click **Generate keys**. You have generated a new key for each key set in the selected group.
 - To delete an existing key set group, select a key set group from the list of existing key set groups, and click **Delete**. You have deleted the key set group.
 - To create a new key set group, go to step 3.

CAUTION:

Do not delete the cell or node LTPAKeySetGroup, which is used by the Lightweight Third Party Authentication (LPTA) mechanism.

3. Click **New** to create a new key set group.
4. Type a key set group name. You can reference this name by using the `com.ibm.websphere.crypto.KeySetHelper` API to retrieve the managed keys from an application.
5. Select one or more key sets from the Key sets list.

Note: If the key set(s) you want is not listed, make sure that it was created at the same scope or a higher scope than where you are creating the new key set group.

6. Click **Add** to add the selected key set(s) to the new key set group.
7. Select **Automatically generate keys** to generate the new keys on a schedule. If you decide to generate keys automatically, then you must specify a scheduled time of day.
8. Specify the scheduled time to generate keys automatically in hours and minutes, A.M. or P.M., or every 24 hours.
9. You can choose to generate new keys on a specific day or at an interval.
 - Select **Generate on a specific day**. Select a day of the week from the drop-down list, and type a repeat interval number for the number of days between each key generation. This choice enables you to schedule key generation when your systems are least busy.
 - Select **Generate at an interval**. Type a repeat interval number for the number of days between each key generation. This choice enables you to schedule key generation more frequently than once a week.

Note: The **Next start date** is a read-only field that specifies the date for the next scheduled generation. You can stop and restart the deployment manager or base application server without resetting this date. If you do not see the next start date appear after changing the configuration, click **OK** to save it, then check that the next start date displays.

10. Click **Save**.

You have created a new key set group to manage key sets and key generation on a schedule.

After you generate new keys from a key set, you can access them programmatically using the `com.ibm.websphere.crypto.KeySetHelper` API. You must have Java 2 Security permissions, if enabled, to access keys in key sets. Specify the key set name within the fine-grained permissions, as in the following code sample: `WebSphereRuntimePermission "getKeySets.keySetName".` For more information, see “Example: Retrieving the generated keys from a key set group.”

Example: Retrieving the generated keys from a key set group

This example shows how applications can use the `com.ibm.websphere.crypto.KeySetHelper` API to retrieve managed keys from the `KeySet` or `KeySetGroup` configurations. Use the `com.ibm.websphere.crypto.KeySetHelper` API to get either the latest set of keys or the all keys in the `KeySet` or `KeySetGroup` object.

Use the latest keys when performing any new cryptographic operations. All of the other keys that are defined in the `KeySet` or `KeySetGroup` object are for the validation of previously performed cryptographic operations.

The following example uses a method that an application might use to initialize the keys in the associated `KeySetGroup` object. The application might want to store the keys in two separate maps, one for generation and one for validation. Refer to the API documentation for `KeySetHelper` API to determine which Java 2 Security requirements are required.

```
/**
 * Initializes the primary and secondary Maps used for initializing the keys.
 */

public void initializeKeySetGroupKeys() throws com.ibm.websphere.crypto.KeyException
{
    java.util.Map generationKeys = null;
    java.util.Map validationKeys = null;

    PublicKey tempPublicKey = null;
    PrivateKey tempPrivateKey = null;
    byte[] tempSharedKey = null;

    keySetGroupName = "ApplicationKeySetGroup";
    com.ibm.websphere.crypto.KeySetHelper ksh = com.ibm.websphere.crypto.KeySetHelper.getInstance();
    generationKeys = ksh.getLatestKeysForKeySetGroup(keySetGroupName);

    /**
     * Latest keys: {
     *   KeyPair_3=com.ibm.websphere.crypto.KeyPair@64ec64ec,
     *   Secret_3=javax.crypto.spec.SecretKeySpec@fffe8aa7
     * }
     */

    if (generationKeys != null)
    {
        Iterator iKeySet = generationKeys.keySet().iterator();

        while (iKeySet.hasNext())
        {
            String keyAlias = (String)iKeySet.next();

            Object key = generationKeys.get(keyAlias);

            if (key instanceof java.security.Key)
            {
                tempSharedKey = ((java.security.Key)key).getEncoded();
            }
            else if (key instanceof com.ibm.websphere.crypto.KeyPair)
            {
                java.security.Key publicKeyAsSecret =
                ((com.ibm.websphere.crypto.KeyPair)key).getPublicKey();
            }
        }
    }
}
```

```

        tempPublicKey = new PublicKey(publicKeyAsSecret.getEncoded());
        java.security.Key privateKeyAsSecret =
((com.ibm.websphere.crypto.KeyPair)key).getPrivateKey();
        tempPrivateKey = new PrivateKey(privateKeyAsSecret.getEncoded());
    }
}

// save these for use later, if necessary
validationKeys = ksh.getAllKeysForKeySetGroup(keySetGroupName);

/**
 * All keys: {
 * version_1=
 *   {Secret_1=javax.crypto.spec.SecretKeySpec@178cf,
 *    KeyPair_1=com.ibm.websphere.crypto.KeyPair@1c121c12},
 * version_2=
 *   {Secret_2=javax.crypto.spec.SecretKeySpec@17a77,
 *    KeyPair_2=com.ibm.websphere.crypto.KeyPair@182e182e},
 * version_3=
 *   {Secret_3=javax.crypto.spec.SecretKeySpec@fffe8aa7,
 *    KeyPair_3=com.ibm.websphere.crypto.KeyPair@4da04da0}
 * }
 */
}
else
{
    throw new com.ibm.websphere.crypto.KeyException("Could not generateKeys.");
}
}
}

```

Example: Developing a key or key pair generation class for automated key generation

A class that generates keys for cryptographic operations can be created automatically. With this capability, the key management infrastructure can maintain a list of keys for a predefined key set, and applications can access these keys.

You can schedule new key generation at predefined frequencies. Remember that key generation frequency affects the security of your data. For example, for persistent data, you might schedule key generation less frequently than for real time communications, which require that the keys be generated more often as old keys expire.

When you develop a key generation class, decide if you are creating a shared key or a key pair because this decision determines the interface you must use.

If you are developing shared keys, refer to the following example, which uses the KeyGenerator class to implement the com.ibm.websphere.crypto.KeyGenerator interface. The interface returns a java.security.Key key, which is stored as a SecretKey in a JCEKS keystore type. You can use any other keystore type that supports storing secret keys.

```

package com.ibm.test;

import java.util.*;
import com.ibm.ws.ssl.core.*;
import com.ibm.ws.ssl.config.*;
import com.ibm.websphere.crypto.KeyException;

public class KeyGenerator implements com.ibm.websphere.crypto.KeyGenerator
{
    private java.util.Properties customProperties = null;
    private java.security.Key secretKey = null;

    public KeyGenerator()
    {
    }
}

```

```

/**
 * <p>
 * This method is called to pass any custom properties configured with
 * the KeySet to the implementation of this interface.
 * </p>
 *
 * @param java.util.Properties
 */
public void init (java.util.Properties customProps)
{
    customProperties = customProps;
}

/**
 * <p>
 * This method is called whenever a key needs to be generated either
 * from the schedule or manually requested. The key is stored in the
 * KeyStore referenced by the configured KeySet that contains the
 * keyGenerationClass implementing this interface. The implementation of
 * this interface manages the key type. The user of the KeySet
 * must know the type that is returned by this keyGenerationClass.
 * </p>
 *
 * @return java.security.Key
 * @throws com.ibm.websphere.crypto.KeyException
 */
public java.security.Key generateKey () throws KeyException
{
    try
    {
        // Assume generate3DESKey is there to create the key.
        byte[] tripleDESKey = generate3DESKey();
        secretKey = new javax.crypto.spec.SecretKeySpec(tripleDESKey, 0, 24, "3DES");

        if (secretKey != null)
        {
            return secretKey;
        }
        else
        {
            throw new com.ibm.websphere.crypto.KeyException ("Key could not be generated.");
        }
    }
    catch (Exception e)
    {
        e.printStackTrace(); // handle exception
    }
}
}

```

If you are developing a key pair, refer to the following example, which uses the KeyPairGenerator class to implement the com.ibm.websphere.crypto.KeyPairGenerator interface.

```

package com.ibm.test;

import java.util.*;
import javax.crypto.spec.SecretKeySpec;
import com.ibm.websphere.crypto.KeyException;

/**
 * <p>
 * This implementation defines the method to generate a java.security.KeyPair.
 * When a keyGeneration class implements this method, the generateKeyPair method
 * is called and a KeyPair is stored in the keystore. The isKeyPair
 * attribute is ignored since the KeyGenerationClass is an
 * implementation of KeyPairGenerator. The isKeyPair attributes is for when
 * the keys already exist in a KeyStore, and are just read (not
 * generating them).
 * </p>
 * @author IBM Corporation

```

```

* @version WebSphere Application Server 6.1
* @since WebSphere Application Server 6.1
**/
public class KeyPairGenerator implements com.ibm.websphere.crypto.KeyPairGenerator
{
    private java.util.Properties customProperties = null;

    public KeyPairGenerator()
    {
    }

    /**
     * <p>
     * This method is called to pass any custom properties configured with
     * the KeySet to the implementation of this interface.
     * </p>
     *
     * @param java.util.Properties
     **/
    public void init (java.util.Properties customProps)
    {
        customProperties = customProps;
    }

    /**
     * <p>
     * This method is called whenever a key needs to be generated either
     * from the schedule or manually requested and isKeyPair=true in the KeySet
     * configuration. The key is stored in the KeyStore referenced by
     * the configured KeySet which contains the keyGenerationClass implementing
     * this interface. The implementation of this interface manages the
     * type of the key. The user of the KeySet must know the type that
     * is returned by this keyGenerationClass.
     * </p>
     *
     * @return com.ibm.websphere.crypto.KeyPair
     * @throws com.ibm.websphere.crypto.KeyException
     **/
    public com.ibm.websphere.crypto.KeyPair generateKeyPair () throws KeyException
    {
        try
        {
            java.security.KeyPair keyPair = generateKeyPair();

            // Store as SecretKeySpec
            if (keyPair != null)
            {
                java.security.PrivateKey privKey = keyPair.getPrivate();
                java.security.PublicKey pubKey = keyPair.getPublic();

                SecretKeySpec publicKeyAsSecretKey = new SecretKeySpec
                    (pubKey.getEncoded(), "RSA_PUBLIC");
                SecretKeySpec privateKeyAsSecretKey = new SecretKeySpec
                    (privKey.getEncoded(), "RSA_PRIVATE");

                com.ibm.websphere.crypto.KeyPair pair = new
com.ibm.websphere.crypto.KeyPair(publicKeyAsSecretKey, privateKeyAsSecretKey);
                return pair;
            }
            else
            {
                throw new com.ibm.websphere.crypto.KeyException ("Key pair could
not be generated.");
            }
        }
        catch (Exception e)
        {
            e.printStackTrace(); // handle exception
        }
    }
}

```

This interface returns a `com.ibm.websphere.crypto.KeyPair` key pair, which can contain either a `X509Certificate` and `PrivateKey` object or `PublicKey` and `PrivateKey` objects. If the `com.ibm.websphere.crypto.KeyPair` interface contains a `X509Certificate` and `PrivateKey` object, the certificate and private key are stored in the keystore. Consequently, they can use any `KeyStore` type.

If the `com.ibm.websphere.crypto.KeyPair` interface contains `PublicKey` and `PrivateKey` objects, you must convert the encoded values to the `SecretKeySpec` object in order to store them. The WebSphere Application Server runtime stores and retrieves the key pair as secret keys. The runtime converts the key pair back to `PublicKey` and `PrivateKey` objects when the server retrieves the pair during the handshake.

Use the following constructors to develop the `com.ibm.websphere.crypto.KeyPair` interface:

- Public and private constructor

```
public KeyPair(java.security.Key publicKey, java.security.Key privateKey)
```
- Certificate and private constructor.

```
public KeyPair(java.security.cert.Certificate[] certChain,
java.security.Key privateKey)
```

The previous example code shows the `KeyPairGenerator` class using the public and private constructor. Each call to this class generates a new and unique key pair, and this class is invoked by a `KeySet` to create a new key pair when `isKeyPair=true`. The version number in the key set increments each time it is called.

Key set groups collection

Use this page to manage groups of public, private, and shared keys. These key groups enable the application server to control multiple sets of Lightweight Third Party Authentication (LTPA) keys.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > ssl_configuration** . Under Related items, click **Key set groups**.

Button	Resulting action
New	Adds a key set group. A key set group combines one or more key sets together as a single key set group. It allows the generation of multiple different types of keys to occur at the same time. A single key set represents one type of key, so a key set group allows you to group the different types.
Delete	Deletes an existing key set group. You must be sure that there are no other references to this key set group before you delete it.
Generate keys	Generates keys for key set group. The system generates keys for each key set within the key set group so that the keys remain synchronized with each other in terms of version. You must configure a valid key generation class and a key store that is writable. See the <code>com.ibm.websphere.crypto.KeySetHelper</code> application programming interfaces (APIs) to enable the use of keys that are managed by a <code>KeySetGroup</code> or <code>KeySet</code> .

Key set group name

Specifies the name of the key set group used to reference it.

Automatically generate keys

Specifies that the keys are to be generated automatically on a schedule.

Key set groups settings

Use this page to create new key set groups.

To view this administrative console page, click **Security > SSL certificate and key management > Manage endpoint security configurations > {Inbound | Outbound} > ssl_configuration** . Under Related items, click **Key set groups > New**.

Key set group name

Specifies the name of key set group used. This name can be referenced using the `com.ibm.websphere.crypto.KeySetHelper` API to retrieve the managed keys from an application.

Data type: Text

Key sets

Specifies a set of key instances of the same type for use in cryptographic operations.

Add

Specifies to add the selected key set part of this key set group.

Remove

Specifies to remove the selection from the **Key sets** list.

Automatically generate keys

Specifies that the keys are generated automatically on a schedule. When a new key is generated, the `security.xml` is updated and saved by the runtime to track the key reference version. This can cause save conflicts when updating the same file from admin applications.

Default: Disabled

Scheduled time for generation

Specifies the scheduled time when the system generates selected key set group or groups. You can specify the scheduled time in hours and minutes; specify either A.M. or P.M., or specify 24-hour. You can also specify the day of the week you want the scheduled event to occur. It is recommended that you set this event to occur during a low peak time, especially for keys that are used by runtime for token validation.

Data type: Integer
Default: 0, 0
Range: 1–12, 0–59

Generate on a specific day

Specifies whether to have the generation occur on a specific day of the week. It is best to auto-generate keys during a low peak day.

Default: Enabled

Weekday

Specifies the day of the week on which the expiration monitor will run if the Check on a specific day option is selected.

Default: Sunday
Range: Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday

Repeat interval

Specifies the period of time between each schedule time to check for expired certificates or the interval between schedule checks.

Default: Daily
Range: Daily, Weekly

Generate at an interval

Specifies to generate keys at the specified frequency regardless of the day of the week on which generation occurs.

Default: Disabled

Next start date

Specifies the date for the next scheduled check. This allows the deployment manager to be stopped and restarted without resetting the date.

Chapter 8. Developing extensions to the WebSphere security infrastructure

WebSphere Application Server provides various plug points so that you can extend the security infrastructure.

The following topics are covered in this section:

- Developing custom user registries
- Developing applications that use programmatic security
- Customizing Web application login forms
- Customizing application login forms with Java Authentication and Authorization Service (JAAS)
- Securing transports with Java Secure Sockets Extension (JSSE) and Java Cryptography Extension (JCE) programming interfaces
- Implementing tokens for security attribute propagation

Developing standalone custom registries

This development provides considerable flexibility in adapting WebSphere Application Server security to various environments where some notion of a user registry, other than LDAP or Local OS, already exists in the operational environment.

WebSphere Application Server security supports the use of standalone custom registries in addition to the local operating system registry, standalone Lightweight Directory Access Protocol (LDAP) registries, and federated repositories for authentication and authorization purposes. A standalone custom-implemented registry uses the `UserRegistry` Java interface as provided by WebSphere Application Server. A standalone custom-implemented registry can support virtually any type or notion of an accounts repository from a relational database, flat file, and so on.

Implementing a standalone custom registry is a software development effort. Use the methods that are defined in the `UserRegistry` interface to make calls to the appropriate registry to obtain user and group information. The interface defines a general set of methods for encapsulating a wide variety of registries. You can configure a standalone custom registry as the selected repository when configuring WebSphere Application Server security on the Secure administration, applications, and infrastructure panel.

In WebSphere Application Server Version 6.1, make sure that your implementation of the standalone custom registry does not depend on any WebSphere Application Server components such as data sources, Enterprise JavaBeans (EJB) and Java Naming and Directory Interface (JNDI). You can 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. For example, if your previous implementation used data sources to connect to a database, use `DriverManager` to connect to the database.

Refer to the “Migrating custom user registries” on page 37 for more information on migrating. If your previous implementation uses data sources to connect to a database, change the implementation to use Java database connectivity (JDBC) connections. However, it is recommended that you use the new interface to implement your custom registry.

1. Implement all the methods in the interface except for the `CreateCredential` method, which is implemented by WebSphere Application Server. `FileRegistrySample.java` file is provided for reference.

Attention: The sample provided is intended to familiarize you with this feature. Do not use this sample in an actual production environment.

2. Build your implementation.

To compile your code, you need the `app_server_install_rootBase/plugins/com.ibm.ws.runtime_6.1.0.jar` and the `app_server_install_rootBase/plugins/com.ibm.ws.security.crypto_6.1.0/cryptosf.jar` files in your class path. For example:

```
%install_root%/java/bin/javac -classpath
%install_root%app_server_install_rootBase/plugins/com.ibm.ws.runtime_6.1.0.jar;
%install_root%app_server_install_rootBase/plugins/com.ibm.ws.security.crypto_6.1.0
/cryptosf.jar your_implementation_file.java
```

3. Copy the class files that are generated in the previous step to the product class path. The preferred location is the `%install_root%/lib/ext` directory. Copy these class files to all of the product process class paths.
4. Follow the steps in “Configuring standalone custom registries” on page 111 to configure your implementation using the administrative console. This step is required to implement custom user registries.

If you enable security, make sure that you complete the remaining steps:

1. Save and synchronize the configuration and restart all of the servers.
2. Try accessing some J2EE resources to verify that the custom registry implementation is correct.

Example: Standalone custom registries

Use these links to view registry examples.

A *standalone custom registry* is a customer-implemented registry that implements the UserRegistry Java interface, as provided by WebSphere Application Server. A custom-implemented registry can support virtually any type or form of an accounts repository from a relational database, flat file, and so on. The custom registry provides considerable flexibility in adapting WebSphere Application Server security to various environments where some form of a registry, other than a federated repository, Lightweight Directory Access Protocol (LDAP) registry, or local operating system registry, already exist in the operational environment.

To view a sample standalone custom registry, refer to the following files:

- FileRegistrySample.java file
- users.props file
- groups.props file

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 more users and groups in the user registry exist than requested.

```
//
// 5639-D57, 5630-A36, 5630-A37, 5724-D18
// (C) COPYRIGHT International Business Machines Corp. 1997, 2005
// 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;
}

```

UserRegistry.java files

The following file is a custom property that is used with a custom user registry.

For more information, see “Configuring standalone custom registries” on page 111.

```

// 5639-D57, 5630-A36, 5630-A37, 5724-D18
// (C) COPYRIGHT International Business Machines Corp. 1997, 2005
// 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)

```

```

* getUniqueId(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 the 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.
*
* 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 a 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 registry.
* 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 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 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 where you are working with a user registry and it is not
* practical to get all of the users from any of your groups (for example if
* a large number of users exist) you can create the NotImplementedException
* for those particular groups. Make sure that if the WebSphere Business
* Integration Server Foundation Process Choreographer is installed (or
* if installed later) that the users 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 NotImplementedException.
*
* @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 create 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;
}

```

Implementing custom password encryption

WebSphere Application Server supports the use of custom password encryption.

An installation can implement any password encryption algorithm it chooses.

Complete the following steps to implement custom password encryption:

1. Build your custom password encryption class. An example of a custom password encryption class follows.

```

// CustomPasswordEncryption
// Encryption and decryption functions
public interface CustomPasswordEncryption {
    public EncryptedInfo encrypt(byte[] clearText) throws PasswordEncryptException;
    public byte[] decrypt(EncryptedInfo cipherTextInfo) throws PasswordEncryptException;
    public void initialize(HashMap initParameters);
};
// Encapsulation of cipher text and label
public class EncryptedInfo {
    public EncryptedInfo(byte[] bytes, String keyAlias);
    public byte[] getEncryptedBytes();
    public String getKeyAlias();
};

```

2. Enable custom password encryption.
 - a. Set the custom property **com.ibm.wsspi.security.crypto.customPasswordEncryptionClass** to the name of the class that is to be given control.
 - b. Enable the function. Set the custom property, **com.ibm.wsspi.security.crypto.customPasswordEncryptionEnabled** to true.

Custom password encryption at the installation is complete.

Developing applications that use programmatic security

For some applications, declarative security is not sufficient to express the security model of the application. Use this topic to develop applications that use programmatic security.

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 that are 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 that is 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 runtime. 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. For more information, see “Developing with programmatic security APIs for Web applications” on page 533.
2. Develop servlet filters for form login processing. For more information, see “Developing servlet filters for form login processing” on page 544.
3. Develop form login pages. For more information, see “Customizing Web application login” on page 541.
4. Develop enterprise bean component applications. For more information, see “Developing with programmatic APIs for EJB applications” on page 537.
5. 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 548.
6. Develop your own Java 2 security mapping module. For more information, see “Configuring programmatic logins for Java Authentication and Authorization Service” on page 552.
7. Develop custom user registries. For more information, see “Developing standalone custom registries” on page 501.
8. Develop a custom interceptor for trust associations.

Protecting system resources and APIs (Java 2 security)

Java 2 security is a programming model that is very pervasive and has a huge impact on application development.

Java 2 security is orthogonal to Java 2 Platform, Enterprise Edition (J2EE) role-based security; you can disable or enable it independently of administrative 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 application programming interfaces (API). Administrators 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. Developers have to know whether or not the APIs that are 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 topic to visit this Web site.
2. Make sure that migrated applications from previous releases are given the required permissions. Because 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. No easy way to find out all the required permissions for the application is available. The following are activities you can perform to determine the extra permissions that are 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 preproduction 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 debugging. Do not use this property in a production environment..

Refer to “Java 2 security” on page 64

The default permission set for applications is the recommended permission set that is defined in the J2EE 1.3 Specification. The default is declared in the `app_server_root/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 that are declared in the `profiles/profile_name/config/cells/cell_name/filter.policy` file. Permissions that are declared in the `filter.policy` file are filtered for applications during the permission check.

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 514 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 > Secure administration, applications, and infrastructure**. The Secure administration, applications, and infrastructure panel is displayed.
2. Select the **Use Java 2 security to restrict application access to local resources** 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. 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. Prevent application code from calling destructive APIs, for example, calling the `System.exit` method brings down the application server.
3. Prevent application code from obtaining privileged information (passwords) or gaining extra privileges (obtaining server credentials).

The Java 2 security manager is enhanced to dump the Java 2 security permissions that are granted to all classes on the call stack when an application is denied access to a resource. The `java.security.AccessControlException` exception is created. However, this tracing capability is disabled by default. You can enable this capability by specifying the server trace service with the `com.ibm.ws.security.core.SecurityManager=all=enabled` trace specification. When the exception is created, the trace dump provides hints to determine whether the application is missing permissions or the product runtime code or the third-party libraries that are used are not properly marked as `privileged` when accessing Java 2 security-protected resources. See the Security Problem Determination Guide for details.

Using PolicyTool to edit policy files

Use the **PolicyTool** utility to update policy files.

Java 2 security uses several policy files to determine the granted permission for each Java program. The Java Development Kit provides the **PolicyTool** tool 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` exception when the application runs, 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.

See “Java 2 security policy files” on page 68 for the list of available policy files.

1. Start the **PolicyTool**.

Windows For example, you can enter the following command at a Windows command prompt:

```
%{was.install.root}/java/jre/bin/policytool
```

The **PolicyTool** window opens. The tool 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, you can enter:

```
app_server_root/InstalledApps/testcase.ear
```


where the `app_server_root` variable represents your installation location.
5. Modify or add the permission specification.
 - a. Modify the permission specification by double-clicking the entry that 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 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.security.auth.policy` Java virtual machine (JVM) system property.

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 the previous steps 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** utility. 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.

Configuring Java 2 security policy files

Use can configure Java 2 security policy files so that the required permission is granted for the specified WebSphere Application Server enterprise application.

Java 2 security uses several policy files to determine the permissions for each Java programs.

See the “Java 2 security policy files” on page 68 topic for the list of available policy files that are supported by WebSphere Application Server.

Two types of policy files are 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. Six dynamic policy files are provided:

Policy file name	Description
app.policy	Contains default permissions for all of the enterprise applications in the cell.
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 that are 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 that are 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 creates a java.security.AccessControlException exception. 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.security.auth.policy Java virtual machine (JVM) system property. 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” on page 527.
- If the permission is required by all of the WebSphere Application Server enterprise applications in the node, refer to “spi.policy file permissions” on page 522.
- 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 the was.policy file” on page 520 and “Adding the was.policy file to applications” on page 525.
- If the permission is required by shared libraries, refer to “library.policy file permissions” on page 523.
- If the permission is required by SPI libraries, refer to “spi.policy file permissions” on page 522.

Tip: 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, rather than static policy files.

Add any permission that you never want granted to the WebSphere Application Server enterprise application in the cell to the filter.policy file. Refer to “filter.policy file permissions” on page 518.

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 `java.security.AccessControlException` exception. 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)
```

The previous two lines were split onto two lines because of the width of the page. Enter the permission 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.

```
grant codeBase "file:user_client_installed_location" {
permission java.io.FilePermission
"C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar", "read";
};
```

The previous permission information lines were split onto more than one line because of their length. Enter the permission on one line.

To decide whether to add a permission, refer to the “Access control exception” on page 73 topic.

app.policy file permissions:

Java 2 security uses several policy files to determine the granted permissions for each Java program.

For the list of available policy files that are supported by WebSphere Application Server, see the “Java 2 security policy files” on page 68 article The app.policy file is a default policy file that is shared by all of the WebSphere Application Server enterprise applications. The union of the permissions that are 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 creates a `java.security.AccessControlException` exception. 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.security.auth.policy` Java virtual machine (JVM) system property. You can statically set the authorization policy files in the `java.security.auth.policy` property 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 that is 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.

To extract the policy file, use a command prompt to enter the following command on one line using the appropriate variable values for your environment:

```
wsadmin> set obj [$AdminConfig extract profiles/profile_name/cells/cell_name/node/
node_name/app.policy c:/temp/test/app.policy]
```

Edit the extracted `app.policy` file with the Policy Tool. For more information, see “Using PolicyTool to edit policy files” on page 512. Changes to the `app.policy` file are local for the node.

To check in the policy file, use a command prompt to enter the following command on one line using the appropriate variable values for your environment:

```
wsadmin> $AdminConfig checkin profiles/profile_name/cells/cell_name/nodes/
node_name/app.policy c:/temp/test/was.policy $obj
```

Several product-reserved symbols are defined to associate the permission lists to a specific type of resource.

Symbol	Meaning
<code>file:\${application}</code>	Permissions apply to all resources within the application
<code>file:\${jars}</code>	Permissions apply to all utility Java archive (JAR) files within the application
<code>file:\${ejbComponent}</code>	Permissions apply to enterprise bean resources within the application

Symbol	Meaning
file:\${webComponent}	Permissions apply to Web resources within the application
file:\${connectorComponent}	Permissions apply to connector resources both within the application and within standalone connector resources.

Five embedded symbols are provided to specify the path and name for the java.io.FilePermission permission. 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

Tip: You cannot use the `${was.module.path}` in the `${application}` entry.

The `app.policy` file that is supplied by WebSphere Application Server resides at `app_server_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 that are related to java.io.FilePermission permission are split into two lines for illustrative purposes only.

```
grant codeBase "file:${application}" {
    // The following are required by Java mail
    permission java.io.FilePermission "${was.install.root}${lib{mail-impl.jar}", "read";
    permission java.io.FilePermission "${was.install.root}${lib{activation-impl.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 `java.security.AccessControlException`

exception. 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 `AccessControlException` topic.

Restart all WebSphere Application Server enterprise applications to ensure that the updated `app.policy` file takes effect.

filter.policy file permissions:

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.

Before modifying the `filter.policy` file, you must start the `wsadmin` tool. See the `Starting the wsadmin scripting client` article for more information.

Refer to “Protecting system resources and APIs (Java 2 security)” on page 511. The filtering policy defined in the `filter.policy` file is cell wide. The `filter.policy` file is the only policy file that is used when restricting the permission instead of granting permission. The permissions that are listed in the filter policy file are filtered out from the `app.policy` file and the `was.policy` file. Permissions that are 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 that are defined in the `app.policy` file and the `was.policy` file are compound permissions like the `java.security.AllPermission` permission, 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`), a warning message is logged and the permission is not removed. You can change the value of the **Warn if applications are granted custom permissions** option on the `Secure administration, applications, and infrastructure` panel. It is not recommended that you use the `AllPermission` permission for the enterprise application.

Some default permissions that are defined in the `filter.policy` file. These permissions are the minimal ones that are 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.

You cannot use the `Policy Tool` to edit the `filter.policy` file. Editing must be completed in a text editor. Be careful and verify that no syntax errors exist in the `filter.policy` file. If any syntax errors exist in the `filter.policy` file, the file is not loaded by the product security runtime, which implies that filtering is disabled.

To extract the `filter.policy` file, enter the following command using information from your environment:

```
set obj [$AdminConfig extract cells/cell_name/filter.policy c:/temp/test/filter.policy]
```

To check in the policy file, enter the following command using information from your environment:

```
$AdminConfig checkin cells/cell_name/filter.policy c:/temp/test/filter.policy $obj
```

An updated `filter.policy` file is applied to all of the WebSphere Application Server enterprise applications after the servers are restarted. The `filter.policy` file is managed by configuration and file replication services.

The `filter.policy` file that is supplied by WebSphere Application Server resides at: `app_server_root/profiles/profile_name/config/cells/cell_name/filter.policy`.

This fill 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 that are defined in `filterMask` filter are for static policy filtering. The security runtime 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 that are defined in the `filterMask` filter, and if scripting is used. The `runtimeFilterMask` filter defines permissions that are used by the security runtime to deny access to those permissions to application thread. Do not add more permissions to the `runtimeFilterMask` filter. Application start failure or incorrect functioning might result. Be careful when adding more permissions to the `runtimeFilterMask` filter. 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 can compromise the integrity of the system. For instance, WebSphere Application Server considers the `exitVM` and `setSecurityManager` permissions as those permissions that most applications never have. If these permissions are granted, the following scenarios are possible:

exitVM

A servlet, JavaServer Pages (JSP) file, enterprise bean, or other library that is used by the aforementioned might call the `System.exit` API and cause the entire WebSphere Application Server process to terminate.

setSecurityManager

An application might install its own security manager and either grant more permissions or bypass the default policy that the WebSphere Application Server security manager enforces.

Important: In application code, do not use the `setSecurityManager` permission to set a security manager. When an application uses the `setSecurityManager` permission, a conflict exists with the internal security manager within WebSphere Application Server. If you must set a security manager in an application for Remote Method Invocation (RMI) purposes, you also must enable the **Enforce Java 2 Security** option on the Global security settings page within the WebSphere Application Server administrative console. WebSphere Application Server then registers a security manager, which the application code can verify is registered by using the `System.getSecurityManager` application programming interface (API).

Important: In application code, do not use the `setSecurityManager` permission to set a security manager. When an application uses the `setSecurityManager` permission, a conflict exists with the internal security manager within WebSphere Application Server. If you must set a security manager in an application for Remote Method Invocation (RMI) purposes, you also must enable the **Use Java 2 security to restrict application access to local resources** option on

the Secure administration, applications, and infrastructure panel within the WebSphere Application Server administrative console. WebSphere Application Server then registers a security manager, which the application code can verify is registered by using the `System.getSecurityManager` application programming interface (API).

For the updated `filter.policy` file to take effect, restart related Java processes.

Configuring the `was.policy` file:

You should update the `was.policy` file if the application has specific resources to access.

Java 2 security uses several policy files to determine the granted permission for each Java program. The `was.policy` file is an application-specific policy file for WebSphere Application Server enterprise applications. This file is embedded in the `META-INF/was.policy` enterprise archive (.EAR) file. The `was.policy` file is located in:

```
profile_root/config/cells/cell_name/applications/  
ear_file_name/deployments/application_name/META-INF/was.policy
```

See “Java 2 security policy files” on page 68 for the list of available policy files that are supported by WebSphere Application Server Version 6.1.

The union of the permissions that are 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.

Several product-reserved symbols are defined to associate the permission lists to a specific type of resources.

Symbol	Definition
<code>file:\${application}</code>	<code>file:\${application}</code>
<code>file:\${jars}</code>	Permissions apply to all utility Java archive (JAR) files within the application
<code>file:\${ejbComponent}</code>	Permissions apply to enterprise bean resources within the application
<code>file:\${webComponent}</code>	Permissions apply to Web resources within the application
<code>file:\${connectorComponent}</code>	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 creates a `java.security.AccessControlException` exception. 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 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 `java.policy`, `server.policy`, and `client.policy` policy file. The JAAS principal keyword is supported in a JAAS policy file when it is specified by the `java.security.auth.policy` Java virtual machine (JVM) system property. You can statically set the authorization policy files in the `java.security.auth.policy` file with the `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";
};
```

Five embedded symbols are provided to specify the path and name for the `java.io.FilePermission` permission. These symbols enable flexible permission specification. The absolute file path is fixed after the application is installed.

Symbol	Definition
<code>\${app.installed.path}</code>	Path where the application is installed
<code>\${was.module.path}</code>	Path where the module is installed
<code>\${current.cell.name}</code>	Current cell name
<code>\${current.node.name}</code>	Current node name
<code>\${current.server.name}</code>	Current server name

If the default permissions for the enterprise application are enough, an action is not required. The default permissions are a union of the permissions that are defined in the `java.policy` file, the `server.policy` file, and the `app.policy` file. 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.

Tip: 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 by using the PolicyTool. For more information, see “Using PolicyTool to edit policy files” on page 512.
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 525. The following instructions describe how to import a `was.policy` file.

 - a. Import the EAR file into an assembly tool.
 - 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 by 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 that is 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 512.

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`, delete the `was.policy` file for that application. The symptom of the missing permission is the `java.security.AccessControlException` exception. The missing permission is listed in the exception data:

```
java.security.AccessControlException: access denied (java.io.FilePermission
app_server_root/lib/mail-impl.jar read)
```

The previous example was split onto several lines for illustrative purposes only.

When a Java program receives this exception and adding this permission is justified, add the following permission to the `was.policy` file:

```
grant codeBase "file:user_client_installed_location" {
permission java.io.FilePermission "app_server_root/lib/mail-impl.jar", "read"; };
```

The previous example was split onto several lines for illustrative purposes only.

To determine whether to add a permission, see “Access control exception” on page 73.

Restart all applications for the updated `app.policy` file to take effect.

spi.policy file permissions:

Java 2 security uses several policy files to determine the granted permission for each Java program.

For the list of available policy files that are supported by WebSphere Application Server Version 6.0.x, see “Java 2 security policy files” on page 68.

Because the default permission for the Service Provider Interface (SPI) is the `AllPermission` permission, 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 java.policy, server.policy, and client.policy policy files. The JAAS Principal keyword is supported in a JAAS policy file that is specified by the java.security.auth.policy Java virtual machine (JVM) system property. 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.

To extract the filter.policy file, enter the following command using information from your environment:

```
set obj [$AdminConfig extract profiles/profile_name/cells/cell_name/nodes/node_name/spi.policy
c:/temp/test/spi.policy]
```

Edit the file using the Policy Tool. For more information, see “Using PolicyTool to edit policy files” on page 512.

To check in the policy file, enter the following command using information from your environment:

```
$AdminConfig checkin profiles/profile_name/cells/cell_name/nodes/node_name/spi.policy
c:/temp/test/spi.policy $obj
```

The updated spi.policy is applied to the Service Provider Interface (SPI) libraries after the Java process is restarted.

Examples

The spi.policy file is the template for SPIs or third-party resources embedded in the product. Examples 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 that are defined in the spi.policy file to the class paths defined in the resources.xml file. The union of the permission that is 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.

You can find the spi.policy file that is supplied by WebSphere Application Server in the following location: *app_server_root/profiles/profile_name/config/cells/cell_name/nodes/node_name/spi.policy*. This file 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.

library.policy file permissions:

Java 2 security uses several policy files to determine the granted permission for each Java program.

For the list of available policy files that are supported by WebSphere Application Server, see “Java 2 security policy files” on page 68.

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 `java.policy`, the `server.policy`, and the `client.policy` 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 the `java.security.auth.policy` file with `auth.policy.url.n=URL` where `URL` is the location of the authorization policy.

To extract the policy file, use a command prompt to enter the following command using the appropriate variable values for your environment:

```
wsadmin> set obj [$AdminConfig extract cells/cell_name/nodes/  
node_name/library.policy c:/temp/test/library.policy]
```

The previous two lines were split onto two lines for illustrative purposes only.

Edit the extracted `library.policy` file with the Policy Tool. For more information, see “Using PolicyTool to edit policy files” on page 512.

To check in the policy file, use a command prompt to enter the following command using the appropriate variable values for your environment:

```
wsadmin> $AdminConfig checkin cells/cell_name/nodes/node_name/library.policy  
c:/temp/test/library.policy $obj
```

An updated `library.policy` is applied to shared libraries after the servers restart.

Example

The union of the permission that is 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.

The `library.policy` file are supplied by WebSphere Application Server resides at: `app_server_root/config/cells/cell_name/nodes/node_name/` directory. The file 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, the `app.policy` file and the `library.policy` file, update the `library.policy` file. The missing permission causes the `java.security.AccessControlException` exception. The missing permission is listed in the exception data.

 For example:

```
java.security.AccessControlException: access denied (java.io.FilePermission  
app_server_root/lib/mail-impl.jar read)
```

The previous lines are split into two lines for illustrative purposes only. The `app_server_root` variable represents your installation directory.

When a Java program receives this exception and adding this permission is justified, add a permission to the `library.policy` file.

 For example:

```
grant { permission java.io.FilePermission "app_server_root/lib/mail-impl.jar", "read"; };
```

The previous lines are split into two lines for illustrative purposes only. The `app_server_root` variable represents your installation directory.

To decide whether to add a permission, refer to “Access control exception” on page 73.

Restart the related Java processes for the changes in the `library.policy` file to become effective.

Adding the `was.policy` file to applications:

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.

When Java 2 security is enabled for a WebSphere Application Server, all the applications that run on 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 you do not change this file. Add any new permissions that are required for an application in the `was.policy` file.

The `app.policy` file is located in the `profile_root/config/cells/cell_name/nodes/node_name` directory. The contents of the `app.policy` file are presented in the following example:

Attention: In the following code sample, the two permissions that are required by JavaMail are split onto two lines for illustrative purposes only.

```
// 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 "
        ${was.install.root}${/}lib${/}mail-impl.jar", "read";
    permission java.io.FilePermission "
        ${was.install.root}${/}lib${/}activation-impl.jar", "read"; };
    // The following permissions apply to all utility .jar files (other
    // than enterprise beans JAR files) in the application.
grant codeBase "file:${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 modules in the application. You can assign additional permissions to each module (`.war` file or `.jar` file), as shown in the following example.

The following example illustrates 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 onto two lines for illustrative purposes only.

```
// 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 app_server_root directory.
};
```

To use a `was.policy` file for your application, perform the following steps:

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 512.
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.
5. 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 `java.security.AccessControlException` exception. The missing permission is listed in the exception data, for example:

```
java.security.AccessControlException: access denied (java.io.FilePermission
app_server_root/lib/mail-impl.jar read)
```

The previous two lines are one continuous line for illustration purposes only.

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
    "app_server_root/lib/mail-impl.jar", "read";
};
```

Lines are split in this example for illustration purposes only.

Install the application.

Configuring static policy files

By configuring the static policy files, the required permission will be granted for all of the Java programs.

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 68 topic for the list of available policy files that are supported by WebSphere Application Server.

Two types of policy files are 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.

Policy file name	Description
<code>java.policy</code>	Contains default permissions for all of the Java programs on the node. This file seldom changes.
<code>server.policy</code>	Contains default permissions for all of the WebSphere Application Server programs on the node. This file is rarely updated.
<code>client.policy</code>	Contains default permissions for all of the applets and client containers on the node.

The static policy file is not a configuration file that is 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 514.
- If the permission is required only by applets and client containers, update the `client.policy` file. Refer to “client.policy file permissions” on page 531.
- If the permission is required only by WebSphere Application Server (servers, agents, managers and application servers), update the `server.policy` file. Refer to “server.policy file permissions” on page 529.
- 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 “java.policy file permissions” on page 528.

2. Stop and restart WebSphere Application Server.

The required permission is granted for all of the Java programs that run 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 results in the creation of the `java.security.AccessControlException` exception. The missing permission is listed in the exception data.

For example:

```
java.security.AccessControlException: access denied (java.io.FilePermission
C:/WAS_HOME/lib/mail-impl.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:/WAS_HOME/lib/mail-impl.jar",
        "read";
};
```

To decide whether to add a permission, refer to “Access control exception” on page 73.

java.policy file permissions:

Java 2 security uses several policy files to determine the granted permission for each Java program.

See “Java 2 security policy files” on page 68 for the list of available policy files that are supported by WebSphere Application Server.

The `java.policy` file is a global default policy file that is shared by all of the Java programs that run 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 the policy tool. For more information, see “Using PolicyTool to edit policy files” on page 512. 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 that run 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 that is 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 that is supplied by WebSphere Application Server is located at `install_root/java/jre/lib/security/java.policy`. This file 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, 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 creation of the `java.security.AccessControlException` exception. 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 73.

Restart all of the Java processes for the updated `java.policy` file to take effect.

server.policy file permissions:

Java 2 security uses several policy files to determine the granted permission for each Java program.

See “Java 2 security policy files” on page 68 for the list of available policy files that are supported by WebSphere Application Server.

The `server.policy` file is a default policy file that is shared by all of the WebSphere Application Servers on a node. The `server.policy` file is not a configuration file that is 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 that is 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 512 topic 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 ibm jdk extensions
grant codeBase "file:${was.install.root}/java/ext/-" {
    permission java.security.AllPermission;
};

// Allow to use ibm tools
grant codeBase "file:${was.install.root}/java/tools/ibmtools.jar" {
    permission java.security.AllPermission;
};

// Allow to use sun tools
grant codeBase "file:/QIBM/ProdData/Java400/jdk14/lib/tools.jar" {
    permission java.security.AllPermission;
};

// Allow to use sun tools (V5R2M0 codebase)
grant codeBase "file:/QIBM/ProdData/OS400/Java400/jdk/lib/tools.jar" {
    permission java.security.AllPermission;
};

// WebSphere system classes
grant codeBase "file:${was.install.root}/plugins/-" {
    permission java.security.AllPermission;
};
grant codeBase "file:${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;
};

// Allow the WebSphere deploy tool all permissions
grant codeBase "file:${was.install.root}/optionalLibraries/-" {
    permission java.security.AllPermission;
};

// Allow Channel Framework classes all permission
grant codeBase "file:${was.install.root}/installedChannels/-" {
    permission java.security.AllPermission;
};

grant codeBase "file:${user.install.root}/lib/-" {
    permission java.security.AllPermission;
};
```

```
grant codeBase "file:${user.install.root}/classes/-" {
    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 creates the `java.security.AccessControlException` exception. 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-impl.jar read)
```

The previous two lines are split into two lines for illustrative purposes only.

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 73.

Restart all of the Java processes for the updated `server.policy` file to take effect.

client.policy file permissions:

Java 2 security uses several policy files to determine the granted permission for each Java program.

For the list of available policy files that are supported by WebSphere Application Server, see “Java 2 security policy files” on page 68.

- The `client.policy` file is a default policy file that is shared by all of the WebSphere Application Server client containers and applets on a node.
- The union of the permissions that is contained in the `java.policy` file and the `client.policy` file are given to all of the client containers for WebSphere Application Server and applets running on the node.
- The `client.policy` file is not a configuration file that is 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 in the `profile_root/properties/client.policy`.
- 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.
- 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 512, to edit policy files. Changes to the `client.policy` file are local for the node.

This file contains these default permissions:

```
grant codeBase "file:${was.install.root}/java/ext/*" {
    permission java.security.AllPermission;
};
```

```
// JDK classes
grant codeBase "file:${was.install.root}/java/ext/-" {
```

```

    permission java.security.AllPermission;
};
grant codeBase "file:${was.install.root}/java/tools/ibmtools.jar" {
    permission java.security.AllPermission;
};
grant codeBase "file:/QIBM/ProdData/Java400/jdk14/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}/plugins/-" {
    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;
};
grant codeBase "file:${user.install.root}/installedConnectors/-" {
    permission java.security.AllPermission;
};

grant codeBase "file:${was.install.root}/installedChannels/-" {
    permission java.security.AllPermission;
};

// J2EE 1.4 permissions for client container WebSphere Application Server applications
// in $WAS_HOME/installedApps
grant codeBase "file:${user.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 1.4 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";
};

```

All of the client containers and applets on the local node are granted the updated permissions when they start. 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 creates the `java.security.AccessControlException` exception. 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 presented as such for illustrative purposes only.

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.jar", "read"; }`;

To decide whether to add a permission, refer to “Access control exception” on page 73.

If you update the policy file, you must restart the browser and any client applications.

Developing with programmatic security APIs for Web applications

Use this information to programmatically secure 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 that 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 that contains the remote user name. If no user is authenticated, it returns `null`.

You can configure several options for Web authentication that determine how the Web client interacts with protected and unprotected Uniform Resource Identifiers (URI). Also, you can specify whether WebSphere Application Server challenges the Web client for basic authentication information if the certificate authentication for the HTTPS client fails. For more information, see “Authentication mechanisms” on page 173.

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 that is passed to this method. Because 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 an assembly tool such as Rational Application Developer (RAD) 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 when a Web application needs 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 the `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`, and `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 899.

getRemoteUser and getAuthType methods

The `getRemoteUser` and `getAuthType` methods are methods of the `javax.servlet.http.HttpServletRequest` interface. If the user has been authenticated, the `getRemoteUser` method returns the login of the user that makes the request. If the user is not authenticated, the `getRemoteUser` method returns null. The `getAuthType` method returns the name of the authentication scheme that is used to protect the servlet (for example, BASIC or SSL). If the servlet is not protected, the `getAuthType` method returns null.

For both methods, the data that is returned depends upon whether security is enabled in the application server where the servlet is deployed. The following possibilities exist:

- If security is not enabled, a servlet is requested and it is configured with Web server protection. The `getRemoteUser` method returns the login and `getAuthType` method returns the authentication scheme.
- If security is enabled and a servlet is requested, both methods return null when WebSphere Application Server protection is not configured for the servlet.

- If security is enabled, a servlet is requested, and the servlet is configured with WebSphere Application Server protection, then the `getRemoteUser` method returns the login and the `getAuthType` method returns the configured authentication scheme.

Example: Web application code

The following example depicts a Web application or servlet using the programmatic security model.

This example illustrates one use and not necessarily the only use of the programmatic security model. The application can use the information that is returned by the `getUserPrincipal`, `isUserInRole`, and the `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` servlet as shown in the following example:

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

Web authentication settings

Use this page to specify the Web authentication settings that are associated with a Web client.

To view this administrative console page, complete the following steps:

1. Click **Security > Secure administration and applications**.
2. Under Authentication, expand **Web security** and click **General settings**.

You can override the global Web authentication setting that you select on this panel by specifying a system property on the server level. To specify the system property, complete the following steps:

1. Click **Servers > Application servers > server_name**.
2. Under Server infrastructure, click **Java and Process Management > Process definition**.
3. Under Additional properties, click **Java Virtual Machine > Custom properties > New**

You can specify the following system properties on the server level for Web authentication.

Table 15. Web authentication system property values

Property name	Value	Explanation
com.ibm.wsspi.security.web.webAuthReq	lazy	This value is equivalent to the Authenticate only when the URI is protected option.
com.ibm.wsspi.security.web.webAuthReq	persisting	This value is equivalent to the Use available authentication data when an unprotected URI is accessed option.
com.ibm.wsspi.security.web.webAuthReq	always	This value is equivalent to the Authenticate when any URI is accessed option.
com.ibm.wsspi.security.web.failOverToBasicAuth	true	This value is equivalent to the Default to basic authentication when certificate authentication for the HTTPS client fails option.

Authenticate only when the URI is protected:

The application server challenges the Web client to provide authentication data when the Web client accesses a Uniform Resource Identifier (URI) that is protected by a Java 2 Platform, Enterprise Edition (J2EE) role. The authenticated identity is available only when the Web client accesses a protected URI.

This option is the default J2EE Web authentication behavior that is also available in previous releases of WebSphere Application Server.

Default: Enabled

Use available authentication data when an unprotected URI is accessed:

The Web client can access validated authenticated data that it previously could not access. This option enables the Web client to call the `getRemoteUser`, `isUserInRole`, and `getUserPrincipal` methods to retrieve an authenticated identity from an unprotected URI.

When you select this option with the **Authenticate only when the URI is protected** option, the Web client can use authenticated data when the URI is protected or not protected.

Important: This option does not challenge the Web client to provide authenticated data if the Web client accesses an unprotected URI without authenticated data.

Default: Disabled

Authenticate when any URI is accessed:

The Web client must provide authentication data regardless of whether the URI is protected.

Default: Disabled

Default to basic authentication when certificate authentication for the HTTPS client fails:

When the required HTTPS client certificate authentication fails, the application server uses the basic authentication method to challenge the Web client to provide a user ID and password.

The HTTP client certification authentication that is performed by the application server security is different from the client authentication that is performed by the Web server plug-in. If you configure the Web server plug-in for mutual authentication and client authentication fails, the following situations will occur:

- The Web server produces a error and the Web request is not processed by application server security.
- The application server cannot fail over to basic authentication.

Default: Disabled

Developing with programmatic APIs for EJB applications

Use this topic to programmatically secure your Enterprise JavaBeans (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` application programming interface (API) 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 security role that is 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 that contains the bean caller name. If the caller is not authenticated, it returns a principal that contains an unauthorized name.

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

When the `isCallerInRole` method is used, declare a `security-role-ref` element in the deployment descriptor with a `role-name` that is subelement containing the role name that is passed to this method. Because 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 an 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 an assembly tool such as Rational Application Developer (RAD) 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.

Performing the previous steps result in 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.

Using J2EE security model capabilities to specify security policies declaratively is useful 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, you can create a `security-role-ref` element:

```
<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 for example `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 919.

Example: Enterprise bean application code

The following Enterprise JavaBeans (EJB) component example illustrates the use of the `isCallerInRole` and the `getCallerPrincipal` methods in an EJB module.

Using that declarative security is recommended. The following example is one way of using the `isCallerInRole` and the `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() {
        return message;
    }

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

        // 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 the 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 919. Use the information under Map security-role-ref and role-name to role-link to create the element.

Customizing Web application login

You can create a form login page and an error page to authenticate a user.

A Web client or a 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 a 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 authentication mechanism is strong 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 request to get the page that is specified in the Web address. 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 542.
 2. Create an error page. You can program error pages to retry authentication or to 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 when the 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

This article provides several examples pertaining to form login.

For the authentication to proceed appropriately, the action of the login form must always have the `j_security_check` action. 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, the server 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>FORM<auth-method>
<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 that shows login and error pages for the previous login configuration follows:

```

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 of the form logout mechanism, access to a protected Web resource requires re-authentication. This feature is not required by J2EE specifications, but it is provided as an additional feature in WebSphere Application Server security.

Suppose that you want 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. The form-logout page is an HTML or a JavaServer Pages (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, the form-logout page has a special post action. This post action is recognized by the Web container, which dispatches the post action to a special internal 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 a JSP file within the same Web application to which the user is redirected after logging out. Additionally, the logout-exit page permits a fully qualified URL in the form of `http://hostname:port/URL`. 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.

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 to use the information that is contained in the requests or responses. One or more servlet filters can be attached to a servlet or to a group of servlets. Servlet filters also can attach to JavaServer Pages (JSP) files and HTML pages. All of 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 that contains the user name and the `j_password` parameter that contains the password. A servlet filter can use the user name parameter and password information to perform more authentication or other special needs.

1. A servlet filter implements the `javax.servlet.Filter` class. Implement three methods in the filter class:
 - **init(javax.servlet.FilterConfig cfg)**. This method is called by the container 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. The `FilterChain` chain that is passed to this method can be used to invoke the next filter in the chain of filters. The original requested servlet runs when the last filter in the chain calls the `chain.doFilter` method. Therefore, all filters call the `chain.doFilter` method for the original servlet to run after filtering. If an additional authentication check is implemented in the filter code and results in failure, the original servlet does not run. The `chain.doFilter` method is not called and can be redirected to some other error page.
2. If a servlet maps to many servlet filters, servlet filters are called in the order that is listed in the `web.xml` deployment descriptor of the application. Place the servlet filter class file in the `WEB-INF/classes` directory of the application.

An example of a servlet filter follows: This login filter can map to the `/j_security_check` URL 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.
    }
}
```

Example of servlet filters

This example illustrates one way that 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 filter is 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` is broken into two lines for illustrative purposes only. The `public void doFilter(ServletRequest request` line and the line after it are one continuous line.

An example of the `web.xml` file that shows the `LoginFilter` filter configured and mapped to the `j_security_check` URL:

```

<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

```

Configuring servlet filters

IBM Rational Application Developer or an assembly tool can configure the servlet filters. Two steps are involved 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 web.xml application deployment descriptor 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 a URL or a servlet.

After mapping the servlet filter to a URL or a servlet, the web.xml application deployment descriptor 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 servlet, 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 \(Samples Gallery\)](#).

Customizing application login with Java Authentication and Authorization Service

The following topics are covered in this section:

- Developing programmatic logins with the Java Authentication and Authorization Service (JAAS)
- Configuring programmatic logins for JAAS
- Configuring a server-side Java Authentication and Authorization Service authentication and login configuration

Developing programmatic logins with the Java Authentication and Authorization Service

Use this topic to develop programmatic logins with the Java Authentication and Authorization Service.

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

JAAS replaces the Common Object Request Broker Architecture (CORBA) programmatic login application programming interfaces (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 a custom JAAS login configuration, verify that the JAAS login configuration is properly defined. See “Configuring programmatic logins for Java Authentication and Authorization Service” on page 552 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.

For details, see the following articles:

- “Adding the `was.policy` file to applications” on page 525
- “Using PolicyTool to edit policy files” on page 512
- “Configuring the `was.policy` file” on page 520

For more details on which APIs are protected by Java 2 security permissions, check the IBM Developer Kit, Java Technology Edition; JAAS and WebSphere Application Server public APIs documentation in Security: Resources for learning.

Some of the APIs that are used in the sample code in this documentation and the Java 2 security permissions that are required by these APIs are in the following list:

- `javax.security.auth.login.LoginContext` constructors are protected by the `javax.security.auth.AuthPermission "createLoginContext"` object.
 - `javax.security.auth.Subject.doAs` and `com.ibm.websphere.security.auth.WSSubject.doAs` methods are protected by the `javax.security.auth.AuthPermission "doAs"` object.
 - `javax.security.auth.Subject.doAsPrivileged` and `com.ibm.websphere.security.auth.WSSubject.doAsPrivileged` methods are protected by the `javax.security.auth.AuthPermission "doAsPrivileged"` object.
- **Enhanced model to Java 2 Platform, Enterprise Edition (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 that is associated with the running thread inside a `java.security.AccessController.doPrivileged` code block. This oversight can present inconsistent behavior, which might have unwanted effects. The `com.ibm.websphere.security.auth.WSSubject` class provides a workaround to associate a Subject to a running thread. The `com.ibm.websphere.security.auth.WSSubject` class extends the JAAS model to Java 2 Platform, Enterprise Edition (J2EE) resources for authorization checks. If the Subject associates with the running thread 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 resource authorization checks.

- **User interface support for defining new JAAS login configuration.**

You can configure a JAAS login configuration in the administrative console and store the JAAS login configuration in a configuration repository. Applications can define a new JAAS login configuration in the administrative console and the data is persisted in the configuration repository. However, WebSphere Application Server still supports the default JAAS login configuration format (plain text file) that is provided by the JAAS default implementation. If duplicate login configurations are defined in both the configuration repository and the plain text file format, the one in the repository takes precedence.

Advantages to defining the login configuration in the configuration repository includes:

- Administrative console support in defining JAAS login configuration
 - Central management of the JAAS login configuration
- **Application support for programmatic authentication.**

WebSphere Application Server provides JAAS login configurations for applications to perform programmatic authentication to the WebSphere security runtime. These configurations perform authentication to the WebSphere Application Server-configured authentication mechanism (Simple WebSphere Authentication Mechanism (SWAM) or Lightweight Third Party Authentication (LTPA)) and

user registry (Local OS, Lightweight Directory Access Protocol (LDAP), custom registries, or federated repositories) based on the authentication data that is supplied. The authenticated Subject from these JAAS login configurations contains the required principal and credentials that the WebSphere security runtime can use to perform authorization checks on J2EE role-based protected resources.

Note: SWAM is deprecated in WebSphere Application Server Version 6.1 and will be removed in a future release.

Here are the JAAS login configurations that are provided by WebSphere Application Server:

- **WSLogin JAAS login configuration.** A generic JAAS login configuration can use Java clients, client container applications, servlets, JavaServer Pages (JSP) files, and Enterprise JavaBeans (EJB) components to perform authentication based on a user ID and password, or a token to the security runtime for WebSphere Application Server. However, this configuration does not honor the CallbackHandler handler that is specified in the client container deployment descriptor.
- **ClientContainer JAAS login configuration.** This JAAS login configuration honors the CallbackHandler handler that is specified in the client container deployment descriptor. The login module of this login configuration uses the CallbackHandler handler in the client container deployment descriptor if one is specified, even if the application code specified one callback handler in the login context. This is for a client container application.

A Subject authenticated with the previously mentioned JAAS login configurations contains a `com.ibm.websphere.security.auth.WSPPrincipal` principal and a `com.ibm.websphere.security.cred.WSCredential` credential. If the authenticated Subject is passed in the `com.ibm.websphere.security.auth.WSSubject.doAs` or the other `doAs` methods, the product security runtime can perform authorization checks on J2EE resources based on the `com.ibm.websphere.security.cred.WSCredential` Subject.

- **Customer-defined JAAS login configurations.**

You can define other JAAS login configurations to perform programmatic authentication to your authentication mechanism. See the “Configuring programmatic logins for Java Authentication and Authorization Service” on page 552 for details. For the product security runtime 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 running 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 to which you are authenticating. Pointing to one cell 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 “Example: Programmatic logins” article.

Example: Programmatic logins

This 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", "password"));
} catch (LoginException le) {
    System.out.println("Cannot create LoginContext. " + le.getMessage());
    // Insert the error processing code
} catch (SecurityException se) {
    System.out.println("Cannot create LoginContext." + se.getMessage());
    // Insert the error processing code
}

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

```

As shown in the example, the new login context is initialized with the WSLogin login configuration and the WSCallbackHandlerImpl callback handler. Use the WSCallbackHandlerImpl instance on a server-side application where you do not want prompting. A WSCallbackHandlerImpl instance is initialized by the specified user ID, password, and realm information. The present WSLoginModuleImpl class implementation that is specified by the WSLogin login configuration can only retrieve authentication information from the specified callback handler. You can construct a login context with a Subject object, but the Subject is disregarded by the present WSLoginModuleImpl implementation. For product client-container applications, replace WSLogin login configuration 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 callback handler 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 callback handler implementations, depending on the particular application environment. You can develop a new callback handler if neither of these implementations fit your particular application requirement.

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

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

Attention: The first line starting with `env.put` was split into two lines for illustration purposes only.

```
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.println("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
}
```

Configuring programmatic logins for Java Authentication and Authorization Service

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.

Java Authentication and Authorization Service (JAAS) is a feature in WebSphere Application Server. JAAS 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 the JAAS login in the administrative console and store this login configuration in the Application Server configuration. However, WebSphere Application Server still supports the default JAAS login configuration format (plain text file) that is provided by the JAAS default implementation. If duplicate login configurations are defined in both the WebSphere Application Server configuration API and the plain text file format, the one in the WebSphere Application Server 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

Due to a design oversight in JAAS Version 1.0, the `javax.security.auth.Subject.getSubject` method does not return the subject that is associated with the running thread inside a `java.security.AccessController.doPrivileged` code block. This problem presents an inconsistent behavior that might cause unfavorable results. 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 module. 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 class path for the Java extension class loader with this proxy LoginModule module.

If you do not want to use the Proxy LoginModule module, you can place the LoginModule module in the `app_server_root/lib/ext/` directory. However, this action is not recommended due to the security risks.

JAAS login configurations are defined in the WebSphere Application Server configuration application programming interface (API) security document. Click **Security > Secure administration, applications, and infrastructure**. Under Java Authentication and Authorization Service, click **Application logins**. The following JAAS login configurations are available:

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. For more information, see “Configuration entry settings for Java Authentication and Authorization Service” on page 557.

DefaultPrincipalMapping,

Defines a special LoginModule module that is typically used by J2EE connectors to map an authenticated WebSphere Application Server 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.

WSLogin

Defines a login configuration and a LoginModule implementation that applications can use in general.

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 (such as, ClientContainer, WSLogin, and DefaultPrincipalMapping). Deleting or removing them can cause other enterprise applications to fail.

1. Delete a JAAS login configuration.

- a. Click **Security > Secure administration, applications, and infrastructure**.
 - b. Under Java Authentication and Authorization Service, click **Application logins**. The Application Login Configuration panel is displayed.
 - c. Select the check box for the login configurations to delete and click **Delete**.
2. Create a new JAAS login configuration.
- a. Click **Security > Secure administration, applications, and infrastructure**.
 - b. Under Java Authentication and Authorization Service, click **Application logins**.
 - c. Click **New**. The Application Login Configuration panel is displayed.
 - 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` context.
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. Under Additional properties, click **JAAS Login Modules**.
 - f. Click **New**.
 - g. Specify the Module class name. Specify the WebSphere Application Server proxy `LoginModule` module because of the limitation of the class loader visibility.
 - h. Specify the `LoginModule` implementation as the delegate property of the Proxy `LoginModule` module. The WebSphere Application Server 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. Under Additional properties, 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 `true` value. These properties are passed to the `LoginModule` class as options to the `initialize` method of the `LoginModule` instance.
 - l. Click **Save**.

Several locations are 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 Platform, Enterprise Edition (J2EE) application.
If you place the login module within the EAR file, the login module is accessible by 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.
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, 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, which is a plain text file, that is 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, which is

located in the *app_server_root/properties/wsjaas.conf* file. 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 *app_server_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 that are defined in the *app_server_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 that are 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 Application Server strategic application programming interface (API) for authentication that replaces the Common Object Request Broker Architecture (CORBA) programmatic login API.

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 or by using the scripting functions and store this configuration in the WebSphere Application Server configuration API. However, WebSphere Application Server still supports the default JAAS login configuration format, a plain text file, which is provided by the JAAS default implementation. If duplicate login configurations are defined in both the WebSphere Application Server configuration API and the plain text file format, the one in the WebSphere Application Server 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 JAAS 1.0, the `javax.security.auth.Subject.getSubject` method does not return the Subject that is associated with the running thread inside a `java.security.AccessController.doPrivileged` code block. This action can present an inconsistent behavior that is problematic. The `com.ibm.websphere.security.auth.WSSubject` extension provides a workaround to associate the Subject to the running thread. The `com.ibm.websphere.security.auth.WSSubject` extension expands the JAAS authorization model to J2EE resources.

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 an `AccessController.doPrivileged` call is contained 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() {
```


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, see the Java 2 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 runtime 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` subdirectory of the root directory under which WebSphere Application Server is installed. Changes made to the `wsjaas.conf` file are used only by the local application server and take effect after the application server restarts. The JAAS configuration in the WebSphere Application Server configuration API security document takes precedence over that defined in the `wsjaas.conf` file. A configuration entry in the `wsjaas.conf` is overridden by an entry of the same alias name in the WebSphere Application Server configuration API security document.

The Java Authentication and Authorization Service (JAAS) login configuration entries in the administrative console are propagated to the server runtime when they are created, not when the configuration is saved. However, the deleted JAAS login configuration entries are not removed from the server runtime. 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 runtime 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 a 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 Java 2 Platform, Enterprise Edition (J2EE) components such as enterprise beans, JavaServer Pages (JSP) files, servlets, resource adapters, and message-driven beans (MDBs).

To view this administrative console page, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Authentication, click **Java Authentication and Authorization Service > Application logins**.

Read the JAAS specifications before you begin defining additional login modules for authenticating to the application server security run time. You can define additional login configurations for your applications. However, if the application server LoginModule `com.ibm.ws.security.common.auth.module.WSLoginModuleImpl` module is not used or the LoginModule module does not produce a credential that is recognized by the application server. The application server security run time cannot use the authenticated subject from these login configurations for an authorization check for resource access.

You must invoke Java client programs that use Java Authentication and Authorization Service (JAAS) for authentication with a JAAS configuration file that is specified. The application server supplies the `wsjaas_client.conf` default JAAS configuration file under the `app_server_root/properties` directory. This configuration file is set in the `app_server_root/bin/launchClient.bat` file as:

```
set JAAS_LOGIN_CONFIG=-Djava.security.auth.login.config=%WAS_HOME%\properties\wsjaas_client.conf
```

ClientContainer:

Specifies the login configuration used by the client container application, which uses the CallbackHandler API that is defined in the client container deployment descriptor.

The ClientContainer configuration is the default login configuration for the application server. Do not remove this default, as other applications that use it fail.

Default: ClientContainer

DefaultPrincipalMapping:

Specifies the login configuration that is used by Java 2 Connectors to map users to principals that are defined in the J2C authentication data entries.

The ClientContainer configuration is the default login configuration for the application server. Do not remove this default, as other applications that use it fail.

Default: ClientContainer

WSLogin:

Indicates whether all of the applications can use the WSLogin configuration to perform authentication for the application server security run time.

This login configuration does not honor the CallbackHandler handler that is 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 application server. Do not remove this default because other administrative applications that use it fail. This login configuration authenticates users for the application server security run time. Use the credentials from the authenticated subject that are returned from this login configuration as an authorization check for access to 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, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Authentication, click **Java Authentication and Authorization Service > System logins**.

Read the Java Authentication and Authorization Service documentation before you begin defining additional login modules for authenticating to the application server security runtime. 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.

RMI_INBOUND

This login configuration handles logins for inbound RMI requests. Typically, these logins are requests for authenticated access to Enterprise JavaBeans (EJB) files. When using the RMI connector, these logins might be requests for Java Management Extensions (JMX).

WEB_INBOUND

This login configuration handles logins for Web application requests, which include 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 that is passed into the WEB_INBOUND login configuration might contain objects that are generated by the TAI.

DEFAULT

This login configuration handles the logins for inbound requests that are made by most of the other protocols and internal authentications.

These three login configurations will 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 the 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 list of the TokenHolder objects that are returned from the call to the WSOpaqueTokenHelper. The callback uses the createTokenHolderListFromOpaqueToken method with the Common Secure Interoperability version 2 (CSlv2) 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. You must enable security attribute propagation for both the outbound and inbound authentication.

You can enable the **Security attribute propagation** option for CSlv2 outbound authentication by completing the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Authentication, expand RMI/IIOP security and click **CSlv2 outbound authentication**.
3. Enable the **Security attribute propagation** option.

You can enable the **Security attribute propagation** option for CSlv2 inbound authentication by completing the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Authentication, expand RMI/IIOP security and click **CSlv2 inbound authentication**.
3. Enable the **Security attribute propagation** option.

In system login configurations, the application server authenticates the user based upon the information that is 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 CSlv2 identity assertion; Web and CSlv2 X509 certificate logins; old-style trust association interceptor logins, and so on. In Web and CSlv2 X509 certificate logins, the application server maps the certificate to a user name. This callback is used by any login type that establishes trust with 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 a single sign-on (SSO) or downstream login. Any time a request originates from the application server, instead of a pure client, the LTPA token flows to the target server. For single sign-on (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 application server login modules and it changes the identity using a credential mapping service, 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 for this validation to be successful. A security exposure is possible 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, the application server uses the `NameCallback` callback and the token holder, which contains all of the propagated attributes, to deserialize most of the objects. The application server uses the `NameCallback` callback 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 > Secure administration, applications, and infrastructure**.
2. Under Authentication, click **CSIV2 inbound authentication**.

A custom login module needs to handle custom serialization. For more information, see "Security attribute propagation" in the information center.

In addition to the callbacks that are defined previously, the `WEB_INBOUND` login configuration can contain the following additional callbacks only:

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 cookie to the response.

Callback

```
callbacks[6] = new com.ibm.websphere.security.auth.callback.  
WSApplicationContextCallback("ApplicationContextCallback: ");
```

Responsibility

Collects the Web application context used during the login. This callback consists of a hashtable, which if present contains the application name and the redirected Web address.

Callback

```
callbacks[7] = new WSRealmNameCallbackImpl("Realm Name: ", <default_realm>);
```

Responsibility

Collects the realm name for the login information. The realm information might not always be provided and should be assumed to be the current realm if it is not provided.

Callback

```
callbacks[8] = new WSX509CertificateChainCallback("X509Certificate[]: ");
```

Responsibility

If the login source is an X509Certificate from SSL client authentication, this callback contains the certificate that was validated by SSL. The ltpaLoginModule calls the same mapping functions as in previous releases. Once it is passed into the login, it provides a custom login module with the opportunity to map the certificate in a custom way. It then perform a hashtable login (see "Example: Custom login module for inbound mapping" on page 300 for an example of a hashtable login).

If you want to use security attribute propagation with the WEB_INBOUND login configuration, you can enable **Web inbound security attribute propagation** option on the Single sign-on panel.

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Authentication, expand Web security and click **Single sign-on (SSO)**.
3. Select the **Web inbound security attribute propagation** option.

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`
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 context.
- 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 callback is present, the login module deserializes the byte token objects and regenerates the serialized Subject contents. The java.util.Hashtable hashtable takes precedence over all of the other forms of login. Be careful to avoid duplicating or overriding what the application server might have propagated previously.

By specifying a java.util.Hashtable 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 callback. 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 the application server as sufficient login information, see "Configuring inbound identity mapping" in the information center.

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, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Authentication, expand RMI/IIOP security and 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 the application server Version 5.1.1 or later (or 5.1.0.2 for z/OS) 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, the application server removes the need to access the user registry again and look up the security attributes of the user for authorization purposes. Additionally, any custom objects that are added at the sending server are present in the Subject at the downstream server.

The following callback is available 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 the application server must map the realm. For more information, see "Configuring outbound mapping to a different target realm" in the information center.

Callback

```
callbacks[0] = new WSPolicyCallback("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 policy 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.Im.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 are processed after this login module acts on the new Subject contents.

For more information, see "Configuring outbound mapping to a different target realm" in the information center.

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, the application server cannot send requests from server to server. In this case, you must use LTPA.

Note: The SWAM login configuration is deprecated and will be removed in a future release.

wssecurity.IDAssertion:

Processes login configuration requests for Web services security using identity assertion.

This login configuration is for Version 5.x systems. For more information, see "Identity assertion authentication method" in the information center.

wssecurity.PKCS7:

Verifies an X.509 certificate with a certificate revocation list in a Public Key Cryptography Standards #7 (PKCS7) object.

This login configuration is for Version 6.0.x systems.

wssecurity.PkiPath:

Verifies an X.509 certificate with a public key infrastructure (PKI) path.

This login configuration is for Version 6.0.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:

Verifies basic authentication (user name and password).

This login configuration is for Version 6.0.x systems.

wssecurity.X509BST:

Verifies an X.509 binary security token (BST) by checking the validity of the certificate and the certificate path.

This login configuration is for Version 6.0.x systems.

LTPA_WEB:

Processes login requests to components in the Web container such as servlets and JavaServer pages (JSP) files.

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 "Example: Customizing a server-side Java Authentication and Authorization Service authentication and logon configuration" in the information center.

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.Im.ItpaLoginModule` 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 "Example: Customizing a server-side Java Authentication and Authorization Service authentication and logon configuration" in the information center.

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, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Authentication, click **Java Authentication and Authorization Service > Application logins** or **System logins > alias_name**.
3. Under Additional properties, click **JAAS login modules**.

Module class name:

Specifies the class name of the given login module.

Data type: String

Use login module proxy:

Specifies that the Java Authentication and Authorization Service (JAAS) loads the login module proxy class. JAAS then delegates calls to the login module classes that are defined in the Module class name field.

Use this option when you use both Version 5.x and Version 6 Application Servers in the same environment. If you migrate a Version 5.x Application Server to Version 6, WebSphere Application Server Version 6 automatically enables this option. If you have Version 6 only cells in your environment, you might choose to deselect this option.

Default: Enabled

Proxy class name:

Specifies the name of the proxy login module class.

The default login modules that are defined by the application server use the `com.ibm.ws.security.common.auth.module.WSLoginModuleProxy` proxy `LoginModule` class. This proxy class loads the 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 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` module is required to succeed. Whether authentication succeeds or fails, the process still continues down the `LoginModule` list for each realm.

Requisite

The `LoginModule` module is required to succeed. If authentication is successful, the process continues down the `LoginModule` list in the realm entry. If authentication fails, control immediately returns to the application. Authentication does not proceed down the `LoginModule` list.

Sufficient

The `LoginModule` module is not required to succeed. If authentication succeeds, control

immediately returns to the application. Authentication does not proceed down the LoginModule list. If authentication fails, the process continues down the list.

Optional

The LoginModule module is not required to succeed. Whether authentication succeeds or fails, the process still continues 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 pass 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 the 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 > Secure administration, applications, and infrastructure**.
2. Under Authentication, click **Java Authentication and Authorization Service > Application logins or System logins > alias**. 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 click **OK** and save the changes, the new order is reflected on either the Application login configuration or the 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, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Authentication, click **Java Authentication and Authorization Service > 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

Managing J2EE Connector Architecture authentication data entries

This task creates and deletes Java 2 Connector (J2C) 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 authentication 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 repository.

Description

A short text description.

1. Delete a J2C authentication data entry.
 - a. Click **Security > Secure administration, applications, and infrastructure**.
 - b. Under Java Authentication and Authorization Service, click **J2C authentication data**. The **J2C Authentication Data Entries** panel is displayed.
 - c. 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 > Secure administration, applications, and infrastructure**.
 - b. Under Java Authentication and Authorization Service, click **J2C authentication data**. The **J2C Authentication Data Entries** panel is displayed.
 - c. Click **New**.
 - d. Enter a unique alias, a valid user ID, a valid password, and a short description (optional).
 - e. Click **OK** or **Apply**. No validation for the user ID and password is required.
 - f. Click **Save**.

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.

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 Java Authentication and Authorization Service (JAAS) configuration page or from other pages for resources that use J2EE Connector (J2C) authentication data entries. To view this administrative page, complete the following steps:

1. Click **Security > Secure administration, applications, and infrastructure**.
2. Under Authentication, click **Java Authentication and Authorization Service > 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
-------------------	--------

Password:

Specifies the password that is associated with the user identity.

This field is not available on the collections panel. However, the panel is available when you create a new J2C authentication data entry.

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

J2C principal mapping modules:

You can develop your own J2EE Connector (J2C) mapping module if your application requires more sophisticated mapping functions. The mapping login module that you might have developed on WebSphere Application Server Version 5.x is still supported in WebSphere Application Server Version 6.0.x and later.

You can use the Version 5.x login modules in the connection factory mapping configuration. These login modules can also be used in the reference mapping configuration for the resource manager connection factory. A version 5.x mapping login module is not able to use the custom mapping properties.

If you want to develop a new mapping login module in Version 6.0.x and later, use the programming interface that is described in the following sections.

transition: Migrate your Version 5.x mapping login module to use the new programming model and the new custom properties as well as the mapping configuration isolation at application scope. Note that mapping login modules that are developed using WebSphere Application Server Version 6.0.x cannot be used in the deprecated mapping configuration for the resource connection factory.

Invoking the login module for the resource reference mapping

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 6.0.x.

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;
}
```

The `WSMappingCallbackHandler` handler can manage two new callback types that are defined in Version 6.0.x:

```
com.ibm.wsspi.security.auth.callback.WSManagedConnectionFactoryCallback
com.ibm.wsspi.security.auth.callback.WSMappingPropertiesCallback
```

The new login modules use the two callback types that are used at the reference mapping configuration for the resource manager connection factory. The `WSManagedConnectionFactoryCallback` callback provides a `ManagedConnectionFactory` instance that you set in the `PasswordCredential` credential. With this setting, the `ManagedConnectionFactory` instance can determine whether a `PasswordCredential` instance is used for signon to the target Enterprise Information Systems (EIS) instance. The `WSMappingPropertiesCallback` callback provides a hash map that contains custom mapping properties. The `com.ibm.mapping.authDataAlias` property name is reserved for setting the authentication data alias.

The WebSphere Application Server `WSMappingCallbackHandle` handle continues to support the two WebSphere Application Server Version 5.x callback types that older mapping login modules can use. The two callbacks defined can be used only by login modules that the login configuration uses at the connection factory. For backward compatibility, WebSphere Application Server Version 6.0.x and later passes the authentication data alias, if defined in the list of custom properties under the `com.ibm.mapping.authDataAlias` property name using the `WSAuthDataAliasCallback` callback to Version 5.x login modules:

```
com.ibm.ws.security.auth.j2c.WSManagedConnectionFactoryCallback
com.ibm.ws.security.auth.j2c.WSAuthDataAliasCallback
```

Invoking the login module for the connection factory mapping

The `WSPrincipalMappingCallbackHandler` class handles two callback types:

```
com.ibm.wsspi.security.auth.callback.WSManagedConnectionFactoryCallback
com.ibm.wsspi.security.auth.callback.WSMappingPropertiesCallback
```

The `WSPrincipalMappingCallbackHandler` handler and the two callbacks are deprecated in WebSphere Application Server Version 6.

Passing the mapping properties for the resource reference to the mapping login module

You can pass arbitrary custom properties to your mapping login module. The following example shows how the WebSphere Application Server default mapping login module 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 default mapping login module for WebSphere Application Server Version 6.0.x requires one mapping property to define the authentication data alias. The mapping property, which is called `MAPPING_ALIAS`, is defined in the `Constants.class` file in the `com.ibm.wsspi.security.auth.callback` package.

```
MAPPING_ALIAS    =    "com.ibm.mapping.authDataAlias"
```

When you click **Use default method > Select authentication data entry authentication** 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 create your own custom login configuration and then use the default mapping login module, you must 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` principal in the principal set and a `WSCredential` credential 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` credential or an `org.ietf.jgss.GSSCredential` instance in the set of private credentials.

The `GenericCredential` interface that is defined in Java Cryptography Architecture (JCA) Specification Version 1.0 is removed in the JCA Version 1.5 specification. The `GenericCredential` interface is supported by WebSphere Application Server Version 6.0.x to support older resource adapters that might be programmed to the `GenericCredential` interface.

Customizing an application login to perform an identity assertion

Using the Java Authentication and Authorization Service (JAAS) login framework, you can create a JAAS login configuration that can be used to perform login to an identity assertion.

You can allow an application or system provider to perform an identity assertion with trust validation. To do this, you use the JAAS login framework, where trust validation is accomplished in one login module and credential creation is accomplished in another module. The two custom login modules allow you to create a JAAS login configuration that can be used to perform a login to an identity assertion.

Two custom login modules are required:

User implemented trust association login module (trust validation)

The user implemented trust association login module performs whatever trust verification the user requires. When trust is verified, the trust verification status and the login identity should be put into a map in the share state of the login module so that the credential creation login module can use the information. This map should be stored in the property:

`com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.state`
(which consists of)

`com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.trusted`
(which is set to **true** if trusted and **false** if not trusted)

`com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.principal`
(which contains the principal of the identity)

`com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.certificates`
(which contains the certificate of the identity)

Identity assertion login module (credential creation)

The `com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule` performs the credential creation. This module relies on the trust state information being in the login context's shared state. This login module is protected by the Java 2 security runtime permissions for:

- `com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.initialize`
- `com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.login`

The identity assertion login module looks for the trust information in the shared state property, `com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.state`, which contains the trust status and the identity to login and should include:

`com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.trusted`
(which when **true** indicates trusted and **false** when not trusted)

`com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.principal`
(which contains the principal of the identity to login, if using a principal)

`com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.certificates`
(which contains a array of a certificate chain that contains the identity to login, if using a certificate)

A `WSLoginFailedException` is returned if the state, trust, or identity information is missing. The login module then performs a login of the identity, and the subject will contain the new identity

1. Delegate trust validation to a user implemented plug point. Trust validation must be accomplished in a custom login module. This custom login module should perform any trust validation required, then set the trust and identity information in the shared state to be passed on to the identity assertion login module. A map is required in the shared state key, `com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.state`. If the state is missing then a `WSLoginFailedException` is thrown by the `IdentityAssertionLoginModule`. This map must include:
 - A trust key called `com.ibm.wsspi.securirty.common.auth.module.IdentityAssertionLoginModule.trust`. If the key is set to **true**, then trust is established. If the key is set to **false**, then no trust is established. If the trust key is not set to true, then the `IdentityAssertionLoginModule` will throw a `WSLoginFailedException`.
 - An identity key is set: A `java.security.Principal` can be set in the `com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.principal` key.
 - Or a `java.security.cert.X509Certificate[]` can be set in the `com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.certificates` key

- If both a principal and certificate are supplied, then the principal is used and a warning is issued.
2. Create a new JAAS configuration for application logins. The JAAS configuration will contain the user implemented trust validation custom login module and the IdentityAssertionLoginModule. Then to configure an application login configuration, perform the following on the administration console:
 - a. Expand **Security > Secure administration, applications, and infrastructure**
 - b. Expand **Java authentication and authorization services > Application logins**
 - c. Select **New**.
 - d. Give the JAAS configuration an alias.
 - e. Click **Apply**.
 - f. Select **JAAS Login Modules**
 - g. Select **New**.
 - h. Enter the **Module class name** of the user implemented trust validation custom login module.
 - i. Click **Apply**.
 - j. Enter the **Module class name** of
com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule
 - k. Make sure the **Module class name** classes are in the correct order. The user implemented trust validation login module should be first and the IdentityAssertionLoginModule should be the second class in the list.
 - l. Click **Save**.

This JAAS configuration is then used by the application to perform an Identity Assertion.

3. Perform the programmable identity assertion. A program can now use the JAAS login configuration to perform a programmatic identity assertion. The application program can create a login context for the JAAS configuration created in step 2, then login to that login context with the identity they would assert to. If the login is successful then that identity can be set in the current running process. Here is an example of how such code would operate:

```
MyCallbackHandler handler = new MyCallbackHandler(new MyPrincipal("Joe"));
LoginContext lc = new LoginContext("MyAppLoginConfig", handler);
lc.login(); //assume successful
Subject s = lc.getSubject();
WSSubject.setRunAsSubject(s);
// From here on , the runas identity is "Joe"
```

Using the JAAS login framework and two user implemented login modules, you can create a JAAS login configuration that can be used to perform login to an identity assertion.

Customization of 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 the WSCredential credential and WSPincipal principal 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 191. To determine which login configuration to use for plugging in your custom login modules, see the descriptions of the login configurations that are located in the “System login configuration entry settings for Java Authentication and Authorization Service” on page 558.

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 > Secure administration, applications, and infrastructure**. Under Java Authentication and Authorization Service, click **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 are split into two lines.

```
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
    $entry {{moduleClassName
33. set newPropertyId [$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 login module before the AuthenLoginModule login module, delete the AuthenLoginModule login module and recreate it after inserting the custom login module. Save the sample script into a sample.jacl file, and run 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 login modules:

```

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 are split into two or more lines for illustrative purposes only.

```
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
    $newJAASConfigurationEntryId
    {{moduleName
    "com.ibm.ws.security.common.auth.module.proxy.WSLoginModuleProxy"}}]
125.
126. set newPropertyId [$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 ItpaLoginModule and AuthenLoginModule login modules use the shared state to save state information so that custom login modules can modify the information. The ItpaLoginModule login module initializes the callback array in the login method using the following code. The callback array is created by the ItpaLoginModule login module only if an array is not defined in the shared state area. In the following code sample, the error handling code is removed to make the sample

concise. If you insert a custom login module before the `ItpaLoginModule` login module, the custom login module might follow the same style to save the callback into the shared state.

Attention: In the following code sample, several lines of code are split into two lines for illustrative purposes only.

```
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.     }
```

The `ItpaLoginModule` and `AuthenLoginModule` login modules generate both a `WSPrincipal` object 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 login module, which is inserted after the `ItpaLoginModule` and the `AuthenLoginModule` login modules, can modify the `WSPrincipal` and `WSCredential` objects before these objects 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 to make security decisions.

`AuthenLoginModule` uses the following code to initialize the callback array:

Attention: In the following code sample, several lines of code are split into two lines for illustrative purposes only.

```
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(
163.             "HttpServletRequest: ");
            callbacks[5] =
            new com.ibm.wsspi.security.auth.callback.WSAppContextCallback(
164.             "ApplicationContextCallback: ");
            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 login module: a `java.util.Map`, an `HttpServletRequest`, and an `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 web address. 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 are split into two lines for illustrative purposes only.

```

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 login module to perform mapping functions. The error page of the form-based login might be modified by a custom login module. 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 login module. 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 running thread during the request processing. When a Web or an Enterprise JavaBeans (EJB) module is configured to use the caller identity, the user identity is propagated to the downstream service in an EJB request. The `WSCredential` credential 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 cannot be modified. For more information on the `WSSubject` subject, see “Example: Getting the caller subject from the thread” on page 589.

Custom login module development for a system login configuration

For WebSphere Application Server, multiple Java Authentication and Authorization Service (JAAS) plug-in points exist for configuring system logins. WebSphere Application Server uses system login configurations to authenticate incoming requests, outgoing requests, and internal server logins.

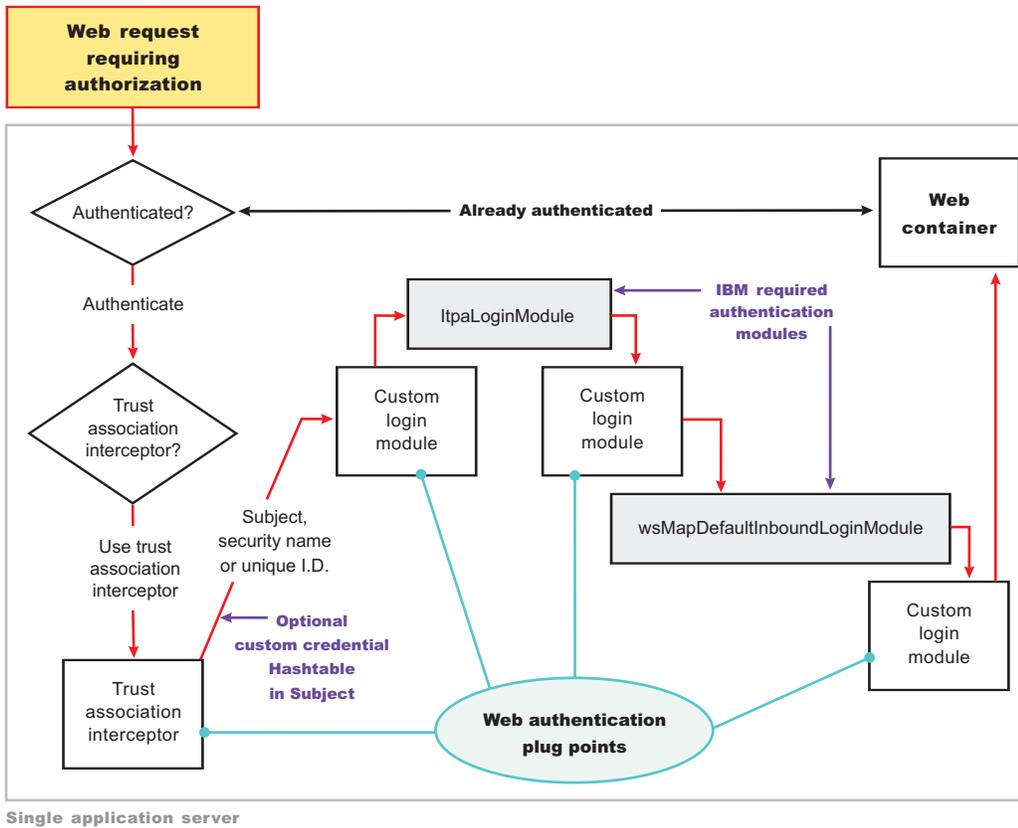
Application login configurations are called by Java 2 Platform, Enterprise Edition (J2EE) applications for obtaining a Subject that is based on specific authentication information. This login configuration enables the application to associate the Subject with a specific protected remote action. The Subject is picked up on the outbound request processing. The following list identifies 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 the login modules combined in Figure 1. Figure 1 shows where you can plug in custom login modules and where the `ltpaLoginModule` and the `wsMapDefaultInboundLoginModule` login modules are required.

Figure 1

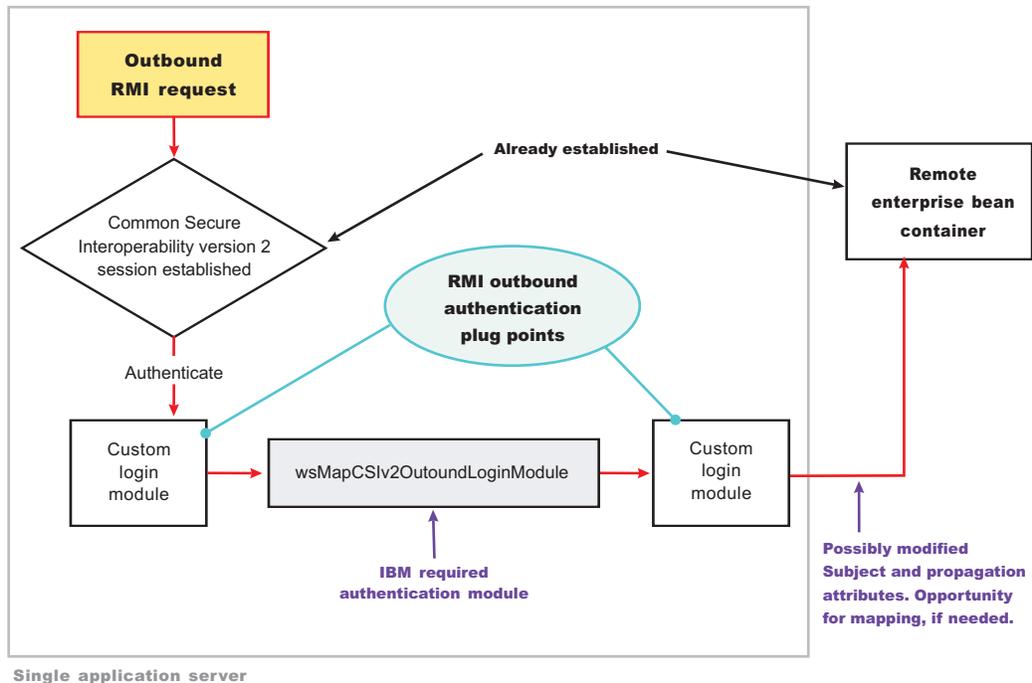


For more detailed information on the WEB_INBOUND configuration including its associated callbacks, see "RMI_INBOUND, WEB_INBOUND, DEFAULT" in "System login configuration entry settings for Java Authentication and Authorization Service" on page 558.

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 invocation Subject passed in and other security context information such as propagation tokens. 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 303. Figure 2 shows where you can plug in custom login modules and shows where the wsMapCSlv2OutboundLoginModule login module is required.

Figure 2

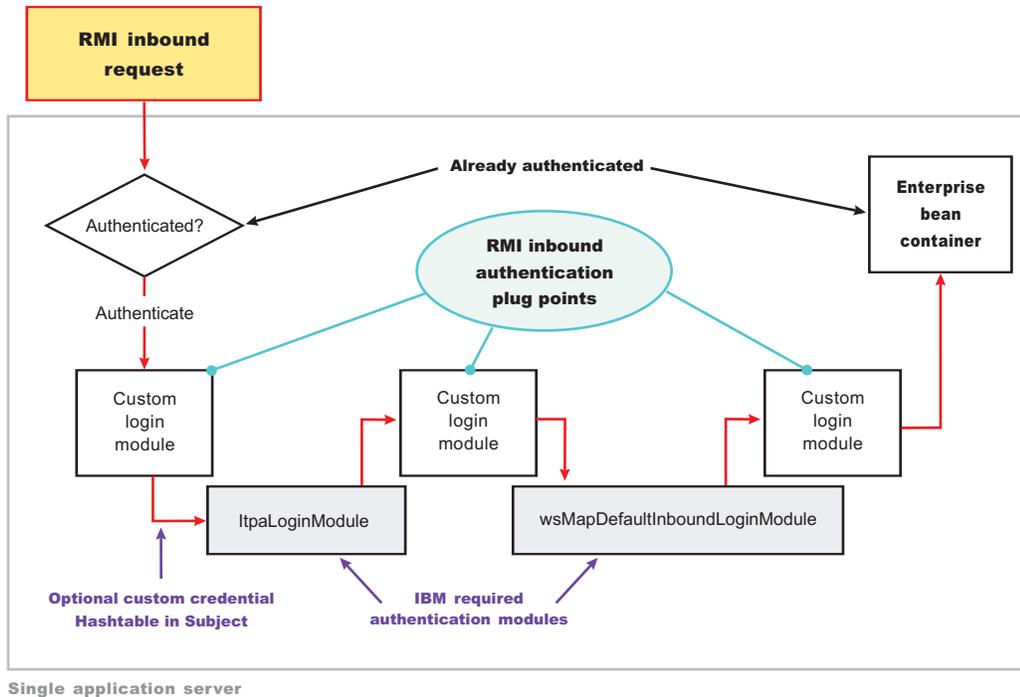


For more information on the RMI_OUTBOUND login configuration, including its associated callbacks, see "RMI_OUTBOUND" in "System login configuration entry settings for Java Authentication and Authorization Service" on page 558.

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 191. During a propagation login, this plug point is used to deserialize the information that is 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 object, which is referenced in figure 3, see "Configuring inbound identity mapping" on page 294. Figure 3 shows where you can plug in custom login modules and shows that the ItpaLoginModule and the wsMapDefaultInboundLoginModule login modules 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 558.

DEFAULT login configuration

The DEFAULT login configuration is a plug point that handles all of the other types of authentication requests, including administrative 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 558.

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, which is 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. 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 is 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 callback 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 callback might not have sufficient attributes for a full login. Check the requiresLogin method on the WSTokenHolderCallback callback 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 list contains the callbacks that might be present in the system login configurations. The list includes the callback name and a description of their responsibility.

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. This callback handler is typically present when a user name and password are 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. This callback handler enables login modules to get information from the HTTP request for use in the login, and is presented from the WEB_INBOUND login configuration only.

callbacks[5] = new

com.ibm.websphere.security.auth.callback.WSServletResponseCallback("HttpServletResponse: ");

This callback handler collects the HTTP servlet response object, if present. This callback handler 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 cookie 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 that is used during the login. This callback handler consists of a HashMap object, which contains the application name and the redirect web address, if present. The callback handler is presented from the WEB_INBOUND login configuration only.

callbacks[7] = new WSRealmNameCallbackImpl("Realm Name: ", default_realm);

This callback handler collects the realm name for the login information. The realm information might not always be provided. If the realm information is not provided, assume that it is the current realm.

callbacks[8] = new WSX509CertificateChainCallback("X509Certificate[]: ");

This callback handler contains the certificate that was validated by Secure Sockets Layer (SSL) if the login source is an X509Certificate from SSL client authentication. The ItpaLoginModule calls the same mapping functions as WebSphere Application Server releases prior to version 6.1.

However, having it passed into the login gives a custom login module the opportunity to map the certificate in a custom way. Then, it performs a Hashtable login. See "Configuring inbound identity mapping" on page 294 for more information on a Hashtable login.

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 a stop 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. Associate a `java.util.Hashtable` with this property. This hashtable contains properties that are 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 294.

If you want to access the objects that WebSphere Application Server creates during a login, refer to the following shared state variables. The variables are set in the following login modules: `ItpaLoginModule`, `swamLoginModule`, and `wsMapDefaultInboundLoginModule`.

Shared state variable

`com.ibm.wsspi.security.auth.callback.Constants.WSPRINCIPAL_KEY`

Purpose

Specifies the `com.ibm.websphere.security.auth.WSPPrincipal` object. See the WebSphere

Application Server API documentation 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.

The 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 API documentation 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 the wsMapDefaultInboundLoginModule login module. The information set here is propagated downstream and is available to the application. See the WebSphere Application Server API documentation 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 callback is present and contains sufficient information from a sending server to recreate all the required objects needed by WebSphere Application Server runtime. In cases where there is sufficient information for the WebSphere Application Server runtime, the information you might add to the Subject is likely to exist from the previous login. To verify if your object is present, you can get access to the ArrayList object that is present in the WSTokenHolderCallback callback, 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 runtime might add additional information at the end of the name to know which Subject is set to add the custom object after deserialization.

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 294.

```
// 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.

```

{
// 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.WSPPrincipal 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 ("HttpServletRequest: ");
callbacks[5] = new com.ibm.wsspi.security.auth.callback.
    WServletResponseCallback ("ApplicationContextCallback: ");
callbacks[6] = new com.ibm.wsspi.security.auth.callback.
    WTokenHolderCallback ("Authz Token List: ");

try
{
    callbackHandler.handle(callbacks);
}
catch (Exception e)
{
    // Handles exceptions
    throw new WLoginFailedException (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 = ((WServletRequestCallback)
    callbacks[3]).getHttpServletRequest();
javax.servlet.http.HttpServletResponse response = ((WServletResponseCallback)
    callbacks[4]).getHttpServletResponse();
java.util.Map appContext = ((WApplicationContextCallback)
    callbacks[5]).getContext();
java.util.List authzTokenList = ((WTokenHolderCallback)
    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 191 // - Add a new custom authorization token (For more information, see // “Security attribute propagation” on page 191 // - 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 = ((WTokenHolderCallback) 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 191. // 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 191) // 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 WSPincipal 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 > Secure administration, applications, and infrastructure**. Under Java Authentication and Authorization Service, click **System logins**. For more information on using the wsadmin utility for system login configuration, see “Customization of a server-side Java Authentication and Authorization Service authentication and login configuration” on page 573. Also refer to the “Customization of a server-side Java Authentication and Authorization Service authentication and login configuration” on page 573 article for information on system login modules and to determine whether to add additional login modules.

Example: Getting the caller subject from the thread

The Caller subject (or “received subject”) contains the user authentication information that is used in the call for this request. This subject is returned after issuing the WSSubject.getCallerSubject application programming interface (API) to prevent replacing existing objects. The subject is marked read-only. This API can be used to get access to the WSCredential credential 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 credential is propagated downstream and a new caller subject is 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 run 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 the WSSubject.getRunAsSubject application programming interface (API) to prevent replacing existing objects. You can use this API to get access to the WSCredential credential, which is documented in the API documentation, 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 credential 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 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 that is 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 running thread.

This extension gives you the flexibility to associate the Subject with all the remote calls on this thread whether you use a `WSSubject.doAs` method to associate the subject with the remote action. 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.

This procedure can be called from another JACL script. The following Java Command Language (JACL) revokes a user when given the realm and the user ID, and cycles through all the security administration

MBean instances that are returned for the entire cell when run from the deployment manager wsadmin command. The command also purges the user from the cache during each process.

Note: When a user is removed from authentication cache, the user can still login to WebSphere Application Server at any time. Removing the cache only removes the user from the runtime cache. It does not remove the user from registry, nor does it lock out the user.

Attention: In some of the following lines of code, the lines are split into two or more lines for illustrative purposes only.

```
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
    }
}
```

Enabling identity assertion with trust validation

By enabling identity assertion with trust validation, an application can use the JAAS login configuration to perform a programmatic identity assertion.

To enable an identity assertion with trust validation, follow these steps:

1. Create a custom login module to perform a trust validation. The login module must set trust and identity information in the shared state, which is then passed on to the IdentityAssertionLoginModule. The trust and identity information is stored in a map in the shared state under the key, `com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.state`. If this key is missing from the shared state, a `WSLoginFailedException` error is thrown by the IdentityAssertionLoginModule module. The custom login module should include the following:
 - A trust key named `com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.trust`. If the trust key is set to `true`, trust is established. If the trust key is set to `false`, the IdentityAssertionLoginModule module creates a `WSLoginFailedException` error.

- The identity of the `java.security.Principal` type set in the `com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.principal` key.
- The identity in the form of a `java.security.cert.X509Certificate[]` certificate set in the `com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule.certificates` key.

Note: If both a principal and a certificate are supplied, the principal is used, and a warning is issued.

2. Create a new Java Authentication and Authorization Service (JAAS) configuration for application logins. It contains the user-implemented trust validation custom login module and the `IdentityAssertionLoginModule` module. To configure an application login configuration from the administrative console, complete the following steps:
 - a. Click **Security > Secure administration, applications, and infrastructure**.
 - b. Under Java Authentication and Authorization Service, click **Application logins > New**.
 - c. Supply the JAAS configuration with an alias, and then click **Apply**.
 - d. Under Additional properties, click **JAAS Login Modules > New**.
 - e. Enter the module class name of the user-implemented trust validation custom login module, and then click **Apply**.
 - f. Enter the `com.ibm.wsspi.security.common.auth.module.IdentityAssertionLoginModule` module class name.
 - g. Make sure that the module class name classes are in the correct order. The user-implemented trust validation login module must be the first class in the list, and the `IdentityAssertionLoginModule` module must be the second class.
 - h. Click **Save**. The new JAAS configuration is used by the application to perform an identity assertion.

An application can now use the JAAS login configuration to perform a programmatic identity assertion. The application can create a login context for the JAAS configuration created in step 2, then login to that login context with the identity it asserts to. If the login is successful, that identity can be set in the current running process, as in the following example:

```
MyCallbackHandler handler = new MyCallbackHandler(new MyPrincipal("Joe"));
LoginContext lc = new LoginContext("MyAppLoginConfig", handler);
lc.login(); //assume successful
Subject s = lc.getSubject();
WSSubject.setRunAsSubject(s);
// From here on, the runas identity is "Joe"
```

Secure transports with JSSE and JCE programming interfaces

This topic provides detailed information about transport security using Java Secure Socket Extension (JSSE) and Java Cryptography Extension (JCE) programming interfaces. Within this topic, there is a description of the IBM version of the Java Cryptography Extension Federal Information Processing Standard (IBMJCEFIPS).

Java Secure Socket Extension

Java Secure Socket Extension (JSSE) provides the transport security for WebSphere Application Server. JSSE provides the 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 5. The API package for JSSE APIs is `javax.net.ssl.*`. Documentation for using JSSE APIs can be found in the J2SE 5 API documentation that is located at <http://java.sun.com/j2se/1.5.0/docs/api/index.html>.

Several JSSE providers ship with the J2SDK Version 5 that comes with WebSphere Application Server. The IBMJSSE provider is used in previous WebSphere Application Server 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 Java Message Service (JMS) and HTTP transports in WebSphere Application Server Version 6.1. These transports take advantage of the J2SDK Verison 5 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.x currently use the IBMJSSE2 provider, but can be switched to the old IBMJSSE provider, if necessary (specified in the SSL repertoire configuration).

For more information on the new IBMJSSE2 provider, please review the documentation located in the <http://www.ibm.com/developerworks/java/jdk/security/142/jsse2docs.zip> file. After it is 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. The following table summarizes which aspects can be customized, what the defaults are, and which mechanisms are used to provide customization. You can customize the following key aspects:

Customizable item	Default	How to customize
X509Certificate	X509Certificate implementation from IBM	The <code>cert.provider.x509v1</code> security property
HTTPS protocol	Implementation from IBM	The <code>java.protocol.handler.pkgs</code> system property
Cryptography Package Provider	IBMJSSE	A <code>security.provider.n=</code> line in security properties file. See description.
Default keystore	None	The <code>* javax.net.ssl.keyStore</code> system property
Default truststore	<code>jssecacerts</code> , if it exists. Otherwise, <code>cacerts</code>	The <code>* javax.net.ssl.trustStore</code> system property
Default key manager factory	<code>IBMJSSE</code>	The <code>ssl.KeyManagerFactory.algorithm</code> security property
Default trust manager factory	<code>IBMJSSE</code>	The <code>ssl.TrustManagerFactory.algorithm</code> 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, which is located in the `app_server_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) that is 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 access the following information:

Version 1.4.2

1. Access the <http://www.ibm.com/developerworks/java/jdk/security/> Web site.
2. Click **Java 1.4.2**.
3. Click **Javadoc HTML documentation** in the Java Secure Socket Extension (JSSE) Guide section.

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. You can access the samples in the following location:

Version 1.4.2

1. Access the <http://www.ibm.com/developerworks/java/jdk/security/> Web site.
2. Click **Java 1.4.2**.
3. Click **jssedocs_samples.zip** in the Java Secure Socket Extension (JSSE) Guide section.

Look 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 <code>com.ibm.net.ssl.www.protocol</code> class. The URL is specified with the <code>http</code> or <code>https</code> prefix. The HTML that is returned from this site is displayed.

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 list is for reference 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 topic for documentation references to JSSE.

JCE

Java Cryptography Extension (JCE) provides cryptographic, key and hash algorithms for WebSphere Application Server. JCE 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, JCERACFKS [z/OS only])
- Key store (JKS, JCEKS, PKCS12, JCERACFKS [z/OS only])

The IBMJCE belongs to the `com.ibm.crypto.provider.*` packages.

For further information, see the information on JCE on the following web site: <http://www.ibm.com/developerworks/java/jdk/security/142/>.

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 that are defined by the JCE. The JCE APIs cover:

- 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 on the <http://www.ibm.com/developerworks/java/jdk/security/> Web site.

Samples using Java Cryptography Extension

There are samples located on the <http://www.ibm.com/developerworks/java/jdk/security/> Web site in the `jceDocs_samples.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 SHA1withDSA 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 for documentation on JCE.

Configuring Federal Information Processing Standard Java Secure Socket Extension files

Use this topic to configure Federal Information Processing Standard Java Secure Socket Extension files.

In WebSphere Application Server, 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 that is shipped in WebSphere Application Server Version 6.1 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

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. When enabling the **Use the United States Federal Information Processing Standard (FIPS) algorithms** option on the server SSL certificate and key management panel, the runtime always uses IBMJSSE2, despite the contextProvider that 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 6.1 because an administrator needs to enable only the **Use the United States Federal Information Processing Standard (FIPS) algorithms** option on the server SSL certificate and key management panel to enable all transports using SSL.

1. Click **Security > SSL certificate and key management**.
2. Select the **Use the United States Federal Information Processing Standard (FIPS) algorithms** option and click **Apply**. This option makes IBMJSSE2 and IBMJCEFIPS the active providers.
3. Accommodate Java clients that must access enterprise beans.

Change the `com.ibm.security.useFIPS` property value from `false` to `true` in the `profile_root/properties/ssl.client.props` file.

4. Ensure that the `java.security` includes the provider.

Edit the `java.security` file to insert the IBMJCEFIPS provider (`com.ibm.crypto.fips.provider.IBMJCEFIPS`) before the IBMJCE provider, and also renumber the other providers in the provider list. The IBMJCEFIPS provider must be in the `java.security` file provider list.

The `java.security` file is located in the `WASHOME/java/jre/lib/security` directory.

The IBM SDK `java.security` file looks like the following example after completing this step:

```
security.provider.1=com.ibm.crypto.fips.provider.IBMJCEFIPS
security.provider.2=com.ibm.crypto.provider.IBMJCE
security.provider.3=com.ibm.jsse.IBMJSSEProvider
security.provider.4=com.ibm.jsse2.IBMJSSEProvider2
security.provider.5=com.ibm.security.jgss.IBMJGSSProvider
security.provider.6=com.ibm.security.cert.IBMCertPath
security.provider.7=com.ibm.crypto.pkcs11.provider.IBMPKCS11
security.provider.8=com.ibm.security.cmskeystore.CMSProvider
security.provider.9=com.ibm.security.jgss.mech.spnego.IBMPNEGO
```

If you are using the Sun JDK, the `java.security` file looks like the following example after completing this step:

```
security.provider.1=sun.security.provider.Sun
security.provider.2=com.ibm.security.jgss.IBMJGSSProvider
security.provider.3=com.ibm.crypto.fips.provider.IBMJCEFIPS
security.provider.4=com.ibm.crypto.provider.IBMJCE
```

```
security.provider.5=com.ibm.jsse.IBMJSSEProvider
security.provider.6=com.ibm.jsse2.IBMJSSEProvider2
security.provider.7=com.ibm.security.cert.IBMCertPath
#security.provider.8=com.ibm.crypto.pkcs11.provider.IBMPKCS11
```

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 might not have 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 line to the *profile_root/properties/soap.client.props* file:
com.ibm.ssl.contextProvider=IBMJSSEFIPS
- When you select the **Use the Federal Information Processing Standard (FIPS)** option on the SSL certificate and key management panel, the Lightweight Third-Party Authentication (LTPA) token format is not backwards-compatible with previous releases of WebSphere Application Server. However, you can import the LTPA keys from a previous version of the application server.

Note: When enabling FIPS, you cannot configure cryptographic token devices in the SSL repertoires. IBMJSSE2 must use IBMJCEFIPS when utilizing cryptographic services for FIPS.

The following FIPS 140-2 approved cryptographic providers that are the only devices that are supported with the FIPS option:

- IBMJCEFIPS (certificate 376)
- IBM Cryptography for C (ICC) (certificate 384)

The relevant certificates are listed on the NIST Web site: Cryptographic Module Validation Program FIPS 140-1 and FIPS 140-2 Pre-validation List

To unconfigure the FIPS provider, reverse the changes that you made in the previous steps. After you reverse the changes, verify that you have made the following changes to the *sas.client.props*, *soap.client.props*, and *java.security* files:

- In the *ssl.client.props* file, you must change the *com.ibm.security.useFIPS* value to false.
- In the *java.security* file, you must change the FIPS provider to a non-FIPS provider.

If you are using the IBM SDK *java.security* file, you must change the first provider to a non-FIPS provider as shown in the following example:

```
#security.provider.1=com.ibm.crypto.fips.provider.IBMJCEFIPS
security.provider.1=com.ibm.crypto.provider.IBMJCE
security.provider.2=com.ibm.jsse.IBMJSSEProvider
security.provider.3=com.ibm.jsse2.IBMJSSEProvider2
security.provider.4=com.ibm.security.jgss.IBMJGSSProvider
security.provider.5=com.ibm.security.cert.IBMCertPath
#security.provider.6=com.ibm.crypto.pkcs11.provider.IBMPKCS11
```

If you are using the Sun JDK *java.security* file, you must change the third provider to a non-FIPS provider as shown in the following example:

```
security.provider.1=sun.security.provider.Sun
security.provider.2=com.ibm.security.jgss.IBMJGSSProvider
security.provider.3=com.ibm.crypto.fips.provider.IBMJCEFIPS
security.provider.4=com.ibm.crypto.provider.IBMJCE
security.provider.5=com.ibm.jsse.IBMJSSEProvider
security.provider.6=com.ibm.jsse2.IBMJSSEProvider2
security.provider.7=com.ibm.security.cert.IBMCertPath
#security.provider.8=com.ibm.crypto.pkcs11.provider.IBMPKCS11
```

Implementing tokens for security attribute propagation

As part of an extensible architecture, WebSphere Application Server enables you to implement your own tokens in which to propagate security attributes.

The following topics are covered in this section:

- Implementing a custom propagation token
- Implementing a custom authorization token
- Implementing a custom a single sign-on token
- Implementing a custom authentication token
- Propagating a custom Java serializable object

Implementing a custom propagation token

This topic explains how you might create your own propagation token implementation, which is set on the running thread and propagated downstream.

The default propagation token 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. Many different methods are available for implementing the `PropagationToken` interface. However, make sure that the methods that are required by the `PropagationToken` interface and the token interface are fully implemented.

After you implement this interface, you can place it in the `app_server_root/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 required permissions for the server code.

Tip: All of the token types that are defined by the propagation framework have similar interfaces. 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 propagation token, might extend the abstract class and then most of the work is complete.

To see an implementation of the propagation token, see “Example: `com.ibm.wsspi.security.token.PropagationToken` implementation” on page 601.

2. Add and receive the custom propagation token 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, to deserialize the information, you need to plug in a custom login module, which is discussed in “Propagating a custom Java serializable object” on page 635. The `WSSecurityPropagationHelper` class has APIs that are used to set a propagation token on the thread and to retrieve the token from the thread to make updates.

The code sample in “Example: Custom propagation token login module” on page 606 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` callback contains propagation data. If the callback does not contain propagation data, initialize a new custom propagation token implementation and set it on the thread. If the callback contains propagation data, look for your specific custom propagation token

TokenHolder instance, convert the byte array back into your custom PropagationToken object, and set it back on the thread. The code sample shows both instances.

You can add attributes any time your custom propagation token is added to the thread. If you add attributes between requests and the getUniqueld method changes, the Common Secure Interoperability Version 2 (CSIv2) client session is invalidated so that it can send the new information downstream. Adding attributes between requests can affect performance. In many cases, you want the downstream requests to receive the new propagation token information.

To add the custom propagation token to the thread, call the WSSecurityPropagationHelper.addPropagationToken token. This call requires the WebSphereRuntimePerMission "setPropagationToken" Java 2 Security permission.

3. Add your custom login module to WebSphere Application Server system login configurations that already contain the com.ibm.ws.security.server.Im.wsMapDefaultInboundLoginModule login module for receiving serialized versions of your custom propagation token. You can also add this login module to any of the application logins where you might want to generate your custom propagation token 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 579

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 propagation token implementation. The following sample code does not extend an abstract class, but 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 considerable differences exist between how you handle the various token implementations.

For information on how to implement a custom propagation token, see "Implementing a custom propagation token" on page 600.

```
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 PropagationToken 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 is checked 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 a 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 propagation token login module

This example 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 579.)
    }

    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 579.)

        // 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 579
    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;
}

```

Implementing a custom authorization token

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()` application programming interface (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 `app_server_root/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 635. After the object is instantiated in the login module, you can add 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 635. 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 614 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.Im.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.Im.wsMapDefaultInboundLoginModule, add this login module after com.ibm.ws.security.server.Im.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 579

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 authorization token” on page 608.

```
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

For information on what to do during initialization, login and commit, see “Custom login module development for a system login configuration” on page 579.

```

public customLoginModule()
{
    public void initialize(Subject subject, CallbackHandler callbackHandler,
        Map sharedState, Map options)
    {
        _sharedState = sharedState;
    }

    public boolean login() throws LoginException
    {
        // 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 principal 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
{
    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;
}

```

Implementing a custom single sign-on token

You can create your own single sign-on token implementation. The single sign-on token implementation 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 single sign-on token 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 single sign-on token when you receive it from a propagation login. 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. 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 single sign-on token, complete the following steps:

1. Write a custom implementation of the `SingleSignonToken` interface.

Many different methods are available for implementing the `SingleSignonToken` interface. However, make sure the methods that are required by the `SingleSignonToken` interface and the token interface are fully implemented.

After you implement this interface, you can place it in the `app_server_root/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 required permissions for the server code.

Tip: All of the token types that are 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 single sign-on token, might extend the abstract class and then most of the work is complete.

To see an implementation of the single sign-on token, see “Example: A `com.ibm.wsspi.security.token.SingleSignonToken` implementation” on page 618

2. Add and receive the custom single sign-on token 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, to deserialize the information, you 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: A custom single sign-on token login module” on page 622, shows how to determine if the login is an initial login or a propagation login. The difference is whether the `WSTokenHolderCallback` callback contains propagation data. If the callback does not contain propagation data, initialize a new custom single sign-on token 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 single sign-on token 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 does not support propagation.

You can make your single sign-on token 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 runtime does not handle cookies that it does not generate, so this cookie is not used by the runtime.
 - 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 that is found in “Example: An HTTP cookie retrieval” on page 624 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.Im.wsspi.DefaultInboundLoginModule` for receiving serialized versions of your custom propagation token. Because this login module relies on information in the `sharedState` state that is added by the `com.ibm.ws.security.server.Im.wsspi.DefaultInboundLoginModule` login module, add this login module after the `com.ibm.ws.security.server.Im.wsspi.DefaultInboundLoginModule` login module.
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 579.

After completing these steps, you have implemented a custom single sign-on token.

Example: A com.ibm.wsspi.security.token.SingleSignonToken implementation

Use this file to see an example of a single sign-on implementation. The following sample code does not extend an abstract class, but 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 considerable differences exist between how you handle the various token implementations.

For information on how to implement a custom single sign-on token, see “Implementing a custom single sign-on token” on page 616.

```
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 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 other token unique IDs that are part of
 * the same Subject.
 *
 * This method should return null if you want the access ID 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 a 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: A custom single sign-on token login module

This file shows how to determine if the login is an initial login or a propagation login.

For information on initialization and on what to do during login and commit, see “Custom login module development for a system login configuration” on page 579.

```

public customLoginModule()
{
    public void initialize(Subject subject, CallbackHandler callbackHandler,

```

```

        Map sharedState, Map options)
    {
        _sharedState = sharedState;
    }

public boolean login() throws LoginException
{
    // 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 sign-on (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
{
    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);
                    }
                }
            });
        }
        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: An HTTP cookie retrieval

The following example shows you 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 what to do during initialization, login and commit, see “Custom login module development for a system login configuration” on page 579.

```

public customLoginModule()
{
    public void initialize(Subject subject, CallbackHandler callbackHandler,
        Map sharedState, Map options)
    {
        _sharedState = sharedState;
    }
}

public boolean login() throws LoginException
{
    // 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())
                    {
                        System.out.println("*** GOT MY CUSTOM SSO TOKEN FROM
                            THE REQUEST ***");
                    }
                }
            }
        }
    }
}

```

```

}

public boolean commit() throws LoginException
{
    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;
}
```

Implementing a custom authentication token

This topic explains how you might create your own authentication token implementation, which is set in the login Subject and propagated downstream.

With this implementation you can 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` application programming interface (API).

Important: Custom authentication token implementations are not used by the security runtime in WebSphere Application Server to enforce authentication. WebSphere Application Security runtime uses this token in the following situations only:

- Call the `getBytes` method for serialization
- Call the `getForwardable` method to determine whether to serialize the authentication token.
- Call the `getUniqueld` method for uniqueness
- Call the `getName` and the `getVersion` methods for adding serialized bytes to the token holder 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. Many different methods are available for implementing the `AuthenticationToken` interface. However, make sure the methods that are 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 the class has the necessary permissions required by the server code.

Tip: All of the token types that are defined by the propagation framework have similar interfaces. 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 authentication token, might extend the abstract class and then most of the work is complete.

To see an implementation of the `AuthenticationToken` interface, see “Example: A `com.ibm.wsspi.security.token.AuthenticationToken` implementation” on page 628.

2. Add and receive the custom authentication token 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, to deserialize the information you must plug in a custom login module. After the object is instantiated in the login module, you can add 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 635. If you want to ensure that the information is propagated, do your own custom serialization, or specify the uniqueness for Subject caching purposes, consider writing your own authentication token implementation.

The code sample in “Example: A custom authentication token login module” on page 633, 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 callback contains propagation data. If the callback does not contain propagation data, initialize a new custom authentication token implementation and set it into the Subject. If the callback contains propagation data, look for your specific custom authentication token TokenHolder instance, convert the byte array back into your custom AuthenticationToken object, and set it back into the Subject. The code sample shows both instances.

You can make your authentication token read-only in the commit phase of the login module. If you do not make the token read-only, 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.Im.wsMapDefaultInboundLoginModule login module for receiving serialized versions of your custom authorization token.

Because this login module relies on information in the shared state that is added by the com.ibm.ws.security.server.Im.wsMapDefaultInboundLoginModule login module, add this login module after the com.ibm.ws.security.server.Im.wsMapDefaultInboundLoginModule login module. 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 579.

After completing these steps, you have implemented a custom authentication token.

Example: A com.ibm.wsspi.security.token.AuthenticationToken implementation

The following example illustrates an authentication token 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 considerable differences exist between how you handle the various token implementations.

```
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 a 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: A custom authentication token login module

This examples shows how to determine if the login is an initial login or a propagation login.

For information on what to do during initialization, login and commit, see “Custom login module development for a system login configuration” on page 579.

```

public customLoginModule()
{
    public void initialize(Subject subject, CallbackHandler callbackHandler,
        Map sharedState, Map options)
    {
        _sharedState = sharedState;
    }

    public boolean login() throws LoginException
    {
        // 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 runtime 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
{
    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

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.

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 sign-on (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 serialized 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. A two-phase process exists 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 on the sending server, deserializes the object on the receiving server, and adds the object back into the Subject downstream. However, some requirements exist for this process to occur successfully. For more information on the JAAS programming model, see the JAAS information provided in *Security: Resources for learning*.

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 permissions. 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 required to run the application programming interfaces (API) that are called from the login module. Because the login module might be run 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 code. For information on what to do during initialization, login and commit, see “Custom login module development for a system login configuration” on page 579.

```
public customLoginModule()
{
    public void initialize(Subject subject, CallbackHandler callbackHandler,
        Map sharedState, Map options)
    {
    }

    public boolean login() throws LoginException
    {
        // 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
{
    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 application programming interface (API)
                    // documentation.
                    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 serialized if you want the object to propagate. For example, the object must implement the `java.io.Serializable` interface. If the object is not serialized, the request does not fail, but the object does not propagate. To make sure an object that is added to the Subject is propagated, implement one of the token interfaces that is defined in “Security attribute propagation” on page 191 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 authorization token” on page 201.

PropagationToken

Add attributes that are specific to an invocation. For more information, see “Default propagation token” on page 195.

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 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 Enterprise JavaBeans (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 579.
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. You must check several places in the administrative console to make sure propagation is fully enabled. For more information, see “Propagating security attributes among application servers” on page 267.
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 > Secure administration, applications, and infrastructure > Custom properties > New**.
 - b. Add `com.ibm.ws.security.propagationExcludeList` in the **Name** field.
 - c. 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 sign-on (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, configure the authentication cache properties. See “Configuring the authentication cache” on page 269. It is recommended that you make your authentication cache timeout value equal to the SSO token timeout.

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 191.

Plug point for custom password encryption

A plug point for custom password encryption can be created to encrypt and decrypt all passwords in WebSphere Application Server that are currently encoded or decoded using Base64-encoding.

The implementation class of this plug point has the responsibility for managing keys, determining the encryption algorithm to use, and for protecting the master secret. The WebSphere Application Server runtime stores the encrypted passwords in their existing locations, preceded with {custom:alias} tags instead of {xor} tags. The custom part of the tag indicates that it is a custom algorithm. The alias part of the tag is specified by the custom implementation, which helps to indicate how the password is encrypted. The implementation can include the key alias, encryption algorithm, encryption mode, or encryption padding.

A custom provider of this plug point must implement an interface that is designed to encrypt and decrypt passwords. The interface is called by the WebSphere Application Server runtime whenever the custom plug point is enabled. The custom algorithm becomes one of the supported algorithms when the plug point is enabled. Other supported algorithms include {xor} (standard base64 encoding) and {os400} which is used on the iSeries platform.

The following example illustrates the com.ibm.wsspi.security.crypto.CustomPasswordEncryption interface:

```
package com.ibm.wsspi.security.crypto;
public interface CustomPasswordEncryption
{
    /**
     * The encrypt operation takes a UTF-8 encoded String in the form of a byte[].
     * The byte[] is generated from String.getBytes("UTF-8").
     * An encrypted byte[] is returned from the implementation in the EncryptedInfo
     * object. Additionally, a logical key alias is returned in the EncryptedInfo
     * object which is passed back into the decrypt method to determine which key was
     * used to encrypt this password. The WebSphere Application Server runtime has
     * no knowledge of the algorithm or the key used to encrypt the data.
     *
     * @param byte[]
     * @return com.ibm.wsspi.security.crypto.EncryptedInfo
     * @throws com.ibm.wsspi.security.crypto.PasswordEncryptException
     */
    public EncryptedInfo encrypt (byte[] decrypted_bytes) throws PasswordEncryptException;

    /**
     * The decrypt operation takes the EncryptedInfo object containing a byte[]
     * and the logical key alias and converts it to the decrypted byte[]. The
     * WebSphere Application Server runtime converts the byte[] to a String
     * using new String (byte[], "UTF-8");
     *
     * @param com.ibm.wsspi.security.crypto.EncryptedInfo
     * @return byte[]
     * @throws com.ibm.wsspi.security.crypto.PasswordDecryptException
     */
    public byte[] decrypt (EncryptedInfo info) throws PasswordDecryptException;

    /**
     * The following is reserved for future use and is currently not
     * called by the WebSphere Application Server runtime.
     *
     * @param java.util.HashMap
     */
    public void initialize (java.util.HashMap initialization_data);
}
```

The `com.ibm.wsspi.security.crypto.EncryptedInfo` class contains the encrypted bytes with the user-defined alias that is associated with the encrypted bytes. This information is passed back into the encryption method to help determine how the password was originally encrypted.

```
package com.ibm.wsspi.security.crypto;
public class EncryptedInfo
{
    private byte[] bytes;
    private String alias;

/**
 * This constructor takes the encrypted bytes and a keyAlias as parameters.
 * This constructor is used to pass to or from the WebSphere Application Server
 * runtime to enable the runtime to associate the bytes with a specific key that
 * is used to encrypt the bytes.
 */

    public EncryptedInfo (byte[] encryptedBytes, String keyAlias)
    {
        bytes = encryptedBytes;
        alias = keyAlias;
    }

/**
 * This command returns the encrypted bytes.
 *
 * @return byte[]
 */
    public byte[] getEncryptedBytes()
    {
        return bytes;
    }

/**
 * This command returns the key alias. The key alias is a logical string that is
 * associated with the encrypted password in the model. The format is
 * {custom:keyAlias}encrypted_password. Typically, just the key alias is placed
 * here, but algorithm information can also be returned.
 *
 * @return String
 */
    public String getKeyAlias()
    {
        return alias;
    }
}
```

The encryption method is called for password processing whenever the custom class is configured and custom encryption is enabled. The decryption method is called whenever the custom class is configured and the password contains the `{custom:alias}` tag. The `custom:alias` tag is stripped prior to decryption. For more information, see [Enabling custom password encryption](#).

Enabling custom password encryption

To view an example code sample that illustrates the `com.ibm.wsspi.security.crypto.CustomPasswordEncryption` interface, see [Plug point for custom password encryption](#).

The encryption method is called for password processing whenever the custom class is configured and custom encryption is enabled. The decryption method is called whenever the custom class is configured and the password contains the `{custom:alias}` tag. The `custom:alias` tag is stripped prior to decryption.

1. To enable custom password encryption, you must configure two properties:

- **property `com.ibm.wsspi.security.crypto.customPasswordEncryptionClass`** - Defines the custom class that implements the `com.ibm.wsspi.security.crypto.CustomPasswordEncryption` password encryption interface.
 - **`com.ibm.wsspi.security.crypto.customPasswordEncryptionEnabled`** - Defines when the custom class is used for default password processing. When the `passwordEncryptionEnabled` option is not specified or set to `false`, and the `passwordEncryptionClass` class is specified, the decryption method is called whenever a `{custom:alias}` tag still exists in the configuration repository.
2. If the custom implementation class defaults to the `com.ibm.wsspi.security.crypto.CustomPasswordEncryptionImpl` interface, and this class is present in the class path, then encryption is enabled by default. This simplifies the enablement process for all nodes. It is not necessary to define any other properties except for those that the custom implementation requires. To disable encryption, but still use this class for decryption, specify the following class.
 - `com.ibm.wsspi.security.crypto.customPasswordEncryptionEnabled=false`
 3. To configure custom password encryption, configure both of these properties in the `security.xml` file. The custom encryption class (`com.acme.myPasswordEncryptionClass`) must be placed in a Java archive (JAR) file in the `#{WAS_INSTALL_ROOT}/classes` directory in all WebSphere Application Server processes. Every configuration document that contains a password (`security.xml` and any application bindings that contain RunAs passwords), must be saved before all of the passwords become encrypted with the custom encryption class.

Whenever a custom encryption class encryption operation is called, and it creates a run-time exception or a defined `PasswordEncryptException` exception, the WebSphere Application Server runtime uses the `{xor}` algorithm to encode the password. This encoding prevents the storage of the password in plain text. After the problem with the custom class has been resolved, it automatically encrypts the password the next time the configuration document is saved.

When a RunAs role is assigned a user ID and password, it currently is encoded using the WebSphere Application Server encoding function. Therefore, after the custom plug point is configured to encrypt the passwords, it encrypts the passwords for the RunAs bindings as well. If the deployed application is moved to a cell that does not have the same encryption keys, or the custom encryption is not yet enabled, a login failure results because the password is not readable.

One of the responsibilities of the custom password encryption implementation is to manage the encryption keys. This class must decrypt any password that it encrypted. Any failure to decrypt a password renders that password to be unusable, and the password must be changed in the configuration. All encryption keys must be available for decryption there no passwords are left using those keys. The master secret must be maintained by the custom password encryption class to protect the encryption keys.

You can manage the master secret by using a stash file for the keystore, or by using a password locator that enables the custom encryption class to locate the password so that it can be locked down.

Chapter 9. Configuring security with scripting

You can configure security with scripting and the wsadmin tool.

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 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 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=
```

Optionally, set the following property:

```
com.ibm.SOAP.loginSource=none
```

The default value for this property is prompt in the `soap.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.

- 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 -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.

Note: 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 Application Server profile.

Enabling and disabling administrative security using scripting

You can use scripting to enable or disable administrative security.

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 administrative security based on LocalOS registry.

- To determine if application security is enabled or disabled by looking at the value of the `appEnabled` field in the WCCM security model, use the **isAppEnabled** command, for example:

- Using Jacl:

```
$AdminTask isAppSecurityEnabled {-interactive}
```

- Using Jython:

```
AdminTask.isAppSecurityEnabled ('[-interactive]')
```

This command returns a value of `true` if `appEnabled` is set to `true`. Otherwise, returns a value of `false`.

- To determine if administrative security is enabled or disabled by looking at the value of the `enabled` field in the WCCM security model, use the **isGlobalSecurity** command, for example:

- Using Jacl:

```
$AdminTask isGlobalSecurity {-interactive}
```

- Using Jython:

```
AdminTask.isGlobalSecurity ('[-interactive]')
```

Returns a value of `true` if `enabled` is set to `true`. Otherwise, returns a value of `false`.

- To set administrative security based on the passed in value, use the **setGlobalSecurity** command. For example:

- Using Jacl:

```
$AdminTask setGlobalSecurity {-interactive}
```

- Using Jython:

```
AdminTask.setGlobalSecurity ('[-interactive]')
```

Returns a value of `true` if the `enabled` field in the WCCM security model is successfully updated. Otherwise, returns a value of `false`.

- 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 administrative 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 administrative security based on the LocalOS registry, use the following procedure call:

- Using Jacl:
securityoff
- Using Jython:
securityoff()

Enabling and disabling LTPA authentication

There are sample scripts located in the <WAS_ROOT>/bin directory on how to enable and disable LTPA authentication. The scripts are:

- LTPA_LDAPSecurityProcs.py (python script)
- LTPA_LDAPSecurityProcs.jacl (jacl script)

Note: The scripts hard code the type of LDAP server and base distinguished name (baseDN). The LDAP server type is hardcoded as IBM_DIRECTORY_SERVER and the baseDN is hardcoded as o=ibm,cn=us.

Enabling and disabling Java 2 security using scripting

You can enable or disable Java 2 security with scripting and the wsadmin tool.

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.

Enabling authentication in the file transfer service using scripting

You can enable authentication in the file transfer service using scripting and the wsadmin tool.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

In WebSphere Application Server Network Deployment, V5.0.1 or later, the file transfer service is enhanced to provide role-based authentication. Two versions of the file transfer Web application are provided. By default, the version that does not authenticate its caller is installed. This default supports compatibility between the WebSphere Application Server Network Deployment, V5.0 and V5.0.1 or later.

Turning the file transfer authentication on is recommended to prevent unauthorized use of the file transfer application; however, if you have any V5.0 clients in your Network Deployment environment, they cannot communicate with the secured file transfer application if global security is turned on.

In WebSphere Application Server V6.x, mixed cells are supported and file transfer has become a system application. If all of the nodes in the cell are of V5.0.1 or later, you can activate authentication in the file transfer service by redeploying the file transfer application at the deployment manager. The compatible version is shipped in the *app_server_root/systemApps/filetransfer.ear* directory. The secured version is provided in the *app_server_root/systemApps/filetransferSecured.ear* directory.

- A wsadmin Jacl script is provided to help you redeploy the file transfer. The script is called *redeployFileTransfer.jacl* and is located in the *app_server_root/bin* directory. After the deployment manager and all the nodes are upgraded to WebSphere Application Server Network Deployment, version 5.0.1 or later, you can deploy the secured file transfer service by running the script. The syntax for running the script from the bin directory is the following:

–

```
wsadmin -profile redeployFileTransfer.jacl -c "fileTransferAuthenticationXxx cellName nodeName serverName"
```

where Xxx is **On** or **Off**.

For Windows systems, use *wsadmin* or *wsadmin.bat*. For Linux and UNIX systems, use *wsadmin.sh*. For OS/400 systems, use *wsadmin*.

- For example, when running the script to enable use of the *filetransferSecured.ear* file, the syntax is similar to the following example:

```
wsadmin -profile redeployFileTransfer.jacl -c "fileTransferAuthenticationOn managedCell managedCellManager dmgr"
```

or

```
wsadmin -profile redeployFileTransfer.jacl -c "fileTransferAuthenticationOn baseCell base server1"
```

- If you want to go return to running the file transfer service without authentication, you can run the script as shown in the following example:

```
wsadmin -profile redeployFileTransfer.jacl -c "fileTransferAuthenticationOff baseNodeCell baseNode server1"
```

or

```
wsadmin -profile redeployFileTransfer.jacl -c "fileTransferAuthenticationOff managedCell managedCellManager dmgr"
```

You must restart the server for the change to take affect.

Propagating security policy of installed applications to a JACC provider using wsadmin scripting

It is possible that you have applications installed prior to enabling the Java Authorization Contract for Containers (JACC)-based authorization. You can start with default authorization and then move to an external provider-based authorization using JACC later.

Also, during application installation or modification you might have had problems propagating the security policy information to the JACC provider. For example, network problems might occur, the JACC provider might not be available, and so on. For these cases, the security policy of the previously installed

applications does not exist in the JACC provider to make the access decisions. One choice is to reinstall the applications involved. However, you can avoid reinstalling by using the wsadmin scripting tool. Use this tool to propagate information to the JACC provider independent of the application installation 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. You can invoke this tool using wsadmin at the base application server for base and deployment manager level for Network Deployment. Note that the SecurityAdmin MBean is available only when the server is running.

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 the *Securing applications and their environment* PDF for more information about these interfaces.

The `appNames` String 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 applications to the Tivoli Access Manager JACC provider, create or import the users and groups that are in the applications to Tivoli Access Manager.
 - Depending on the application or the number of applications that are propagated, you might have to increase the request time-out period either in the `soap.client.props` file in the directory `profile_root/properties` (if using SOAP) or in the `sas.client.props` file (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 the *Securing applications and their environment* PDF 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 connect to the appropriate process (Deployment Manager or
//base application server)
wsadmin -user serverID -password serverPWD
// To get the SecurityAdmin MBean for Deployment Manager
wsadmin> set secadm [${AdminControl} queryNames type=SecurityAdmin,process=dmgr,*]
// or to get the SecurityAdmin MBean for a unmanaged base application server
//(replace the process name to match your configuration)
wsadmin> set secadm [${AdminControl} queryNames
    type=SecurityAdmin,process=server1,*]
// 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 the JACC provider for Tivoli Access Manager using the wsadmin utility

You can use the wsadmin utility to configure Tivoli Access Manager security for WebSphere Application Server.

1. Start WebSphere Application Server.
2. Start the **wsadmin** command-line utility.
Run the **wsadmin** command from the *app_server_root/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.
Tivoli Access Manager Policy Server	Enter the name of the Tivoli Access Manager policy server and the connection port. Use the format, <i>policy_server : port</i> . 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 <i>auth_server : port : priority</i> . 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: <i>auth_server1:7136:1,auth_server2:7137:2</i> . 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 the "Creating the security administrative user" topic in the <i>Securing applications and their environment</i> PDF. For example: <i>cn=wasadmin,o=organization,c=country</i>
Tivoli Access Manager user registry distinguished name suffix	For example: <i>o=organization,c=country</i>
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, <i>sec_master</i> .
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.

Option	Description
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 <i>yes</i> , this option defers the configuration of the management server until the next restart. Set to <i>no</i> , 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 topic in the *Securing applications and their environment* PDF.

Disabling embedded Tivoli Access Manager client using wsadmin

Follow these steps to unconfigure the Java Authorization Contract for Containers (JACC) provider for Tivoli Access Manager.

1. Start the wsadmin command-line utility. The wsadmin command is found in the *install_dir/bin* directory
2. 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 an asterisk (*) to select all nodes.
Tivoli Access Manager administrator user name	Enter the Tivoli Access Manager administration user ID, as created at the time of Tivoli Access Manager configuration. This name is usually, <i>sec_master</i> .
Tivoli Access Manager administrator user password	Enter the password for the Tivoli Access Manager administrator.
Force	Enter <i>yes</i> , if you want to ignore errors when unconfiguring the JACC provider for Tivoli Access Manager. Enter this option as <i>yes</i> only when the Tivoli Access Manager domain is in an irreparable state.
Defer	Enter <i>no</i> , to force the unconfiguration of the connected server. Enter <i>No</i> for the unconfiguration to proceed correctly.

3. When all information is entered, enter **F** to save the properties or **C** to cancel from the unconfiguration process and discard the entered information.
4. Restart all WebSphere Application Server instances for the changes to take effect.

Creating an SSL configuration at the node scope using scripting

An Secure Socket Layer (SSL) configuration references many other configuration objects. To help you make valid selections for the new SSL configuration before you create it, view information about existing configuration objects. Information about existing objects is also useful when you create a node scoped SSL configuration using the **createSSLConfig** command of the AdminTask object.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

To use the information in this task effectively, familiarize yourself with the instructions in the Creating a Secure Sockets Layer configuration topic in the *Securing applications and their environment* PDF. Perform the following task to create an Secure Socket Layer (SSL) configuration at the node scope:

1. List the existing configuration objects. Perform any of the following:
 - List some of the configuration objects that you may need when you create a new SSL configuration. For example, you want to see which management scopes have already been defined. If the one you need does not exist you will need to create it.

– Using Jacl:

```
$AdminTask listManagementScopes {-scopeName (cell):BIRKT40Cell102:(node):BIRKT40Node02}
```

– Using Jython:

```
AdminTask.listManagementScopes ('[-scopeName (cell):BIRKT40Cell102:(node):BIRKT40Node02]')
```

This shows an existing cell scope and existing node scope that you can use. If you want to create a different scope, use the **createManagementScope** command of the AdminTask object to define a different one. The valid scope parameters are cell, nodegroup, node, server, cluster, and endpoint. See the Central management of Secure Sockets Layer configurations topic in the *Securing applications and their environment* PDF for more information on scope definitions.

- List the key stores that exist in the configuration including key stores and trust stores.

– Using Jacl:

```
$AdminTask listKeyStores
```

– Using Jython:

```
AdminTask.listKeyStores()
```

Example output:

```
CellDefaultKeyStore(cells/BIRKT40Cell102|security.xml#KeyStore_1)
CellDefaultTrustStore(cells/BIRKT40Cell102|security.xml#KeyStore_2)
CellLTPAKeys(cells/BIRKT40Cell102|security.xml#KeyStore_3)
```

The previous example only lists the key stores for the default management scope which is also known as the cell scope. To obtain key stores for other scopes, specify the scopeName parameter, for example:

– Using Jacl:

```
$AdminTask listKeyStores {-scopeName (cell):BIRKT40Cell102:(node):BIRKT40Node02 }
```

– Using Jython:

```
$AdminTask listKeyStores ('[-scopeName (cell):BIRKT40Cell102:(node):BIRKT40Node02]')
```

Example output:

```
CellDefaultKeyStore(cells/BIRKT40Cell102|security.xml#KeyStore_1)
CellDefaultTrustStore(cells/BIRKT40Cell102|security.xml#KeyStore_2)
CellLTPAKeys(cells/BIRKT40Cell102|security.xml#KeyStore_3)
NodeDefaultKeyStore(cells/BIRKT40Cell102|security.xml#KeyStore_1134610924357)
NodeDefaultTrustStore(cells/BIRKT40Cell102|security.xml#KeyStore_1134610924377)
```

- List specific trust or key managers. Be sure to display the object name for the trust managers. You will need the object name for the SSL configuration because you can specify multiple trust manager instances.

- Using Jacl:


```
$AdminTask listTrustManagers {-scopeName (cell):BIRKT40Cell02:(node):BIRKT40Node02 -displayObjectName true }
```
- Using Jython:


```
AdminTask.listTrustManagers ('[-scopeName (cell):BIRKT40Cell02:(node):BIRKT40Node02 -displayObjectName true]')
```

Example output:

```

IbmX509(cells/BIRKT40Cell02|security.xml#TrustManager_1)
IbmPKIX(cells/BIRKT40Cell02|security.xml#TrustManager_2)
IbmX509(cells/BIRKT40Cell02|security.xml#TrustManager_1134610924357)
IbmPKIX(cells/BIRKT40Cell02|security.xml#TrustManager_1134610924377)

```

2. Create the node-scoped SSL configuration in interactive mode. Now that we have the information we need to choose from, we need to decide if these objects are sufficient or if we need to create new ones. For now, we will reuse what we've already got in the configuration and save creating new instances to task documents specific to those objects.

- Using Jacl:


```
$AdminTask createSSLConfig -interactive
```
- Using Jython:


```
AdminTask.createSSLConfig ('[-interactive]')
```

Example output:

Create a SSL Configuration.

```

*SSL Configuration Alias (alias): BIRKT40Node02SSLConfig
Management Scope Name (scopeName): (cell):BIRKT40Cell02:(node):BIRKT40Node02
Client Key Alias (clientKeyAlias): default
Server Key Alias (serverKeyAlias): default
SSL Type (type): [JSSE]
Client Authentication (clientAuthentication): [false]
Security Level of the SSL Configuration (securityLevel): [HIGH]
Enabled Ciphers SSL Configuration (enabledCiphers):
JSSE Provider (jsseProvider): [IBMJSSE2]
Client Authentication Support (clientAuthenticationSupported): [false]
SSL Protocol (sslProtocol): [SSL_TLS]
Trust Manager Object Names (trustManagerObjectNames): (cells/BIRKT40Cell02|security.xml#TrustManager_1)
*Trust Store Name (trustStoreName): NodeDefaultTrustStore
Trust Store Scope (trustStoreScopeName): (cell):BIRKT40Cell02:(node):BIRKT40Node02
*Key Store Name (keyStoreName): NodeDefaultKeyStore
Key Store Scope Name (keyStoreScopeName): (cell):BIRKT40Cell02:(node):BIRKT40Node02
Key Manager Name (keyManagerName): IbmX509
Key Manager Scope Name (keyManagerScopeName): (cell):BIRKT40Cell02:(node):BIRKT40Node02

```

Create SSL Configuration

F (Finish)
C (Cancel)

```

Select [F, C]: [F] F
WASX7278I: Generated command line: $AdminTask createSSLConfig {-alias BIRKT40Node02SSLConfig -scopeName
(cell):BIRKT40Cell02:(node):BIRKT40Node02 -clientKeyAlias default -serverKeyAlias default
-trustManagerObjectNames (cells/BIRKT40Cell02|security.xml#TrustManager_1) -trustStoreName
NodeDefaultTrustStore -trustStoreScopeName (cell):BIRKT40Cell02:(node):BIRKT40Node02 -keyStoreName
NodeDefaultKeyStore -keyStoreScopeName (cell):BIRKT40Cell02:(node):BIRKT40Node02 -keyManagerName
IbmX509 -keyManagerScopeName (cell):BIRKT40Cell02:(node):BIRKT40Node02 }

```

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.

The name of the SSL configuration object that you created, for example, (cells/BIRKT40Cell02|security.xml#SSLConfig_1136652770753), appears in the security.xml file.

Example security.xml file output:

```
<repertoire xmi:id="SSLConfig_1136652770753" alias="BIRKT40Node02SSLConfig" type="JSSE"
managementScope="ManagementScope_1134610924357">
<setting xmi:id="SecureSocketLayer_1136652770924" clientKeyAlias="default" serverKeyAlias="default"
clientAuthentication="false" securityLevel="HIGH" jsseProvider="IBMJSSE2" sslProtocol="SSL_TLS"
keyStore="KeyStore_1134610924357" trustStore="KeyStore_1134610924377" trustManager="TrustManager_1"
keyManager="KeyManager_1134610924357"/>
</repertoire>
```

Once you create the SSL configuration object, the next step is to use it. There are several different ways that you can associate SSL configurations with protocols, for example:

- Set the SSL configuration on the thread programmatically.
- Associate the SSL configuration with an outbound protocol or a target host and port.
- Directly associating the SSL configuration using the alias.
- Centrally managing the SSL configurations by associating them with SSL configuration groups or zones so that they are used based upon the group from where the end point exists.

Creating self-signed certificates using scripting

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

You can create self-signed certificates using the scripting and the AdminTask object. You can run the commands in interactive or batch mode. Interactive mode provides a way to discover the flags that you need to run the task in batch mode.

Certificates reside inside of key stores. To run the commands, you will need the name of the key store to be supplied. Use the **listKeyStore** command of the AdminTask object to get a list of key stores. If you need a new key store, use the **createKeyStore** command of the AdminTask object.

To create a personal key store, use the following examples:

- Interactive mode:
 - Using Jacl:

```
$AdminTask createSelfSignedCertificate -interactive
```
 - Using Jython:

```
AdminTask.createSelfSignedCertificate ('[-interactive]')
```

Example output:

```
*Key Store Name (keyStoreName): keyStore
Key Store Scope Name (keyStoreScope):
*Certificate Alias (certificateAlias): newCert
"Certificate Version" (certificateVersion): 3
*Key Size (certificateSize): [1024]
*Common Name (certificateCommonName): localhost
*Organization (certificateOrganization): workgroup
Organizational Unit (certificateOrganizationalUnit): testing
certLocality (certificateLocality): austin
State (certificateState): Texas
Zip (certificateZip): 78757
Country (certificateCountry): [US]
Validity Period (certificateValidDays): [365]
Create Self-Signed Certificate
```

```
F (Finish)
C (Cancel)
```

```
Select [F, C]: [F]
```

```
WASX7278I: Generated command line: $AdminTask createSelfSignedCertificate
{-keyStoreName keyStore -certificateAlias newCert -certificateVersion 3
-certificateCommonName localhost -certificateOrganization ibm
-certificateOrganizationalUnit testing -certificateLocality austin
-certificateState Texas -certificateZip 78757 }
true
```

At the end of the output, the batch mode parameters are provided.

- Batch mode:

- Using Jacl:

```
$AdminTask createSelfSignedCertificate {-keyStoreName keyStore
-certificateAlias newCert -certificateVersion 3 -certificateCommonName localhost
-certificateOrganization ibm -certificateOrganizationalUnit testing
-certificateLocality austin -certificateState Texas -certificateZip 78757 }
```

- Using Jython:

```
AdminTask.createSelfSignedCertificate ('[-keyStoreName keyStore
-certificateAlias newCert -certificateVersion 3 -certificateCommonName localhost
-certificateOrganization ibm -certificateOrganizationalUnit testing
-certificateLocality austin -certificateState Texas -certificateZip 78757]')
```

Automating SSL configurations using scripting

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

SSL configuration is needed for WebSphere to perform SSL connections with other servers. A SSL configuration can be configured through the Admin Console. But if an automated way to create a SSL configuration is desired then AdminTask should be used.

AdminTask can be used in a interactive mode and batch mode. For automation the batch mode options should be used. AdminTask batch mode can be called in a JACL or Python script. Interactive mode will step through all the parameter the task needs, requires ones are marked with a '*'. Before the interactive task executes the task it echoes the batch mode syntax of the task to the screen. This can be helpful when writing batch mode scripts.

There attributes needed to create an ssl configurations:

- A key store
- Default client certificate alias
- Default server certificate alias
- Trust store
- The handshake protocol
- The ciphers needed during handshake
- Supporting client authentication or not

If automating the creation of a SSL Configuration it may be needed to create some of the attribute values needed like the key store, trust store, key manager, and trust managers.

- To create a SSL configuration the createSSLConfig AdminTask can be used. To make changes to the SSL configurations use the modifySSLConfig AdminTask.

- Interactive mode:

Interactive mode steps you through all attributes and tell you the default value of the attribute if there is one. The default value is in '['] on the prompt line. The actual flag used in batch mode is in '(')' on each prompt line. If you are using the default value then the flag will not show up on the batch command line.

Using Jacl:

```
$AdminTask createSSLConfig -interactive
```

– Using Jython:

```
AdminTask.createSSLConfig ('[interactive]')
```

Example output:

```
*SSL Configuration Alias (alias): testSSLConfig
Management Scope Name (scopeName): (cell):HOSTNode01Cell:(node):HOSTNode01
Client Key Alias (clientKeyAlias): clientCert
Server Key Alias (serverKeyAlias): serverCert
SSL Type (type): [JSSE]
Client Authentication (clientAuthentication): [false]
Security Level of the SSL Configuration (securityLevel): [HIGH] HIGH
Enabled Ciphers SSL Configuration (enabledCiphers):
JSE Provider (jsseProvider): [IBMJSSE2]
Client Authentication Support (clientAuthenticationSupported): [false]
SSL Protocol (sslProtocol): [SSL_TLS] SSL_TLS
Trust Manager Object Names (trustManagerObjectNames):
*Trust Store Name (trustStoreName): testTrustStore
Trust Store Scope (trustStoreScopeName): (cell):HOSTNode01Cell:(node):HOSTNode01
*Key Store Name (keyStoreName): testKeyStore
Key Store Scope Name (keyStoreScopeName): (cell):HOSTNode01Cell:(node):HOSTNode01
Key Manager Name (keyManagerName): IbmX509
Key Manager Scope Name (keyManagerScopeName): (cell):HOSTNode01Cell:(node):HOSTNode01
```

Create SSL Configuration

F (Finish)

C (Cancel)

Select [F, C]: [F]

```
WASX7278I: Generated command line: $AdminTask createSSLConfig {-alias testSSLConfig
-scopeName (cell):HOSTNode01Cell:(node):HOSTNode01 -clientKeyAlias clientCert
-serverKeyAlias serverCert -trustStoreName testTrustStore
-trustStoreScopeName (cell):HOSTNode01Cell:(node):HOSTNode01
-keyStoreName testKeyStore -keyStoreScopeName (cell):HOSTNode01Cell:(node):HOSTNode01
-keyManagerName IbmX509 -keyManagerScopeName (cell):HOSTNode01Cell:(node):HOSTNode01 }
(cells/HOSTNode01Cell|security.xml#SSLConfig_1137687301834)
```

At the end of the output, the batch mode parameters are provided.

– Batch mode:

Using Jacl:

```
$AdminTask createSSLConfig {-alias testSSLConfig
-scopeName (cell):HOSTNode01Cell:(node):HOSTNode01 -clientKeyAlias clientCert
-serverKeyAlias serverCert -trustStoreName testTrustStore
-trustStoreScopeName (cell):HOSTNode01Cell:(node):HOSTNode01
-keyStoreName testKeyStore -keyStoreScopeName (cell):HOSTNode01Cell:(node):HOSTNode01
-keyManagerName IbmX509 -keyManagerScopeName (cell):HOSTNode01Cell:(node):HOSTNode01}
```

– Using Jython:

```
AdminTask.createSSLConfig ('[-alias testSSLConfig
-scopeName (cell):HOSTNode01Cell:(node):HOSTNode01 -clientKeyAlias clientCert
-serverKeyAlias serverCert -trustStoreName testTrustStore
-trustStoreScopeName (cell):HOSTNode01Cell:(node):HOSTNode01
-keyStoreName testKeyStore -keyStoreScopeName (cell):HOSTNode01Cell:(node):HOSTNode01
-keyManagerName IbmX509 -keyManagerScopeName (cell):HOSTNode01Cell:(node):HOSTNode01]')
```

Example output:

```
(cells/HOSTNode01Cell|security.xml#SSLConfig_1137687301834)
```

- **Key Stores and Trust Stores** The key store and trust store may already exist or a new one may need to be created. To create a new key store or trust store use the `createKeyStore` AdminTask. It will create a key store file and store the configuration object in the system configuration. A trust store is just a key store that usually only has signer certificates in it. To create a key store enter:

– Using Jacl:

```
$AdminTask createKeyStore {-keyStoreName testKeyStore -keyStoreType PKCS12
-keyStoreLocation $(USER_INSTALL_ROOT)\testKeyStore.p12 -keyStorePassword abcd
-keyStorePasswordVerify abcd -keyStoreIsFileBased true -keyStoreReadOnly false}
```

– Using Jython:

```
AdminTask.createKeyStore ('[-keyStoreName testKeyStore -keyStoreType PKCS12
-keyStoreLocation $(USER_INSTALL_ROOT)\testKeyStore.p12 -keyStorePassword abcd
-keyStorePasswordVerify abcd -keyStoreIsFileBased true -keyStoreReadOnly false]')
```

To populate the key store with certificates see “Managing Certificates using AdminConsole and Admin Task” The key store and trust store are required to create a SSL configuration. Use the ‘-keyStoreName’ and ‘-trustStoreName’ flags on the createSSLConfig. There scopes can be added with the ‘-keyStoreScope’ flag and ‘-trustStoreScope’ flags.

- Key Manager Key manager are used to determine how a certificate is selected. The IbmX509 key manager is in the security configuration by default. If a different key manager is needed then use createKeyManager AdminTask to create it. To create a key manager enter:

– Using Jacl:

```
$AdminTask createKeyManager {-name testKeyManager
-scopeName (cell):HOSTNode01Cell:(node):HOSTNode01
-provider IBMJSSE2 -algorithm specialAlgorithm }
```

– Using Jython:

```
AdminTask.createKeyManager ('[-name testKeyManager
-scopeName (cell):HOSTNode01Cell:(node):HOSTNode01
-provider IBMJSSE2 -algorithm specialAlgorithm]')
```

To supply a key manager on the createSSLConfig AdminTask use the ‘-keyManagerName’ along with the ‘-keyManagerScope’ flag.

- Trust Manager Trust managers are use to determine how trust is established during ssl communication. The IbmX509 and IbmPKIX are trust managers are in the security configuration by default. If a different or additional trust manager is needed then use the createTrustManger AdminTask to create it. To create a trust manager enter:

– Using Jacl:

```
$AdminTask createTrustManager {-name testTrustManager
-scopeName (cell):HOSTNode01Cell:(node):HOSTNode01
-provider IBMJSSE2 -algorithm specialAlgorithm }
```

– Using Jython:

```
AdminTask.createTrustManager ('[-name testTrustManager
-scopeName (cell):HOSTNode01Cell:(node):HOSTNode01
-provider IBMJSSE2 -algorithm specialAlgorithm]')
```

The SSL Configuration can have multiple trust managers. To supply multiple trust managers give a comma separated list of the trust managers configuration IDs with the -trustManagerObjectNames flag. When you create a trust manager the configuration object ID is returned. To get a list of trust managers object IDs use the **listTrustManagers** command of the AdminTask object with the -displayObjectName true flag. For example:

```
wsadmin>$AdminTask listTrustManagers -interactive
List Trust Managers
```

List trust managers.

```
Management Scope Name (scopeName):
Display list in ObjectName Format (displayObjectName): [false] true
```

List Trust Managers

```
F (Finish)
C (Cancel)
```

```
Select [F, C]: [F]
Inside generate script command
```

```
WASX7278I: Generated command line: $AdminTask listTrustManagers {-displayObjectName true }
IbmX509(cells/IBM-0AF8DABCF16Node01Cell|security.xml#TrustManager_IBM-0AF8DABCF16Node01_1)
IbmPKIX(cells/IBM-0AF8DABCF16Node01Cell|security.xml#TrustManager_IBM-0AF8DABCF16Node01_2)
```

Updating default key store passwords using scripting

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

When you install WebSphere Application Server, each server creates a key store and trust store for the default SSL configuration with the default password WebAS. To protect the security of the key store files and the SSL configuration, you must change the password. The following examples update the default password:

- Change multiple key stores passwords. The **changeMultipleKeyStorePasswords** command updates all of the key stores that have the same password. For example:

- Using Jacl:

```
$AdminTask changeMultipleKeyStorePasswords {-keyStorePassword WebAS
-newKeyStorePassword secretPwd -newKeyStorePasswordVerify secretPwd}
```

- Using Jython:

```
AdminTask.changeMultipleKeyStorePasswords ['-keyStorePassword WebAS
-newKeyStorePassword secretPwd -newKeyStorePasswordVerify secretPwd']
```

- Change the password of a single key store. The **changeKeyStorePassword** command updates the password of an individual key store. For example:

- Using Jacl:

```
$AdminTask changeKeyStorePassword {-keyStoreName testKS
-keyStoreScope (cell):localhost:(server):server1
-keyStorePassword WebAS -newKeyStorePassword secretPwd
-newKeyStorePasswordVerify secretPwd}
```

- Using Jython:

```
AdminTask.changeKeyStorePassword ('[-keyStoreName testKS
-keyStoreScope (cell):localhost:(server):server1
-keyStorePassword WebAS -newKeyStorePassword secretPwd
-newKeyStorePasswordVerify secretPwd]')
```

Commands for the IdMgrConfig group of the AdminTask object

Use the commands in the IdMgrConfig group to configure the virtual member manager. The commands for this group do not require a target object. To see the additional commands related to the virtual member manager, see the Commands for the IdMgrRepositoryConfig group of the AdminTask object and the Commands for the IdMgrRealmConfig group of the AdminTask object articles.

For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the IdMgrConfig group of the AdminTask object:

Table 16.

Command name:	Description:	Parameters and return values:	Examples:
<p>createIdMgr Supported EntityType</p>	<p>The createIdMgr Supported EntityType command creates a supported entity type configuration. To validate the result, check for duplicate entity type names.</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - name The name of the supported entity type. (String, required) - defaultParent The default parent node for the supported entity type. (String, required) - rdnProperties The RDN attribute name for the supported entity type in the entity domain name. Separate the RDN properties with semicolons (;). (String, required) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask createIdMgrSupported EntityType {-name <i>entity1</i> -defaultParent <i>node1</i>}</pre> • Using Jython string: <pre>AdminTask.createIdMgrSupported EntityType ('[-name <i>entity1</i> -defaultParent <i>node1</i>']')</pre> • Using Jython list: <pre>AdminTask.createIdMgrSupported EntityType (['-name', '<i>entity1</i>', '-defaultParent', '<i>node1</i>'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask createIdMgrSupported EntityType {-interactive}</pre> • Using Jython string: <pre>AdminTask.createIdMgrSupported EntityType ('[-interactive]')</pre> • Using Jython list: <pre>AdminTask.createIdMgrSupported EntityType (['-interactive'])</pre>

Table 16. (continued)

Command name:	Description:	Parameters and return values:	Examples:
<p>deleteIdMgr Supported EntityType</p>	<p>The deleteIdMgr Supported EntityType command deletes the supported entity type configuration that you specify.</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - name The name of the supported entity type. The value of this parameter must be one of the supported entity types. (String, required) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask deleteIdMgrSupported EntityType {-name <i>entity1</i>} • Using Jython string: AdminTask.deleteIdMgrSupported EntityType ('[-name <i>entity1</i>']') • Using Jython list: AdminTask.deleteIdMgrSupportedEntityType (['-name', '<i>entity1</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask deleteIdMgrSupportedEntityType {-interactive} • Using Jython string: AdminTask.deleteIdMgrSupportedEntityType ('[-interactive]') • Using Jython list: AdminTask.deleteIdMgrSupportedEntityType (['-interactive'])
<p>getIdMgr Supported EntityType</p>	<p>The getIdMgr Supported EntityType command returns the configuration of the supported entity type that you specify.</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - name The name of the supported entity type. The value of this parameter must be one of the supported entity types. (String, required) • Returns: A hash map that has the parameters of the createIdMgrSupportedEntityType command as keys. Since the rdnProperties parameter has multiple values, its values are returned in a list. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask getIdMgrSupported EntityType {-name <i>entity1</i>} • Using Jython string: AdminTask.getIdMgrSupported EntityType ('[-name <i>entity1</i>']') • Using Jython list: AdminTask.getIdMgrSupportedEntityType (['-name', '<i>entity1</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask getIdMgrSupportedEntityType {-interactive} • Using Jython string: AdminTask.getIdMgrSupportedEntityType ('[-interactive]') • Using Jython list: AdminTask.getIdMgrSupportedEntityType (['-interactive'])

Table 16. (continued)

Command name:	Description:	Parameters and return values:	Examples:
listIdMgr Supported EntityTypes	The listIdMgr Supported EntityTypes command lists all of the supported entity types that are configured.	<ul style="list-style-type: none"> • Parameters: None • Returns: A list that contains the names of the supported entity types. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask listIdMgr SupportedEntityTypes • Using Jython string: AdminTask.listIdMgr SupportedEntityTypes() • Using Jython list: AdminTask.listIdMgr SupportedEntityTypes() <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask listIdMgrSupported EntityTypes {-interactive} • Using Jython string: AdminTask.listIdMgrSupported EntityTypes (['-interactive']) • Using Jython list: AdminTask.listIdMgrSupported EntityTypes (['-interactive'])
resetId MgrConfig	The resetId MgrConfig command resets the current configuration to the last configuration that was saved.	<ul style="list-style-type: none"> • Parameters: None • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask resetIdMgrConfig • Using Jython string: AdminTask.resetIdMgrConfig() • Using Jython list: AdminTask.resetIdMgrConfig() <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask resetIdMgrConfig {-interactive} • Using Jython string: AdminTask.resetIdMgrConfig (['-interactive']) • Using Jython list: AdminTask.resetIdMgrConfig (['-interactive'])

Table 16. (continued)

Command name:	Description:	Parameters and return values:	Examples:
showId MgrConfig	The showId MgrConfig command returns the current configuration XML in string format.	<ul style="list-style-type: none"> • Parameters: None • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask showIdMgrConfig • Using Jython string: AdminTask.showIdMgrConfig() • Using Jython list: AdminTask.showIdMgrConfig() <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask showIdMgrConfig {-interactive} • Using Jython string: AdminTask.showIdMgrConfig (['-interactive']) • Using Jython list: AdminTask.showIdMgrConfig (['-interactive'])
updateIdMgr Supported EntityType	The updateIdMgr Supported EntityType command updates the configuration that you specify for a supported entity type.	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - name The name of the supported entity type. The value of this parameter must be one of the supported entity types. (String, required) - defaultParent The default parent node for the supported entity type. (String, optional) - rdnProperties The RDN attribute name for the supported entity type in the entity domain name. To reset all the values of the rdnProperties parameter, specify a blank string (""). (String, optional) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask updateIdMgrSupported EntityType {-name <i>entity1</i>} • Using Jython string: AdminTask.updateIdMgrSupported EntityType ('[-name <i>entity1</i>']) • Using Jython list: AdminTask.updateIdMgrSupported EntityType (['-name', '<i>entity1</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask updateIdMgrSupported EntityType {-interactive} • Using Jython string: AdminTask.updateIdMgrSupported EntityType ('[-interactive]') • Using Jython list: AdminTask.updateIdMgrSupported EntityType (['-interactive'])

Commands for the IdMgrRepositoryConfig group of the AdminTask object

Use the commands in the IdMgrRepositoryConfig to configure the virtual member manager. The commands for this group do not require a target object. To see the additional commands related to the virtual member manager, see the “Commands for the IdMgrConfig group of the AdminTask object” on page 656 and the “Commands for the IdMgrRealmConfig group of the AdminTask object” on page 750 articles.

For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the IdMgrRepositoryConfig group of the AdminTask object:

Table 17.

Command name:	Description:	Parameters and return values:	Examples:
addIdMgrLDAPBackupServer	The addIdMgrLDAPBackupServer command adds or updates backup LDAP servers.	<ul style="list-style-type: none"> Parameters: - id The ID of the repository. (String, required) - primary_host The host name for the primary LDAP server. (String, required) - host The host name for the LDAP server. (String, required) - port The port number for the LDAP server. (Integer, optional) Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: \$AdminTask addIdMgrLDAPBackupServer {-id <i>id1</i> -primary_host <i>myprimaryhost</i> -host <i>myhost.ibm.com</i>} Using Jython string: AdminTask.addIdMgrLDAPBackupServer ('[-id <i>id1</i> -primary_host <i>myprimaryhost</i> -host <i>myhost.ibm.com</i>']') Using Jython list: AdminTask.addIdMgrLDAPBackupServer (['-id', '<i>id1</i>', '-primary_host', '<i>myprimaryhost</i>', '-host', '<i>myhost.ibm.com</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: \$AdminTask addIdMgrLDAPBackupServer {-interactive} Using Jython string: AdminTask.addIdMgrLDAPBackupServer ('[-interactive]') Using Jython list: AdminTask.addIdMgrLDAPBackupServer (['-interactive'])

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
<p>addIdMgr LDAPEntity Type</p>	<p>The addIdMgr LDAPEntity Type command adds an LDAP entity type definition.</p>	<ul style="list-style-type: none"> • Parameters: - id The ID of the repository. (String, required) - name The name of the entity type. (String, required) - searchFilter The search filter that you want to use to search the entity type. (String, optional) - objectClasses One or more object classes for the entity type. (String, required) - objectClassesForCreate The object class to use when an entity type is created. If the value of this parameter is the same as the objectClass parameter, you do not need to specify this parameter. (String, optional) - searchBases The search base or bases to use while searching the entity type. (String, optional) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <code>\$AdminTask addIdMgrLDAPEntity Type {-id <i>id1</i> -name <i>name1</i> -objectClasses <i>objectclass</i>}</code> • Using Jython string: <code>AdminTask.addIdMgrLDAPEntity Type ('[-id <i>id1</i> -name <i>name1</i> -objectClasses <i>objectclass</i>']')</code> • Using Jython list: <code>AdminTask.addIdMgrLDAPEntity Type (['-id', '<i>id1</i>', '-name', '<i>name1</i>', '-objectClasses', '<i>objectclass</i>'])</code> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <code>\$AdminTask addIdMgrLDAPEntityType {-interactive}</code> • Using Jython string: <code>AdminTask.addIdMgrLDAPEntityType ('[-interactive]')</code> • Using Jython list: <code>AdminTask.addIdMgrLDAPEntityType (['-interactive'])</code>

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
<p>addId MgrLDAP EntityType RDNAttr</p>	<p>The addId MgrLDAP EntityType RDNAttr command adds RDN attribute configuration to an LDAP entity type definition.</p>	<ul style="list-style-type: none"> • Parameters: <li style="padding-left: 20px;">- id The ID of the repository. (String, required) <li style="padding-left: 20px;">- entityTypeName The name of the entity type. (String, required) <li style="padding-left: 20px;">- name The attribute name that is used to build the relative distinguished name (RDN) for the entity type. (String, required) <li style="padding-left: 20px;">- objectClass The object class to use for the entity type for the relative distinguished name (RDN) attribute name that you specify. Use this parameter to map one entity type to multiple structural object classes. (String, optional) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask addIdMgrLDAPEntityTypeRDNAttr {-id <i>id1</i> -entityTypeName <i>entitytype</i> -name <i>name1</i>}</pre> • Using Jython string: <pre>AdminTask.addIdMgrLDAPEntityTypeRDNAttr ('[-id <i>id1</i> -entityTypeName <i>entitytype</i> -name <i>name1</i>']')</pre> • Using Jython list: <pre>AdminTask.addIdMgrLDAPEntityTypeRDNAttr (['-id', '<i>id1</i>', '-entityTypeName', '<i>entitytype</i>', '-name', '<i>name1</i>'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask addIdMgrLDAPEntityTypeRDNAttr {-interactive}</pre> • Using Jython string: <pre>AdminTask.addIdMgrLDAPEntityTypeRDNAttr ('[-interactive]')</pre> • Using Jython list: <pre>AdminTask.addIdMgrLDAPEntityTypeRDNAttr (['-interactive'])</pre>

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
<p>addIdMgr LDAPGroup Dynamic Member Attr</p>	<p>The addIdMgr LDAPGroup Dynamic Member Attr command adds a dynamic member attribute configuration to an LDAP group configuration.</p>	<ul style="list-style-type: none"> • Parameters: - id The ID of the repository. (String, required) - name The name of the LDAP attribute that is used as the group member attribute. For example, member or uniqueMember. (String, required) - objectClass The group object class that contains the member attribute. For example, groupOfNames or groupOfUniqueNames. If you do not define this parameter, the member attribute applies to all group object classes. (String, optional) 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask addIdMgrLDAPGroup DynamicMemberAttr {-id <i>id1</i> -name <i>name1</i> -objectClass <i>objectclass</i>} • Using Jython string: AdminTask.addIdMgrLDAPGroup DynamicMemberAttr ('[-id <i>id1</i> -name <i>name1</i> -objectClass <i>objectclass</i>']') • Using Jython list: AdminTask.addIdMgrLDAPGroup DynamicMemberAttr (['-id', '<i>id1</i>', '-name', '<i>name1</i>', '-objectClass', '<i>objectclass</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask addIdMgrLDAPGroup DynamicMemberAttr {-interactive} • Using Jython string: AdminTask.addIdMgrLDAPGroup DynamicMemberAttr ('[-interactive]') • Using Jython list: AdminTask.addIdMgrLDAPGroup DynamicMemberAttr (['-interactive'])

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<p>- scope</p> <p>The scope of the member attribute. The valid values for this parameter include the following:</p> <ul style="list-style-type: none"> • direct - The member attribute only contains direct members, therefore, this value refers to the member directly contained by the group and not contained through the nested group. For example, if Group1 contains Group2 and Group2 contains User1, then Group2 is a direct member of Group1 but User1 is not a direct member of Group1. Both member and uniqueMember are direct member attributes. • nested - The member attribute that contains the direct members and the nested members. 	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<ul style="list-style-type: none"> • all - The member attribute that contains the direct members, the nested members, and the dynamic members. For example, the <code>ibm-allMembers</code> attribute which is supported by the IBM Tivoli Directory Server. (String, optional) - dummyMember Indicates that if you create a group without specifying a member, a dummy member will be filled in to avoid creating an exception about missing a mandatory attribute. (String, optional) • Returns: None 	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
<p>addIdMgr LDAPGroup MemberAttr</p>	<p>The addIdMgr LDAPGroup MemberAttr command adds a member attribute configuration to an LDAP group configuration.</p>	<ul style="list-style-type: none"> • Parameters: - id The ID of the repository. (String, required) - name The name of the LDAP attribute that is used as the group member attribute. For example, member or uniqueMember. (String, required) - objectClass The group object class that contains the member attribute. For example, groupOfNames or groupOfUniqueNames. If you do not define this parameter, the member attribute applies to all group object classes. (String, optional) 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask addIdMgrLDAPGroup MemberAttr {-id <i>id1</i> -name <i>name1</i>} • Using Jython string: AdminTask.addIdMgrLDAPGroup MemberAttr ('[-id <i>id1</i> -name <i>name1</i>]') • Using Jython list: AdminTask.addIdMgrLDAPGroup MemberAttr (['-id', '<i>id1</i>', '-name', '<i>name1</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask addIdMgrLDAPGroup MemberAttr {-interactive} • Using Jython string: AdminTask.addIdMgrLDAPGroup MemberAttr ('[-interactive]') • Using Jython list: AdminTask.addIdMgrLDAPGroup MemberAttr (['-interactive'])

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<p>- scope</p> <p>The scope of the member attribute. The valid values for this parameter include the following:</p> <ul style="list-style-type: none"> • direct - The member attribute only contains direct members, therefore, this value refers to the member directly contained by the group and not contained through the nested group. For example, if Group1 contains Group2 and Group2 contains User1, then Group2 is a direct member of Group1 but User1 is not a direct member of Group1. Both member and uniqueMember are direct member attributes. • nested - The member attribute that contains the direct members and the nested members. 	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<ul style="list-style-type: none"> • all - The member attribute that contains the direct members, the nested members, and the dynamic members. For example, the <code>ibm-allMembers</code> attribute which is supported by the IBM Tivoli Directory Server. (String, optional) - dummyMember Indicates that if you create a group without specifying a member, a dummy member will be filled in to avoid creating an exception about missing a mandatory attribute. (String, optional) • Returns: None 	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
addId MgrLDAP Server	<p>The addId MgrLDAP Server command adds an LDAP server to the LDAP repository ID that you specify.</p>	<ul style="list-style-type: none"> • Parameters: - id The ID of the repository. (String, required) - host The host name for the primary LDAP server. (String, required) - port The port number for the LDAP server. (Integer, optional) - bindDN The binding distinguished name for the LDAP server. (String, optional) - bindPassword The binding password. (String, optional) - authentication Indicates the authentication method to use. The default value is simple. Valid values include: none or strong. (String, optional) - referral The LDAP referral. The default value is ignore. Valid values include: follow, throw, or false. (String, optional) 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask addIdMgrLDAPServer {-id <i>id1</i> -host <i>myhost.ibm.com</i>} • Using Jython string: AdminTask.addIdMgrLDAPServer ('[-id <i>id1</i> -host <i>myhost.ibm.com</i>]') • Using Jython list: AdminTask.addIdMgrLDAPServer (['-id', '<i>id1</i>', '-host', '<i>myhost.ibm.com</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask addIdMgrLDAPServer {-interactive} • Using Jython string: AdminTask.addIdMgrLDAPServer ('[-interactive]') • Using Jython list: AdminTask.addIdMgrLDAPServer (['-interactive'])

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<p>- derefAliases Controls how aliases are dereferenced. The default value is always. Valid values include:</p> <ul style="list-style-type: none"> • never - never deference aliases • finding - deferences aliases only during name resolution • searching - deferences aliases only after name resolution <p>(String, optional)</p> <p>- sslEnabled Indicates to enable SSL or not. The default value is false. (Boolean, optional)</p> <p>- connectionPool The connection pool. The default value is false. (Boolean, optional)</p> <p>- connectTimeout The connection timeout in seconds. The default value is 0. (Integer, optional)</p>	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<p>- ldapServerType The type of LDAP server being used. The default value is IDS51. (String, optional)</p> <p>- sslConfiguration The SSL configuration. (String, optional)</p> <p>-</p> <p>certificateMapMode Specifies whether to map X.509 certificates into a LDAP directory by exact distinguished name or by certificate filter. The default value is exactdn. To use the certificate filter for the mapping, specify certificatefilter. (String, optional)</p> <p>- certificateFilter If certificateMapMode has the value certificatefilter, then this property specifies the LDAP filter which maps attributes in the client certificate to entries in LDAP. (String, optional)</p> <ul style="list-style-type: none"> • Returns: None 	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
<p>addIdMgr Repository BaseEntry</p>	<p>The addIdMgr Repository BaseEntry command adds a base entry to the specified repository.</p>	<ul style="list-style-type: none"> • Parameters: <li style="padding-left: 20px;">- id The ID of the repository. (String, required) <li style="padding-left: 20px;">- name The distinguished name of a base entry. (String, required) <li style="padding-left: 20px;">- nameInRepository The distinguished name in the repository that uniquely identifies the base entry name. (String, optional) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask addIdMgrRepository BaseEntry {-id <i>id1</i> -name <i>name1</i>} • Using Jython string: AdminTask.addIdMgrRepository BaseEntry ('[-id <i>id1</i> -name <i>name1</i>]') • Using Jython list: AdminTask.addIdMgrRepository BaseEntry (['-id', '<i>id1</i>', '-name', '<i>name1</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask addIdMgrRepository BaseEntry {-interactive} • Using Jython string: AdminTask.addIdMgrRepository BaseEntry ('[-interactive]') • Using Jython list: AdminTask.addIdMgrRepository BaseEntry (['-interactive'])

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
<p>createId MgrDB Repository</p>	<p>The createId MgrDB Repository command creates a database repository configuration.</p>	<ul style="list-style-type: none"> • Parameters: - id The ID of the repository. (String, required) - dataSourceName The name of the data source. The default value is jdbc/wimDS. (String, required) - databaseType The type of the database. The default value is DB2. (String, required) - dbURL The URL of the database. (String, required) - dbAdminId The database administrator ID. (String, required if database type is not Cloudscape.) - dbAdminPassword The database administrator password. (String, required if database type is not Cloudscape.) - adapterClassName The default value is com.ibm.ws.wim.adapter.db.DBAdapter. (String, optional) 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask createIdMgrDB Repository {-id <i>id1</i> -dataSourceName <i>datasource name</i> -databaseType <i>DB2</i>} • Using Jython string: AdminTask.createIdMgrDB Repository ('[-id <i>id1</i> -dataSourceName <i>datasource name</i> -databaseType <i>DB2</i>']') • Using Jython list: AdminTask.createIdMgrDB Repository (['-id', '<i>id1</i>', '-dataSourceName', '<i>datasource name</i>', '-databaseType', '<i>DB2</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask createIdMgrDB Repository {-interactive} • Using Jython string: AdminTask.createIdMgrDB Repository ('[-interactive]') • Using Jython list: AdminTask.createIdMgrDB Repository (['-interactive'])

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<ul style="list-style-type: none"> <li data-bbox="769 302 1000 422">- JDBCDriverClass The JDBC driver class name. (String, optional) <li data-bbox="769 436 1000 667">- supportSorting Indicates if sorting is supported or not. The default value is false. (Boolean, optional) <li data-bbox="769 682 1000 940">- supportTransaction Indicates if transactions are supported or not. The default value is false. (Boolean, optional) <li data-bbox="769 955 1000 1161">- isExtIdUnique Specifies if the external ID is unique. The default value is true. (Boolean, optional) <li data-bbox="769 1176 1000 1434">- supportExternalName Indicates if external names are supported or not. The default value is false. (Boolean, optional) 	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<ul style="list-style-type: none"> - entityRetrievalLimit Indicates the value of the retrieval limit on database entries. The default value is 200. (Integer, optional) - saltLength The salt length in bits. The default value is 12. (Integer, optional) - encryptionKey The default value is rZ15ws0e1y9yHk3zCs3sTMv/ho8fY17s. (String, optional) • Returns: None 	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
<p>createId MgrFile Repository</p>	<p>The createId MgrFile Repository command creates a file repository configuration.</p>	<ul style="list-style-type: none"> • Parameters: - id The ID of the repository. (String, required) - messageDigestAlgorithm The message digest algorithm that will be used for hashing the password. The default value is SHA-1. Valid values include the following: SHA-245, SHA-384, or SHA-512. (String, required) - adapterClassName The default value is com.ibm.ws.wim.adapter.file.was.FileAdapter. (String, optional) - supportPaging Indicates if paging is supported or not. The default value is false. (Boolean, optional) - supportSorting Indicates if sorting is supported or not. The default value is false. (Boolean, optional) 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask createIdMgrFile Repository {-id <i>id1</i> -messageDigestAlgorithm <i>SHA-245</i>} • Using Jython string: AdminTask.createIdMgrFile Repository ('[-id <i>id1</i> -messageDigestAlgorithm <i>SHA-245</i>]') • Using Jython list: AdminTask.createIdMgrFile Repository (['-id', '<i>id1</i>', '-messageDigestAlgorithm', '<i>SHA-245</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask createIdMgrFile Repository {-interactive} • Using Jython string: AdminTask.createIdMgrFile Repository ('[-interactive]') • Using Jython list: AdminTask.createIdMgrFile Repository (['-interactive'])

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<ul style="list-style-type: none"> - supportTransaction Indicates if transaction is supported or not. The default value is false. (Boolean, optional) - isExtIdUnique Specifies if the external ID is unique or not. The default value is true. (Boolean, optional) - supportExternalName Indicates if external names are supported or not. The default value is false. (Boolean, optional) - baseDirectory The base directory where the fill will be created in order to store the data. The default is to be dynamically built during run time using user.install.root and cell name. (String, optional) - fileName The file name of the repository. The default value is fileRegistry.xml. (String, optional) 	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<ul style="list-style-type: none"> - saltLength The salt length of the randomly generated salt for password hashing. The default value is 12. (Integer, optional) • Returns: None 	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
<p>createIdMgrLDAP Repository</p>	<p>The create IdMgrLDAP Repository command creates an LDAP repository configuration.</p>	<ul style="list-style-type: none"> • Parameters: - id The unique identifier for the repository. (String, required) - ldapServerType The type of LDAP server that is being used. The default value is <code>IDS51</code>. (String, required) - adapterClassName The default value is <code>com.ibm.ws.wim.adapter.db.DBAdapter</code>. (String, optional) - supportSorting Indicates if sorting is supported or not. The default value is <code>false</code>. (Boolean, optional) - supportPaging Indicates if paging is supported or not. The default value is <code>false</code>. (Boolean, optional) - supportTransaction Indicates if transactions are supported or not. The default value is <code>false</code>. (Boolean, optional) 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <code>\$AdminTask createIdMgrLDAP Repository {-id <i>id1</i> -ldapServerType <i>IDS51</i>}</code> • Using Jython string: <code>AdminTask.createIdMgrLDAP Repository ('[-id <i>id1</i> -ldapServerType <i>IDS51</i>']')</code> • Using Jython list: <code>AdminTask.createIdMgrLDAP Repository (['-id', '<i>id1</i>', '-ldapServerType', '<i>IDS51</i>'])</code> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <code>\$AdminTask createIdMgrLDAP Repository {-interactive}</code> • Using Jython string: <code>AdminTask.createIdMgrLDAP Repository ('[-interactive]')</code> • Using Jython list: <code>AdminTask.createIdMgrLDAP Repository (['-interactive'])</code>

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<ul style="list-style-type: none"> <li data-bbox="769 302 1011 506"> <p>- isExtIdUnique Specifies if the external ID is unique. The default value is true. (Boolean, optional)</p> <li data-bbox="769 527 1011 783"> <p>- supportExternalName Indicates if external names are supported or not. The default value is false. (Boolean, optional)</p> <li data-bbox="769 804 1011 1060"> <p>- authentication Indicates the authentication method to use. The default value is simple. Valid values include: none or strong. (String, optional)</p> <li data-bbox="769 1081 1011 1337"> <p>- referral The LDAP referral. The default value is ignore. Valid values include: follow, throw, or false. (String, optional)</p> <li data-bbox="769 1358 1011 1562"> <p>- sslEnabled Indicates to enable SSL or not. The default value is false. (Boolean, optional)</p> <li data-bbox="769 1583 1011 1692"> <p>- sslConfiguration The SSL configuration. (String, optional)</p> 	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<ul style="list-style-type: none"> - connectionPool The connection pool. The default value is false. (Boolean, optional) - translateRDN Indicates to translate RDN or not. The default value is false. (Boolean, optional) - searchTimeLimit The value of search time limit. (Integer, optional) - searchCountLimit The value of search count limit. (Integer, optional) - searchPageSize The value of search page size. (Integer, optional) - returnToPrimaryServer (Integer, optional) - primaryServerQueryTimeInterval (Integer, optional) - default If you set this parameter to true, the default values will be set for the remaining configuration properties of the LDAP repository. (Boolean, optional) • Returns: None 	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
deleteId MgrLDAP EntityType	The deleteId MgrLDAP EntityType command deletes the LDAP entity type configuration data for a specified entity type for a specific LDAP repository.	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - id The ID of the repository. (String, required) - name The name of the entity type. (String, required) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <code>\$AdminTask deleteIdMgrLDAP EntityType {-id <i>id1</i> -name <i>name1</i>}</code> • Using Jython string: <code>AdminTask.deleteIdMgrLDAP EntityType ('[-id <i>id1</i> -name <i>name1</i>']')</code> • Using Jython list: <code>AdminTask.deleteIdMgrLDAP EntityType (['-id', '<i>id1</i>', '-name', '<i>name1</i>'])</code> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <code>\$AdminTask deleteIdMgrLDAP EntityType {-interactive}</code> • Using Jython string: <code>AdminTask.deleteIdMgrLDAP EntityType ('[-interactive]')</code> • Using Jython list: <code>AdminTask.deleteIdMgrLDAP EntityType (['-interactive'])</code>

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
<p>deleteId MgrLDAP EntityType RDNAttr</p>	<p>The deleteId MgrLDAP EntityType RDNAttr command deletes the relative distinguished name (RDN) attribute configuration from an LDAP entity type configuration.</p>	<ul style="list-style-type: none"> • Parameters: - id The ID of the repository. (String, required) - entityTypeName The name of the entity type. (String, required) - name The attribute name that is used to build the relative distinguished name (RDN) for the entity type. (String, required) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask deleteIdMgrLDAPEntityTypeRDNAttr {-id <i>id1</i> -name <i>name1</i> -entityTypeName <i>entityType</i>}</pre> • Using Jython string: <pre>AdminTask.deleteIdMgrLDAPEntityTypeRDNAttr ('[-id <i>id1</i> -name <i>name1</i> -entityType <i>entityType</i>']')</pre> • Using Jython list: <pre>AdminTask.deleteIdMgrLDAPEntityTypeRDNAttr (['-id', '<i>id1</i>', '-name', '<i>name1</i>', '-entityType', '<i>entityType</i>'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask deleteIdMgrLDAPEntityTypeRDNAttr {-interactive}</pre> • Using Jython string: <pre>AdminTask.deleteIdMgrLDAPEntityTypeRDNAttr ('[-interactive]')</pre> • Using Jython list: <pre>AdminTask.deleteIdMgrLDAPEntityTypeRDNAttr (['-interactive'])</pre>

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
<p>deleteId MgrLDAP GroupConfig</p>	<p>The deleteId MgrLDAP GroupConfig command deletes the LDAP group configuration.</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - id The ID of the repository. (String, required) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask deleteIdMgrLDAP GroupConfig {-id <i>id1</i>} • Using Jython string: AdminTask.deleteIdMgrLDAP GroupConfig ('[-id <i>id1</i>']') • Using Jython list: AdminTask.deleteIdMgrLDAP GroupConfig (['-id', '<i>id1</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask deleteIdMgrLDAP GroupConfig {-interactive} • Using Jython string: AdminTask.deleteIdMgrLDAP GroupConfig ('[-interactive]') • Using Jython list: AdminTask.deleteIdMgrLDAP GroupConfig (['-interactive'])
<p>deleteIdMgr LDAPGroup MemberAttr</p>	<p>The deleteIdMgr LDAPGroup MemberAttr command deletes a member attribute configuration from an LDAP group configuration.</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - id The ID of the repository. (String, required) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask deleteIdMgrLDAP GroupMemberAttr {-id <i>id1</i>} • Using Jython string: AdminTask.deleteIdMgrLDAP GroupMemberAttr ('[-id <i>id1</i>']') • Using Jython list: AdminTask.deleteIdMgrLDAP GroupMemberAttr (['-id', '<i>id1</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask deleteIdMgrLDAP GroupMemberAttr {-interactive} • Using Jython string: AdminTask.deleteIdMgrLDAP GroupMemberAttr ('[-interactive]') • Using Jython list: AdminTask.deleteIdMgrLDAP GroupMemberAttr (['-interactive'])

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
<p>deleteIdMgr LDAPGroup Dynamic MemberAttr</p>	<p>The deleteIdMgr LDAPGroup Dynamic MemberAttr command deletes a dynamic member attribute configuration from an LDAP group configuration.</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - id The ID of the repository. (String, required) - name The name of the LDAP attribute that is used as the group member attribute. For example, memberURL. (String, required) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask deleteIdMgrLDAPGroupDynamicMemberAttr {-id <i>id1</i> -name <i>name1</i>}</pre> • Using Jython string: <pre>AdminTask.deleteIdMgrLDAPGroupDynamicMemberAttr ('[-id <i>id1</i> -name <i>name1</i>']')</pre> • Using Jython list: <pre>AdminTask.deleteIdMgrLDAPGroupDynamicMemberAttr (['-id', '<i>id1</i>', '-name', '<i>name1</i>'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask deleteIdMgrLDAPGroupDynamicMemberAttr {-interactive}</pre> • Using Jython string: <pre>AdminTask.deleteIdMgrLDAPGroupDynamicMemberAttr ('[-interactive]')</pre> • Using Jython list: <pre>AdminTask.deleteIdMgrLDAPGroupDynamicMemberAttr (['-interactive'])</pre>

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
deleteld MgrLDAP Server	The deleteld MgrLDAP Server command deletes the configuration for the LDAP server that you specify from the LDAP repository ID that you specify.	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - id The ID of the repository. (String, required) - host The host name for the primary LDAP server. (String, required) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask deleteIdMgrLDAP Server {-id <i>id1</i> -host <i>myhost.ibm.com</i>} • Using Jython string: AdminTask.deleteIdMgrLDAP Server ('[-id <i>id1</i> -host <i>myhost.ibm.com</i>]') • Using Jython list: AdminTask.deleteIdMgrLDAP Server (['-id', '<i>id1</i>', '-host', '<i>myhost.ibm.com</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask deleteIdMgrLDAP Server {-interactive} • Using Jython string: AdminTask.deleteIdMgrLDAP Server ('[-interactive]') • Using Jython list: AdminTask.deleteIdMgrLDAP Server (['-interactive'])
deleteldMgr Repository	The deleteldMgr Repository command deletes a repository that you specify.	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - id The ID of the repository. Valid values include existing repository IDs. (String, required) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask deleteIdMgr Repository {-id <i>id1</i>} • Using Jython string: AdminTask.deleteIdMgr Repository ('[-id <i>id1</i>]') • Using Jython list: AdminTask.deleteIdMgr Repository (['-id', '<i>id1</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask deleteIdMgrRepository {-interactive} • Using Jython string: AdminTask.deleteIdMgrRepository ('[-interactive]') • Using Jython list: AdminTask.deleteIdMgrRepository (['-interactive'])

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
<p>deleteIdMgr Repository BaseEntry</p>	<p>The deleteIdMgr Repository BaseEntry command deletes a base entry from the specified repository.</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - id The ID of the repository. (String, required) - name The distinguished name of a base entry. (String, required) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask deleteIdMgrRepository BaseEntry {-id <i>id1</i> -name <i>name1</i>} • Using Jython string: AdminTask.deleteIdMgrRepository BaseEntry ('[-id <i>id1</i> -name <i>name1</i>']') • Using Jython list: AdminTask.deleteIdMgrRepository BaseEntry (['-id', '<i>id1</i>', '-name', '<i>name1</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask deleteIdMgrRepository BaseEntry {-interactive} • Using Jython string: AdminTask.deleteIdMgrRepository BaseEntry ('[-interactive]') • Using Jython list: AdminTask.deleteIdMgrRepository BaseEntry (['-interactive'])
<p>getIdMgr LDAPAttr Cache</p>	<p>The getIdMgr LDAPAttr Cache command returns the LDAP attribute cache configuration.</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - id The ID of the repository. (String, required) • Returns: A hash map with the parameters of the setIdMgr LDAPAttr Cache command as keys. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask getIdMgrLDAPAttr Cache {-id <i>id1</i>} • Using Jython string: AdminTask.getIdMgrLDAPAttr Cache ('[-id <i>id1</i>']') • Using Jython list: AdminTask.getIdMgrLDAPAttr Cache (['-id', '<i>id1</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask getIdMgrLDAPAttrCache {-interactive} • Using Jython string: AdminTask.getIdMgrLDAPAttrCache ('[-interactive]') • Using Jython list: AdminTask.getIdMgrLDAPAttrCache (['-interactive'])

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
getIdMgr LDAP ContextPool	The getIdMgr LDAP Context Pool command returns the LDAP context pool configuration.	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - id The ID of the repository. (String, required) • Returns: A hash map that includes the parameters of the setIdMgrLDAPContextPool command as the keys. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask getIdMgrLDAPContextPool {-id <i>id1</i>} • Using Jython string: AdminTask.getIdMgrLDAPContextPool ('[-id <i>id1</i>']') • Using Jython list: AdminTask.getIdMgrLDAPContextPool (['-id', '<i>id1</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask getIdMgrLDAPContextPool {-interactive} • Using Jython string: AdminTask.getIdMgrLDAPContextPool ('[-interactive]') • Using Jython list: AdminTask.getIdMgrLDAPContextPool (['-interactive'])
getIdMgr LDAP EntityType	The getIdMgr LDAP EntityType command returns the LDAP entity type configuration data.	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - id The ID of the repository. (String, required) - name The name of the entity type. (String, required) • Returns: A hash map with the keys the same as the property name of the addIdMgrLDAPEntityType command. Multi-valued parameters are returned as list. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask getIdMgrLDAPEntityType {-id <i>id1</i> -name <i>name1</i>} • Using Jython string: AdminTask.getIdMgrLDAPEntityType ('[-id <i>id1</i> -name <i>name1</i>]') • Using Jython list: AdminTask.getIdMgrLDAPEntityType (['-id', '<i>id1</i>', '-name', '<i>name1</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask getIdMgrLDAPEntityType {-interactive} • Using Jython string: AdminTask.getIdMgrLDAPEntityType ('[-interactive]') • Using Jython list: AdminTask.getIdMgrLDAPEntityType (['-interactive'])

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
<p>getIdMgr LDAPEntity TypeRDNAttr</p>	<p>The getIdMgr LDAPEntity TypeRDNAttr command returns the relative distinguished name (RDN) attribute configuration for an LDAP entity type definition.</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - id The ID of the repository. (String, required) - entityTypeName The name of the entity name. (String, required) • Returns: A hash map with the RDN attribute names as the key. If the object class is set, the value of the key will be set to the value of the object class. Otherwise, the value will be null. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask getIdMgrLDAPEntity TypeRDNAttr {-id <i>id1</i> -entityTypeName <i>name1</i>}</pre> • Using Jython string: <pre>AdminTask.getIdMgrLDAPEntity TypeRDNAttr ('[-id <i>id1</i> -entityTypeName <i>name1</i>']')</pre> • Using Jython list: <pre>AdminTask.getIdMgrLDAPEntity TypeRDNAttr (['-id', '<i>id1</i>', '-entityTypeName', '<i>name1</i>'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask getIdMgrLDAPEntity TypeRDNAttr {-interactive}</pre> • Using Jython string: <pre>AdminTask.getIdMgrLDAPEntity TypeRDNAttr ('[-interactive]')</pre> • Using Jython list: <pre>AdminTask.getIdMgrLDAPEntity TypeRDNAttr (['-interactive'])</pre>
<p>getIdMgr LDAPGroupConfig</p>	<p>The getIdMgr LDAPGroupConfig command returns the LDAP group configuration.</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - id The ID of the repository. (String, required) • Returns: A hash map that contains the parameters of the setIdMgr LDAPGroupConfig command as the keys. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask getIdMgrLDAPGroup Config {-id <i>id1</i>}</pre> • Using Jython string: <pre>AdminTask.getIdMgrLDAPGroup Config ('[-id <i>id1</i>']')</pre> • Using Jython list: <pre>AdminTask.getIdMgrLDAPGroup Config (['-id', '<i>id1</i>'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask getIdMgrLDAPGroup Config {-interactive}</pre> • Using Jython string: <pre>AdminTask.getIdMgrLDAPGroup Config ('[-interactive]')</pre> • Using Jython list: <pre>AdminTask.getIdMgrLDAPGroup Config (['-interactive'])</pre>

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
getIdMgr LDAPGroup Dynamic Member Attrs	The getIdMgr LDAPGroup Dynamic Member Attrs command returns the dynamic member attribute configuration from the LDAP group configuration.	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - id The ID of the repository. (String, required) • Returns: A hash map list that contains the parameters of the addIdMgr LDAPGroup Dynamic MemberAttr command as the keys. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask getIdMgrLDAPGroupDynamicMemberAttrs {-id <i>id1</i>} • Using Jython string: AdminTask.getIdMgrLDAPGroupDynamicMemberAttrs ('[-id <i>id1</i>']') • Using Jython list: AdminTask.getIdMgrLDAPGroupDynamicMemberAttrs (['-id', '<i>id1</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask getIdMgrLDAPGroupDynamicMemberAttrs {-interactive} • Using Jython string: AdminTask.getIdMgrLDAPGroupDynamicMemberAttrs ('[-interactive]') • Using Jython list: AdminTask.getIdMgrLDAPGroupDynamicMemberAttrs (['-interactive'])
getIdMgr LDAPGroup MemberAttrs	The getIdMgr LDAPGroup MemberAttrs command returns the member attribute configuration for the LDAP group configuration.	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - id The ID of the repository. (String, required) • Returns: A hash map list that contains the parameters of the addIdMgr LDAPGroup MemberAttr command as the keys. 	<p>Batch mode example usage:</p> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask getIdMgrLDAPGroupMemberAttrs {-interactive} • Using Jython string: AdminTask.getIdMgrLDAPGroupMemberAttrs ('[-interactive]') • Using Jython list: AdminTask.getIdMgrLDAPGroupMemberAttrs (['-interactive'])

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
getIdMgr LDAPSearch ResultCache	The getIdMgr LDAPSearch ResultCache command returns the LDAP search result cache configuration.	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - id The ID of the repository. (String, required) • Returns: A hash map with the parameters of the setIdMgr LDAPSearch ResultCache command as the keys. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask getIdMgrLDAPSearchResultCache {-id <i>id1</i>} • Using Jython string: AdminTask.getIdMgrLDAPSearchResultCache ('[-id <i>id1</i>']') • Using Jython list: AdminTask.getIdMgrLDAPSearchResultCache (['-id', '<i>id1</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask getIdMgrLDAPSearchResultCache {-interactive} • Using Jython string: AdminTask.getIdMgrLDAPSearchResultCache ('[-interactive]') • Using Jython list: AdminTask.getIdMgrLDAPSearchResultCache (['-interactive'])
getIdMgr LDAPServer	The getIdMgr LDAPServer command returns the configuration for the LDAP server that you specify for the LDAP repository ID that you specify.	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - id The ID of the repository. (String, required) - host The host name for the primary LDAP server. (String, required) • Returns: A hash map with the keys the same as the parameter names for the addIdMgr LDAPServer command. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask getIdMgrLDAPServer {-id <i>id1</i> -host <i>myhost.ibm.com</i>} • Using Jython string: AdminTask.getIdMgrLDAPServer ('[-id <i>id1</i> -host <i>myhost.ibm.com</i>]') • Using Jython list: AdminTask.getIdMgrLDAPServer (['-id', '<i>id1</i>', '-host', '<i>myhost.ibm.com</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask getIdMgrLDAPServer {-interactive} • Using Jython string: AdminTask.getIdMgrLDAPServer ('[-interactive]') • Using Jython list: AdminTask.getIdMgrLDAPServer (['-interactive'])

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
getIdMgr Repository	<p>The getIdMgr Repository command returns the configuration of the specified repository.</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - id The ID of the repository. (String, required) • Returns: A hash map. The keys will vary depending on the type of repository. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask getIdMgrRepository {-id <i>id1</i>} • Using Jython string: AdminTask.getIdMgrRepository ('[-id <i>id1</i>']') • Using Jython list: AdminTask.getIdMgrRepository (['-id', '<i>id1</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask getIdMgrRepository {-interactive} • Using Jython string: AdminTask.getIdMgrRepository ('[-interactive]') • Using Jython list: AdminTask.getIdMgrRepository (['-interactive'])
listIdMgr Custom Properties	<p>The listIdMgr Custom Properties command returns a list of custom properties for the repository that you specify.</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - id The ID of the repository. (String, required) • Returns: A hash map that contains keys as the custom property names and values as custom property values. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask listIdMgrCustomProperties {-id <i>id1</i>} • Using Jython string: AdminTask.listIdMgrCustomProperties ('[-id <i>id1</i>']') • Using Jython list: AdminTask.listIdMgrCustomProperties (['-id', '<i>id1</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask listIdMgrCustomProperties {-interactive} • Using Jython string: AdminTask.listIdMgrCustomProperties ('[-interactive]') • Using Jython list: AdminTask.listIdMgrCustomProperties (['-interactive'])

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
listIdMgr LDAPBackupServers	The listIdMgr LDAPBackupServers command returns a list of the backup LDAP server or servers.	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - id The ID of the repository. (String, required) - primary_host The host name for the primary LDAP server. (String, required) • Returns: A list of hash maps. The hash maps contains the names of the backup servers as the key and the port numbers as the value. Returning all the data in a hash map does not maintain the order of backup servers. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask listIdMgrLDAPBackupServer {-id <i>id1</i> -primary_host <i>hostname</i>} • Using Jython string: AdminTask.listIdMgrLDAPBackupServer ('[-id <i>id1</i> -primary_host <i>hostname</i>']') • Using Jython list: AdminTask.listIdMgrLDAPBackupServer (['-id', '<i>id1</i>', '-primary_host', '<i>hostname</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask listIdMgrLDAPBackupServer {-interactive} • Using Jython string: AdminTask.listIdMgrLDAPBackupServer ('[-interactive]') • Using Jython list: AdminTask.listIdMgrLDAPBackupServer (['-interactive'])
listIdMgr LDAPEntityTypeTypes	The listIdMgr LDAPEntityTypeTypes command lists the name of all of the configured LDAP entity type definitions.	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - id The ID of the repository. (String, required) • Returns: A list that contains the names of the configured LDAP entity types. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask listIdMgrLDAPEntityType {-id <i>id1</i>} • Using Jython string: AdminTask.listIdMgrLDAPEntityType ('[-id <i>id1</i>']') • Using Jython list: AdminTask.listIdMgrLDAPEntityType (['-id', '<i>id1</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask listIdMgrLDAPEntityType {-interactive} • Using Jython string: AdminTask.listIdMgrLDAPEntityType ('[-interactive]') • Using Jython list: AdminTask.listIdMgrLDAPEntityType (['-interactive'])

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
listIdMgr LDAP Servers	The listIdMgr LDAP Servers command lists all of the configured primary LDAP servers.	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - id The ID of the repository. (String, required) • Returns: A list that contains the primary LDAP server names. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask listIdMgrLDAP Servers {-id <i>id1</i>} • Using Jython string: AdminTask.listIdMgrLDAP Servers ('[-id <i>id1</i>']') • Using Jython list: AdminTask.listIdMgrLDAP Servers (['-id', '<i>id1</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask listIdMgrLDAP Servers {-interactive} • Using Jython string: AdminTask.listIdMgrLDAP Servers ('[-interactive]') • Using Jython list: AdminTask.listIdMgrLDAP Servers (['-interactive'])

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
listIdMgr Repositories	<p>The listIdMgr Repositories command lists names and types of all configured repositories.</p>	<ul style="list-style-type: none"> • Parameters: None • Returns: A hash map with key as the name of the repository and value as another hash map that includes the following keys: <ul style="list-style-type: none"> – repositoryType - The type of repository. For example, File, LDAP, DB, and so on. – specificRepositoryType - The specific type of repository. For example, LDAP, IDS51, NDS, and so on. – host - The host name where the repository resides. For File, it is LocalHost and for DB it is dataSourceName. <p>This command will not return the Property Extension and Entry Mapping repository data.</p>	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask listIdMgrRepositories</pre> • Using Jython string: <pre>AdminTask.listIdMgrRepositories()</pre> • Using Jython list: <pre>AdminTask.listIdMgrRepositories()</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask listIdMgrRepositories {-interactive}</pre> • Using Jython string: <pre>AdminTask.listIdMgrRepositories (['-interactive'])</pre> • Using Jython list: <pre>AdminTask.listIdMgrRepositories (['-interactive'])</pre>

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
listIdMgr Repository BaseEntries	The listIdMgr Repository BaseEntries command lists the base entries for a specified repository.	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - id The ID of the repository. (String, required) • Returns: A hash map that contains base entry name as key and nameInRepository as value. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask listIdMgrRepository BaseEntries {-id <i>id1</i>} • Using Jython string: AdminTask.listIdMgrRepository BaseEntries ('[-id <i>id1</i>']') • Using Jython list: AdminTask.listIdMgrRepository BaseEntries (['-id', '<i>id1</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask listIdMgrRepository BaseEntries {-interactive} • Using Jython string: AdminTask.listIdMgrRepository BaseEntries ('[-interactive]') • Using Jython list: AdminTask.listIdMgrRepository BaseEntries (['-interactive'])
listIdMgr Supported DBTypes	The listIdMgr Supported DBTypes command returns a list of supported database types.	<ul style="list-style-type: none"> • Parameters: None • Returns: A list of supported database types. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask listIdMgrSupportedDBTypes • Using Jython string: AdminTask.listIdMgrSupportedDBTypes () • Using Jython list: AdminTask.listIdMgrSupportedDBTypes () <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask listIdMgrSupported DBTypes {-interactive} • Using Jython string: AdminTask.listIdMgrSupported DBTypes ('[-interactive]') • Using Jython list: AdminTask.listIdMgrSupported DBTypes (['-interactive'])

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
listIdMgr rSupported Message Digest Algorithms	The listIdMgr Supported Message Digest Algorithms command returns a list of supported message digest algorithms.	<ul style="list-style-type: none"> Parameters: None Returns: A list of supported message digest algorithms. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: \$AdminTask listIdMgrSupported MessageDigestAlgorithms Using Jython string: AdminTask.listIdMgrSupported MessageDigestAlgorithms () Using Jython list: AdminTask.listIdMgrSupported MessageDigestAlgorithms () <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: \$AdminTask listIdMgrSupportedMessageDigestAlgorithms {-interactive} Using Jython string: AdminTask.listIdMgrSupportedMessageDigestAlgorithms ('[-interactive]') Using Jython list: AdminTask.listIdMgrSupportedMessageDigestAlgorithms (['-interactive'])
listIdMgr Supported LDAPServerTypes	The listIdMgr Supported LDAP ServerTypes command returns a list of supported LDAP server types.	<ul style="list-style-type: none"> Parameters: None Returns: A list of supported LDAP server types. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: \$AdminTask listIdMgrSupported LDAPServerTypes Using Jython string: AdminTask.listIdMgrSupported LDAPServerTypes () Using Jython list: AdminTask.listIdMgrSupported LDAPServerTypes () <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: \$AdminTask listIdMgrSupported LDAPServerTypes {-interactive} Using Jython string: AdminTask.listIdMgrSupported LDAPServerTypes ('[-interactive]') Using Jython list: AdminTask.listIdMgrSupported LDAPServerTypes (['-interactive'])

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
removeldMgr LDAPBackupServer	The removeldMgr LDAPBackupServer command removes the backup LDAP server or servers.	<ul style="list-style-type: none"> • Parameters: - id The ID of the repository. (String, required) - primary_host The host name for the primary LDAP server. (String, required) - host The name of the backup host name. Use an asterisk (*) if you want to remove all backup servers. (String, required) - port The port number of the LDAP server. (Integer, optional) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <code>\$AdminTask removeIdMgrLDAPBackupServer {-id id1 -primary_host myprimaryhost.ibm.com -host myhost.ibm.com}</code> • Using Jython string: <code>AdminTask.removeIdMgrLDAPBackupServer ('[-id id1 -primary_host myprimaryhost.ibm.com -host myhost.ibm.com]')</code> • Using Jython list: <code>AdminTask.removeIdMgrLDAPBackupServer (['-id', 'id1', '-primary_host', 'myprimaryhost.ibm.com', '-host', 'myhost.ibm.com'])</code> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <code>\$AdminTask removeIdMgrLDAPBackupServer {-interactive}</code> • Using Jython string: <code>AdminTask.removeIdMgrLDAPBackupServer (['-interactive'])</code> • Using Jython list: <code>AdminTask.removeIdMgrLDAPBackupServer (['-interactive'])</code>

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
<p>setIdMgr Custom Property</p>	<p>The setIdMgr Custom Property command adds the custom properties to a repository configuration.</p>	<ul style="list-style-type: none"> • Parameters: <li style="padding-left: 20px;">- id The ID of the repository. Valid values include the existing repository IDs. (String, required) <li style="padding-left: 20px;">- name The name of the additional property for the repository that are not defined OOTB. (String, required) <li style="padding-left: 20px;">- value If this parameter is an empty string, the property will be deleted from the repository configuration. If this parameter is not an empty string and name does not exist, it will be added. If name is an empty string, all the custom properties will be deleted. (String, required) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask setIdMgrCustomProperty {-id <i>id1</i> -name <i>name1</i> -value <i>value</i>} • Using Jython string: AdminTask.setIdMgrCustomProperty ('[-id <i>id1</i> -name <i>name1</i> -value <i>value</i>']') • Using Jython list: AdminTask.setIdMgrCustomProperty (['-id', '<i>id1</i>', '-name', '<i>name1</i>', '-value', '<i>value</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask setIdMgrCustomProperty {-interactive} • Using Jython string: AdminTask.setIdMgrCustomProperty ('[-interactive]') • Using Jython list: AdminTask.setIdMgrCustomProperty (['-interactive'])

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
setIdMgr LDAPA ttrCache	<p>The setIdMgr LDAPA ttrCache command configures the LDAP attribute cache configuration.</p>	<ul style="list-style-type: none"> • Parameters: - id The ID of the repository. (String, required) - cachesDiskOffLoad (String, optional) - enabled Indicates if you want to enable attribute caching. The default value is true. (Boolean, optional) - cacheSize The maximum size of the attribute cache defined by the number of attribute objects that are permitted in the attribute cache. The minimum value of this parameter is 100. The default value is 4000. (Integer, optional) 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask setIdMgrLDAPAttr Cache {-id <i>id1</i>} • Using Jython string: AdminTask.setIdMgrLDAPAttr Cache ('[-id <i>id1</i>']') • Using Jython list: AdminTask.setIdMgrLDAPAttr Cache (['-id', '<i>id1</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask setIdMgrLDAPAttr Cache {-interactive} • Using Jython string: AdminTask.setIdMgrLDAPAttr Cache ('[-interactive]') • Using Jython list: AdminTask.setIdMgrLDAPAttr Cache (['-interactive'])

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<p>- cacheTimeOut The amount of time in seconds before the cached entries that are located in the attributes cache can be not valid. The minimum value of this parameter is 0. The attribute objects that are cached will remain in the attributes cache until the virtual member manager changes the attribute objects. The default value is 1200. (Integer, optional)</p> <p>- attributeSizeLimit An integer that represents the maximum number of attribute object values that can cache in the attributes cache.</p> <p>Some attributes, for example, the member attribute, contain many values. The attributeSizeLimit parameter prevents the attributes cache to cache large attributes. The default value is 2000. (Integer, optional)</p>	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<p>- serverTTLAttribute</p> <p>The name of the ttl attribute that is supported by the LDAP server. The attributes cache uses the value of this attribute to determine when the cached entries in the attributes cache will time out.</p> <p>The ttl attribute contains the time, in seconds, that any information from the entry should be kept by a client before it is considered stale and a new copy is fetched. A value of 0 implies that the object will not be cached.</p> <p>For more information about this attribute, go to:</p> <p>http://www.ietf.org/proceedings/98aug/I-D/draft-ietf-asid-ldap-cache-01.txt.</p> <p>The ttl attribute is not supported by all LDAP servers. If this attribute is supported by an LDAP server, you can set the value of the serverTTLAttribute parameter to the name of the ttl attribute in order to allow the value of the ttl attribute to determine when cached entries will time out. The time out value for different entries in attributes cache can be different.</p>	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<p>For example, if the value of the serverTTLAttribute parameter is ttl and the attributes cache retrieves attributes of a user from an LDAP server, it will also retrieve the value of the ttl attribute of this user. If the value is 200, the WMM uses this value to set the time out for the attributes of the user in the attributes cache instead of using the value of cacheTimeout. You can set different ttl attribute values for different users. (String, optional)</p> <ul style="list-style-type: none"> • Returns: None 	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
<p>setIdMgrLDAPContextPool</p>	<p>The setIdMgr LDAPContextPool command sets up the LDAP context pool configuration.</p>	<ul style="list-style-type: none"> • Parameters: - id The ID of the repository. (String, required) - enabled By default, the context pool is enabled. If you set this parameter to false, the context pool is disabled. When the context pool is disabled, new context instances will be created for each request. The default value is true. (Boolean, optional) - initPoolSize The number of context instances that the the virtual member manager LDAP adapter creates when it creates the pool. The valid range for this parameter is 1 to 50. The default value is 1. (Integer, optional) 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask setIdMgrLDAPContextPool {-id <i>id1</i>} • Using Jython string: AdminTask.setIdMgrLDAPContextPool ('[-id <i>id1</i>']') • Using Jython list: AdminTask.setIdMgrLDAPContextPool (['-id', '<i>id1</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask setIdMgrLDAPContextPool {-interactive} • Using Jython string: AdminTask.setIdMgrLDAPContextPool ('[-interactive]') • Using Jython list: AdminTask.setIdMgrLDAPContextPool (['-interactive'])

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<p>- maxPoolSize</p> <p>The maximum number of context instances that the context pool will maintain. Context instances that are in use and those that are idle contribute to this number. When the pool size reaches this number, new context instances cannot be created for new requests. The new request is blocked until a context instance is released by another request or is removed. The request checks periodically if there are context instances available in the pool according to the amount of time that you specify using the poolWaitTime parameter.</p> <p>The minimum value for this parameter is 0. There is no maximum value. Setting the value of this parameter to 0 means that there is no maximum size and a request for a pooled context instance will use an existing pooled idle context instance or a newly created pooled context instance. The default value is 20. (Integer, optional)</p>	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<p>- prefPoolSize</p> <p>The preferred number of context instances that the context pool will maintain. Context instances that are in use and those that are idle contribute to this number. When there is a request for the use of a pooled context instance and the pool size is less than the preferred size, the context pool creates and uses a new pooled context instance regardless of whether an idle connection is available. When a request finishes with a pooled context instance and the pool size is greater than the preferred size, the context pool closes and removes the pooled context instance from the pool.</p> <p>The valid range for this parameter is from 0 to 100. Setting the value of this parameter to 0 means that there is no preferred size and a request for a pooled context instance results in a newly created context instance only if no idle ones are available. The default value is 3.(Integer, optional)</p>	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<p>- poolTimeout An integer that represents the number of milliseconds that an idle context instance may remain in the pool without being closed and removed from the pool. When a context instance is requested from the pool, if this context already exists in the pool for more than the time defined by poolTimeout, this connection will be closed no matter this context instance is stale or active. A new context instance will be created and put back to the pool after it has been released from the request.</p> <p>The minimum value for this parameter is 0. There is no maximum value. Setting the value of this parameter to 0 means that the context instances in the pool will remain in the pool until they are staled. The context pool catches the communication exception and recreates a new context instance. The default value is 0.(Integer, optional)</p>	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<p>- poolWaitTime The time interval in milliseconds that the request waits until the context pool rechecks if there are idle context instances available in the pool when the number of context instances reaches the maximum pool size. If no idle context instance, the request will continue waiting for the same period of time until next checking.</p> <p>The minimum value for the poolWaitout parameter is 0. There is no maximum value. A value of 0 for this parameter means that the context pool will not check if idle context exists. The request will be notified when a context instance releases from other requests. The default value is 3000.(Integer, optional)</p> <ul style="list-style-type: none"> Returns: None 	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
<p>setIdMgr LDAPGroupConfig</p>	<p>The setIdMgr LDAPGroupConfig command sets up the LDAP group configuration.</p>	<ul style="list-style-type: none"> • Parameters: <li style="padding-left: 20px;">- id The ID of the repository. (String, required) <li style="padding-left: 20px;">- updateGroupMembership Updates the group membership if the member is deleted or renamed. Some LDAP servers, for example, Domino server, do not clean up the membership of the user when a user is deleted or renamed. If you choose these LDAP server types in the <code>ldapServerType</code> property, the value of this parameter is set to true. Use this parameter to change the value. The default value is false. (Boolean, optional) <li style="padding-left: 20px;">- name The name of the membership attribute. For example, <code>memberOf</code> in an active directory server and <code>ibm-allGroups</code> in IDS. (String, optional) 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <code>\$AdminTask setIdMgrLDAPGroupConfig {-id id1}</code> • Using Jython string: <code>AdminTask.setIdMgrLDAPGroupConfig ('[-id id1]')</code> • Using Jython list: <code>AdminTask.setIdMgrLDAPGroupConfig (['-id', 'id1'])</code> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <code>\$AdminTask setIdMgrLDAPGroupConfig {-interactive}</code> • Using Jython string: <code>AdminTask.setIdMgrLDAPGroupConfig ('[-interactive]')</code> • Using Jython list: <code>AdminTask.setIdMgrLDAPGroupConfig (['-interactive'])</code>

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<p>- scope</p> <p>The scope of the membership attribute. The following are the possible values for this parameter:</p> <ul style="list-style-type: none"> • direct - The membership attribute only contains direct groups. Direct groups contain the member and are not contained through a nested group. For example, if group1 contains group2, group2 contains user1, then group2 is a direct group of user1, but group1 is not a direct group of user1. • nested - The membership attribute contains both direct groups and nested groups. • all - The membership attribute contains direct groups, nested groups, and dynamic members. <p>The default value is <code>direct</code>. (String, optional)</p> <ul style="list-style-type: none"> • Returns: None 	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
<p>setIdMgr LDAPSearch ResultCache</p>	<p>The setIdMgr LDAPSearch ResultCache command sets up the LDAP search result cache configuration.</p>	<ul style="list-style-type: none"> • Parameters: - id The ID of the repository. (String, required) - cachesDiskOffLoad Loads the attributes caches and the search results onto hard disk. By default, when the number of cache entries reaches the maximum size of the cache, cache entries are evicted to allow new entries to enter the caches. If you enable this parameter, the evicted cache entries will be copied to disk for future access. The default value is false. (Boolean, optional) - enabled Enables the search results cache. The default value is true. (Boolean, optional) 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask setIdMgrLDAPSearch ResultCache {-id <i>id1</i>} • Using Jython string: AdminTask.setIdMgrLDAPSearch ResultCache (['-id <i>id1</i>']) • Using Jython list: AdminTask.setIdMgrLDAPSearch ResultCache (['-id', '<i>id1</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask setIdMgrLDAPSearch ResultCache {-interactive} • Using Jython string: AdminTask.setIdMgrLDAPSearch ResultCache (['-interactive']) • Using Jython list: AdminTask.setIdMgrLDAPSearch ResultCache (['-interactive'])

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<p>- cacheSize The maximum size of the search results cache. The number of naming enumeration objects that can be put into the search results cache. The minimum value of this parameter is 100. The default value is 2000. (Integer, optional)</p> <p>- cacheTimeout The amount of time in seconds before the cached entries in the search results cache can be not valid. The minimum value for this parameter is 0. A value of 0 means that the cached naming enumeration objects will stay in the search results cache until there are configuration changes. The default value is 600. (Integer, optional)</p>	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<p>- searchResultSizeLimit The maximum number of entries contained in the naming enumeration object that can be cached in the search results cache. For example, if the results from a search contains 2000 users, the search results will not cache in the search results cache if the value of the of this property is set to 1000. The default value is 1000. (Integer, optional)</p> <ul style="list-style-type: none"> • Returns: None 	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
<p>setIdMgr Entry Mapping Repository</p>	<p>The setIdMgr Entry Mapping Repository command sets or updates an entry mapping repository configuration.</p>	<ul style="list-style-type: none"> • Parameters: - dataSourceName The name of the data source. The default value is jdbc/wimDS. The parameter is required if the property extension is not set. The parameter is not required if the command is used to update the existing configuration. (String) - databaseType The type of the database. The default value is DB2. The parameter is required if the property extension is not set. The parameter is not required if the command is used to update the existing configuration. (String) - dbURL The URL of the database. The parameter is required if the property extension is not set. The parameter is not required if the command is used to update the existing configuration. (String) 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask setIdMgrEntry MappingRepository {-dbAdminId <i>dbid1</i> -dbAdminPassword <i>pw1</i>} • Using Jython string: AdminTask.setIdMgrEntry MappingRepository ('[-dbAdminId <i>dbid1</i> -dbAdminPassword <i>pw1</i>]') • Using Jython list: AdminTask.setIdMgrEntry MappingRepository (['-dbAdminId', '<i>dbid1</i>', '-dbAdmin Password', '<i>pw1</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask setIdMgrEntryMapping Repository {-interactive} • Using Jython string: AdminTask.setIdMgrEntryMapping Repository ('[-interactive]') • Using Jython list: AdminTask.setIdMgrEntryMapping Repository (['-interactive'])

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<ul style="list-style-type: none"> - dbAdminId The database administrator ID. (String, required if database type is not Cloudscape.) - dbAdminPassword The database administrator password. (String, required if database type is not Cloudscape.) - JDBCDriverClass The JDBC driver class name. (String, optional) • Returns: None 	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
<p>setIdMgr Property Extension Repository</p>	<p>The setIdMgr Property Extension Repository command sets or updates the property extension repository configuration.</p>	<ul style="list-style-type: none"> • Parameters: - dataSourceName The name of the data source. The default value is jdbc/wimDS. The parameter is required if the property extension is not set. The parameter is not required if the command is used to update the existing configuration. (String) - databaseType The type of the database. The default value is DB2. The parameter is required if the property extension is not set. The parameter is not required if the command is used to update the existing configuration. (String) - dbURL The URL of the database. The parameter is required if the property extension is not set. The parameter is not required if the command is used to update the existing configuration. (String) 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask setIdMgrProperty ExtensionRepository {-entity RetrievalLimit 10 -JDBC DriverClass classname} • Using Jython string: AdminTask.setIdMgrProperty ExtensionRepository ('[-entity RetrievalLimit 10 -JDBC DriverClass classname]') • Using Jython list: AdminTask.setIdMgrProperty ExtensionRepository (['-entity RetrievalLimit', '10', '-JDBCdriverClass', 'classname']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask setIdMgrProperty ExtensionRepository {-interactive} • Using Jython string: AdminTask.setIdMgrProperty ExtensionRepository (['-interactive']) • Using Jython list: AdminTask.setIdMgrPropertyExt ensionRepository (['-interactive'])

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<ul style="list-style-type: none"> - dbAdminId The database administrator ID. (String, required if database type is not Cloudscape.) - dbAdminPassword The database administrator password. (String, required if database type is not Cloudscape.) - entityRetrievalLimit The limit for the retrieval of entities. (Integer, required) - JDBCDriverClass The JDBC driver class name. (String, required) • Returns: None 	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
<p>updateId MgrDB Repository</p>	<p>The updateId MgrDB Repository command updates the configuration for the database repository that you specify.</p>	<ul style="list-style-type: none"> • Parameters: - id The ID of the repository. (String, required) - dataSourceName The name of the data source. The default value is jdbc/wimDS. (String, optional) - databaseType The type of the database. The default value is DB2. (String, optional) - dbURL The URL of the database. (String, optional) - dbAdminId The database administrator ID. (String, optional) - dbAdminPassword The database administrator password. (String, optional) - entityRetrievalLimit Indicates the value of the retrieval limit on database entries. The default value is 200. (Integer, optional) 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask updateIdMgrDB Repository {-id <i>id1</i>} • Using Jython string: AdminTask.updateIdMgrDB Repository ('[-id <i>id1</i>']') • Using Jython list: AdminTask.updateIdMgrDB Repository (['-id', '<i>id1</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask updateIdMgrDB Repository {-interactive} • Using Jython string: AdminTask.updateIdMgrDB Repository ('[-interactive]') • Using Jython list: AdminTask.updateIdMgrDB Repository (['-interactive'])

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<ul style="list-style-type: none"> <li data-bbox="737 302 977 422">- JDBCDriverClass The JDBC driver class name. (String, optional) <li data-bbox="737 436 977 579">- saltLength The salt length in bits. The default value is 12. (Integer, optional) <li data-bbox="737 594 977 737">- encryptionKey The default value is rZ15ws0e1y9yHk3zCs3sTMv/ho8fY17s. (String, optional) <li data-bbox="737 751 922 781">• Returns: None 	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
<p>updateId MgrFile Repository</p>	<p>The updateId MgrFile Repository command updates the configuration for the file repository that you specify. To update other properties of the file repository use the update IdMgr Repository command.</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - id The ID of the repository. (String, required) - messageDigest Algorithm The message digest algorithm that will be used for hashing the password. The default value is SHA-1. Valid values include the following: SHA-245, SHA-384, or SHA-512. (String, optional) - baseDirectory The base directory where the fill will be created in order to store the data. The default is to be dynamically built during run time using user.install.root and cell name. (String, optional) - fileName The file name of the repository. The default value is fileRegistry.xml. (String, optional) - saltLength The salt length of the randomly generated salt for password hashing. The default value is 12. (Integer, optional) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask updateIdMgrFile Repository {-id <i>id1</i>} • Using Jython string: AdminTask.updateIdMgrFile Repository ('[-id <i>id1</i>']') • Using Jython list: AdminTask.updateIdMgrFile Repository (['-id', '<i>id1</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask updateIdMgrFile Repository {-interactive} • Using Jython string: AdminTask.updateIdMgrFile Repository ('[-interactive]') • Using Jython list: AdminTask.updateIdMgrFile Repository (['-interactive'])

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
<p>updateId MgrLDAP AttrCache</p>	<p>The updateId MgrLDAP AttrCache command updates the LDAP attribute cache configuration.</p>	<ul style="list-style-type: none"> • Parameters: - id The ID of the repository. (String, required) - cachesDiskOffLoad (String, optional) - enabled Indicates if you want to enable attribute caching. The default value is true. (Boolean, optional) - cacheSize The maximum size of the attribute cache defined by the number of attribute objects that are permitted in the attribute cache. The minimum value of this parameter is 100. The default value is 4000. (Integer, optional) 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask updateIdMgrLDAP AttrCache {-id <i>id1</i>} • Using Jython string: AdminTask.updateIdMgrLDAP AttrCache ('[-id <i>id1</i>']') • Using Jython list: AdminTask.updateIdMgrLDAP AttrCache (['-id', '<i>id1</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask updateIdMgrLDAP AttrCache {-interactive} • Using Jython string: AdminTask.updateIdMgrLDAP AttrCache ('[-interactive]') • Using Jython list: AdminTask.updateIdMgrLDAP AttrCache (['-interactive'])

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<p>- cacheTimeOut The amount of time in seconds before the cached entries that are located in the attributes cache can be not valid. The minimum value of this parameter is 0. The attribute objects that are cached will remain in the attributes cache until the virtual member manager changes the attribute objects. The default value is 1200. (Integer, optional)</p> <p>- attributeSizeLimit An integer that represents the maximum number of attribute object values that can cache in the attributes cache.</p> <p>Some attributes, for example, the member attribute, contain many values. The attributeSizeLimit parameter prevents the attributes cache to cache large attributes. The default value is 2000. (Integer, optional)</p>	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<p>- serverTTLAttribute</p> <p>The name of the ttl attribute that is supported by the LDAP server. The attributes cache uses the value of this attribute to determine when the cached entries in the attributes cache will time out.</p> <p>The ttl attribute contains the time, in seconds, that any information from the entry should be kept by a client before it is considered stale and a new copy is fetched. A value of 0 implies that the object will not be cached.</p> <p>For more information about this attribute, go to:</p> <p>http://www.ietf.org/proceedings/98aug/I-D/draft-ietf-asid-ldap-cache-01.txt.</p> <p>The ttl attribute is not supported by all LDAP servers. If this attribute is supported by an LDAP server, you can set the value of the serverTTLAttribute parameter to the name of the ttl attribute in order to allow the value of the ttl attribute to determine when cached entries will time out. The time out value for different entries in attributes cache can be different.</p>	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<p>For example, if the value of the serverTTLAttribute parameter is ttl and the attributes cache retrieves attributes of a user from an LDAP server, it will also retrieve the value of the ttl attribute of this user. If the value is 200, the WMM uses this value to set the time out for the attributes of the user in the attributes cache instead of using the value of cacheTimeout. You can set different ttl attribute values for different users. (String, optional)</p> <ul style="list-style-type: none"> • Returns: None 	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
<p>updateId MgrLDAP ContextPool</p>	<p>The updateId MgrLDAP ContextPool command updates the LDAP context pool configuration.</p>	<ul style="list-style-type: none"> • Parameters: - id The ID of the repository. (String, required) - enabled By default, the context pool is enabled. If you set the value of this parameter to false, the context pool is disabled which means that a new context instance will be created for each request. The default value is true. (Boolean, optional) - initPoolSize The number of context instances that the virtual member manager LDAP adapter creates when it creates the pool. The valid range for this parameter is 1 to 50. The default value is 1. (Integer, optional) 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <code>\$AdminTask updateIdMgrLDAP ContextPool {-id id1}</code> • Using Jython string: <code>AdminTask.updateIdMgrLDAP ContextPool (['-id id1'])</code> • Using Jython list: <code>AdminTask.updateIdMgrLDAP ContextPool (['-id', 'id1'])</code> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <code>\$AdminTask updateIdMgrLDAP ContextPool {-interactive}</code> • Using Jython string: <code>AdminTask.updateIdMgrLDAP ContextPool (['-interactive'])</code> • Using Jython list: <code>AdminTask.updateIdMgrLDAP ContextPool (['-interactive'])</code>

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<p>- maxPoolSize</p> <p>The maximum number of context instances that can be maintained concurrently by the context pool. Both in-use and idle context instances contribute to this number. When the pool size reaches this number, new context instances cannot be created for new requests. The new request is blocked until a context instance is released by another request or is removed. The request checks periodically if there are context instances available in the pool according to the value defined for the poolWaitTime parameter. The minimum value of the maxPoolSize parameter is 0. There is no maximum value. A maximum pool size of 0 means that there is no maximum size and that a request for a pooled context instance will use an existing pooled idle context instance or a newly created pooled context instance. The default value is 20. (Integer, optional)</p>	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<p>- prefPoolSize The preferred number of context instances that the Context Pool should maintain. Both in-use and idle context instances contribute to this number. When there is a request for the use of a pooled context instance and the pool size is less than the preferred size, Context Pool will create and use a new pooled context instance regardless of whether an idle connection is available. When a request is finished with a pooled context instance and the pool size is greater than the preferred size, the Context Pool will close and remove the pooled context instance from the pool. The valid range of the prefPoolSize parameter is 0 to 100. A preferred pool size of 0 means that there is no preferred size: A request for a pooled context instance will result in a newly created context instance only if no idle ones are available. The default value is 3. (Integer, optional)</p>	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<p>- poolTimeOut An integer that represents the number of milliseconds that an idle context instance may remain in the pool without being closed and removed from the pool. When a context instance is requested from the pool, if this context already exists in the pool for more than the time defined by poolTimeout, this connection will be closed no matter this context instance is stale or active. A new context instance will be created and put back to the pool after it has been released from the request. The minimum value of poolTimeout is 0. There is no maximum value. A poolTimeout of 0 means that the context instances in the pool will remain in the pool until they are staled. In this case, Context Pool will catch the communication exception and recreate a new context instance. The default value is 0. (Integer, optional)</p>	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<p>- poolWaitTime The time interval (in milliseconds) that the request will wait until the Context Pool checks again if there are idle context instance available in the pool when the number of context instances reaches the maximum pool size. If there is still no idle context instance, the request will continue waiting for the same period of time until next checking. The minimum value of poolWaitout is 0. There is no maximum value. A poolWaitTime of 0 means the Context Pool will not check if there are idle context. Instead, the request will be notified when there is a context instance is released from other requests. The default value is 3000. (Integer, optional)</p> <ul style="list-style-type: none"> • Returns: None 	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
<p>updateId MgrLDAP EntityType</p>	<p>The updateId MgrLDAP EntityType command updates an existing LDAP entity type definition to LDAP repository configuration. You can use this command to add more values to multi-valued parameters. If the property already exists, the value of the property will be replaced. If the property does not exist, it will be added.</p>	<ul style="list-style-type: none"> • Parameters: - id The ID of the repository. (String, required) - name The name of the entity type. (String, required) - searchFilter The search filter that you want to use to search the entity type. (String, optional) - objectClasses One or more object classes for the entity type. (String, optional) - objectClassesForCreate The object class that will be when you create an entity type object. You do not have to specify the value of this parameter if it is the same as the value of the objectClasses parameter. (String, optional) - searchBases The search base or bases to use while searching the entity type. (String, optional) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <code>\$AdminTask updateIdMgrLDAPEntityType {-id <i>id1</i> -name <i>name1</i>}</code> • Using Jython string: <code>AdminTask.updateIdMgrLDAPEntityType ('[-id <i>id1</i> -name <i>name1</i>]')</code> • Using Jython list: <code>AdminTask.updateIdMgrLDAPEntityType (['-id', '<i>id1</i>', '-name', '<i>name1</i>'])</code> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <code>\$AdminTask updateIdMgrLDAPEntityType {-interactive}</code> • Using Jython string: <code>AdminTask.updateIdMgrLDAPEntityType ('[-interactive]')</code> • Using Jython list: <code>AdminTask.updateIdMgrLDAPEntityType (['-interactive'])</code>

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
<p>updateIdMgr LDAPGroup Dynamic MemberAttr</p>	<p>The updateIdMgr LDAPGroup Dynamic MemberAttr command updates a dynamic member attribute configuration to an LDAP group configuration.</p>	<ul style="list-style-type: none"> • Parameters: - id The ID of the repository. (String, required) - name The name of the LDAP attribute that is used as the group member attribute. For example, memberURL. (String, required) - objectClass The group object class that contains the dynamic member attribute. For example groupOfURLs. If you do not define this parameter, the dynamic member attribute will apply to all group object classes. (String, required) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask updateIdMgrLDAPGroup DynamicMemberAttr {-id <i>id1</i> -name <i>name1</i> -objectClass <i>groupOfURLs</i>} • Using Jython string: AdminTask.updateIdMgrLDAPGroup DynamicMemberAttr (['-id <i>id1</i> -name <i>name1</i> -objectClass <i>groupOfURLs</i>']) • Using Jython list: AdminTask.updateIdMgrLDAPGroup DynamicMemberAttr (['-id', '<i>id1</i>', '-name', '<i>name1</i>', '-objectClass', '<i>groupOfURLs</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask updateIdMgrLDAPGroup DynamicMemberAttr {-interactive} • Using Jython string: AdminTask.updateIdMgrLDAPGroup DynamicMemberAttr (['-interactive']) • Using Jython list: AdminTask.updateIdMgrLDAPGroup DynamicMemberAttr (['-interactive'])

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
updateIdMgr LDAPGroup MemberAttr	<p>The updateIdMgr LDAPGroup MemberAttr command updates a member attribute configuration of an LDAP group configuration.</p>	<ul style="list-style-type: none"> • Parameters: - id The ID of the repository. (String, required) - name The name of the LDAP attribute that is used as the group member attribute. For example, member or uniqueMember. (String, required) - objectClass The group object class that contains the member attribute. For example, groupOfNames or groupOfUniqueNames. If you do not define this parameter, the member attribute applies to all group object classes. (String, optional) 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask updateIdMgrLDAPGroupMemberAttr {-id <i>id1</i> -name <i>name1</i>} • Using Jython string: AdminTask.updateIdMgrLDAPGroupMemberAttr ('[-id <i>id1</i> -name <i>name1</i>]') • Using Jython list: AdminTask.updateIdMgrLDAPGroupMemberAttr (['-id', '<i>id1</i>', '-name', '<i>name1</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask updateIdMgrLDAPGroupMemberAttr {-interactive} • Using Jython string: AdminTask.updateIdMgrLDAPGroupMemberAttr ('[-interactive]') • Using Jython list: AdminTask.updateIdMgrLDAPGroupMemberAttr (['-interactive'])

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<p>- scope</p> <p>The scope of the member attribute. The following are the valid values:</p> <ul style="list-style-type: none"> • direct - The member attribute only contains direct members whereby the member is directly contained by the group and not contained in a nested group. For example, if group1 contains group2, group2 contains user1, then group2 is a direct member of group1 but user1 is not a direct member of group1. Both member and uniqueMember are direct member attributes. • nested - The member attribute contains both direct members and nested members. 	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<ul style="list-style-type: none"> • all - The member attribute contains direct members, nested members and dynamic members. One example is the <code>ibm-allMembers</code> attribute that IBM Tivoli Directory Server supports. <p>The default value is <code>direct</code>. (String, optional)</p> <p>- dummyMember When you create a group without specifying a member, a dummy member will be filled in automatically to avoid receiving an exception that indicates that there is a mandatory attribute missing. (String, optional)</p> <ul style="list-style-type: none"> • Returns: None 	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
<p>updateId MgrLDAP Repository</p>	<p>The updateId MgrLDAP Repository command updates an LDAP repository configuration.</p>	<ul style="list-style-type: none"> • Parameters: - id The ID of the repository. (String, required) - ldapServerType The type of LDAP server that is being used. The default value is IDS51. (String, optional) - adapterClassName The default value is com.ibm.ws.wim.adapter.ldap.LdapAdapter. (String, optional) - certificateMapMode Specifies whether to map X.509 certificates into a LDAP directory by exact distinguished name or by certificate filter. The default value is exactdn. To use the certificate filter for the mapping, specify certificatefilter. (String, optional) 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask updateIdMgrLDAP Repository {-id <i>id1</i>} • Using Jython string: AdminTask.updateIdMgrLDAP Repository ('[-id <i>id1</i>']') • Using Jython list: AdminTask.updateIdMgrLDAP Repository (['-id', '<i>id1</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask updateIdMgrLDAP Repository {-interactive} • Using Jython string: AdminTask.updateIdMgrLDAP Repository ('[-interactive]') • Using Jython list: AdminTask.updateIdMgrLDAP Repository (['-interactive'])

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<p>- certificateFilter If certificateMapMode has the value certificatefilter, then this property specifies the LDAP filter which maps attributes in the client certificate to entries in LDAP. (String, optional)</p> <p>- isExtIdUnique Specifies if the external ID is unique. The default value is true. (Boolean, optional)</p> <p>- loginProperties Indicates the property name used for login. (String , optional)</p> <p>-</p> <p>primaryServerQueryTimeInterval Indicates the polling interval for testing the primary server availability. The value of this parameter is specified in minutes. The default value is 15. (Integer, optional)</p>	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<ul style="list-style-type: none"> - returnToPrimaryServer Indicates to return to the primary LDAP server when it is available. The default value is true. (Boolean, optional) - supportAsyncMode Indicates if the async mode is supported or not. The default value is false. (Boolean, optional) - supportSorting Indicates if sorting is supported or not. The default value is false. (Boolean, optional) - supportPaging Indicates if paging is supported or not. The default value is false. (Boolean, optional) - supportTransaction Indicates if transactions are supported or not. The default value is false. (Boolean, optional) 	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<ul style="list-style-type: none"> - supportExternalName Indicates if external names are supported or not. The default value is <code>false</code>. (Boolean, optional) - sslConfiguration The SSL configuration. (String, optional) - translateRDN Indicates to translate RDN or not. The default value is <code>false</code>. (Boolean, optional) - searchTimeLimit The value of search time limit. (Integer, optional) - searchCountLimit The value of search count limit. (Integer, optional) - searchPageSize The value of search page size. (Integer, optional) • Returns: None 	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
<p>updateIdMgr LDAPSearch ResultCache</p>	<p>The updateIdMgr LDAPSearch ResultCache command updates the LDAP search result cache configuration.</p>	<ul style="list-style-type: none"> • Parameters: - id The ID of the repository. (String, required) - cachesDiskOffLoad Loads the attributes caches and the search results onto hard disk. By default, when the number of cache entries reaches the maximum size of the cache, cache entries are evicted to allow new entries to enter the caches. If you enable this parameter, the evicted cache entries will be copied to disk for future access. The default value is false. (Boolean, optional) - enabled Enables the search results cache. The default value is true. (Boolean, optional) 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask updateIdMgrLDAP SearchResultCache {-id id1} • Using Jython string: AdminTask.updateIdMgrLDAPSearch ResultCache ('[-id id1]') • Using Jython list: AdminTask.updateIdMgrLDAPSearch ResultCache (['-id', 'id1']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask updateIdMgrLDAP SearchResultCache {-interactive} • Using Jython string: AdminTask.updateIdMgrLDAPSearch ResultCache ('[-interactive]') • Using Jython list: AdminTask.updateIdMgrLDAPSearch ResultCache (['-interactive'])

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<p>- cacheSize The maximum size of the search results cache. The number of naming enumeration objects that can be put into the search results cache. The minimum value of this parameter is 100. The default value is 2000. (Integer, optional)</p> <p>- cacheTimeOut The amount of time in seconds before the cached entries in the search results cache can be not valid. The minimum value for this parameter is 0. A value of 0 means that the cached naming enumeration objects will stay in the search results cache until there are configuration changes. The default value is 600. (Integer, optional)</p>	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<p>- searchResultSizeLimit The maximum number of entries contained in the naming enumeration object that can be cached in the search results cache. For example, if the results from a search contains 2000 users, the search results will not cache in the search results cache if the value of the of this property is set to 1000. The default value is 1000. (Integer, optional)</p> <ul style="list-style-type: none"> • Returns: None 	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
updateIdMgr LDAPServer	<p>The updateIdMgr LDAPServer command updates an LDAP server configuration for the LDAP repository ID that you specify.</p>	<ul style="list-style-type: none"> • Parameters: - id The ID of the repository. (String, required) - host The host name for the LDAP server that contains the properties that you want to modify. (String, required) - port The port number for the LDAP server. (Integer, optional) - authentication Indicates the authentication method to use. The default value is simple. Valid values include: none or strong. (String, optional) - bindDN The binding domain name for the LDAP server. (String, optional) - bindPassword The binding password. The password is encrypted before it is stored. (String, optional) 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <code>\$AdminTask updateIdMgrLDAPServer {-id <i>id1</i> -host <i>myhost.ibm.com</i>}</code> • Using Jython string: <code>AdminTask.updateIdMgrLDAPServer ('[-id <i>id1</i> -host <i>myhost.ibm.com</i>']')</code> • Using Jython list: <code>AdminTask.updateIdMgrLDAPServer (['-id', '<i>id1</i>', '-host', '<i>myhost.ibm.com</i>'])</code> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <code>\$AdminTask updateIdMgrLDAPServer {-interactive}</code> • Using Jython string: <code>AdminTask.updateIdMgrLDAPServer ('[-interactive]')</code> • Using Jython list: <code>AdminTask.updateIdMgrLDAPServer (['-interactive'])</code>

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<p>-</p> <p>certificateMapMode Specifies whether to map X.509 certificates into a LDAP directory by exact distinguished name or by certificate filter. The default value is exactdn. To use the certificate filter for the mapping, specify certificatefilter. (String, optional)</p> <p>- certificateFilter If certificateMapMode has the value certificatefilter, then this property specifies the LDAP filter which maps attributes in the client certificate to entries in LDAP. (String, optional)</p> <p>- connectTimeout The connection timeout measured in seconds. The default value is 0. (Integer, optional)</p> <p>- connectionPool The connection pool. The default value is false. (Boolean, optional)</p>	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<p>- derefAliases Controls how aliases are dereferenced. The default value is always. Valid values include:</p> <ul style="list-style-type: none"> • never - never deference aliases • finding - deferences aliases only during name resolution • searching - deferences aliases only after name resolution <p>(String, optional)</p> <p>- ldapServerType The type of LDAP server being used. The default value is IDS51. (String, optional)</p> <p>- primary_host The host name for the primary LDAP server. (String, optional)</p> <p>- referral The LDAP referral. The default value is ignore. Valid values include: follow, throw, or false. (String, optional)</p>	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<ul style="list-style-type: none"><li data-bbox="737 302 964 422">- sslConfiguration The SSL configuration. (String, optional)<li data-bbox="737 436 964 638">- sslEnabled Indicates to enable SSL or not. The default value is false. (Boolean, optional)<li data-bbox="737 646 922 674">• Returns: None	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
updateIdMgr Repository	The updateIdMgr Repository command updates the common repository configuration.	<ul style="list-style-type: none"> • Parameters: - id The ID of the repository. (String, required) - adapterClassName The implementation class name for the repository adapter. (String, optional) - EntityTypesNotAllowCreate The name of the entity type that should not be created in this repository. (String, optional) - EntityTypesNotAllowUpdate The name of the entity type that should not be updated in this repository. (String, optional) - EntityTypesNotAllowRead The name of the entity type that should not be read from this repository. (String, optional) 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask updateIdMgrRepository {-id <i>id1</i>} • Using Jython string: AdminTask.updateIdMgrRepository ('[-id <i>id1</i>']') • Using Jython list: AdminTask.updateIdMgrRepository (['-id', '<i>id1</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask updateIdMgrRepository {-interactive} • Using Jython string: AdminTask.updateIdMgrRepository ('[-interactive]') • Using Jython list: AdminTask.updateIdMgrRepository (['-interactive'])

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<ul style="list-style-type: none"> - EntityTypesNotAllowDelete The name of the entity type that should not be deleted from this repository. (String, optional) - loginProperties (String, optional) - readOnly Indicates if this is a read only repository. The default value is false. (Boolean, optional) - repositoriesForGroups The repository ID where group data is stored. (String, optional) - supportPaging Indicates if the repository supports paging or not. (Boolean, optional) - supportSorting Indicates if the repository supports sorting or not. (Boolean, optional) 	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<ul style="list-style-type: none"> - supportTransaction Indicates if the repository supports transaction or not. (Boolean, optional) - isExtIdUnique Specifies if the external ID is unique or not. (Boolean, optional) - supportedExternalName Indicates if the repository supports external names or not. (Boolean, optional) - supportAsyncMode Indicates if the adapter supports async mode or not. The default value is false. (Boolean, optional) <ul style="list-style-type: none"> • Returns: None 	

Table 17. (continued)

Command name:	Description:	Parameters and return values:	Examples:
updateIdMgr Repository BaseEntry	The updateIdMgr Repository BaseEntry command updates a base entry to the specified repository.	<ul style="list-style-type: none"> • Parameters: - id The ID of the repository. (String, required) - name The distinguished name of a base entry. (String, required) - nameInRepository The distinguished name in the repository that uniquely identifies the base entry name. (String, optional) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <code>\$AdminTask updateIdMgrRepositoryBaseEntry {-id <i>id1</i> name <i>name1</i>}</code> • Using Jython string: <code>AdminTask.updateIdMgrRepositoryBaseEntry ('[-id <i>id1</i> name <i>name1</i>']')</code> • Using Jython list: <code>AdminTask.updateIdMgrRepositoryBaseEntry (['-id', '<i>id1</i>', 'name', '<i>name1</i>'])</code> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <code>\$AdminTask updateIdMgrRepositoryBaseEntry {-interactive}</code> • Using Jython string: <code>AdminTask.updateIdMgrRepositoryBaseEntry ('[-interactive]')</code> • Using Jython list: <code>AdminTask.updateIdMgrRepositoryBaseEntry (['-interactive'])</code>

Commands for the IdMgrRealmConfig group of the AdminTask object

Use the commands in the IdMgrConfig group to configure the member manager realm and realms. The commands for this group do not require a target object. To see the additional commands related to the member manager, see the `Commands for the IdMgrRepositoryConfig group of the AdminTask object` and the `Commands for the IdMgrConfig group of the AdminTask object` articles.

For more information about the AdminTask object, see the `Commands for the AdminTask object` article.

The following commands are available for the IdMgrRealmConfig group of the AdminTask object:

Table 18.

Command name:	Description:	Parameters and return values:	Examples:
<p>addIdMgrRealmBaseEntry</p>	<p>The addIdMgrRealmBaseEntry command adds a base entry to a specified realm configuration.</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - name The realm name. (String, required) - baseEntry The name of a base entry. (String, required) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask addIdMgrRealmBaseEntry {-name realm1 -baseEntry entry1}</pre> • Using Jython string: <pre>AdminTask addIdMgrRealmBaseEntry ('[-name realm1 -baseEntry entry1]')</pre> • Using Jython list: <pre>AdminTask addIdMgrRealmBaseEntry (['-name', 'realm1', '-base Entry', 'entry1'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask addIdMgrRealm BaseEntry {-interactive}</pre> • Using Jython string: <pre>AdminTask.addIdMgrReaImBase Entry ('[-interactive]')</pre> • Using Jython list: <pre>AdminTask.addIdMgrReaImBase Entry (['-interactive'])</pre>

Table 18. (continued)

Command name:	Description:	Parameters and return values:	Examples:
createIdMgrRealm	<p>The createIdMgrRealm command creates a realm configuration.</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - name The realm name. (String, required) - securityUse A string that indicates if this virtual realm will be used in security now, later, or never. The default value is active. Additional values includes: inactive and nonSelectable. (String, optional) - delimiter The delimiter used for this realm. The default value is @. (String, optional) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask createIdMgrRealm {-name <i>realm1</i>} • Using Jython string: AdminTask.createIdMgrRealm ('[-name <i>realm1</i>']') • Using Jython list: AdminTask.createIdMgrRealm (['-name', '<i>realm1</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask createIdMgrRealm {-interactive} • Using Jython string: AdminTask.createIdMgrRealm ('[-interactive]') • Using Jython list: AdminTask.createIdMgrRealm (['-interactive'])
deleteIdMgrRealm	<p>The deleteIdMgrRealm command deletes the realm configuration that you specified.</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - name The realm name. (String, required) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask deleteIdMgrRealm {-name <i>realm1</i>} • Using Jython string: AdminTask.deleteIdMgrRealm ('[-name <i>realm1</i>']') • Using Jython list: AdminTask.deleteIdMgrRealm (['-name', '<i>realm1</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask deleteIdMgrRealm {-interactive} • Using Jython string: AdminTask.deleteIdMgrRealm ('[-interactive]') • Using Jython list: AdminTask.deleteIdMgrRealm (['-interactive'])

Table 18. (continued)

Command name:	Description:	Parameters and return values:	Examples:
<p>deleteIdMgr RealmBaseEntry</p>	<p>The deleteIdMgr RealmBaseEntry command deletes a base entry from a realm configuration that you specified.</p> <p>The realm must always contain at least one base entry, thus you cannot remove every entry.</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - name The realm name. (String, required) - baseEntry The name of a base entry. (String, required) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask deleteIdMgrRealmBaseEntry {-name realm1 -baseEntry entry1}</pre> • Using Jython string: <pre>AdminTask.deleteIdMgrRealmBaseEntry (['-name realm1 -baseEntry entry1'])</pre> • Using Jython list: <pre>AdminTask.deleteIdMgrRealmBaseEntry (['-name', 'realm1', '-baseEntry', 'entry1'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask deleteIdMgrRealmBaseEntry {-interactive}</pre> • Using Jython string: <pre>AdminTask.deleteIdMgrRealmBaseEntry (['-interactive'])</pre> • Using Jython list: <pre>AdminTask.deleteIdMgrRealmBaseEntry (['-interactive'])</pre>
<p>getIdMgrDefaultRealm</p>	<p>The getIdMgrDefaultRealm command returns the default realm name.</p>	<ul style="list-style-type: none"> • Parameters: None • Returns: The name of the default realm. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask getIdMgrDefaultRealm</pre> • Using Jython string: <pre>AdminTask.getIdMgrDefaultRealm()</pre> • Using Jython list: <pre>AdminTask.getIdMgrDefaultRealm()</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask getIdMgrDefaultRealm {-interactive}</pre> • Using Jython string: <pre>AdminTask.getIdMgrDefaultRealm (['-interactive'])</pre> • Using Jython list: <pre>AdminTask.getIdMgrDefaultRealm (['-interactive'])</pre>

Table 18. (continued)

Command name:	Description:	Parameters and return values:	Examples:
<p>getIdMgrRepositories ForRealm</p>	<p>The getIdMgrRepositories ForRealm command returns repository specific details for the repositories configured for a specified realm.</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - name The realm name. (String, required) • Returns: A hash map with the following keys: <ul style="list-style-type: none"> - id - The repository ID. - repositoryType - The type of repository, for example, File, LDAP, DB, and so on. - specificRepositoryType - The specific type of repository, for example, for LDAP, IDS51, NDS, and so on. - host - The host name where the repository resides. For example, for File, LocalHost or for DB, dataSourceName. - port - The port number. This only applies to LDAP. - name - The name of the base entry. - nameInRepository - The name in repository for the base entry. <p>This command will not return the property extension and entry mapping repository data.</p>	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask getIdMgrRepositories ForRealm {-name realm1}</pre> • Using Jython string: <pre>AdminTask.getIdMgrRepositories ForRealm ('[-name realm1]')</pre> • Using Jython list: <pre>AdminTask.getIdMgrRepositories ForRealm (['-name', 'realm1'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask getIdMgrRepositories ForRealm {-interactive}</pre> • Using Jython string: <pre>AdminTask.getIdMgrRepositories ForRealm (['-interactive'])</pre> • Using Jython list: <pre>AdminTask.getIdMgrRepositories ForRealm (['-interactive'])</pre>

Table 18. (continued)

Command name:	Description:	Parameters and return values:	Examples:
getIdMgrRealm	The getIdMgrRealm command returns the configuration parameters for the realm that you specified.	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - name The realm name. (String, required) • Returns: A hash map that contains keys as the parameters of the createIdMgrRealm command. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask getIdMgrRealm {-name <i>realm1</i>} • Using Jython string: AdminTask.getIdMgrRealm ('[-name <i>realm1</i>']') • Using Jython list: AdminTask.getIdMgrRealm (['-name', '<i>realm1</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask getIdMgrRealm {-interactive} • Using Jython string: AdminTask.getIdMgrRealm ('[-interactive]') • Using Jython list: AdminTask.getIdMgrRealm (['-interactive'])
listIdMgrRealms	The listIdMgrRealms command returns all of the names of the configured realms.	<ul style="list-style-type: none"> • Parameters: None • Returns: A list that contains the name of the configured realms. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask listIdMgrRealms • Using Jython string: AdminTask.listIdMgrRealms() • Using Jython list: AdminTask.listIdMgrRealms() <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask listIdMgrRealms {-interactive} • Using Jython string: AdminTask.listIdMgrRealms ('[-interactive]') • Using Jython list: AdminTask.listIdMgrRealms (['-interactive'])

Table 18. (continued)

Command name:	Description:	Parameters and return values:	Examples:
listIdMgrRealmBaseEntries	The listIdMgrRealmBaseEntries command returns all of the names of the configured realms.	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - name The realm name. (String, required) • Returns: A list of all the base entries of the specified realm. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask listIdMgrRealmBaseEntries {-name <i>realm1</i>} • Using Jython string: AdminTask.listIdMgrRealmBaseEntries ('[-name <i>realm1</i>']') • Using Jython list: AdminTask.listIdMgrRealmBaseEntries (['-name', '<i>realm1</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask listIdMgrRealmBaseEntries {-interactive} • Using Jython string: AdminTask.listIdMgrRealmBaseEntries ('[-interactive]') • Using Jython list: AdminTask.listIdMgrRealmBaseEntries (['-interactive'])
renameIdMgrRealm	The renameIdMgrRealm command renames the name of the realm that you specified.	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - name The realm name. (String, required) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask renameIdMgrRealm {-name <i>realm1</i>} • Using Jython string: AdminTask.renameIdMgrRealm ('[-name <i>realm1</i>']') • Using Jython list: AdminTask.renameIdMgrRealm (['-name', '<i>realm1</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask renameIdMgrRealm {-interactive} • Using Jython string: AdminTask.renameIdMgrRealm ('[-interactive]') • Using Jython list: AdminTask.renameIdMgrRealm (['-interactive'])

Table 18. (continued)

Command name:	Description:	Parameters and return values:	Examples:
setIdMgrDefaultRealm	The setIdMgrDefaultRealm command sets up the default realm configuration.	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - name The name of the realm that is used as a default realm when the caller does not specify any in context. (String, required) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask setIdMgrDefaultRealm {-name <i>realm1</i>} • Using Jython string: AdminTask.setIdMgrDefaultRealm ('[-name <i>realm1</i>']') • Using Jython list: AdminTask.setIdMgrDefaultRealm (['-name', '<i>realm1</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask setIdMgrDefaultRealm {-interactive} • Using Jython string: AdminTask.setIdMgrDefaultRealm ('[-interactive]') • Using Jython list: AdminTask.setIdMgrDefaultRealm (['-interactive'])

Table 18. (continued)

Command name:	Description:	Parameters and return values:	Examples:
updateIdMgrRealm	<p>The updateIdMgrRealm command updates the configuration for a realm that you specify.</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - name The realm name. (String, required) - securityUse A string that indicates if this realm will be used in security now, later, or never. The default value is active. Additional values includes: inactive and nonSelectable. (String, optional) - delimiter The delimiter used for this realm. The default value is @. (String, optional) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask updateIdMgrRealm {-name <i>realm1</i>} • Using Jython string: AdminTask.updateIdMgrRealm ('[-name <i>realm1</i>]') • Using Jython list: AdminTask.updateIdMgrRealm (['-name', '<i>realm1</i>']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask updateIdMgrRealm {-interactive} • Using Jython string: AdminTask.updateIdMgrRealm ('[-interactive]') • Using Jython list: AdminTask.updateIdMgrRealm (['-interactive'])

Commands for the WIMManagementCommands group of the AdminTask object

Use the commands in the WIMManagementCommands group to configure the virtual member manager. The commands for this group do not require a target object. For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the WIMManagementCommands group of the AdminTask object:

Table 19.

Command name:	Description:	Parameters and return values:	Examples:
addMemberToGroup	<p>The addMemberToGroup command adds a user or a group to a group.</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - memberUniqueName Specifies the unique name value for the user or group that you want to add to the specified group. This parameter maps to the uniqueName property in the virtual member manager. (String, required) - groupUniqueName Specifies the unique name value for the group to which you want to add the user or group that you specified in the memberUniqueName parameter. This parameter maps to the uniqueName property in virtual member manager. (String, required) • Returns: Void if the command is successful. Returns an exception if the command fails. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask addMemberToGroup {-memberUniqueName uid= meyersd,cn=users,dc=yourco, dc=com -groupUniqueName cn=admins,cn=groups, dc=yourco,dc=com}</pre> • Using Jython string: <pre>AdminTask.addMemberToGroup (['-memberUniqueName uid=meyersd,cn=users, dc=yourco,dc=com -group UniqueName cn=admins, cn=groups,dc=yourco, dc=com']')</pre> • Using Jython list: <pre>AdminTask.addMemberToGroup (['-memberUniqueName', 'uid=meyersd,cn=users,dc= yourco,dc=com', '-group UniqueName', 'cn=admins, cn=groups,dc=yourco,dc=com'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask addMemberToGroup {-interactive}</pre> • Using Jython string: <pre>AdminTask.addMemberToGroup (['-interactive']')</pre> • Using Jython list: <pre>AdminTask.addMemberToGroup (['-interactive'])</pre>

Table 19. (continued)

Command name:	Description:	Parameters and return values:	Examples:
createGroup	<p>The createGroup command creates a new group in the virtual member manager. After the command completes, the new group will appear in the repository. For LDAP, a group must contain a member. The <code>memberUniqueName</code> parameter is optional in this case. If you set the <code>memberUniqueName</code> parameter to the unique name of a group or a user, the group or user will be added as a member of the group.</p>	<ul style="list-style-type: none"> • Parameters: <li style="margin-left: 20px;">- cn Specifies the common name for the group that you want to create. This parameter maps to the <code>cn</code> property in virtual member manager. (String, required) <li style="margin-left: 20px;">- description Specifies additional information about the group that you want to create. This parameter maps to the <code>description</code> property in a virtual member manager object. (String, optional) <li style="margin-left: 20px;">- parent Specifies the repository in which you want to create the group. This parameter maps to the <code>parent</code> property in the virtual member manager. (String, optional) <li style="margin-left: 20px;">- memberUniqueName Specifies the unique name value for the user or group that you want to add to the new group. This parameter maps to the <code>uniqueName</code> property in the virtual member manager. (String, optional) • Returns: The unique name of the group that you created. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <code>\$AdminTask createGroup {-cn groupA -description a group of admins}</code> • Using Jython string: <code>AdminTask.createGroup (['-cn groupA -description a group of admins'])</code> • Using Jython list: <code>AdminTask.createGroup (['-cn', 'groupA', '-description', 'a group of admins'])</code> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <code>\$AdminTask createGroup {-interactive}</code> • Using Jython string: <code>AdminTask.createGroup (['-interactive'])</code> • Using Jython list: <code>AdminTask.createGroup (['-interactive'])</code>

Table 19. (continued)

Command name:	Description:	Parameters and return values:	Examples:
createUser	<p>The createUser command creates a new user in the default repository or a repository that the parent command parameter specifies. This command creates a person entity and a login account entity in the virtual member manager.</p>	<ul style="list-style-type: none"> • Parameters: - uid Specifies the unique ID for the user that you want to create. Virtual member manager then creates a uniqueId value and a uniqueName value for the user. This parameter maps to the uid property in the virtual member manager. (String, required) - password Specifies the password for the user. This parameter maps to the password property in the virtual member manager. (String, required) - confirmPassword Specifies the password again to validate how it was entered for the password parameter. This parameter maps to the password property in virtual member manager. (String, optional) - cn Specifies the first name or given name of the user. This parameter maps to the cn property in virtual member manager. (String, optional) 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask createUser {-uid 123 -password tempPass -confirmPassword tempPass -cn Jane -surname Doe -ibm-primaryEmail janedoe@acme.com}</pre> • Using Jython string: <pre>AdminTask.createUser ('[-uid 123 -password tempPass -confirmPassword tempPass -cn Jane -surname Doe -ibm-primaryEmail janedoe@acme.com]')</pre> • Using Jython list: <pre>AdminTask.createUser (['-uid', '123', '-password', 'tempPass', '-confirmPassword', 'tempPass', '-cn', 'Jane', '-surname', 'Doe', '-ibm-primaryEmail', 'janedoe@acme.com'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask createUser {-interactive}</pre> • Using Jython string: <pre>AdminTask.createUser ('[-interactive]')</pre> • Using Jython list: <pre>AdminTask.createUser (['-interactive'])</pre>

Table 19. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<ul style="list-style-type: none"> <li data-bbox="721 304 1029 562"> <p>- surname Specifies the last name or family name of the user. This parameter maps to the sn property in virtual member manager. (String, optional)</p> <li data-bbox="721 583 1029 842"> <p>- ibm-primaryEmail Specifies the e-mail address of the user. This parameter maps to the ibm-PrimaryEmail property in the virtual member manager. (String, optional)</p> <li data-bbox="721 863 1029 1121"> <p>- parent Specifies the repository in which you want to create the user. This parameter maps to the parent property in the virtual member manager. (String, optional)</p> <li data-bbox="721 1142 1029 1209"> <p>• Returns: The unique name of the user that you created.</p> 	

Table 19. (continued)

Command name:	Description:	Parameters and return values:	Examples:
deleteGroup	<p>The deleteGroup command deletes a group in the virtual member manager. You cannot use this command to delete descendants. When this command completes, the group will be deleted from the repository.</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - uniqueName Specifies the unique name value for the group that you want to delete. This parameter maps to the uniqueName property in virtual member manager. (String, required) • Returns: Void if the command is successful. Returns an error if the command fails. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask deleteGroup {-uniqueName cn=opera tors,cn=users,dc=yourco, dc=com}</pre> • Using Jython string: <pre>AdminTask.deleteGroup (['-uniqueName cn=ope rators,cn=users,dc=you rco,dc=com']')</pre> • Using Jython list: <pre>AdminTask.deleteGroup (['-uniqueName', 'cn =operators,cn=users,dc =yourco,dc=com']')</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask deleteGroup {-interactive}</pre> • Using Jython string: <pre>AdminTask.deleteGroup (['-interactive']')</pre> • Using Jython list: <pre>AdminTask.deleteGroup (['-interactive']')</pre>

Table 19. (continued)

Command name:	Description:	Parameters and return values:	Examples:
deleteUser	<p>The deleteUser command deletes a user from the virtual member manager. This includes a person object and an account object in the non-merged repositories.</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - uniqueName Specifies the unique name value for the user that you want to delete. This parameter maps to the uniqueName property in virtual member manager. (String, required) • Returns: Void if the command is successful and an exception if the command fails. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask deleteUser {-uniqueName uid=dmeyers,cn=users,dc=yourco,dc=com}</pre> • Using Jython string: <pre>AdminTask.deleteUser ('[-uniqueName uid=dmeyers,cn=users,dc=yourco,dc=com]')</pre> • Using Jython list: <pre>AdminTask.deleteUser (['-uniqueName', 'uid=dmeyers,cn=users,dc=yourco,dc=com'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask deleteUser {-interactive}</pre> • Using Jython string: <pre>AdminTask.deleteUser ('[-interactive]')</pre> • Using Jython list: <pre>AdminTask.deleteUser (['-interactive'])</pre>

Table 19. (continued)

Command name:	Description:	Parameters and return values:	Examples:
<p>duplicate MembershipOfGroup</p>	<p>Use the duplicate MembershipOfGroup command to make a one group a member of all of the same groups as another group. For example, group A is in group B and group C. To add group D to the same groups as group A, use the duplicate MembershipOfGroup command.</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - copyToName Specifies the name of the group to which you want to add the memberships of the group specified in the copyFromName parameter. (String, required) - copyFromName Specifies the name of the group from which you want to copy the group memberships for another group to use. (String, required) • Returns: Void if the command is successful. Returns an exception if the command fails. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask duplicateMembershipOfGroup {-copyToName cn=operators,cn=groups,dc=yourco,dc=com -copyFromName cn=admins,cn=groups,dc=yourco,dc=com}</pre> • Using Jython string: <pre>AdminTask.duplicateMembershipOfGroup ('[-copyToName cn=operators,cn=groups,dc=yourco,dc=com -copyFromName cn=admins,cn=groups,dc=yourco,dc=com]')</pre> • Using Jython list: <pre>AdminTask.duplicateMembershipOfGroup (['-copyToName', 'cn=operators,cn=groups,dc=yourco,dc=com', '-copyFromName', 'cn=admins,cn=groups,dc=yourco,dc=com'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask duplicateMembershipOfGroup {-interactive}</pre> • Using Jython string: <pre>AdminTask.duplicateMembershipOfGroup ('[-interactive]')</pre> • Using Jython list: <pre>AdminTask.duplicateMembershipOfGroup (['-interactive'])</pre>

Table 19. (continued)

Command name:	Description:	Parameters and return values:	Examples:
<p>duplicate Membership OfUser</p>	<p>Use the duplicate Membership OfUser command to make a one user a member of all of the same groups as another user. For example, user 1 is in group B and group C. To add user 2 to the same groups as user 1, use the duplicate Membership OfUser command.</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - copyToName Specifies the name of the user to which you want to add the memberships of the user specified in the copyFromName parameter. (String, required) - copyFromName Specifies the name of the user from which you want to copy the group memberships for another user to use. (String, required) • Returns: Void if the command is successful. Returns an exception if the command fails. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask duplicateMembershipOfUser {-copyToName uid=meyersd,cn=users,dc=yourco,dc=com -copyFromName uid=jhart,cn=users,dc=yourco,dc=com}</pre> • Using Jython string: <pre>AdminTask.duplicateMembershipOfUser ('[-copyToName uid=meyersd,cn=users,dc=yourco,dc=com -copyFromName uid=jhart,cn=users,dc=yourco,dc=com]')</pre> • Using Jython list: <pre>AdminTask.duplicateMembershipOfUser (['-copyToName', 'uid=meyersd,cn=users,dc=yourco,dc=com', '-copyFromName', 'uid=jhart,cn=users,dc=yourco,dc=com'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask duplicateMembershipOfUser {-interactive}</pre> • Using Jython string: <pre>AdminTask.duplicateMembershipOfUser ('[-interactive]')</pre> • Using Jython list: <pre>AdminTask.duplicateMembershipOfUser (['-interactive'])</pre>

Table 19. (continued)

Command name:	Description:	Parameters and return values:	Examples:
getGroup	<p>The getGroup command retrieves the common name and description of a group.</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - uniqueName Specifies the unique name value for the group that you want to view. This parameter maps to the <code>uniqueName</code> property in virtual member manager. (String, required) • Returns: A map of common name and description properties. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask getGroup {-uniqueName cn=operators, cn=groups,dc=yourco,dc=com}</pre> • Using Jython string: <pre>AdminTask.getGroup ('[-uniqueName cn=operators, cn=groups,dc=yourco, dc=com]')</pre> • Using Jython list: <pre>AdminTask.getGroup ([' -uniqueName', 'cn=opera tors,cn=groups,dc=your co,dc=com'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask getGroup {-interactive}</pre> • Using Jython string: <pre>AdminTask.getGroup ('[-interactive]')</pre> • Using Jython list: <pre>AdminTask.getGroup (['-interactive'])</pre>

Table 19. (continued)

Command name:	Description:	Parameters and return values:	Examples:
getMembership OfGroup	<p>The getMembership OfGroup command retrieves the groups of which a group is a member.</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - uniqueName Specifies the unique name value for the group whose group memberships you want to view. This parameter maps to the uniqueName property in virtual member manager. (String, required) • Returns: A list of unique names of each of the groups of which the group is a member. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask getMebmership OfGroup {-uniqueName uid=dmeyers,cn=users,dc=yourco,dc=com}</pre> • Using Jython string: <pre>AdminTask.getMebmership OfGroup ('[-uniqueName uid=dmeyers,cn=users,dc=yourco,dc=com]')</pre> • Using Jython list: <pre>AdminTask.getMebmership OfGroup (['-uniqueName', 'uid=dmeyers,cn=users,dc=yourco,dc=com'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask getMembership OfGroup {-interactive}</pre> • Using Jython string: <pre>AdminTask.getMembership OfGroup ('[-interactive]')</pre> • Using Jython list: <pre>AdminTask.getMembership OfGroup (['-interactive'])</pre>

Table 19. (continued)

Command name:	Description:	Parameters and return values:	Examples:
getMembershipOfUser	The getMembershipOfUser command retrieves the groups of which a user is a member.	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - uniqueName Specifies the unique name value for the user whose group memberships you want to view. This parameter maps to the <code>uniqueName</code> property in virtual member manager. (String, required) • Returns: A list of unique names for each group of which the user is a member. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask getMembershipOfUser {-uniqueName uid=dmeyers,cn=users,dc=yourco,dc=com}</pre> • Using Jython string: <pre>AdminTask.getMembershipOfUser ('[-uniqueName uid=dmeyers,cn=users,dc=yourco,dc=com]')</pre> • Using Jython list: <pre>AdminTask.getMembershipOfUser (['-uniqueName', 'uid=dmeyers,cn=users,dc=yourco,dc=com'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask getMembershipOfUser {-interactive}</pre> • Using Jython string: <pre>AdminTask.getMembershipOfUser ('[-interactive]')</pre> • Using Jython list: <pre>AdminTask.getMembershipOfUser (['-interactive'])</pre>

Table 19. (continued)

Command name:	Description:	Parameters and return values:	Examples:
getMembers OfGroup	<p>The getMembers OfGroup command retrieves the members of a group.</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - uniqueName Specifies the unique name value for the group whose members you want to view. This parameter maps to the uniqueName property in virtual member manager. (String, required) • Returns: The unique name of each of the members of the group and the type of each member. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask getMembersOfGroup {-uniqueName cn=operators,cn=groups ,dc=yourco,dc=com}</pre> • Using Jython string: <pre>AdminTask.getMembersOfGroup [('-uniqueName cn=operators,cn=groups ,dc=yourco,dc=com')]</pre> • Using Jython list: <pre>AdminTask.getMembersOfGroup [('-uniqueName', 'cn=operators,cn=groups ,dc=yourco,dc=com')]</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask getMembersOfGroup {-interactive}</pre> • Using Jython string: <pre>AdminTask.getMembersOfGroup ('[-interactive]')</pre> • Using Jython list: <pre>AdminTask.getMembersOfGroup (['-interactive'])</pre>

Table 19. (continued)

Command name:	Description:	Parameters and return values:	Examples:
<p>getUser</p>	<p>The getUser command retrieves information about a user in the virtual member manager.</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - uniqueName Specifies the unique name value for the user that you want to view. This parameter maps to the uniqueName property in the virtual member manager. (String, required) • Returns: A map that contains the following properties: uniqueName, cn, sn, uid, and ibm-primaryEmail. These attributes are fixed and you cannot change them. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask getUser {-user Name uid=dmeyers,cn=users,dc=yourco,dc=com} • Using Jython string: AdminTask.getUser ('[-use rName uid=dmeyers,cn=users,dc=yourco,dc=com]') • Using Jython list: AdminTask.getUser (['-use rName', 'uid=dmeyers,cn=users,dc=yourco,dc=com']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask getUser {-interactive} • Using Jython string: AdminTask.getUser (['-interactive']) • Using Jython list: AdminTask.getUser (['-interactive'])

Table 19. (continued)

Command name:	Description:	Parameters and return values:	Examples:
<p>removeMemberFromGroup</p>	<p>The removeMemberFromGroup command removes a user or a group from a group.</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - memberUniqueName Specifies the unique name value for the user or group that you want to remove from the specified group. This parameter maps to the <code>uniqueName</code> property in virtual member manager. (String, required) - groupUniqueName Specifies the unique name value for the group from which you want to remove the user or group that you specified with the <code>memberUniqueName</code> parameter. This parameter maps to the <code>uniqueName</code> property in virtual member manager. (String, required) • Returns: Void if the command is successful. Returns an exception if the command fails. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask removeMemberFromGroup {-memberUniqueName uid=meyersd,cn=users,dc=yourco,dc=com -groupUniqueName cn=admins,cn-groups,dc=yourco,dc=com}</pre> • Using Jython string: <pre>AdminTask.removeMemberFromGroup (['-memberUniqueName uid=meyersd,cn=users,dc=yourco,dc=com -groupUniqueName cn=admins,cn-groups,dc=yourco,dc=com'])</pre> • Using Jython list: <pre>AdminTask.removeMemberFromGroup (['-memberUniqueName', 'uid=meyersd,cn=users,dc=yourco,dc=com', '-groupUniqueName', 'cn=admins,cn-groups,dc=yourco,dc=com'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask removeMemberFromGroup {-interactive}</pre> • Using Jython string: <pre>AdminTask.removeMemberFromGroup (['-interactive'])</pre> • Using Jython list: <pre>AdminTask.removeMemberFromGroup (['-interactive'])</pre>

Table 19. (continued)

Command name:	Description:	Parameters and return values:	Examples:
searchGroups	<p>Use the searchGroups command to find groups in the virtual member manager that match criteria that you provide. For example, you can use the searchGroups command to find all of the groups with a common name that begins with IBM. You can search for any virtual member manager property because the command is generic.</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - cn The first name or given name of the user. This parameter maps to the cn property in the virtual member manager. You must set this parameter or the description parameter, but not both. (String, optional) - description Specifies information about the group. This parameter maps to the description entity in a virtual member manager object. You must set this parameter or the cn parameter, but not both. (String, optional) - timeLimit Specifies the maximum amount of time in milliseconds that the search can run. The default value is no time limit. (String, optional) - countLimit Specifies the maximum number of results that you want returned from the search. By default, all groups found in the search are returned. (String, optional) • Returns: A list of unique names of all of the groups that match the search criteria that you provided. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <code>\$AdminTask searchGroups {cn *IBM*}</code> • Using Jython string: <code>AdminTask.searchGroups ('[cn *IBM*]')</code> • Using Jython list: <code>AdminTask.searchGroups (['cn', '*IBM*'])</code> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <code>\$AdminTask searchGroups {-interactive}</code> • Using Jython string: <code>AdminTask.searchGroups (['-interactive'])</code> • Using Jython list: <code>AdminTask.searchGroups (['-interactive'])</code>

Table 19. (continued)

Command name:	Description:	Parameters and return values:	Examples:
searchUsers	<p>Use the searchUsers command to find users in the virtual member manager that match criteria that you provide. For example, you can use the searchUsers command to find all of the telephone numbers that contain 919. You can search for any virtual member manager property because the command is generic.</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - principalName Specifies the principal name of the user that is used as the logon ID for the user in the system. This parameter maps to the principalName property in virtual member manager. You must specify only one of the following parameters: principalName, uid, cn, sn, or ibm-primaryEmail. (String, optional) - uid Specifies the unique ID value for the user for whom you want to search. This parameter maps to the uid property in virtual member manager. You must specify only one of the following parameters: principalName, uid, cn, sn, or ibm-primaryEmail. (String, optional) - cn Specifies the first name or given name of the user. This parameter maps to the cn property in virtual member manager. You must specify only one of the following parameters: principalName, uid, cn, sn, or ibm-primaryEmail. (String, optional) 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask searchUsers {-principalName */IBM/US*} • Using Jython string: AdminTask.searchUsers ('[-principalName */IBM/US*]') • Using Jython list: AdminTask.searchUsers (['-principalName', '*/IBM/US*']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask searchUsers {-interactive} • Using Jython string: AdminTask.searchUsers ('[-interactive]') • Using Jython list: AdminTask.searchUsers (['-interactive'])

Table 19. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<p>- sn Specifies the last name or family name of the user. This parameter maps to the sn property in virtual member manager. You must specify only one of the following parameters: principalName, uid, cn, sn, or ibm-primaryEmail. (String, optional)</p> <p>- ibm-primaryEmail Specifies the email address of the user. This parameter maps to the ibm-PrimaryEmail property in the virtual member manager. You must specify only one of the following parameters: principalName, uid, cn, sn, or ibm-primaryEmail. (String, optional)</p> <p>- timeLimit Specifies the maximum amount of time in milliseconds that the search can run. The default is not time limit. (String, optional)</p> <p>- countLimit Specifies the maximum number of results that you want returned from the search. By default, all users found in the search are returned. (String, optional)</p> <ul style="list-style-type: none"> • Returns: A list of unique names of all of the users that match the search criteria that you provided. 	

Table 19. (continued)

Command name:	Description:	Parameters and return values:	Examples:
updateGroup	<p>The updateGroup command updates the common name or the description of a group.</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - uniqueName Specifies the unique name value for the group for which you want to modify the properties. This parameter maps to the uniqueName property in virtual member manager. (String, required) - cn Specifies the new common name used for the group. This parameter maps to the cn property in virtual member manager. (String, optional) - description Specifies the new information about the group. This parameter maps to the description entity in a virtual member manager object. (String, optional) • Returns: Void if the command is successful. Returns an exception if the command fails. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask updateGroup {-uniqueName cn=opera tors,cn=groups,dc=yourco ,dc=com -cn groupA}</pre> • Using Jython string: <pre>AdminTask.updateGroup (['-uniqueName cn=oper ators,cn=groups,dc=your co,dc=com -cn groupA'])</pre> • Using Jython list: <pre>AdminTask.updateGroup ([' -uniqueName', 'cn=oper ators,cn=groups,dc=yourco, dc=com', '-cn', 'groupA'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask updateGroup {-interactive}</pre> • Using Jython string: <pre>AdminTask.updateGroup (['-interactive'])</pre> • Using Jython list: <pre>AdminTask.updateGroup (['-interactive'])</pre>

Table 19. (continued)

Command name:	Description:	Parameters and return values:	Examples:
updateUser	<p>The updateUser command updates the following properties: uniqueName, uid, password, cn, sn, or ibm-primaryEmail.</p>	<ul style="list-style-type: none"> • Parameters: - uniqueName Specifies the unique name value for the user for which you want to modify the properties. This parameter maps to the uniqueName property in virtual member manager. (String, required) - uid Specifies the new unique ID value for the user. This parameter maps to the uid property in virtual member manager. (String, optional) - password Specifies the new password for the user. This parameter maps to the password property in virtual member manager. (String, optional) - confirmPassword Specifies the password again to validate how it was entered on the password parameter. This parameter maps to the password property in virtual member manager. (String, optional) - cn Specifies the new first name or given name of the user. This parameter maps to the cn property in virtual member manager. (String, optional) 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <code>\$AdminTask updateUser {-uniqueName uid=dmeyers,cn=users,dc=yourco,dc=com -uid 123}</code> • Using Jython string: <code>AdminTask.updateUser ('[-uniqueName uid=dmeyers,cn=users,dc=yourco,dc=com -uid 123]')</code> • Using Jython list: <code>AdminTask.updateUser (['-uniqueName', 'uid=dmeyers,cn=users,dc=yourco,dc=com', '-uid', '123'])</code> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <code>\$AdminTask updateUser {-interactive}</code> • Using Jython string: <code>AdminTask.updateUser (['-interactive'])</code> • Using Jython list: <code>AdminTask.updateUser (['-interactive'])</code>

Table 19. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<ul style="list-style-type: none"> - surname Specifies the new last name or family name of the user. This parameter maps to the sn property in virtual member manager. (String, optional) - ibm-primaryEmail Specifies the new e-mail address of the user. This parameter maps to the mail property in virtual member manager. (String, optional) • Returns: Void if the command is successful. Returns an exception if the command fails. 	

Commands for the KeyStoreCommands group of the AdminTask object

Use the commands in the KeyStoreCommands group to create or delete key stores. The commands for this group do not require a target object. For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the KeyStoreCommands group of the AdminTask object:

Table 20.

Command name:	Description:	Parameters and return values:	Examples:
<p>changeKey Store Password</p>	<p>The changeKey Store Password command changes the password on the key store.</p>	<ul style="list-style-type: none"> • Parameters: - keyStoreName The name that uniquely identifies the key store configuration object. (String, required) - keyStoreScope The scope name of the key store. (String, optional) - keyStorePassword The password that protects the key store that you want to change. (String, required) - newkeyStorePassword The new password that protects the key store. (String, required) - newkeyStorePassword Verify The new password that protects the key store. (String, required) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask changeKeyStore Password {-keyStoreName testKS -keyStorePassword testpwd -newKeyStorePassword keyPWD -newKeyStorePasswordVerify keyPWD}</pre> • Using Jython string: <pre>AdminTask.changeKeyStore Password ('[-keyStoreName testKS -keyStorePassword testpwd -newKeyStorePassword keyPWD -newKeyStorePassword Verify keyPWD]')</pre> • Using Jython list: <pre>AdminTask.changeKeyStore Password (['-keyStoreName', 'testKS', '-keyStorePassword', 'testpwd', '-newKeyStorePassword', 'keyPWD', '-newKeyStorePasswordVerify', 'keyPWD'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask changeKeyStore Password {-interactive}</pre> • Using Jython string: <pre>AdminTask.changeKeyStore Password (['-interactive'])</pre> • Using Jython list: <pre>AdminTask.changeKeyStore Password (['-interactive'])</pre>

Table 20. (continued)

Command name:	Description:	Parameters and return values:	Examples:
<p>change Multiple KeyStore Passwords</p>	<p>The change Multiple KeyStore Passwords command updates all of the key stores in the configuration that have a give password and changed them to a new password. This is useful because when you create key store files on the system, they will have WebAS as a password by default.</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - keyStorePassword Specifies the name of the password that you want to change. (String, required) - newKeyStorePassword Specifies the new password that you will use to access the key store. (String, required) - newKeyStorePassword Verify Confirms the new key store password. (String, required) • Returns: A list of key store aliases that where changed 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask changeMultipleKeyStorePasswords {-keyStorePassword WebAS -newKeyStorePassword newpwd -newKeyStorePassword Verify newpwd}</pre> • Using Jython string: <pre>AdminTask.changeMultipleKeyStorePasswords ('[-keyStorePassword WebAS -newKeyStorePassword newpwd -newKeyStorePasswordVerify newpwd]')</pre> • Using Jython list: <pre>AdminTask.changeMultipleKeyStorePasswords (['-keyStorePassword', 'WebAS', '-newKeyStorePassword', 'newpwd', '-newKeyStorePasswordVerify', 'newpwd'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask changeMultipleKeyStorePasswords {-interactive}</pre> • Using Jython string: <pre>AdminTask.changeMultipleKeyStorePasswords (['-interactive'])</pre> • Using Jython list: <pre>AdminTask.changeMultipleKeyStorePasswords (['-interactive'])</pre>

Table 20. (continued)

Command name:	Description:	Parameters and return values:	Examples:
createKeyStore	<p>The createKeyStore command creates the key store settings in the configuration and the key store database.</p>	<ul style="list-style-type: none"> • Parameters: - keyStoreName The name that uniquely identifies the key store configuration object. (String, required) - keyStoreType The implementation of the key store management. (String, required) - keyStoreLocation The location of the key store. For file based, the location is the files system path to the key store database. For hardware key store, the location is the path to the token library. (String, required) - keyStorePassword The password that protects the key store. (String, required) - keyStorePasswordVerify The password that protects the key store. (String, required) - keyStoreProvider The provider used to implement the key store. (String, optional) - isKeyStoreFileBased Set the value of this parameter to true if the key store is file based. Set the value of this parameter to false for hardware crypto key stores. (Boolean, optional) 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask createKeyStore {-keyStoreName testKS -location c:\temp\test KeyFile.p12 keyStorePass word testpwd -keyStore PasswordVerify testpwd -isKeyStoreFileBased true -keyStoreInitAtStartup true -keyStoreReadOnly false}</pre> • Using Jython string: <pre>AdminTask.createKeyStore (['-keyStoreName testKS -location c:\temp\testKey File.p12 keyStorePass word testpwd -keyStorePass wordVerify testpwd -isKey StoreFileBased true -key StoreInitAtStartup true -keyStoreReadOnly false'])</pre> • Using Jython list: <pre>AdminTask.createKeyStore (['-keyStoreName', 'testKS', '-location', 'c:\temp\test KeyFile.p12', 'keyStorePass word', 'testpwd', '-keySto rePasswordVerify', 'testpwd', '-isKeyStoreFileBased', 'true', '-keyStoreInitAt Startup', 'true', '-key StoreReadOnly', 'false'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask createKey Store {-interactive}</pre> • Using Jython string: <pre>AdminTask.createKeySt ore ['-interactive']</pre> • Using Jython list: <pre>AdminTask.createKeyS tore (['-interactive'])</pre>

Table 20. (continued)

Command name:	Description:	Parameters and return values:	Examples:
		<ul style="list-style-type: none"> - keyStoreHostList A list of host names that indicate from where the key store is remotely managed, separated by commas. (String, optional) - keyStoreInitAtStartup Set the value of this parameter to true if the key store is initialized at startup. Otherwise, set the value of this parameter to false. (Boolean, optional) - keyStoreReadOnly Set the value of this parameter to true if you cannot write to the key store. Otherwise, set the value of this parameter to false. (Boolean, optional) - keyStoreStashFile Set the value of this parameter to true if you want to create stash files for CMS type key store. Otherwise, set the value of this parameter to false. (Boolean, optional) - scopeName The name of the scope. (String, optional) - - enableCryptoOperations Specifies if the key store object will be used for hardware cryptographic operations or not. The default value is false. (Boolean, optional) • Returns: The configuration object name of the key store object that you created. 	

Table 20. (continued)

Command name:	Description:	Parameters and return values:	Examples:
createCMSKeyStore	<p>The createCMSKeyStore command creates a CMS key store database and the key store settings in the configuration.</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - cmsKeyStoreURI The URI of the CMS key store. (String, required) - pluginHostName The host name of the plug-in. (String, required) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask createCMSKeyStore • Using Jython: AdminTask.createCMSKeyStore() <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask createCMSKeyStore {-interactive} • Using Jython string: AdminTask.createCMSKeyStore ('[-interactive]') • Using Jython list: AdminTask.createCMSKeyStore (['-interactive'])
deleteKeyStore	<p>The deleteKeyStore command deletes the settings of a key store from the configuration and the key store file.</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - name The name that uniquely identifies the key store that you want to delete. (String, required) - scopeName The name of the scope. (String, optional) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask deleteKeyStore {-name testKS} • Using Jython string: AdminTask.deleteKeyStore ('[-name testKS]') • Using Jython list: AdminTask.deleteKeyStore (['-name', 'testKS']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask deleteKeyStore {-interactive} • Using Jython string: AdminTask.deleteKeyStore ('[-interactive]') • Using Jython list: AdminTask.deleteKeyStore (['-interactive'])

Table 20. (continued)

Command name:	Description:	Parameters and return values:	Examples:
exchangeSigners	The exchangeSigners command exchange signer certificate between key stores.	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - keyStoreName1 The name that uniquely identifies a key store. You must specify a second key store name using the <code>keyStoreName2</code> parameter. (String, required) - keyStoreScope1 The scope name of the key store that you specified with the <code>keyStoreName1</code> parameter. (String, required) - certificateAlaisList1 A list of aliases separated by a comma. (String, optional) - keyStoreName2 The name that uniquely identifies a key store. You must specify a second key store name using the <code>keyStoreName1</code> parameter. (String, required) - keyStoreScope2 The scope name of the key store that you specified with the <code>keyStoreName2</code> parameter. (String, required) - certificateAliasList2 A list of aliases separated by a comma. (String, optional) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask exchangeSigners {-keyStoreName1 testKS -certificateAliasList1 testCert1 -keyStoreName2 secondKS -certificate AlaisList2 certAlis}</pre> • Using Jython string: <pre>AdminTask.exchangeSigners ('[-keyStoreName1 testKS -certificateAliasList1 testCert1 -keyStoreName2 secondKS -certificateAlais List2 certAlis]')</pre> • Using Jython list: <pre>AdminTask.exchangeSigners (['-keyStoreName1', 'testKS', '-certificateAliasList1', 'testCert1', '-keyStoreName 2', 'secondKS', '-certifica teAlaisList2', 'certAlis'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask exchangeSigners {-interactive}</pre> • Using Jython string: <pre>AdminTask.exchangeSigners ('[-interactive]')</pre> • Using Jython list: <pre>AdminTask.exchangeSigners (['-interactive'])</pre>

Table 20. (continued)

Command name:	Description:	Parameters and return values:	Examples:
getKeyStoreInfo	The getKeyStoreInfo command displays the settings of a particular key store.	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - name The name that uniquely identifies the key store. (String, required) - scopeName The name of the scope. (String, optional) • Returns: The settings of the key store that you specified. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask getKeyStore {-name testKS} • Using Jython string: AdminTask.getKeyStore ('[-name testKS]') • Using Jython list: AdminTask.getKeyStore (['-name', 'testKS']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask getKeyStore Info {-interactive} • Using Jython string: AdminTask.getKeyStore Info ('[-interactive]') • Using Jython list: AdminTask.getKeyStore Info (['-interactive'])

Table 20. (continued)

Command name:	Description:	Parameters and return values:	Examples:
listKeyFileAliases	<p>The listKeyFileAliases command lists the certificates in a key store file.</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - keyFilePath The path of the key file. (String, required) - keyFilePassword The password for the key file. (String, required) - keyFileType The key file type. (String, required) • Returns: A list of certificate aliases. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask listKeyFileAliases {-keyFilePaht c:\temp\testKeyFile.p12 -keyFilePassword testPwd -keyFileType PKCS12}</pre> • Using Jython string: <pre>AdminTask.listKeyFileAliases ('[-keyFilePaht c:\temp\testKeyFile.p12 -keyFilePassword testPwd -keyFileType PKCS12]')</pre> • Using Jython list: <pre>AdminTask.listKeyFileAliases (['-keyFilePaht', 'c:\temp\testKeyFile.p12', '-keyFilePassword', 'testPwd', '-keyFileType', 'PKCS12'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask listKeyFileAliases {-interactive}</pre> • Using Jython string: <pre>AdminTask.listKeyFileAliases ('[-interactive]')</pre> • Using Jython list: <pre>AdminTask.listKeyFileAliases (['-interactive'])</pre>

Table 20. (continued)

Command name:	Description:	Parameters and return values:	Examples:
listKeyStores	The listKeyStores command lists the key store for a particular scope.	<ul style="list-style-type: none"> Parameters: <ul style="list-style-type: none"> - displayObjectName Set the value of this parameter to true to list the key store configuration objects within a scope. Set the value of this parameter to false to list the strings that contain the key store name and management scope. (String, optional) - scopeName The name of the scope. (String, optional) Returns: A list of key stores. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: <code>\$AdminTask listKeyStores</code> Using Jython: <code>AdminTask.listKeyStores()</code> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: <code>\$AdminTask listKeyStores {-interactive}</code> Using Jython string: <code>AdminTask.listKeyStores ('[-interactive]')</code> Using Jython list: <code>AdminTask.listKeyStores (['-interactive'])</code>
listKeyStoresTypes	The listKeyStoresTypes command lists all valid key store types.	<ul style="list-style-type: none"> Parameters: None Returns: A list of key store types. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: <code>\$AdminTask listKeyStoreTypes</code> Using Jython: <code>AdminTask.listKeyStoreTypes()</code> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: <code>\$AdminTask listKeyStores Types {-interactive}</code> Using Jython string: <code>AdminTask.listKeyStores Types ('[-interactive]')</code> Using Jython list: <code>AdminTask.listKeyStores Types (['-interactive'])</code>

Commands for the SSLConfigCommands group of the AdminTask object

Use the commands in the SSLConfigCommands group to create and delete SSL configurations. For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the SSLConfigCommands group of the AdminTask object:

Table 21.

Command name:	Description:	Target object:	Parameters and return values:	Examples:
createSSLConfig	<p>The createSSLConfig command creates an SSL configuration that is based on key store and trust store settings. You can use the SSL configuration settings to make the SSL connections.</p>	None	<ul style="list-style-type: none"> • Parameters: - alias The name of the alias. (String, required) - scopeName The name of the scope. (String, optional) - clientKeyAlias The certificate alias name for the client. (String, optional) - serverKeyAlias The certificate alias name for the server. (String, optional) - type The type of SSL configuration. (String, optional) - clientAuthentication Set the value of this parameter to true to request client authentication. Otherwise, set the value of this parameter to false. (Boolean, optional) - securityLevel The cipher group that you want to use. Valid values include: HIGH, MEDIUM, LOW, and CUSTOM. (String, optional) - enabledCiphers A list of ciphers used during SSL handshake. (String, optional) - jsseProvider One of the JSSE providers. (String, optional) 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask createSSLConfig {-alias testSSLCfg -clientKeyAlias key1 -serverKeyAlias key2 -trustStoreName trustKS -keyStoreName testKS -keyManagerName testKeyMgr} • Using Jython string: AdminTask.createSSLConfig ('[-alias testSSLCfg -clientKeyAlias key1 -serverKeyAlias key2 -trustStoreName trustKS -keyStoreName testKS -keyManagerName testKeyMgr]') • Using Jython list: AdminTask.createSSLConfig (['-alias', 'testSSLCfg', '-clientKeyAlias', 'key1', '-serverKeyAlias', 'key2', '-trustStoreName', 'trustKS', '-keyStoreName', 'testKS', '-keyManagerName', 'testKeyMgr']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask createSSLConfig {-interactive} • Using Jython string: AdminTask.createSSLConfig ('[-interactive]') • Using Jython list: AdminTask.createSSLConfig (['-interactive'])

Table 21. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
			<p>-</p> <p>clientAuthenticationSupported Set the value of this parameter to true to support client authentication. Otherwise, set the value of this parameter to false. (Boolean, optional)</p> <p>- sslProtocol The protocol type for the SSL handshake. Valid values include: SSL_TLS, SSL, SSLv2, SSLv3, TLS, TLSv1. (String, optional)</p> <p>-</p> <p>trustManagerObjectName A list of trust managers separated by commas. (String, optional)</p> <p>- trustStoreNames The key store that holds trust information used to validate the trust from remote connections. (String, required)</p> <p>-</p> <p>trustStoreScopeName The management scope name of the trust store. (String, optional)</p>	

Table 21. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
			<ul style="list-style-type: none"> - keyStoreName The key store that holds the personal certificates that provide identity for the connection. (String, required) - keyStoreScopeName The management scope name of the key store. (String, optional) - ssslKeyRingName Specifies a system SSL (SSSL) key ring name. The value for this parameter has no affect unless the SSL configuration type is SSSL. (String, optional) • Returns: The configuration object name of the SSL configuration object that you created. 	

Table 21. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
createSSL Config Property	The createSSL Config Property command creates a property for an SSL configuration. Use this command to set SSL configuration settings that are different than the settings in the SSL configuration object.	None	<ul style="list-style-type: none"> • Parameters: - sslConfigAliasName The alias name of the SSL configuration. (String, required) - scopeName The name of the scope. (String, optional) - propertyName The name of the property. (String, required) - propertyValue The value of the property. (String, required) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask createSSLConfigProperty {-sslConfigAliasName NodeDefaultSSLSettings -scopeName (cell):localhostNode01Cell:(node):localhostNode01 -propertyName test.property -propertyValue testValue} • Using Jython string: AdminTask.createSSLConfigProperty ('[-sslConfigAliasName NodeDefaultSSLSettings -scopeName (cell):localhostNode01Cell:(node):localhostNode01 -propertyName test.property -propertyValue testValue]') • Using Jython list: AdminTask.createSSLConfigProperty (['-sslConfigAliasName', 'NodeDefaultSSLSettings', '-scopeName', '(cell):localhostNode01Cell:(node):localhostNode01', '-propertyName', 'test.property', '-propertyValue', 'testValue']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask createSSLConfigProperty {-interactive} • Using Jython string: AdminTask.createSSLConfigProperty ('[-interactive]') • Using Jython list: AdminTask.createSSLConfigProperty (['-interactive'])

Table 21. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
deleteSSLConfig	The deleteSSLConfig command deletes the SSL configuration object that you specify from the configuration.	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - alias The name of the alias. (String, required) - scopeName The name of the scope. (String, optional) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask deleteSSLConfig {-alias NodeDefaultSSL Settings -scopeName (cell):localhostNode01Cell:(no de):localhostNode01}</pre> • Using Jython string: <pre>AdminTask.deleteSSLConfig (['-alias NodeDefaultSSL Settings -scopeName (cell):localhostNode01Cell:(no de):localhostNode01'])</pre> • Using Jython list: <pre>AdminTask.deleteSSLConfig (['-alias', 'NodeDefault SSLSettings', '-scopeName', '(cell):localhostNode01 Cell:(node):localhost Node01'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask deleteSSLConfig {-interactive}</pre> • Using Jython string: <pre>AdminTask.deleteSSLConfig (['-interactive'])</pre> • Using Jython list: <pre>AdminTask.deleteSSLConfig (['-interactive'])</pre>

Table 21. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
getSSLConfig	The getSSLConfig command obtains information about an SSL configuration and displays the settings.	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - alias The name of the alias. (String, required) - scopeName The name of the scope. (String, optional) • Returns: Information about the SSL configuration that you specified. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask getSSLConfig {-alias NodeDefaultSSL Settings -scopeName (cell):localhostNode01 Cell:(node):localhost Node01 }</pre> • Using Jython string: <pre>AdminTask.getSSLConfig (['-alias NodeDefault SSLSettings -scopeName (cell):localhostNode01 Cell:(node):localhost Node01'])</pre> • Using Jython list: <pre>AdminTask.getSSLConfig (['-alias', 'Node DefaultSSLSettings', '-scopeName', '(cell): localhostNode01Cell: (node):localhostNode01'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask getSSLConfig {-interactive}</pre> • Using Jython string: <pre>AdminTask.getSSLConfig (['-interactive'])</pre> • Using Jython list: <pre>AdminTask.getSSLConfig (['-interactive'])</pre>

Table 21. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
<p>getSSLConfig Properties</p>	<p>The getSSLConfig Properties command obtains information about SSL configuration properties.</p>	<p>None</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - alias The name of the alias. (String, required) - scopeName The name of the scope. (String, optional) • Returns: Information about SSL configuration properties. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask getSSLConfig Properties {-sslConfig AliasName NodeDefault SSLSettings -scopeName (cell):localhostNode01 Cell:(node):localhost Node01}</pre> • Using Jython string: <pre>AdminTask.getSSLConfig Properties ('[-sslConfigAliasName NodeDefaultSSLSettings -scopeName (cell):localhost Node01Cell:(node):localhostNode01]')</pre> • Using Jython list: <pre>AdminTask.getSSLConfig Properties (['-sslConfigAliasName', 'NodeDefaultSSLSettings', '-scopeName', '(cell):localhostNode01Cell:(node):localhostNode01'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask getSSLConfig Properties {-interactive}</pre> • Using Jython string: <pre>AdminTask.getSSLConfig Properties ('[-interactive]')</pre> • Using Jython list: <pre>AdminTask.getSSLConfig Properties (['-interactive'])</pre>

Table 21. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
listSSLCiphers	The listSSLCiphers command lists the SSL ciphers.	None	<ul style="list-style-type: none"> • Parameters: - sslConfigAliasName The alias name of the SSL configuration. (String, required) - scopeName The name of the scope. (String, optional) - securityLevel The cipher group that you want to use. Valid values include: HIGH, MEDIUM, LOW, and CUSTOM. (String, required) - provider (String, optional) • Returns: A list of SSL ciphers. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask listSSLCiphers {-sslConfigAliasName test SSLCfg -securityLevel HIGH} • Using Jython string: AdminTask.listSSLCiphers ('[-sslConfigAliasName testSSLCfg -securityLevel HIGH]') • Using Jython list: AdminTask.listSSLCiphers (['-sslConfigAliasName', 'testSSLCfg', '-securityLevel', 'HIGH']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask listSSLCiphers {-interactive} • Using Jython string: AdminTask.listSSLCiphers ('[-interactive]') • Using Jython list: AdminTask.listSSLCiphers (['-interactive'])

Table 21. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
listSSLConfig	The listSSLConfig command lists the defined SSL configurations within a management scope.	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - scopeName The name of the scope. (String, optional) - displayObjectName Set the value of this parameter to true to list the SSL configuration objects within the scope. Set the value of this parameter to false to list the strings that contain the SSL configuration alias and management scope. (Boolean, optional) • Returns: A list of the defined SSL configurations. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask listSSLConfig {-scopeName (cell): localhostNode01Cell:(node): localhostNode01 -displayObjectName true}</pre> • Using Jython string: <pre>AdminTask.listSSLConfig (['-scopeName (cell): localhostNode01Cell:(node):localhostNode01 -displayObjectName true'])</pre> • Using Jython list: <pre>AdminTask.listSSLConfig (['-scopeName', '(cell):localhostNode01Cell:(node):localhostNode01', '-displayObjectName', 'true'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask listSSLConfig {-interactive}</pre> • Using Jython string: <pre>AdminTask.listSSLConfig (['-interactive'])</pre> • Using Jython list: <pre>AdminTask.listSSLConfig (['-interactive'])</pre>

Table 21. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
listSSLConfig Properties	The listSSLConfig Properties command lists the properties for a SSL configuration.	None	<ul style="list-style-type: none"> • Parameters: - • sslConfigAliasName The alias name of the SSL configuration. (String, required) - • scopeName The name of the scope. (String, optional) - • displayObjectName Set the value of this parameter to true to list the SSL configuration objects within the scope. Set the value of this parameter to false to list the strings that contain the SSL configuration alias and management scope. (Boolean, optional) • Returns: A list of properties. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask listSSLConfig Property {-alias No deDefaultSSLSettings -scope Name (cell):localhostNode01:(node):localhostNode01 -displayObjectName true} • Using Jython string: AdminTask.listSSLConfig Property ('[-alias No deDefaultSSLSettings -scopeName (cell):localhostNode01:(node):localhostNode01 -displayObjectName true]') • Using Jython list: AdminTask.listSSLConfigProperty (['-alias', 'No', 'deDefaultSSLSettings', '-scopeName', '(cell):localhostNode01:(node):localhostNode01', '-displayObjectName', 'true']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask listSSLConfig Properties {-interactive} • Using Jython string: AdminTask.listSSLConfig Properties ('[-interactive]') • Using Jython list: AdminTask.listSSLConfig Properties (['-interactive'])

Table 21. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
<p>modifySSLConfig</p>	<p>The modifySSLConfig command modifies the settings of an existing SSL configuration.</p>	<p>None</p>	<ul style="list-style-type: none"> • Parameters: - alias The name of the alias. (String, required) - scopeName The name of the scope. (String, optional) - clientKeyAlias The certificate alias name for the client. (String, optional) - serverKeyAlias The certificate alias name for the server. (String, optional) - type The type of SSL configuration. (String, optional) - clientAuthentication Set the value of this parameter to true to request client authentication. Otherwise, set the value of this parameter to false. (Boolean, optional) - securityLevel The cipher group that you want to use. Valid values include: HIGH, MEDIUM, LOW, and CUSTOM. (String, optional) - enabledCiphers A list of ciphers used during SSL handshake. (String, optional) - jsseProvider One of the JSSE providers. (String, optional) 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask modifySSLConfig {-alias test SSLCfg -clientKeyAlias tstKey1 -serverKeyAlias tstKey2 -securityLevel LOW} • Using Jython string: AdminTask.modifySSLConfig ('[-alias testSSLCfg -clientKeyAlias tstKey1 -serverKeyAlias tstKey2 -securityLevel LOW]') • Using Jython list: AdminTask.modifySSLConfig (['-alias', 'testSSLCfg', '-clientKeyAlias', 'tstKey1', '-serverKeyAlias', 'tstKey2', '-securityLevel', 'LOW']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask modifySSLConfig {-interactive} • Using Jython string: AdminTask.modifySSLConfig ('[-interactive]') • Using Jython list: AdminTask.modifySSLConfig (['-interactive'])

Table 21. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
			<p>-</p> <p>clientAuthenticationSupported Set the value of this parameter to true to support client authentication. Otherwise, set the value of this parameter to false. (Boolean, optional)</p> <p>- sslProtocol The protocol type for the SSL handshake. Valid values include: SSL_TLS, SSL, SSLv2, SSLv3, TLS, TLSv1. (String, optional)</p> <p>-</p> <p>trustManagerObjectNames A list of trust managers separated by commas. (String, optional)</p> <p>- trustStoreName The key store that holds trust information used to validate the trust from remote connections. (String, optional)</p> <p>-</p> <p>trustStoreScopeName The management scope name of the trust store. (String, optional)</p>	

Table 21. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
			<ul style="list-style-type: none"> - keyStoreName The key store that holds the personal certificates that provide identity for the connection. (String, optional) - keyStoreScopeName The management scope name of the key store. (String, optional) - ssslKeyRingName Specifies a system SSL (SSSL) key ring name. The value for this parameter has no affect unless the SSL configuration type is SSSL. (String, optional) <ul style="list-style-type: none"> • Returns: None 	

Commands for the DescriptivePropCommands group of the AdminTask object

For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the DescriptivePropCommands group of the AdminTask object:

Table 22.

Command name:	Description:	Target object:	Parameters and return values:	Examples:
createDescriptiveProp	The createDescriptiveProp command creates key manager settings in the configuration. Use this command during SSL handshake to determine which certificate to use.	None	<ul style="list-style-type: none"> Parameters: <ul style="list-style-type: none"> - parentDataType (String, required) - parentClassName (String, required) - parentScopeName (String, optional) - name (String, required) - value (String, required) - type (String, required) - displayNameKey (String, required) - nlsRangeKey (String, optional) - hoverHelpKey (String, optional) - range (String, optional) - inclusive (Boolean, optional) - firstClass (Boolean, optional) Returns: The configuration object name of the key manager object that you created. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: <pre>\$AdminTask createKeyManager {-name testKM -keyManager Class com.ibm.ws.security. ltpa.LTPAKeyPairGenerator}</pre> Using Jython string: <pre>AdminTask.createKeyManager (['-name testKM -keyManag erClass com.ibm.ws.securi ty.ltpa.LTPAKeyPairGenera tor'])</pre> Using Jython list: <pre>AdminTask.createKeyManager (['-name', 'testKM', '-key ManagerClass', 'com.ibm.ws .security.ltpa.LTPAKeyPair Generator'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: <pre>\$AdminTask createDescrip tiveProp {-interactive}</pre> Using Jython string: <pre>AdminTask.createDescripti veProp (['-interactive'])</pre> Using Jython list: <pre>AdminTask.createDescripti veProp (['-interactive'])</pre>
deleteDescriptiveProp	The deleteDescriptiveProp command deletes key manager settings from the configuration.	None	<ul style="list-style-type: none"> Parameters: <ul style="list-style-type: none"> - parentDataType (String, required) - parentClassName (String, required) - parentScopeName (String, optional) - name (String, required) Returns: None 	<p>Interactive mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: <pre>\$AdminTask deleteDescrip tiveProp {-interactive}</pre> Using Jython string: <pre>AdminTask.deleteDescrip tiveProp (['-interactive'])</pre> Using Jython list: <pre>AdminTask.deleteDescrip tiveProp (['-interactive'])</pre>

Table 22. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
getDescriptiveProp	The getDescriptiveProp command obtains information about key manager settings.	None	<ul style="list-style-type: none"> Parameters: <ul style="list-style-type: none"> - parentDataType (String, required) - parentClassName (String, required) - parentScopeName (String, optional) - name (String, required) Returns: Information about key manager settings. 	<p>Interactive mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: <pre>\$AdminTask getDescriptiveProp {-interactive}</pre> Using Jython string: <pre>AdminTask.getDescriptiveProp ('[-interactive]')</pre> Using Jython list: <pre>AdminTask.getDescriptiveProp (['-interactive'])</pre>
listDescriptiveProp	The listDescriptiveProp command lists the key managers within a particular management scope.	None	<ul style="list-style-type: none"> Parameters: <ul style="list-style-type: none"> - parentDataType (String, required) - parentClassName (String, required) - parentScopeName (String, optional) - displayObjectName Set the value of this parameter to true to list the key manager objects within the scope. Set the value of this parameter to false to list the strings that contain the key manager name and management scope. (Boolean, optional) Returns: A list of key managers. 	<p>Interactive mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: <pre>\$AdminTask listDescriptiveProp {-interactive}</pre> Using Jython string: <pre>AdminTask.listDescriptiveProp ('[-interactive]')</pre> Using Jython list: <pre>AdminTask.listDescriptiveProp (['-interactive'])</pre>

Table 22. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
modifyDescriptiveProp	The modifyDescriptiveProp command modifies the settings of an existing key manager.	None	<ul style="list-style-type: none"> Parameters: <ul style="list-style-type: none"> - parentDataType (String, required) - parentClassName (String, required) - parentScopeName (String, optional) - name (String, required) - value (String, optional) - type (String, optional) - displayNameKey (String, optional) - nlsRangeKey (String, optional) - hoverHelpKey (String, optional) - range (String, optional) - inclusive (Boolean, optional) - firstClass (Boolean, optional) Returns: None 	Interactive mode example usage: <ul style="list-style-type: none"> Using Jacl: <pre>\$AdminTask modifyDescriptiveProp {-interactive}</pre> Using Jython string: <pre>AdminTask.modifyDescriptiveProp ('[-interactive]')</pre> Using Jython list: <pre>AdminTask.modifyDescriptiveProp (['-interactive'])</pre>

Commands for the TrustManagerCommands group of the AdminTask object

Use the commands in the TrustManagerCommands group to create and delete a trust manager. For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the TrustManagerCommands group of the AdminTask object:

Table 23.

Command name:	Description:	Target object:	Parameters and return values:	Examples:
createTrustManager	<p>The createTrustManagerInfo command creates trust manager settings in the configuration. Use this command during SSL handshake to make trust decisions about remote endpoints.</p>	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - name The name that uniquely identifies the trust manager. (String, required) - scopeName The name of the scope. (String, optional) - provider The provider name of the trust manager. (String, optional) - algorithm The algorithm name of the trust manager. (String, optional) - trustManagerClass Specifies a class that implements the <code>javax.net.ssl.X509TrustManager</code> interface. You cannot use this parameter with the provider or algorithm parameters. (String, optional) • Returns: The configuration object name of the trust manager object that you created. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask createTrustManager {-name testTM -provider IBMJSSE2 -algorithm IbmX509}</pre> • Using Jython string: <pre>AdminTask.createTrustManager ('[-name testTM -provider IBMJSSE2 -algorithm IbmX509]')</pre> • Using Jython list: <pre>AdminTask.createTrustManager (['-name', 'testTM', '-provider', 'IBMJSSE2', '-algorithm', 'IbmX509'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask createTrustManager {-interactive}</pre> • Using Jython string: <pre>AdminTask.createTrustManager ('[-interactive]')</pre> • Using Jython list: <pre>AdminTask.createTrustManager (['-interactive'])</pre>

Table 23. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
deleteTrustManager	The deleteTrustManager command deletes the trust manager settings from the configuration.	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - name The name that uniquely identifies the trust manager. (String, optional) - scopeName The name of the scope. (String, optional) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask deleteTrustManager {-name testTM} • Using Jython string: AdminTask.deleteTrustManager (['-name testTM']) • Using Jython list: AdminTask.deleteTrustManager (['-name', 'testTM']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask deleteTrustManager {-interactive} • Using Jython string: AdminTask.deleteTrustManager (['-interactive']) • Using Jython list: AdminTask.deleteTrustManager (['-interactive'])
getTrustManager	The getTrustManager command obtains the setting of a trust manager.	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - name The name that uniquely identifies the trust manager. (String, optional) - scopeName The name of the scope. (String, optional) • Returns: The settings of the trust manager group that you specified. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask getTrustManager {-name testTM} • Using Jython string: AdminTask.getTrustManager (['-name testTM']) • Using Jython list: AdminTask.getTrustManager (['-name', 'testTM']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask getTrustManager {-interactive} • Using Jython string: AdminTask.getTrustManager (['-interactive']) • Using Jython list: AdminTask.getTrustManager (['-interactive'])

Table 23. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
listTrustManagers	The listTrustManagers command lists the trust managers within a particular management scope.	None	<ul style="list-style-type: none"> • Parameters: - scopeName The name of the scope. (String, optional) - displayObjectName Set the value of this parameter to true to list the trust manager objects within a scope. Set the value of this parameter to false to list the strings that contain the trust manager name and management scope. (Boolean, optional) • Returns: A list of trust managers that are found within the management scope that you specified. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask listTrustManagers {-displayObjectName true} • Using Jython string: AdminTask.listTrustManagers ('[-displayObjectName true]') • Using Jython list: AdminTask.listTrustManagers (['-displayName', 'true']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask listTrustManagers {-interactive} • Using Jython string: AdminTask.listTrustManagers ('[-interactive]') • Using Jython list: AdminTask.listTrustManagers (['-interactive'])

Table 23. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
modifyTrustManager	The modifyTrustManager command changes existing trust manager settings.	None	<ul style="list-style-type: none"> Parameters: <ul style="list-style-type: none"> - name The name that uniquely identifies the trust manager. (String, required) - scopeName The name of the scope. (String, optional) - provider The provider name of the trust manager. (String, optional) - algorithm The algorithm name of the trust manager. (String, optional) - trustManagerClass Specifies a class that implements the <code>javax.net.ssl.X509TrustManager</code> interface. You cannot use this parameter with the provider or algorithm parameters. (String, optional) Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: <pre>\$AdminTask modifyTrustManager {-name testTM -trustManagerClass test.trust.manager}</pre> Using Jython string: <pre>AdminTask.modifyTrustManager ('[-name testTM -trustManagerClass test.trust.manager]')</pre> Using Jython list: <pre>AdminTask.modifyTrustManager (['-name', 'testTM', '-trustManagerClass', 'test.trust.manager'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: <pre>\$AdminTask modifyTrustManager {-interactive}</pre> Using Jython string: <pre>AdminTask.modifyTrustManager ('[-interactive]')</pre> Using Jython list: <pre>AdminTask.modifyTrustManager (['-interactive'])</pre>

Commands for the keyManagerCommands group of the AdminTask object

Use the commands in the keyManagerCommands group to manage key managers. You can use these commands to create, modify, list, or obtain information about key managers. For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the keyManagerCommands group of the AdminTask object:

Table 24.

Command name:	Description:	Target object:	Parameters and return values:	Examples:
createKeyManager	<p>The createKeyManager command creates the key manager settings in the configuration. Use this command during SSL handshake to determine which certificate alias to use.</p>	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - name The name that uniquely identifies the key manager. (String, required) - scopeName The name of the scope. (String, optional) - provider The provider name of the key manager. (String, optional) - algorithm The algorithm name of the key manager. (String, optional) - keyManagerClass The name of the key manager implementation class. You can not use this parameter with the provider or the algorithm parameter. (String, optional) • Returns: The configuration object name of the key manager object that you created. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask createKeyManager {-name testKM -keyManagerClass com.ibm.ws.security.ltpa.LTPAKeyPairGenerator}</pre> • Using Jython string: <pre>AdminTask.createKeyManager ('[-name testKM -keyManagerClass com.ibm.ws.security.ltpa.LTPAKeyPairGenerator]')</pre> • Using Jython list: <pre>AdminTask.createKeyManager (['-name', 'testKM', '-keyManagerClass', 'com.ibm.ws.security.ltpa.LTPAKeyPairGenerator'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask createKeyManager {-interactive}</pre> • Using Jython string: <pre>AdminTask.createKeyManager ('[-interactive]')</pre> • Using Jython list: <pre>AdminTask.createKeyManager (['-interactive'])</pre>

Table 24. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
deleteKeyManager	The deleteKeyManager command deletes the key manager settings from the configuration.	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - name The name that uniquely identifies the key manager. (String, optional) - scopeName The name of the scope. (String, optional) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask deleteKeyMa nager {-name testKM} • Using Jython string: AdminTask.deleteKeyMa nager ('[-name testKM]') • Using Jython list: AdminTask.deleteKeyMa nager (['-name', 'testKM']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask deleteKeyM anager {-interactive} • Using Jython string: AdminTask.deleteKeyMa nager ('[-interactive]') • Using Jython list: AdminTask.deleteKeyM anager (['-interactive'])
getKeyManager	The getKeyManager command obtains the settings of a key manager.	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - name The name that uniquely identifies the key manager. (String, optional) - scopeName The name of the scope. (String, optional) • Returns: The settings of the key manager that you specified. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask getKeyMana ger {-name testKM} • Using Jython string: AdminTask.getKeyMana ger ('[-name testKM]') • Using Jython list: AdminTask.getKeyMana ger (['-name', 'testKM']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask getKeyManag er {-interactive} • Using Jython string: AdminTask.getKeyManag er ('[-interactive]') • Using Jython list: AdminTask.getKeyManag er (['-interactive'])

Table 24. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
listKeyManagers	The listKeyManagers command lists the key managers within a particular management scope.	None	<ul style="list-style-type: none"> • Parameters: - scopeName The name of the scope. (String, optional) - - displayObjectName Set the value of this parameter to true to list the key manager objects within the scope. Set the value of this parameter to false to list the strings that contain the key manager name and the management scope. (Boolean, optional) • Returns: A list of key managers. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask listKeyManagers • Using Jython: AdminTask.listKeyManagers() <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask listKeyManagers {-interactive} • Using Jython string: AdminTask.listKeyManagers ('[-interactive]') • Using Jython list: AdminTask.listKeyManagers (['-interactive'])

Table 24. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
modifyKeyManagers	The modifyKeyManagers command changes existing key manager settings.	None	<ul style="list-style-type: none"> Parameters: <ul style="list-style-type: none"> - name The name that uniquely identifies the key manager. (String, required) - scopeName The name of the scope. (String, optional) - provider The provider name of the key manager. (String, optional) - algorithm The algorithm name of the key manager. (String, optional) - keyManagerClass The name of the key manager implementation class. You can not use this parameter with the provider or the algorithm parameter. (String, optional) Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: \$AdminTask modifyKeyManagers {-name testKM -provider IBMJSSE2 -algorithm IbmX509} Using Jython string: AdminTask.modifyKeyManagers ('[-name testKM -provider IBMJSSE2 -algorithm IbmX509]') Using Jython list: AdminTask.modifyKeyManagers (['-name', 'testKM', '-provider', 'IBMJSSE2', '-algorithm', 'IbmX509']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: \$AdminTask modifyKeyManagers {-interactive} Using Jython string: AdminTask.modifyKeyManagers ('[-interactive]') Using Jython list: AdminTask.modifyKeyManagers (['-interactive'])

Commands for the SSLConfigGroupCommands group of the AdminTask object

Use the commands in the SSLConfigGroupCommands group to create or delete a SSL configuration group. For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the SSLConfigGroupCommands group of the AdminTask object:

Table 25.

Command name:	Description:	Target object:	Parameters and return values:	Examples:
createSSLConfigGroup	<p>The createSSLConfigGroup command creates a SSL configuration group.</p>	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - name The name that uniquely identifies the SSL configuration group. (String, required) - direction The direction to which the SSL configuration applies. Valid values include inbound or outbound. (String, required) - certificateAlias (String, required) - scopeName The name of the scope. (String, optional) - sslConfigAliasName The alias that uniquely identifies the SSL configurations in the group. (String, required) - sslConfigScopeName The scope that uniquely identifies the SSL configurations in the group. (String, optional) • Returns: The configuration object name of the SSL configuration group object that you created. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask createSSLConfigGroup { -name testSSLCfgGrp -direction inbound -certificateAlias alias1 -sslConfigAliasName testSSLCfg}</pre> • Using Jython string: <pre>AdminTask.createSSLConfigGroup [('-name testSSLCfgGrp -direction inbound -certificateAlias alias1 -sslConfigAliasName testSSLCfg)']</pre> • Using Jython list: <pre>AdminTask.createSSLConfigGroup [('-name', 'testSSLCfgGrp', '-direction', 'inbound', '-certificateAlias', 'alias1', '-sslConfigAliasName', 'testSSLCfg)']</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask createSSLConfigGroup {-interactive}</pre> • Using Jython string: <pre>AdminTask.createSSLConfigGroup ('[-interactive]')</pre> • Using Jython list: <pre>AdminTask.createSSLConfigGroup (['-interactive'])</pre>

Table 25. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
deleteSSLConfigGroup	The deleteSSLConfigGroup command deletes a SSL configuration group from the configuration.	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - name The name that uniquely identifies the SSL configuration group. (String, required) - direction The direction to which the SSL configuration applies. Valid values include inbound or outbound. (String, required) - scopeName The name of the scope. (String, optional) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask deleteSSLConfigGroup {-name createSSLCfgGrp -direction inbound} • Using Jython string: AdminTask.deleteSSLConfigGroup [('-name createSSLCfgGrp -direction inbound)'] • Using Jython list: AdminTask.deleteSSLConfigGroup [('-name', 'createSSLCfgGrp', '-direction', 'inbound)'] <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask deleteSSLConfigGroup {-interactive} • Using Jython string: AdminTask.deleteSSLConfigGroup ('[-interactive]') • Using Jython list: AdminTask.deleteSSLConfigGroup (['-interactive'])

Table 25. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
getSSLConfigGroup	The getSSLConfigGroup command returns information about a SSL configuration setting.	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - name The name that uniquely identifies the SSL configuration group. (String, required) - direction The direction to which the SSL configuration applies. Valid values include inbound or outbound. (String, required) - scopeName The name of the scope. (String, optional) • Returns: The settings of the SSL configuration group that you specified. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask getSSLConfig Group {-name createSSLCfgGrp -direction inbound} • Using Jython string: AdminTask.getSSLConfig Group ['-name createSSLCfgGrp -direction inbound'] • Using Jython list: AdminTask.getSSLConfig Group [('-name', 'createSSLCfgGrp', '-direction', 'inbound')] <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask getSSLConfig Group {-interactive} • Using Jython string: AdminTask.getSSLConfig Group ('[-interactive]') • Using Jython list: AdminTask.getSSLConfig Group (['-interactive'])

Table 25. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
listSSLConfigGroups	The listSSLConfigGroups command lists the SSL configuration groups within a scope and a direction.	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - direction The direction to which the SSL configuration applies. Valid values include inbound or outbound. (String, optional) - scopeName The name of the scope. (String, optional) - displayObjectName If you set this parameter to true, the command returns a list of all of the SSL configuration group objects within the scope. If you set this parameter to false, the command returns a list of strings that contain the SSL configuration name and management scope. (Boolean, optional) • Returns: A list of SSL configuration groups. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <code>\$AdminTask -listSSLConfigGroups {-displayObjectName true}</code> • Using Jython string: <code>AdminTask.listSSLConfigGroups [('-displayObjectName true)']</code> • Using Jython list: <code>AdminTask.listSSLConfigGroups [('-displayObjectName' 'true')]</code> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <code>\$AdminTask listSSLConfigGroups {-interactive}</code> • Using Jython string: <code>AdminTask.listSSLConfigGroups ('[-interactive]')</code> • Using Jython list: <code>AdminTask.listSSLConfigGroups (['-interactive'])</code>

Table 25. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
modifySSLConfigGroup	The modifySSLConfigGroup command modifies the setting of an existing SSL configuration group.	None	<ul style="list-style-type: none"> Parameters: <ul style="list-style-type: none"> - name The name that uniquely identifies the SSL configuration group. (String, required) - direction The direction to which the SSL configuration applies. Valid values include inbound or outbound. (String, required) - certificateAlias (String, optional) - scopeName The name of the scope. (String, optional) - sslConfigAliasName The alias that uniquely identifies the SSL configurations in the group. (String, optional) - sslConfigScopeName The scope that uniquely identifies the SSL configurations in the group. (String, optional) Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: \$AdminTask modifySSLConfigGroup {-name createSSLCfgGrp -direction inbound -certificateAlias alias2} Using Jython string: AdminTask.modifySSLConfigGroup ['(-name createSSLCfgGrp -direction inbound -certificateAlias alias2)'] Using Jython list: AdminTask.modifySSLConfigGroup [('-name', 'createSSLCfgGrp', '-direction', 'inbound', '-certificateAlias', 'alias2')] <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: \$AdminTask modifySSLConfigGroup {-interactive} Using Jython string: AdminTask.modifySSLConfigGroup ('[-interactive]') Using Jython list: AdminTask.modifySSLConfigGroup (['-interactive'])

Commands for the DynamicSSLConfigSelections group of the AdminTask object

Use the commands in the DynamicSSLConfigSelections group to create or delete a dynamic SSL configuration selection. For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the DynamicSSLConfigSelections group of the AdminTask object:

Table 26.

Command name:	Description:	Target object:	Parameters and return values:	Examples:
<p>createDynamic SSLConfig Selection</p>	<p>The createDynamic SSLConfig Selection command creates the configuration settings for the dynamic SSL configuration selection.</p>	<p>None</p>	<ul style="list-style-type: none"> • Parameters: - dynSSLConfigSelectionName The name that uniquely identifies the dynamic SSL configuration selection. (String, required) - scopeName The name of the scope. (String, optional) - dynSSLConfigSelectionDescription The description of the dynamic SSL configuration selection. (String, optional) - dynSSLConfigSelectionInfo The information for the dynamic SSL configuration selection. (String, required) - sslConfigName The name of the SSL configuration. (String, required) - sslConfigScope The scope of the SSL configuration. (String, optional) - certificateAlias The alias name to identify the certificate. (String, required) • Returns: The configuration object name of the dynamic SSL configuration selection object that you created. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <code>\$AdminTask createDynamicSSLConfigSelection</code> • Using Jython: <code>AdminTask.createDynamicSSLConfigSelection()</code> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <code>\$AdminTask createDynamicSSLConfigSelection {-interactive}</code> • Using Jython string: <code>AdminTask.createDynamicSSLConfigSelection (['-interactive'])</code> • Using Jython list: <code>AdminTask.createDynamicSSLConfigSelection (['-interactive'])</code>

Table 26. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
<p>deleteDynamic SSLConfig Selection</p>	<p>The deleteDynamic SSLConfig Selection command deletes the dynamic SSL configuration selection from the configuration.</p>	<p>None</p>	<ul style="list-style-type: none"> • Parameters: None • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask deleteDynamic SSLConfigSelection</pre> • Using Jython: <pre>AdminTask.deleteDynamic SSLConfigSelection()</pre> • <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask deleteDynamic SSLConfigSelection {-interactive}</pre> • Using Jython string: <pre>AdminTask.deleteDynamicSSLConfigSelection ('[-interactive]')</pre> • Using Jython list: <pre>AdminTask.deleteDynamic SSLConfigSelection (['-interactive'])</pre>
<p>getDynamic SSLConfig Selection</p>	<p>The getDynamic SSLConfig Selection command obtains information about a particular dynamic SSL configuration selection.</p>	<p>None</p>	<ul style="list-style-type: none"> • Parameters: None • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask getDynamicSSLConfigSelection</pre> • Using Jython: <pre>AdminTask.getDynamicSSLConfigSelection()</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask getDynamicSSLConfigSelection {-interactive}</pre> • Using Jython string: <pre>AdminTask.getDynamicSSLConfigSelection ('[-interactive]')</pre> • Using Jython list: <pre>AdminTask.getDynamicSSLConfigSelection (['-interactive'])</pre>

Table 26. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
listDynamic SSLConfig Selections	The listDynamic SSLConfig Selections command lists the configuration objects name for a dynamic SSL configuration selection.	None	<ul style="list-style-type: none"> Parameters: None Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: \$AdminTask listDynamic SSLConfigSelection Using Jython: AdminTask.listDynamic SSLConfigSelection() <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: \$AdminTask listDynamicSSL ConfigSelections {-inter active} Using Jython string: AdminTask.listDynamicSSL ConfigSelections ('[-inte ractive]') Using Jython list: AdminTask.listDynamicSSL ConfigSelections (['-int eractive'])

Commands for the ManagementScopeCommands group of the AdminTask object

Use the commands in the ManagementScopeCommands group to create or delete a management scope. For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the ManagementScopeCommands group of the AdminTask object:

Table 27.

Command name:	Description:	Target object:	Parameters and return values:	Examples:
createManagementScope	The createManagementScope command creates a management scope setting in the configuration.	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - scopeName The name that uniquely identifies the management scope. (String, required) - scopeType The type of the management scope. Valid types include cell, node, nodegroup, cluster, server, and endpoint. (String, optional) • Returns: The configuration object name of the management scope object that you created. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask createManagementScope {-name (cell):localhostNode01Cell -scopeType cell} • Using Jython string: AdminTask.createManagementScope [('-name (cell):localhostNode01Cell -scopeType cell)'] • Using Jython list: AdminTask.createManagementScope [('-name', '(cell):localhostNode01Cell', '-scopeType', 'cell')] <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask createManagementScope {-interactive} • Using Jython string: AdminTask.createManagementScope ('[-interactive]') • Using Jython list: AdminTask.createManagementScope (['-interactive'])

Table 27. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
deleteManagementScope	The deleteManagementScope command deletes a management object from the configuration.	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - scopeName The name that uniquely identifies the management scope. (String, required) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask deleteManagementScope {-scopeName (cell):localhostNode01Cell} • Using Jython string: AdminTask.deleteManagementScope ('[-scopeName (cell):localhostNode01Cell]') • Using Jython list: AdminTask.deleteManagementScope (['-scopeName', '(cell):localhostNode01Cell']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask deleteManagementScope {-interactive} • Using Jython string: AdminTask.deleteManagementScope ('[-interactive]') • Using Jython list: AdminTask.deleteManagementScope (['-interactive'])

Table 27. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
getManagementScope	The getManagementScope command displays the setting of a management scope object.	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - scopeName The name that uniquely identifies the management scope. (String, required) • Returns: The settings of the management scope object. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask getManagementScope {-scopeName (cell):localhostNode01Cell} • Using Jython string: AdminTask.getManagementScope ('[-scopeName (cell):localhostNode01Cell]') • Using Jython list: AdminTask.getManagementScope (['-scopeName', '(cell):localhostNode01Cell']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask getManagementScope {-interactive} • Using Jython string: AdminTask.getManagementScope ('[-interactive]') • Using Jython list: AdminTask.getManagementScope (['-interactive'])

Table 27. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
listManagementScopes	The listManagementScopes command lists the management scopes in the configuration.	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - displayObjectName Set the value to true to display the object names of the management scope. (Boolean, optional) • Returns: A list that contains all of the management scope names. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask listManagementScopes { -name testKM -provider IBMJSSE2 -algorithm IbmX509}</pre> • Using Jython string: <pre>AdminTask.listManagementScopes ('[-name testKM -provider IBMJSSE2 -algorithm IbmX509]')</pre> • Using Jython list: <pre>AdminTask.listManagementScopes (['-name', 'testKM', '-provider', 'IBMJSSE2', '-algorithm', 'IbmX509'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask listManagementScopes {-interactive}</pre> • Using Jython string: <pre>AdminTask.listManagementScopes ('[-interactive]')</pre> • Using Jython list: <pre>AdminTask.listManagementScopes (['-interactive'])</pre>

Commands for the WSCertExpMonitorCommands group of the AdminTask object

Use the commands in the WSCertExpMonitorCommands group to start or update the certificate expiration monitor. For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the WSCertExpMonitorCommands group of the AdminTask object:

Table 28.

Command name:	Description:	Target object:	Parameters and return values:	Examples:
<p>createWSCert ExpMonitor</p>	<p>The createWSCert ExpMonitor command creates the certificate expiration monitor settings in the configuration.</p>	<p>None</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - name The name that uniquely identifies the certificate expiration monitor. (String, required) - autoReplace Set the value of this parameter to true if you want to replace a certificate within a certificate expiration date. If not, set the value of this parameter to false. (Boolean, required) - deleteOld Set the value of this parameter to true if you want to delete an old certificate during certificate expiration monitoring. If not, set the value of this parameter to false. (Boolean, required) - daysBeforeNotification The number of days before a certificate expires that you want to be notified of the expiration. (Integer, required) - wsScheduleName The name of the scheduler to use for certificate expiration. (String, required) 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask createWSCert ExpMonitor {-name test CertMon -autoReplace true -deleteOld true -daysBeforeNotification 30 -wsScheduleName testSchedule -wsNotificationName testNotifier -isEnabled false}</pre> • Using Jython string: <pre>AdminTask.createWSCert ExpMonitor ('[-name testCertMon -autoReplace true -deleteOld true -daysBeforeNotification 30 -wsScheduleName testSchedule -wsNotificationName testNotifier -isEnabled false]')</pre> • Using Jython list: <pre>AdminTask.createWSCert ExpMonitor (['-name', 'testCertMon', '-autoReplace', 'true', '-deleteOld', 'true', '-daysBeforeNotification', '30', '-wsScheduleName', 'testSchedule', '-wsNotificationName', 'testNotifier', '-isEnabled', 'false'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask createWSCert ExpMonitor {-interactive}</pre> • Using Jython string: <pre>AdminTask.createWSCert ExpMonitor ('[-interactive]')</pre> • Using Jython list: <pre>AdminTask.createWSCert ExpMonitor (['-interactive'])</pre>

Table 28. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
			<ul style="list-style-type: none"> - wsNotificationName The name of the notifier to use for certificate expiration. (String, required) - isEnabled Set the value of this parameter to true if the certificate expiration monitor is enabled. If not, set the value of this parameter to false. (Boolean, optional) • Returns: The configuration object name of the certificate expiration monitor object that you created. 	
deleteWSCertExpMonitor	The deleteWSCertExpMonitor command deletes the settings of a scheduler from the configuration.	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - name The name that uniquely identifies the certificate expiration monitor. (String, required) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <code>\$AdminTask deleteWSCertExpMonitor {-name test CertMon}</code> • Using Jython string: <code>AdminTask.deleteWSCertExpMonitor ('[-name test CertMon]')</code> • Using Jython list: <code>AdminTask.deleteWSCertExpMonitor (['-name', 'testCertMon'])</code> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <code>\$AdminTask deleteWSCertExpMonitor {-interactive}</code> • Using Jython string: <code>AdminTask.deleteWSCertExpMonitor ('[-interactive]')</code> • Using Jython list: <code>AdminTask.deleteWSCertExpMonitor (['-interactive'])</code>

Table 28. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
getWSCertExpMonitor	The getWSCertExpMonitor command displays the settings of a particular scheduler.	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - name The name that uniquely identifies the certificate expiration monitor. (String, required) • Returns: The settings of the scheduler that you specified. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask getWSCertExpMonitor {-name testCertMon} • Using Jython string: AdminTask getWSCertExpMonitor ('[-name testCertMon]') • Using Jython list: AdminTask getWSCertExpMonitor (['-name', 'testCertMon']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask getWSCertExpMonitor {-interactive} • Using Jython string: AdminTask.getWSCertExpMonitor ('[-interactive]') • Using Jython list: AdminTask.getWSCertExpMonitor (['-interactive'])

Table 28. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
listWSCertExpMonitor	The listWSCertExpMonitor command lists the scheduler in the configuration.	None	<ul style="list-style-type: none"> • Parameters: - displayObjectNames If you set the value of this parameter to true, the command returns the certificate expiration monitor configuration object. If you set the value of this parameter to false, the command returns the name of the certificate expiration monitor. (Boolean, optional) • Returns: The scheduler in the configuration. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask listWSCertExpMonitor {-displayObjectName false} • Using Jython string: AdminTask.listWSCertExpMonitor ('[-displayObjectName false]') • Using Jython list: AdminTask.listWSCertExpMonitor (['-displayObjectName', 'false']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask listWSCertExpMonitor {-interactive} • Using Jython string: AdminTask.listWSCertExpMonitor ('[-interactive]') • Using Jython list: AdminTask.listWSCertExpMonitor (['-interactive'])

Table 28. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
<p>modifyWSCert ExpMonitor</p>	<p>The modifyWSCert ExpMonitor command changes the setting of an existing scheduler.</p>	<p>None</p>	<ul style="list-style-type: none"> • Parameters: - name The name that uniquely identifies the certificate expiration monitor. (String, required) - autoReplace Set the value of this parameter to true if you want to replace a certificate within a certificate expiration date. If not, set the value of this parameter to false. (Boolean, required) - deleteOld Set the value of this parameter to true if you want to delete an old certificate during certificate expiration monitoring. If not, set the value of this parameter to false. (Boolean, required) - daysBeforeNotification The number of days before a certificate expires that you want to be notified of the expiration. (Integer, required) - wsScheduleName The name of the scheduler to use for certificate expiration. (String, required) 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask modifyWSCert ExpMonitor {-name testCertMon -autoReplace false -deleteOld false -daysBeforeNotification 20 -isEnabled true} • Using Jython string: AdminTask.modifyWSCert ExpMonitor ('[-name testCertMon -autoReplace false -deleteOld false -daysBeforeNotification 20 -isEnabled true]') • Using Jython list: AdminTask.modifyWSCert ExpMonitor (['-name', 'testCertMon', '-autoReplace', 'false', '-deleteOld', 'false', '-daysBeforeNotification', '20', '-isEnabled', 'true']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask modifyWSCert ExpMonitor {-interactive} • Using Jython string: AdminTask.modifyWSCert ExpMonitor ('[-interactive]') • Using Jython list: AdminTask.modifyWSCert ExpMonitor (['-interactive'])

Table 28. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
			<ul style="list-style-type: none"> - wsNotificationName The name of the notifier to use for certificate expiration. (String, required) - isEnabled Set the value of this parameter to true if the certificate expiration monitor is enabled. If not, set the value of this parameter to false. (Boolean, optional) • Returns: None 	
startCertificateExpMonitor	The startCertificateExpMonitor command performs certificate monitoring. This command visits all key stores and checks to see if they are within certificate expiration range.	None	<ul style="list-style-type: none"> • Parameters: None • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask startCertificateExpMonitor • Using Jython: AdminTask.startCertificateExpMonitor() <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask startCertificateExpMonitor {-interactive} • Using Jython string: AdminTask.startCertificateExpMonitor ('[-interactive]') • Using Jython list: AdminTask.startCertificateExpMonitor (['-interactive'])

Commands for the KeySetGroupCommands group of the AdminTask object

For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the KeySetGroupCommands group of the AdminTask object:

Table 29.

Command name:	Description:	Target object:	Parameters and return values:	Examples:
createKeySetGroup	<p>The createKeySetGroup command creates the key set group settings in the configuration. Use this command to manage groups of public, private, and shared keys.</p>	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - name The name that uniquely identifies the key set group. (String, required) - scopeName The name of the scope. (String, optional) - autoGenerate Set the value of this parameter to true if you want to automatically generate keys. If not, set the value to false. (Boolean, optional) - wsScheduleName The name of the scheduler to use to perform key generation. (String, required) - keySetObjectNames A list of key set configuration names separated by colons (:). (String, required) • Returns: The configuration object name of the key set group object that you created. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask createKeySetGroup {-name keySetGrp -autoGenerate true -wsScheduleName testSchedule -keySetObjectNames testKeySet(cells/localhostNode01Cell security.xml#KeySet_1130354347825)}</pre> • Using Jython string: <pre>AdminTask.createKeySetGroup(['-name keySetGrp -autoGenerate true -wsScheduleName testSchedule -keySetObjectNames testKeySet(cells/localhostNode01Cell security.xml#KeySet_1130354347825)'])</pre> • Using Jython list: <pre>AdminTask.createKeySetGroup(['-name', 'keySetGrp', '-autoGenerate', 'true', '-wsScheduleName', 'testSchedule', '-keySetObjectNames', 'testKeySet(cells/localhostNode01Cell security.xml#KeySet_1130354347825)'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask createKeySetGroup {-interactive}</pre> • Using Jython string: <pre>AdminTask.createKeySetGroup(['-interactive'])</pre> • Using Jython list: <pre>AdminTask.createKeySetGroup(['-interactive'])</pre>

Table 29. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
deleteKeySetGroup	The deleteKeySetGroup command deletes the settings of a key set group from the configuration.	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - name The name that uniquely identifies the key set group. (String, required) - scopeName The name of the scope. (String, optional) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask deleteKeySet Group {-name keySetGrp } • Using Jython string: AdminTask.deleteKeySet Group ('[-name keySetGrp]') • Using Jython list: AdminTask.deleteKeySet Group (['-name', 'key SetGrp']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask deleteKey SetGroup {-interactive} • Using Jython string: AdminTask.deleteKeySet Group ('[-interactive]') • Using Jython list: AdminTask.deleteKeySet Group (['-interactive'])

Table 29. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
generateKeys ForKey SetGroup	<p>The generateKeys ForKey SetGroup command generates keys for all of the keys in the key sets that make up the key set group.</p>	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - keySetGroupName The name of the key set group. (String, required) - keySetGroupScope The scope of the key set group. (String, optional) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask generateKey ForKeySetGroup {-keySetGroupName keySetGrp}</pre> • Using Jython string: <pre>AdminTask.generateKey ForKeySetGroup ('[-keySetGroupName keySetGrp]')</pre> • Using Jython list: <pre>AdminTask.generateKey ForKeySetGroup (['-keySetGroupName', 'keySetGrp'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask generateKeys ForKeySetGroup {-interactive}</pre> • Using Jython string: <pre>AdminTask.generateKeys ForKeySetGroup ('[-interactive]')</pre> • Using Jython list: <pre>AdminTask.generateKeys ForKeySetGroup (['-interactive'])</pre>

Table 29. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
getKeySetGroups	<p>The getKeySetGroups command displays the settings of a particular key set group.</p>	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - name The name that uniquely identifies the key set group. (String, required) - scopeName The name of the scope. (String, optional) • Returns: The settings of the specified key set group. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask getKeySetGroup { -name keySetGrp } • Using Jython string: AdminTask.getKeySetGroup ('[-name keySetGrp]') • Using Jython list: AdminTask.getKeySetGroup (['-name', 'keySetGrp']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask getKeySetGroups {-interactive} • Using Jython string: AdminTask.getKeySetGroups (['-interactive']) • Using Jython list: AdminTask.getKeySetGroups (['-interactive'])

Table 29. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
listKeySetGroups	The listKeySetGroups command lists the key set groups for a particular scope.	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - scopeName The name of the scope. (String, optional) - displayObjectNames If you set the value of this parameter to true, the command returns a list of all of the key set group objects within a scope. If you set the value of this parameter to false, the command returns a list of strings that contain the key set group name and management scope. (Boolean, optional) • Returns: A list of key set groups. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask listKeySetGroup {-displayObjectName true} • Using Jython string: AdminTask.listKeySetGroup ('[-displayObjectName true]') • Using Jython list: AdminTask.listKeySetGroup (['-displayObjectName', 'true']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask listKeySetGroups {-interactive} • Using Jython string: AdminTask.listKeySetGroups ('[-interactive]') • Using Jython list: AdminTask.listKeySetGroups (['-interactive'])

Table 29. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
modifyKeySetGroup	The modifyKeySetGroup command changes the settings of an existing key set group.	None	<ul style="list-style-type: none"> Parameters: <ul style="list-style-type: none"> - name The name that uniquely identifies the key set group. (String, required) - scopeName The name of the scope. (String, optional) - autoGenerate Set the value of this parameter to true if you want to automatically generate keys. If not, set the value to false. (Boolean, optional) - wsScheduleName The name of the scheduler to use to perform key generation. (String, optional) - keySetObjectNames A list of key set configuration names separated by colons (:). (String, optional) Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: \$AdminTask modifyKeySet Group {-name keySetGrp -autoGenerate false} Using Jython string: AdminTask.modifyKeySet Group ('[-name keySetGrp -autoGenerate false]') Using Jython list: AdminTask.modifyKeySet Group (['-name', 'keySetGrp', '-autoGenerate', 'false']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: \$AdminTask modifyKeySet Group {-interactive} Using Jython string: AdminTask.modifyKeySet Group ('[-interactive]') Using Jython list: AdminTask.modifyKeySet Group (['-interactive'])

Commands for the KeySetCommands group of the AdminTask object

For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the KeySetCommands group of the AdminTask object:

Table 30.

Command name:	Description:	Target object:	Parameters and return values:	Examples:
createKeySet	The createKeySet command creates the key set settings in the configuration. Use this command to control key instances that have the same type.	None	<ul style="list-style-type: none"> • Parameters: - name The name that uniquely identifies the key set. (String, required) - scopeName The name of the scope. (String, optional) - aliasPrefix The prefix for the key alias when a new key generates. (String, required) - password The password that protects the key in the key store. (String, required) - maxKeyReferences The maximum number of key references returned keys from this key set. (Integer, required) - deleteOldKeys Set the value of this parameter to true to delete old keys when new keys are generated. Otherwise, set the value of this parameter to false. (Boolean, optional) - keyGenerationClass The class that is used to generate new keys in the key set. (String, optional) 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask createKeySet {-name testKeySet -alias Prefix test -password pwd -maxKeyReferences 2 -del eteOldKeys true -keyStor eName testKeyStore -isK eyPair false}</pre> • Using Jython string: <pre>AdminTask.createKeySet(' [-name testKeySet -alias Prefix test -password pwd -maxKeyReferences 2 -deleteOldKeys true -key StoreName testKeyStore -isKeyPair false]')</pre> • Using Jython list: <pre>AdminTask.createKeySet (['-name', 'testKeySet', '-aliasPrefix', 'test', '-password', 'pwd', '-max KeyReferences', '2', '-deleteOldKeys', 'true', '-keyStoreName', 'test KeyStore', '-isKeyPair', 'false'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask createKeySet {-interactive}</pre> • Using Jython string: <pre>AdminTask.createKeySet ('[-interactive]')</pre> • Using Jython list: <pre>AdminTask.createKeySet (['-interactive'])</pre>

Table 30. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
			<ul style="list-style-type: none"> - keyStoreName The key store that contains the keys. (String, required) - - keyStoreScopeName The management scope where the key store is located. (String, optional) - isKeyPair Set the value of this parameter to true if the keys in the key set are key pairs. Otherwise, set the value of this parameter to false. (Boolean, optional) • Returns: The configuration object name of the key set object that you created. 	
deleteKeySet	The deleteKeySet command deletes the settings of a key set from the configuration.	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - name The name that uniquely identifies the key set. (String, required) - scopeName The name of the scope. (String, optional) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask deleteKeySet {-name testKeySet} • Using Jython string: AdminTask.deleteKeySet (['-name testKeySet']) • Using Jython list: AdminTask.deleteKeySet (['-name', 'testKeySet']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask deleteKeySet {-interactive} • Using Jython string: AdminTask.deleteKeySet (['-interactive']) • Using Jython list: AdminTask.deleteKeySet (['-interactive'])

Table 30. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
generateKeyForKeySet	The generateKeyForKeySet command generates keys for the keys in the key set.	None	<ul style="list-style-type: none"> Parameters: <ul style="list-style-type: none"> - keySetName The name of the key set. (String, required) - keySetScope The scope of the key set. (String, optional) - keySetSaveConfig Set the value of this parameter to true to save the configuration of the key set. Otherwise, set the value of this parameter to false. (Boolean, optional) Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: \$AdminTask generateKeyForKeySet { -keySetName testKeySet } Using Jython string: AdminTask.generateKeyForKeySet ('[-keySetName testKeySet]') Using Jython list: AdminTask.generateKeyForKeySet (['-keySetName', 'testKeySet']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: \$AdminTask generateKeyForKeySet {-interactive} Using Jython string: AdminTask.generateKeyForKeySet ('[-interactive]') Using Jython list: AdminTask.generateKeyForKeySet (['-interactive'])
getKeySet	The getKeySet command displays the settings of a particular key set.	None	<ul style="list-style-type: none"> Parameters: <ul style="list-style-type: none"> - name The name that uniquely identifies the key set. (String, required) - scopeName The name of the scope. (String, optional) Returns: The settings of the specified key set group. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: \$AdminTask getKeySet {-name testKeySet} Using Jython string: AdminTask.getKeySet ('[-name testKeySet]') Using Jython list: AdminTask.getKeySet (['-name', 'testKeySet']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: \$AdminTask getKeySet {-interactive} Using Jython string: AdminTask.getKeySet ('[-interactive]') Using Jython list: AdminTask.getKeySet (['-interactive'])

Table 30. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
listKeySets	The listKeySets command lists the key sets in a particular scope.	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - scopeName The name of the scope. (String, optional) - displayObjectNames Set the value of this parameter to true to list the key set configuration objects within the scope. Set the value of this parameter to false if you want to list the strings that contain the key set group name and management scope. (Boolean, optional) • Returns: The key sets for the scope that you specified. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask listKeySets {-displayObjectName true} • Using Jython string: AdminTask.listKeySets (['-displayObjectName true']) • Using Jython list: AdminTask.listKeySets (['-displayObjectName', 'true']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask listKeySets {-interactive} • Using Jython string: AdminTask.listKeySets (['-interactive']) • Using Jython list: AdminTask.listKeySets (['-interactive'])

Table 30. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
modifyKeySet	The modifyKeySet command changes the settings of an existing key set.	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - name The name that uniquely identifies the key set. (String, required) - scopeName The name of the scope. (String, optional) - aliasPrefix The prefix for the key alias when a new key generates. (String, optional) - password The password that protects the key in the key store. (String, optional) - maxKeyReferences The maximum number of key references returned keys from this key set. (Integer, optional) - deleteOldKeys Set the value of this parameter to true to delete old keys when new keys are generated. Otherwise, set the value of this parameter to false. (Boolean, optional) - keyGenerationClass The class that is used to generate new keys in the key set. (String, optional) 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask modifyKeySet {-name testKeySet -maxKeyReferences 3 -deleteOldKeys false} • Using Jython string: AdminTask.modifyKeySet ('[-name testKeySet -maxKeyReferences 3 -deleteOldKeys false]') • Using Jython list: AdminTask.modifyKeySet (['-name', 'testKeySet', '-maxKeyReferences', '3', '-deleteOldKeys', 'false']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask modifyKeySet {-interactive} • Using Jython string: AdminTask.modifyKeySet ('[-interactive]') • Using Jython list: AdminTask.modifyKeySet (['-interactive'])

Table 30. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
			<ul style="list-style-type: none"> - keyStoreName The key store that contains the keys. (String, optional) - - keyStoreScopeName The management scope where the key store is located. (String, optional) - isKeyPair Set the value of this parameter to true if the keys in the key set are key pairs. Otherwise, set the value of this parameter to false. (Boolean, optional) <ul style="list-style-type: none"> • Returns: None 	

Commands for the KeyReferenceCommands group of the AdminTask object

For more information about the AdminTask object, see the [Commands for the AdminTask object](#) article.

The following commands are available for the KeyReferenceCommands group of the AdminTask object:

Table 31.

Command name:	Description:	Target object:	Parameters and return values:	Examples:
createKeyReference	<p>The createKeyReference command creates the key reference setting in the configuration for key set objects.</p>	None	<ul style="list-style-type: none"> • Parameters: - keySetName The name that uniquely identifies the key set to which the key reference belongs. (String, required) - keySetScope The management scope of the key set. (String, optional) - keyAlias The alias name that identifies the key for the key set that you specify. (String, required) - keyPassword The password used for encrypting the key. (String, optional) - keyPasswordVerify The password used for encrypting the key. (String, optional) - version The version of the key reference. (String, optional) - keyReferenceSaveConfig Set the value of this parameter to true to save the key reference to the configuration. Otherwise, set the value to false. (String, optional) 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask createKeyReference {-keySetName testKeySet -keyAlias testKey -password testPWD -passwordVerify testPWD -keyReferenceSaveConfig true} • Using Jython string: AdminTask.createKeyReference ('[-keySetName testKeySet -keyAlias testKey -password testPWD -passwordVerify testPWD -keyReferenceSaveConfig true]') • Using Jython list: AdminTask.createKeyReference (['-keySetName', 'testKeySet', '-keyAlias', 'testKey', '-password', 'testPWD', '-passwordVerify', 'testPWD', '-keyReferenceSaveConfig', 'true']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask createKeyReference {-interactive} • Using Jython string: AdminTask.createKeyReference ('[-interactive]') • Using Jython list: AdminTask.createKeyReference (['-interactive'])

Table 31. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
			<ul style="list-style-type: none"> Returns: The configuration object name of the key reference scope object that you created. 	
deleteKeyReference	<p>The deleteKeyReference command deletes a key reference object from the key set object in the configuration.</p>	None	<ul style="list-style-type: none"> Parameters: <ul style="list-style-type: none"> - keySetName The name that uniquely identifies the key set to which the key reference belongs. (String, required) - keySetScope The management scope of the key set. (String, optional) - keyAlias The alias name that identifies the key for the key set that you specify. (String, required) Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: <pre>\$AdminTask deleteKeyReference { -keySetName testKeySet -keyAlias testKey }</pre> Using Jython string: <pre>AdminTask.deleteKeyReference (['-keySetName testKeySet -keyAlias testKey'])</pre> Using Jython list: <pre>AdminTask.deleteKeyReference (['-keySetName', 'testKeySet', '-keyAlias', 'testKey'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: <pre>\$AdminTask deleteKeyReference {-interactive}</pre> Using Jython string: <pre>AdminTask.deleteKeyReference (['-interactive'])</pre> Using Jython list: <pre>AdminTask.deleteKeyReference (['-interactive'])</pre>

Table 31. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
getKeyReference	The getKeyReference command displays the setting of a key reference object.	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - keySetName The name that uniquely identifies the key set to which the key reference belongs. (String, required) - keySetScope The management scope of the key set. (String, optional) - keyAlias The alias name that identifies the key for the key set that you specify. (String, required) • Returns: The settings of the key reference object. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask getKeyReference { -keySetName test KeySet -keyAlias testKey } • Using Jython string: AdminTask.getKeyReference ('[-keySetName test KeySet -keyAlias testKey]') • Using Jython list: AdminTask.getKeyReference (['-keySetName', 'testKeySet', '-keyAlias', 'testKey']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask getKeyReference {-interactive} • Using Jython string: AdminTask.getKeyReference ('[-interactive]') • Using Jython list: AdminTask.getKeyReference (['-interactive'])

Table 31. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
listKeyReferences	The listKeyReferences command lists the key references for a particular key set in the configuration.	None	<ul style="list-style-type: none"> Parameters: <ul style="list-style-type: none"> - keySetName The name that uniquely identifies the key set to which the key reference belongs. (String, required) - keySetScope The management scope of the key set. (String, optional) Returns: The configuration object name of the key reference scope object that you created. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: \$AdminTask listKeyReferences { -keySetName testKeySet } Using Jython string: AdminTask.listKeyReferences ('[-keySetName testKeySet]') Using Jython list: AdminTask.listKeyReferences (['-keySetName', 'testKeySet']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: \$AdminTask listKeyReferences {-interactive} Using Jython string: AdminTask.listKeyReferences ('[-interactive]') Using Jython list: AdminTask.listKeyReferences (['-interactive'])

Commands for the securityEnablement group of the AdminTask object

For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the securityEnablement group of the AdminTask object:

Table 32.

Command name:	Description:	Target object:	Parameters and return values:	Examples:
GlobalSettings		None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - secureApps Enables application security. The default value is true. (Boolean, required) - secureLocalResources Enables Java 2 Security. The default value is true. (Boolean, required) - ltpaPassword Defines the LTPA password. Valid values include the password used by LTPA. (String, required) - userRegistryType Defines the type of user registry to use. Valid values include: Embedded, LDAP, LocalOS, or Custom. (String, required) • Returns: None 	<p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask GlobalSettings {-interactive} • Using Jython string: AdminTask.GlobalSettings ('[-interactive]') • Using Jython list: AdminTask.GlobalSettings (['-interactive'])

Table 32. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
CustomRegistry	<p>The CustomRegistry command checks that the specified name for the primary administrator does not already exist in the custom registry.</p>	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - primaryAdminName Represents a name of the primary administrator. The value must be a valid administrator name. (String, required) - serverUserName Represents the user name used to connect to internal security routines. The value must be a valid user name. (String, required) - customRegistryClass Represents the class name of the custom user registry implementation of the UserRegistry. The value must be a valid class name for a custom registry. (String, required) - customProperties Custom user registry properties. An array or name-value properties. (String, required) - customName The custom user registry property name. (String, required) - customValue The custom user registry property value that is associated with the property name. (String, required) • Returns: A value of true if the primary administrator does not already exist in the custom registry. Otherwise, returns a value of false. 	<p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask CustomRegistry {-interactive} • Using Jython string: AdminTask.CustomRegistry ('[-interactive]') • Using Jython list: AdminTask.CustomRegistry (['-interactive'])

Table 32. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
EmbeddedRegistry	<p>The EmbeddedRegistry command checks that the specified name for the primary administrator does not already exist in the WIM file-based registry. Returns true if it does not; false, otherwise.</p>	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - primaryAdminName This string represents a name of the primary administrator. The value must be a valid administrator name. (String, required) - adminPassword The password associated with the primary administrator. The value must be a valid administrator password. (String, required) - serverUserName This string represents the user name used to connect to internal security routines. The value must be a valid user name. (String, required) • Returns: A value of true if the specified name for the primary administrator does not already exist in the WIM file-based registry. Otherwise, returns a value of false. 	<p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask EmbeddedRegistry {-interactive} • Using Jython string: AdminTask.EmbeddedRegistry ('[-interactive]') • Using Jython list: AdminTask.EmbeddedRegistry (['-interactive'])

Table 32. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
LDAPRegistry	<p>The LDAPRegistry command checks, after all the inputs for the LDAP registry have been defined, that a connection can be made successfully to the LDAP server. An SSL connection test to LDAP is also supported using this command.</p>	None	<ul style="list-style-type: none"> • Parameters: - IdapServerType The type of LDAP user registry to be connected to WebSphere Application Server. Valid values include: IBM Tivoli Directory Server, SecureWay, Sun ONE, Domino, Active Directory, eDirectory, Custom. (String, required) - IdapHostname The host name for the LDAP server. (String, required) - IdapPort The port to connect to the LDAP server. (Integer, required) - IdapBaseDN The base distinguished name of the directory service, the starting point for searches. (String, required) - IdapBindDN The bind distinguished name, used to bind to the directory server. (String, required) 	<p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask LDAPRegistry {-interactive} • Using Jython string: AdminTask.LDAPRegistry ('[-interactive]') • Using Jython list: AdminTask.LDAPRegistry (['-interactive'])

Table 32. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
			<ul style="list-style-type: none"> - IdapBindPassword The password for the application server, used to bind to the directory service (String, required) - IdapServerUserName The user ID that is used to run WebSphere Application Server for security purposes. (String, required) • Returns: A value of true if the connection was successful. Otherwise, returns a value of false. 	
LocalOSRegistry	The LocalOSRegistry command checks that the specified name for the primary administrator does not already exist in the LocalOS registry.	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - primaryAdminName This string represents a name of the primary administrator. The value must be a valid administrator name. (String, required) - serverUserName This string represents the user name used to connect to internal security routines. The value must be a valid user name. (String, required) • Returns: A value of true if the specified name for the primary administrator does not already exist in the LocalOS registry. Otherwise, returns a value of false. 	Interactive mode example usage: <ul style="list-style-type: none"> • Using Jacl: \$AdminTask LocalOSRegistry {-interactive} • Using Jython string: AdminTask.LocalOSRegistry ('[-interactive]') • Using Jython list: AdminTask.LocalOSRegistry (['-interactive'])

Table 32. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
CurrentSettings	The CurrentSettings command returns a string that represents all of the existing settings set by the wizard that will be read from the workspace instead of from the configuration repository of the <code>security.xml</code> file.	None	<ul style="list-style-type: none"> Parameters: <ul style="list-style-type: none"> - currentWizardSettings A properties object that contains all the current settings selected through the wizard. (Properties, required) Returns: A string of security settings as name-value pairs. 	Interactive mode example usage: <ul style="list-style-type: none"> Using Jacl: <code>\$AdminTask CurrentSettings {-interactive}</code> Using Jython string: <code>AdminTask.CurrentSettings ('[-interactive]')</code> Using Jython list: <code>AdminTask.CurrentSettings (['-interactive'])</code>

Commands for the CertificateRequestCommands group of the AdminTask object

For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the CertificateRequestCommands group of the AdminTask object:

Table 33.

Command name:	Description:	Target object:	Parameters and return values:	Examples:
<p>createCertificateRequest</p>	<p>The createCertificateRequest command creates a certificate request that is associated with a particular key store.</p>	<p>None</p>	<ul style="list-style-type: none"> • Parameters: - keyStoreName The name that uniquely identifies the key store configuration object. (String, required) - keyStoreScope The scope name of the key store. (String, optional) - certificateAlias The name that uniquely identifies the certificate request in a key store. (String, required) - certificateVersion The certificate version. (String, required) - certificateSize (Integer, required) - certificateCommonName (String, required) - certificateOrganization (String, optional) - certificateOrganizationUnit (String, optional) - certificateLocality (String, optional) - certificateState The state code for the certificate. (String, optional) - certificateZip The zip code for the certificate. (String, optional) 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask createCertificateRequest {-keyStoreName testKeyStore -certificateAlias certReq -certificateSize 1024 -certificateCommonName localhost -certificateOrganization testing -certificateRequestFilePath c:\temp\testCertReq.arm} • Using Jython string: AdminTask.createCertificateRequest ('[-keyStoreName testKeyStore -certificateAlias certReq -certificateSize 1024 -certificateCommonName localhost -certificateOrganization testing -certificateRequestFilePath c:\temp\testCertReq.arm] • Using Jython list: AdminTask.createCertificateRequest (['-keyStoreName', 'testKeyStore', '-certificateAlias', 'certReq', '-certificateSize', '1024', '-certificateCommonName', 'localhost', '-certificateOrganization', 'testing', '-certificateRequestFilePath', 'c:\temp\testCertReq.arm']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask createCertificateRequest {-interactive} • Using Jython string: AdminTask.createCertificateRequest ('[-interactive]') • Using Jython list: AdminTask.createCertificateRequest (['-interactive'])

Table 33. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
			<ul style="list-style-type: none"> - certificateCountry The country for the certificate. (String, optional) - certificateValidDays The amount of time in days for which the certificate is valid. (Integer, optional) - - certificateRequestFilePath The file location of the certificate request that can be sent to a certificate authority. (String, required) • Returns: The configuration object name of the key store object that you created. 	
deleteCertificate Request	The deleteCertificateRequest command deletes a certificate request from a key store.	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - keyStoreName The name that uniquely identifies the key store configuration object. (String, required) - keyStoreScope The scope name of the key store. (String, optional) - certificateAlias The name that uniquely identifies the certificate request in a key store. (String, required) • Returns: None 	<p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <code>\$AdminTask deleteCertificateRequest {-interactive}</code> • Using Jython string: <code>AdminTask.deleteCertificateRequest ('[-interactive]')</code> • Using Jython list: <code>AdminTask.deleteCertificateRequest (['-interactive'])</code>

Table 33. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
extractCertificateRequest	The extractCertificateRequest command extracts a certificate request to a file.	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - keyStoreName The name that uniquely identifies the key store configuration object. (String, required) - keyStoreScope The scope name of the key store. (String, optional) - certificateAlias The name that uniquely identifies the certificate request in a key store. (String, required) - certificateRequestFilePath The file location of the certificate request that can be sent to a certificate authority. (String, required) • Returns: A certificate request file is created that contains the extracted certificate. 	<p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask extractCertificateRequest {-interactive} • Using Jython string: AdminTask.extractCertificateRequest ('[-interactive]') • Using Jython list: AdminTask.extractCertificateRequest (['-interactive'])

Table 33. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
getCertificateRequest	The getCertificateRequest command obtains information about a particular certificate request in a key store.	None	<ul style="list-style-type: none"> Parameters: <ul style="list-style-type: none"> - keyStoreName The name that uniquely identifies the key store configuration object. (String, required) - keyStoreScope The scope name of the key store. (String, optional) - certificateAlias The name that uniquely identifies the certificate request in a key store. (String, required) Returns: Information about the certificate request. 	Interactive mode example usage: <ul style="list-style-type: none"> Using Jacl: \$AdminTask getCertificateRequest {-interactive} Using Jython string: AdminTask.getCertificateRequest (['-interactive']) Using Jython list: AdminTask.getCertificateRequest (['-interactive'])
listCertificateRequest	The listCertificateRequest command lists all the certificate requests associated with a particular key store.	None	<ul style="list-style-type: none"> Parameters: <ul style="list-style-type: none"> - keyStoreName The name that uniquely identifies the key store configuration object. (String, required) - keyStoreScope The scope name of the key store. (String, optional) Returns: An attribute list for each certificate request in a key store. 	Interactive mode example usage: <ul style="list-style-type: none"> Using Jacl: \$AdminTask listCertificateRequest {-interactive} Using Jython string: AdminTask.listCertificateRequest (['-interactive']) Using Jython list: AdminTask.listCertificateRequest (['-interactive'])

Commands for the SignerCertificateCommands group of the AdminTask object

For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the SignerCertificateCommands group of the AdminTask object:

Table 34.

Command name:	Description:	Target object:	Parameters and return values:	Examples:
addSignerCertificate	The addSignerCertificate command adds a signer certificate from a certificate file to a key store.	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - keyStoreName The name of the key store where the signer certificate is located. (String, required) - keyStoreScope The management scope of the key store. (String, optional) - certificateFilePath The full path name of the file that contains the signer certificate. (String, required) - certificateAlias The alias name of the signer certificate in the key store. (String, required) - base64Encoded Set the value of this parameter to true if the certificate is ascii base 64 encoded. Set the value of this parameter to false if the certificate is binary. (String, required) • Returns: The configuration object name of the key store object that you created. 	<p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask addSignerCertificate {-interactive} • Using Jython string: AdminTask.addSignerCertificate ('[-interactive]') • Using Jython list: AdminTask.addSignerCertificate (['-interactive'])

Table 34. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
deleteSignerCertificate	The deleteSignerCertificate command deletes a signer certificate from a key store.	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - keyStoreName The name of the key store where the signer certificate is located. (String, required) - keyStoreScope The management scope of the key store. (String, optional) - certificateAlias The alias name of the signer certificate in the key store. (String, required) • Returns: None 	<p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask deleteSignerCertificate {-interactive} • Using Jython string: AdminTask.deleteSignerCertificate ('[-interactive]') • Using Jython list: AdminTask.deleteSignerCertificate (['-interactive'])

Table 34. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
extractSignerCertificate	The extractSignerCertificate command extracts a signer certificate from a key store to a file.	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - keyStoreName The name of the key store where the signer certificate is located. (String, required) - keyStoreScope The management scope of the key store. (String, optional) - certificateAlias The alias name of the signer certificate in the key store. (String, required) - certificateFilePath The full path name of the file that contains the signer certificate. (String, required) - base64Encoded Set the value of this parameter to true if the certificate is ascii base 64 encoded. Set the value of this parameter to false if the certificate is binary. (String, required) • Returns: The certificate file is created and contains the signer certificate. 	<p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask extractSignerCertificate {-interactive} • Using Jython string: AdminTask.extractSignerCertificate ('[-interactive]') • Using Jython list: AdminTask.extractSignerCertificate (['-interactive'])

Table 34. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
getSignerCertificate	The getSignerCertificate command obtains information about a signer certificate from a key store.	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - keyStoreName The name of the key store where the signer certificate is located. (String, required) - keyStoreScope The management scope of the key store. (String, optional) - certificateAlias The alias name of the signer certificate in the key store. (String, required) • Returns: Information about a signer certificate. 	<p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask getSignerCertificate {-interactive} • Using Jython string: AdminTask.getSignerCertificate ('[-interactive]') • Using Jython list: AdminTask.getSignerCertificate (['-interactive'])
listSignerCertificates	The listSignerCertificates command lists all signer certificates in a particular key store.	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - keyStoreName The name of the key store where the signer certificate is located. (String, required) - keyStoreScope The management scope of the key store. (String, optional) • Returns: A list of signer certificate aliases. 	<p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask listSignerCertificates {-interactive} • Using Jython string: AdminTask.listSignerCertificates ('[-interactive]') • Using Jython list: AdminTask.listSignerCertificates (['-interactive'])

Table 34. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
retrieveSignerFromPort	<p>The retrieveSignerFromPort command retrieves a signer from a remote host and stores the signer in a key store.</p>	None	<ul style="list-style-type: none"> • Parameters: - host The host name of the system from where the signer certificate will be retrieved. (String, required) - port The port of the remote system from where the signer certificate will be retrieved. (Integer, required) - keyStoreName The name of the key store where the signer certificate is located. (String, required) - keyStoreScope The management scope of the key store. (String, required) - sslConfigName The name of the SSL configuration object. (String, optional) - sslConfigScopeName The management scope where the SSL configuration object is located. (String, optional) • Returns: The signer certificate is created in the key store file. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask retrieveSignerFromPort {-host serverHost -port 443 -keyStoreName testKeyStore -certificateAlias serverHostSigner} • Using Jython string: AdminTask.retrieveSignerFromPort ('[-host serverHost -port 443 -keyStoreName testKeyStore -certificateAlias serverHostSigner]') • Using Jython list: AdminTask.retrieveSignerFromPort (['-host', 'serverHost', '-port', '443', '-keyStoreName', 'testKeyStore', '-certificateAlias', 'serverHostSigner']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask retrieveSignerFromPort {-interactive} • Using Jython string: AdminTask.retrieveSignerFromPort ('[-interactive]') • Using Jython list: AdminTask.retrieveSignerFromPort (['-interactive'])

Table 34. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
retrieveSigner InfoFromPort	The retrieveSigner InfoFromPort command retrieves signer information from a port on a remote host.	None	<ul style="list-style-type: none"> Parameters: <ul style="list-style-type: none"> - host The host name of the system from where the signer certificate will be retrieved. (String, required) - port The port of the remote system from where the signer certificate will be retrieved. (Integer, required) - sslConfigName The name of the SSL configuration object. (String, optional) - sslConfigScopeName The management scope where the SSL configuration object is located. (String, optional) Returns: Information about the signer certificate from the remote host port. 	Interactive mode example usage: <ul style="list-style-type: none"> Using Jacl: \$AdminTask retrieveSigner InfoFromPort {-interactive} Using Jython string: AdminTask.retrieveSigner InfoFromPort ('[-interactive]') Using Jython list: AdminTask.retrieveSigner InfoFromPort (['-interactive'])

Commands for the PersonalCertificateCommands group of the AdminTask object

For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the personalCertificateCommands group of the AdminTask object:

Table 35.

Command name:	Description:	Target object:	Parameters and return values:	Examples:
<p>createSelfSignedCertificate</p>	<p>The createSelfSignedCertificate command creates a personal certificate in a key store.</p>	<p>None</p>	<ul style="list-style-type: none"> • Parameters: - keyStoreName The name that uniquely identifies the key store configuration object. (String, required) - keyStoreScope The scope name of the key store. (String, optional) - certificateAlias The name that uniquely identifies the certificate request in a key store. (String, required) - certificateVersion The version of the certificate. (String, required) - certificateSize The size of the certificate. (Integer, required) - certificateCommonName The common name of the certificate. (String, required) - certificateOrganization The organization of the certificate. (String, optional) - certificateOrganizationUnit The organizational unit of the certificate. (String, optional) 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask createSelfSignedCertificate {-keyStoreName testKeyStore -certificateAlias default -certificateCommonName localhost -certificateOrganization ibm} • Using Jython string: AdminTask.createSelfSignedCertificate (['-keyStoreName testKeyStore -certificateAlias default -certificateCommonName localhost -certificateOrganization ibm']) • Using Jython list: AdminTask.createSelfSignedCertificate (['-keyStoreName', 'testKeyStore', '-certificateAlias', 'default', '-certificateCommonName', 'localhost', '-certificateOrganization', 'ibm']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask createSelfSignedCertificate {-interactive} • Using Jython string: AdminTask.createSelfSignedCertificate (['-interactive']) • Using Jython list: AdminTask.createSelfSignedCertificate (['-interactive'])

Table 35. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
			<ul style="list-style-type: none"> - certificateLocality The locality of the certificate. (String, optional) - certificateState The state of the certificate. (String, optional) - certificateZip The zip code of the certificate. (String, optional) - certificateCountry The country of the certificate. (String, optional) - certificateValidDays The amount of time in days for which the certificate is valid. (Integer, optional) • Returns: None 	
deleteCertificate	The deleteCertificate command deletes a personal certificate from a key store.	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - keyStoreName The name that uniquely identifies the key store configuration object. (String, required) - keyStoreScope The scope name of the key store. (String, optional) - certificateAlias The name that uniquely identifies the certificate request in a key store. (String, required) • Returns: None 	<p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask deleteCertificate {-interactive} • Using Jython string: AdminTask.deleteCertificate ('[-interactive]') • Using Jython list: AdminTask.deleteCertificate (['-interactive'])

Table 35. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
exportCertificate	<p>The exportCertificate command exports a personal certificate from one key store to another.</p>	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - keyStoreName The name that uniquely identifies the key store configuration object. (String, required) - keyStoreScope The scope name of the key store. (String, optional) - keyStorePassword The password to the key store. (String, required) - keyFilePath The full path to a key store file that is located in a file system. The store from where a certificate will be imported or exported. (String, required) - keyFilePassword The password to the key store file. (String, required) - keyFileType The type of the key file. (String, required) - certificateAlias The name that uniquely identifies the certificate request in a key store. (String, required) - aliasInKeyStore (String, optional) • Returns: None 	<p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <code>\$AdminTask exportCertificate {-interactive}</code> • Using Jython string: <code>AdminTask.exportCertificate ('[-interactive]')</code> • Using Jython list: <code>AdminTask.exportCertificate (['-interactive'])</code>

Table 35. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
extractCertificate	<p>The extractCertificate command extracts the signer part of a personal certificate to a file.</p>	None	<ul style="list-style-type: none"> • Parameters: - keyStoreName The name that uniquely identifies the key store configuration object. (String, required) - keyStoreScope The scope name of the key store. (String, optional) - certificateAlias The name that uniquely identifies the certificate request in a key store. (String, required) - certificateRequestFilePath The full path of the request file that contains the certificate. (String, required) - base64Encoded Set the value of this parameter to true if the certificate is ascii base 64 encoded. Set the value of this parameter to false if the certificate is binary. (Boolean, required) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask extractCertificate {-keyStoreName testKeyStore -certificateFilePath c:\temp\CertFile.arm -certificateAlias testCertificate} • Using Jython string: AdminTask.extractCertificate (['-keyStoreName testKeyStore -certificateFilePath c:\temp\CertFile.arm -certificateAlias testCertificate']) • Using Jython list: AdminTask.extractCertificate (['-keyStoreName', 'testKeyStore', '-certificateFilePath', 'c:\temp\CertFile.arm', '-certificateAlias', 'testCertificate']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask extractCertificate {-interactive} • Using Jython string: AdminTask.extractCertificate (['-interactive']) • Using Jython list: AdminTask.extractCertificate (['-interactive'])

Table 35. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
getCertificate	The getCertificate command obtains information about a particular personal certificate in a key store.	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - keyStoreName The name that uniquely identifies the key store configuration object. (String, required) - keyStoreScope The scope name of the key store. (String, optional) - certificateAlias The name that uniquely identifies the certificate request in a key store. (String, required) • Returns: Information about the certificate request. 	<p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask getCertificate {-interactive} • Using Jython string: AdminTask.getCertificate ('[-interactive]') • Using Jython list: AdminTask.getCertificate (['-interactive'])

Table 35. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
importCertificate	The importCertificate command imports a personal certificate from a key store.	None	<ul style="list-style-type: none"> • Parameters: - keyStoreName The name that uniquely identifies the key store configuration object. (String, required) - keyStoreScope The scope name of the key store. (String, optional) - keyFilePath The full path to a key store file that is located in a file system. The store from where a certificate will be imported or exported. (String, required) - keyFilePassword The password to the key store file. (String, required) - keyFileType The type of the key file. (String, required) - certificateAliasFromKeyFile The certificate alias in the key file from which the certificate is being imported. (String, required) - certificateAlias The name that uniquely identifies the certificate request in a key store. (String, required) • Returns: None 	<p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask importCertificate {-interactive} • Using Jython string: AdminTask.importCertificate ('[-interactive]') • Using Jython list: AdminTask.importCertificate (['-interactive'])

Table 35. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
listPersonalCertificates	<p>The listPersonalCertificates command lists the personal certificates in a particular key store.</p>	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - keyStoreName The name that uniquely identifies the key store configuration object. (String, required) - keyStoreScope The scope name of the key store. (String, optional) • Returns: A list of attributes for each personal certificate in a key store. 	<p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask listPersonalCertificates {-interactive} • Using Jython string: AdminTask.listPersonalCertificates ('[-interactive]') • Using Jython list: AdminTask.listPersonalCertificates (['-interactive'])

Table 35. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
receiveCertificate	<p>The receiveCertificate command receives a signer certificate from a file to a personal certificate.</p>	None	<ul style="list-style-type: none"> • Parameters: - keyStoreName The name that uniquely identifies the key store configuration object. (String, required) - keyStoreScope The scope name of the key store. (String, optional) - certificateAlias The name that uniquely identifies the certificate request in a key store. (String, required) - certificateFilePath The full path of the file that contains the certificate. (String, required) - base64Encoded Set the value of this parameter to true if the certificate is ascii base 64 encoded. Set the value of this parameter to false if the certificate is binary. (Boolean, required) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask receiveCertificate {-keyStoreName testKeyStore -certificateFilePath c:\temp\CertFile.arm} • Using Jython string: AdminTask.receiveCertificate (['-keyStoreName testKeyStore -certificateFilePath c:\temp\CertFile.arm']) • Using Jython list: AdminTask.receiveCertificate (['-keyStoreName', 'testKeyStore', '-certificateFilePath', 'c:\temp\CertFile.arm']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask receiveCertificate {-interactive} • Using Jython string: AdminTask.receiveCertificate (['-interactive']) • Using Jython list: AdminTask.receiveCertificate (['-interactive'])

Table 35. (continued)

Command name:	Description:	Target object:	Parameters and return values:	Examples:
replaceCertificate	<p>The replaceCertificate command replaces a personal certificate with a new one. Replaces all signer certificates from the personal certificate.</p>	None	<ul style="list-style-type: none"> • Parameters: - keyStoreName The name that uniquely identifies the key store configuration object. (String, required) - keyStoreScope The scope name of the key store. (String, optional) - certificateAlias The name that uniquely identifies the certificate request in a key store. (String, required) - replacementCertificateAlias The alias of the certificate that is used to replace a different certificate. (String, required) - deleteOldCert Set the value of this parameter to true if you want to delete the old signer certificates during certificate replacement. Otherwise, set the value of this parameter to false. (Boolean, optional) - deleteOldSigners Set the value of this parameter to true if you want to delete the old certificates during certificate replacement. Otherwise, set the value of this parameter to false. (Boolean, optional) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask replaceCertificate {-keyStoreName testKeyStore -certificateAlias default -replacementCertificateAlias replaceCert -deleteOldCert true -deleteOldSigners true} • Using Jython string: AdminTask.replaceCertificate ('[-keyStoreName testKeyStore -certificateAlias default -replacementCertificateAlias replaceCert -deleteOldCert true -deleteOldSigners true]') • Using Jython list: AdminTask.replaceCertificate (['-keyStoreName', 'testKeyStore', '-certificateAlias', 'default', '-replacementCertificateAlias', 'replaceCert', '-deleteOldCert', 'true', '-deleteOldSigners', 'true']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: \$AdminTask replaceCertificate {-interactive} • Using Jython string: AdminTask.replaceCertificate ('[-interactive]') • Using Jython list: AdminTask.replaceCertificate (['-interactive'])

Commands for the SPNEGO TAI group of the AdminTask object

For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the SPNEGO TAI group of the AdminTask object:

Command name:	Description:	Target object:	Parameters and return values:	Examples:
addSpnegoTAI Properties	The addSpnegoTAI Properties command adds properties in the configuration of the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) for WebSphere Application Server.	None	<ul style="list-style-type: none"> Parameters: <ul style="list-style-type: none"> - spnld This is the SPN identifier for the group of custom properties that are to be defined with this command. If you do not specify this parameter, an unused SPN identifier is assigned. (String, optional) - host Specifies the host name portion in the SPN used by the SPNEGO TAI to establish a Kerberos secure context. (String, required) - filter Defines the filtering criteria used by the class specified with the above attribute. If no filter is specified, all HTTP requests are subject to SPNEGO authentication. (String, optional) - filterClass Specifies the name of the Java class used by the SPNEGO TAI to select which HTTP requests will be subject to SPNEGO authentication. If no filter class is specified, the default filter class, <code>com.ibm.ws.security.spnego.HTTPHeaderFilter</code>, is used. (String, optional) 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: <code>\$AdminTask addSpnegoTAIProperties -host myhost.ibm.com -filter user-agent%=IE 6</code> Using Jython string: <code>AdminTask.addSpnegoTAIProperties ('[-host myhost.ibm.com -filter user-agent%=IE 6]')</code> Using Jython list: <code>AdminTask.addSpnegoTAIProperties (['-host', 'myhost.ibm.com', '-filter', 'user-agent%=IE', '6'])</code> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: <code>\$AdminTask addSpnegoTAIProperties -interactive</code> Using Jython string: <code>AdminTask.addSpnegoTAIProperties (['-interactive'])</code> Using Jython list: <code>AdminTask.addSpnegoTAIProperties ['-interactive'])</code>

			<p>- noSpnegoPage Specifies the URL of a resource that contains the content the SPNEGO TAI will include in the HTTP response to be displayed by the (browser) client application if it does not support SPNEGO authentication. (String, optional).</p> <p>If you do not specify the noSpnegoPage attribute then the default is used:</p> <pre>"<html><head><title> SPNEGO authentication is not supported. </title></head>" + "<body>SPNEGO authe ntication is not supported on this client.</body> </html>";</pre> <p>- ntlmTokenPage Specifies the URL of a resource that contains the content the SPNEGO TAI will include in the HTTP response to be displayed by the (browser) client application when the SPNEGO token received by the interceptor after the challenge-response handshake contains a NT LAN manager (NTLM) token instead of the expected SPNEGO token. (String, optional).</p> <p>If you do not specify the ntlmTokenPage attribute then the default is used:</p> <pre>"<html><head><title> An NTLM Token was received.</title> </head>" + "<body>Your bro wser configuration is correct, but you have not logged into a supported Windows Domain." + "<p>Please login to the application using the normal login page.</html>";</pre>	
--	--	--	--	--

			<p>- trimUserName Specifies whether (true) or not (false) the SPNEGO TAI is to remove the suffix of the principal user name, starting from the @ that precedes the Kerberos realm name. If this attribute is set to true, the suffix of the principal user name is removed. If this attribute is set to false, the suffix of the principal name is retained. The default value used is true. (String, optional)</p> <ul style="list-style-type: none">• Returns: None	
--	--	--	--	--

<p>createKrb ConfigFile</p>	<p>The createKrb ConfigFile command creates the Kerberos configuration file for use with the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) for WebSphere Application Server.</p>	<p>None</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - krbPath Provides the fully qualified file system location of the Kerberos configuration (krb5.ini or krb5.conf) file. (String, required) - realm Provides the Kerberos realm name. The value of this attribute is used by the SPNEGO TAI to form the Kerberos service principal name for each of the hosts specified with the property <code>com.ibm.ws.security.spnego.SPN<id>.hostname</code> (String, required) - kdcHost Provides the host name of the Kerberos Key Distribution Center (KDC). (String, required) - kdcPort Provides the port number of the KDC. The default value, if not specified, is 88. (String, optional) - dns Provides the default domain name service (DNS) that is used to produce a fully qualified host name. (String, required) - keytabPath Provides the file system location of the Kerberos keytab file. (String, required) - encryption Identifies the list of supported encryption types, separated by a space. The specified value is used for the <code>default_tkt_ectypes</code> and <code>default_tgs_ectypes</code>. The default encryption types, if not specified, are <code>des-cbc-md5</code> and <code>rc4-hmac</code>. (String, optional) • Returns: None 	<p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <code>\$AdminTask createKrbConfigFile -interactive</code> • Using Jython string: <code>AdminTask.createKrbConfigFile ('[-interactive]')</code> • Using Jython list: <code>AdminTask.createKrbConfigFile ['-interactive']</code>
---------------------------------	---	-------------	--	---

deleteSpnego TAIProperties	The deleteSpnegoTAIProperties command deletes properties in the configuration of the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) for WebSphere Application Server.	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - spnId The SPN identifier for the group of custom properties that are to be deleted with this command. If you do not specify this parameter, all SPNEGO TAI custom properties are deleted. (String, optional) • Returns: None 	Batch mode example usage: <ul style="list-style-type: none"> • Using Jacl: \$AdminTask deleteSpnegoTAIProperties {-spnId 2} • Using Jython string: AdminTask.deleteSpnegoTAIProperties ('[-spnId 2]') • Using Jython list: AdminTask.deleteSpnegoTAIProperties (['-spnId', '2']) Interactive mode example usage: <ul style="list-style-type: none"> • Using Jacl: \$AdminTask deleteSpnegoTAIProperties -interactive • Using Jython string: AdminTask.deleteSpnegoTAIProperties (['-interactive']) • Using Jython list: AdminTask.deleteSpnegoTAIProperties ['-interactive'])
-------------------------------	---	------	---	---

<p>modifySpnegoTAIProperties</p>	<p>The modifySpnegoTAIProperties command modifies the properties in the configuration of the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) for WebSphere Application Server.</p>	<p>None</p>	<ul style="list-style-type: none"> • Parameters: - spnId The SPN identifier for the group of custom properties that are to be defined with this command. (String, required) - host Specifies the host name portion in the SPN used by the SPNEGO TAI to establish a Kerberos secure context. (String, optional) - filter Defines the filtering criteria used by the class specified with the above attribute. (String, optional) - filterClass Specifies the name of the Java class used by the SPNEGO TAI to select which HTTP requests will be subject to SPNEGO authentication. If no class is specified, all HTTP requests will be subject to SPNEGO authentication. (String, optional) - noSpnegoPage Specifies the URL of a resource that contains the content the SPNEGO TAI will include in the HTTP response to be displayed by the (browser) client application if it does not support SPNEGO authentication. (String, optional) 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <code>\$AdminTask modifySpnegoTAI PROPERTIES -spnId 1 -filter host==myhost.company.com</code> • Using Jython string: <code>AdminTask.modifySpnegoTAI PROPERTIES (['-spnId 1 -filter host==myhost.com pany.com']')</code> • Using Jython list: <code>AdminTask.modifySpnegoTAI PROPERTIES (['-spnId', '1', '-filter', 'host==my host.company.com']')</code> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <code>\$AdminTask modifySpnegoTAI Properties -interactive</code> • Using Jython string: <code>AdminTask.modifySpnegoTAI Properties (['-interactive']')</code> • Using Jython list: <code>AdminTask.modifySpnegoTAI Properties ['-interactive'])</code>
----------------------------------	---	-------------	---	---

			<p>- ntlmTokenPage Specifies the URL of a resource that contains the content the SPNEGO TAI will include in the HTTP response to be displayed by the (browser) client application when the SPNEGO token received by the interceptor after the challenge-response handshake contains a NT LAN manager (NTLM) token instead of the expected SPNEGO token. (String, optional)</p> <p>- trimUserName Specifies whether (true) or not (false) the SPNEGO TAI is to remove the suffix of the principal user name, starting from the "@" that precedes the Kerberos realm name. If this attribute is set to true, the suffix of the principal user name is removed. If this attribute is set to false, the suffix of the principal name is retained. The default value used is true. (String, optional)</p> <ul style="list-style-type: none"> • Returns: None 	
--	--	--	--	--

showSpnego TAI Properties	The showSpnego TAI Properties command displays the properties in the configuration of the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) trust association interceptor (TAI) for WebSphere Application Server.	None	<ul style="list-style-type: none"> Parameters: <ul style="list-style-type: none"> - spnid The service principal name (SPN) identifier for the group of custom properties that are to be displayed with this command. If you do not specify this parameter, all SPNEGO TAI custom properties are displayed. (String, optional) Returns: A list of properties. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: <code>\$AdminTask showSpnegoTAI Properties -spnid 1</code> Using Jython string: <code>AdminTask.showSpnegoTAI Properties (['-spnid 1'])</code> Using Jython list: <code>AdminTask.showSpnegoTAI Properties (['-spnid', '1'])</code> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: <code>\$AdminTask showSpnegoTAI Properties -interactive</code> Using Jython string: <code>AdminTask.showSpnegoTAI Properties (['-interactive'])</code> Using Jython list: <code>AdminTask.showSpnegoTAI Properties ['-interactive'])</code>
---------------------------------	--	------	---	---

Commands for the AuthorizationGroupCommands group of the AdminTask object

For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the AuthorizationGroupCommands group of the AdminTask object:

Command name:	Description:	Target object:	Parameters and return values:	Examples:
---------------	--------------	----------------	-------------------------------	-----------

<p>addResourceToAuthorizationGroup</p>	<p>The addResourceToAuthorizationGroup command adds a resource instance to an existing authorization group. A resource instance cannot belong to more than one authorization group.</p>	<p>None</p>	<ul style="list-style-type: none"> Parameters: <ul style="list-style-type: none"> - authorizationGroupName The name of the authorization group. (String, required) - resourceName The name of the resource instance that you want to add to an authorization group. (String, required) <p>The resourceName parameter should be in the following format: ResourceType= ResourceName</p> <p>where ResourceType is one of the following values: Application, Server, ServerCluster, Node, NodeGroup</p> <p>ResourceName is the name of the resource instance, for example, server1.</p> <p>The following are example uses of the resourceName parameter:</p> <ul style="list-style-type: none"> - Node=node1:Server=server1 <p>This example uniquely identifies server1. node1 is required if another server1 exists on a different node.</p> <ul style="list-style-type: none"> - Application=app1 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: <code>\$AdminTask addResourceToAuthorizationGroup {-authorizationGroupName groupName -resourceName Application=app1}</code> Using Jython string: <code>AdminTask.addResourceToAuthorizationGroup(['-authorizationGroupName groupName -resourceName Application=app1'])</code> Using Jython list: <code>AdminTask.addResourceToAuthorizationGroup(['-authorizationGroupName', 'groupName', '-resourceName', 'Application=app1'])</code> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: <code>\$AdminTask addResourceToAuthorizationGroup {-interactive}</code> Using Jython string: <code>AdminTask.addResourceToAuthorizationGroup (['-interactive'])</code> Using Jython list: <code>AdminTask.addResourceToAuthorizationGroup (['-interactive'])</code>
			<ul style="list-style-type: none"> Returns: None 	

<p>createAuthorizationGroup</p>	<p>The createAuthorizationGroup command creates a new authorization group. When you create a new authorization group, no members are associated with it. Also, no user to administrative role mapping for the authorization table is associated with the authorization group.</p>	<p>None</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - authorizationGroupName The name of the authorization group that you want to create. (String, required) • Returns: The configuration ID of the authorization group that you created. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask createAuthorizationGroup {-authorizationGroupName groupName}</pre> • Using Jython string: <pre>AdminTask.createAuthorizationGroup(['-authorizationGroupName groupName'])</pre> • Using Jython list: <pre>AdminTask.createAuthorizationGroup(['-authorizationGroupName', 'groupName'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask createAuthorizationGroup -interactive</pre> • Using Jython string: <pre>AdminTask.createAuthorizationGroup(['-interactive'])</pre> • Using Jython list: <pre>AdminTask.createAuthorizationGroup(['-interactive'])</pre>
---------------------------------	--	-------------	--	---

deleteAuthorizationGroup	The deletes an existing authorization group. When you delete an authorization group, the authorization table that corresponds is also deleted.	None	<ul style="list-style-type: none"> Parameters: <ul style="list-style-type: none"> - authorizationGroup Name The name of the authorization group that you want to delete. (String, required) Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: <code>\$AdminTask deleteAuthorizationGroup {-authorizationGroupName groupName}</code> Using Jython string: <code>AdminTask.deleteAuthorizationGroup('[-authorizationGroupName groupName]')</code> Using Jython list: <code>AdminTask.deleteAuthorizationGroup(['-authorizationGroupName', 'groupName'])</code> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: <code>\$AdminTask deleteAuthorizationGroup {-interactive}</code> Using Jython string: <code>AdminTask.deleteAuthorizationGroup ('[-interactive]')</code> Using Jython list: <code>AdminTask.deleteAuthorizationGroup (['-interactive'])</code>
	The command lists the existing authorization groups.	None	<ul style="list-style-type: none"> Parameters: None Returns: A list of short names of all existing authorization groups. (String []) 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: <code>\$AdminTask listAuthorizationGroups</code> Using Jython: <code>AdminTask.listAuthorizationGroups()</code> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: <code>\$AdminTask listAuthorizationGroups {-interactive}</code> Using Jython string: <code>AdminTask.listAuthorizationGroups ('[-interactive]')</code> Using Jython list: <code>AdminTask.listAuthorizationGroups (['-interactive'])</code>

listAuthorizationGroupsForGroupID	The listAuthorizationGroupsForGroupID command lists all of the authorization groups to which a given user group has access. This command lists the authorization groups and the granted roles for each authorization group. The group ID can be a short name or a fully qualified domain name if the LDAP user registry is being used. This command will list cell as a group if the user has cell level access.	None	<ul style="list-style-type: none"> Parameters: <ul style="list-style-type: none"> - groupid The ID of the user group. (String, required) Returns: The map of the authorization group and granted roles. (Map[String=String[]]) 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: <code>\$AdminTask listAuthorizationGroupsForGroupID {-groupid <i>userGroupName</i>}</code> Using Jython string: <code>AdminTask.listAuthorizationGroupsForGroupID(['-groupid <i>userGroupName</i>'])</code> Using Jython list: <code>AdminTask.listAuthorizationGroupsForGroupID(['-groupid', '<i>userGroupName</i>'])</code> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: <code>\$AdminTask listAuthorizationGroupsForGroupID {-interactive}</code> Using Jython string: <code>AdminTask.listAuthorizationGroupsForGroupID ('[-interactive]')</code> Using Jython list: <code>AdminTask.listAuthorizationGroupsForGroupID (['-interactive'])</code>
-----------------------------------	---	------	---	---

listAuthorizationGroupsForUser ID	The listAuthorizationGroupsForUser ID command lists all of the authorization groups to which a given user has access. This command lists the authorization groups and the granted roles for each authorization group. The user ID and the group ID can be a short name or a fully qualified domain name if the LDAP user registry is being used. This command will list cell as a group if the user has cell level access.	None	<ul style="list-style-type: none"> Parameters: <ul style="list-style-type: none"> - userid The ID of the user. (String, required) Returns: The map of the authorization group and granted roles. (Map[String=String[]]) 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: <code>\$AdminTask listAuthorizationGroupsForUserID{-userid userName}</code> Using Jython string: <code>AdminTask.listAuthorizationGroupsForUserID(['-userid userName'])</code> Using Jython list: <code>AdminTask.listAuthorizationGroupsForUserID(['-userid', 'userName'])</code> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: <code>\$AdminTask listAuthorizationGroupsForUserID {-interactive}</code> Using Jython string: <code>AdminTask.listAuthorizationGroupsForUserID (['-interactive'])</code> Using Jython list: <code>AdminTask.listAuthorizationGroupsForUserID (['-interactive'])</code>
-----------------------------------	---	------	--	---

listAuthorizationGroupsOfResource	The listAuthorizationGroupsOfResource command lists authorization groups for a given resource. If the value of the <code>traverseContainedObjects</code> parameter is false, only the authorization group of the resource is returned. If the value of the <code>traverseContainedObjects</code> parameter is true, it returns the authorization group of the resource and the authorization groups of all the parent resources in the containment tree.	None	<ul style="list-style-type: none"> Parameters: <ul style="list-style-type: none"> - resourceName The name of the resource. (String, required) The <code>resourceName</code> parameter must be in the following format: ResourceType=ResourceName where <code>ResourceType</code> can be any one of the following values: Application, Server, ServerCluster, Node, or NodeGroup. <code>ResourceName</code> is the name of the resource instance, for example, <code>server1</code>. The following are examples of the <code>resourceName</code> parameter: Node=node1: Server=server This example uniquely identifies <code>server1</code>. The name of the node is required if a server on a different node uses the same server name. Application=app1 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: <code>\$AdminTask listAuthorizationGroupsOfResource {-resourceName Application=app1}</code> Using Jython string: <code>AdminTask.listAuthorizationGroupsOfResource(['-resourceName Application=app1'])</code> Using Jython list: <code>AdminTask.listAuthorizationGroupsOfResource(['-resourceName', 'Application=app1'])</code> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: <code>\$AdminTask listAuthorizationGroupsOfResource {-interactive}</code> Using Jython string: <code>AdminTask.listAuthorizationGroupsOfResource(['-interactive'])</code> Using Jython list: <code>AdminTask.listAuthorizationGroupsOfResource(['-interactive'])</code>
-----------------------------------	---	------	---	--

			<p>- traverseContainedResources</p> <p>Finds the authorization groups of all the parent resources by traversing the resource containment tree upwards. The default value is false. (Boolean, optional)</p> <ul style="list-style-type: none"> Returns: The short names of all of the authorization groups for which the resource belongs. If you do not specify the traverseContainedResources parameter, the result of this command will only contain one value because a resource instance can only belong to one authorization group. (String[]) 	
listResourcesOfAuthorizationGroup	The listResourcesOfAuthorizationGroup command lists all of the resources within the given authorization group.	None	<ul style="list-style-type: none"> Parameters: <ul style="list-style-type: none"> - authorizationGroupName The name of the authorization group. (String, required) Returns: The configuration IDs of all of the resource instances within the authorization group. (String[]) 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: <pre>\$AdminTask listResourcesOfAuthorizationGroup {-authorizationGroupName groupName}</pre> Using Jython string: <pre>AdminTask.listResourcesOfAuthorizationGroup(['-authorizationGroupName groupName'])</pre> Using Jython list: <pre>AdminTask.listResourcesOfAuthorizationGroup(['-authorizationGroupName', 'groupName'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: <pre>\$AdminTask listResourcesOfAuthorizationGroup {-interactive}</pre> Using Jython string: <pre>AdminTask.listResourcesOfAuthorizationGroup(['-interactive'])</pre> Using Jython list: <pre>AdminTask.listResourcesOfAuthorizationGroup(['-interactive'])</pre>

listResourcesForGroupID	<p>The listResourcesForGroupID command lists all the objects that a given group has access to. This command lists the resources and the granted roles for each resource. The resources that this command returns include the resources from the authorization groups to which the user group is granted roles and the resources that are descendants of the resources with in authorization groups to which the user group is granted access to any role. The group ID can be a short name or fully qualified domain name if a LDAP user registry is used.</p>	None	<ul style="list-style-type: none"> Parameters: <ul style="list-style-type: none"> - groupid The ID of the user group. (String, required) Returns: The map of the granted role and resources. (Map[String=String[]]) 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: <code>\$AdminTask listResourcesForGroupID {-groupid <i>userGroupName</i>}</code> Using Jython string: <code>AdminTask.listResourcesForGroupID(['-groupid <i>userGroupName</i>'])</code> Using Jython list: <code>AdminTask.listResourcesForGroupID(['-groupid', 'userGroupName'])</code> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: <code>\$AdminTask listResourcesForGroupID {-interactive}</code> Using Jython string: <code>AdminTask.listResourcesForGroupID ('[-interactive]')</code> Using Jython list: <code>AdminTask.listResourcesForGroupID (['-interactive'])</code>
-------------------------	---	------	--	--

listResourcesForUserID	<p>The listResourcesForUserID command lists all the objects that a given user has access to. This command lists the resources and the granted roles for each resource. The resources that this command returns include the resources from the authorization groups to which the user is granted roles and the resources that are descendants of the resources with in authorization groups to which the user is granted access to any role. The user ID can be a short name or fully qualified domain name if a LDAP user registry is used.</p>	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - userid The ID of the user. (String, required). • Returns: The map of granted role and resources. (Map[String=String[]]) 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <code>\$AdminTask listResourcesForUserID {-userid userName }</code> • Using Jython string: <code>AdminTask.listResourcesForUserID('[-userid userName]')</code> • Using Jython list: <code>AdminTask.listResourcesForUserID(['-userid', 'userName'])</code> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <code>\$AdminTask listResourcesForUserID {-interactive}</code> • Using Jython string: <code>AdminTask.listResourcesForUserID ('[-interactive]')</code> • Using Jython list: <code>AdminTask.listResourcesForUserID (['-interactive'])</code> <p>Example output:</p> <pre>{deployer=[], operator=[], administrator=[cells/IBM-LP16L31HVE8Cell107/clusters/C1 cluster.xml, cells/IBM-LP16L31HVE8Cell107/nodes/IBM-LP16L31HVE8Node05/servers/cm1 server.xml], monitor=[], configurator=[]}</pre>
------------------------	--	------	--	---

<p>mapGroupsToAdminRole</p>	<p>The mapGroupsToAdminRole command maps group IDs in an administrative roles in an authorization group. The name of the authorization group that you provide determines which authorization table will be used. If you do not specify an authorization group name, the mapping is done to the cell level authorization table. The group ID can be a short name or a fully qualified domain name if the LDAP user registry is used.</p>	<p>None</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - authorizationGroupName The name of the authorization group. If you do not specify this parameters, the cell level authorization group is assumed. (String, optional) - roleName The name of the administrative role. (String, required) - groupids The list of group IDs that will mapped to the administrative role. (String[], required) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask mapGroupsToAdminRole {-authorizationGroupName group Name - roleName administrator -groupids group1}</pre> • Using Jython string: <pre>AdminTask.mapGroupsToAdminRole ('[-authorizationGroupName group Name -roleName administrator -groupids group1]')</pre> • Using Jython list: <pre>AdminTask.mapGroupsToAdminRole (['-authorizationGroupName', 'groupName', '-roleName', 'administrator', '-groupids', 'group1'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask mapGroupsToAdminRole {-interactive}</pre> • Using Jython string: <pre>AdminTask.mapGroupsToAdminRole ('[-interactive]')</pre> • Using Jython list: <pre>AdminTask.mapGroupsToAdminRole (['-interactive'])</pre>
-----------------------------	--	-------------	---	---

<p>mapUsersToAdminRole</p>	<p>The mapUsersToAdminRole command maps user IDs to one or more administrative roles in the authorization group. The name of the authorization group that you provide determines the authorization table. If you do not specify the name of the authorization group, the mapping is done to the cell level authorization table. The user ID can be a short name or fully qualified domain name in case LDAP user registry is used.</p>	<p>None</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - authorizationGroupName The name of the authorization group. If you do not specify this parameter, the cell level authorization group is assumed. (String, optional) - roleName The name of the administrative role. (String, required) - userids The list of user IDs that will be mapped to the administrative role (String[], required) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask mapUsersToAdminRole {-authorizationGroupName group Name - roleName administrator -userids user1}</pre> • Using Jython string: <pre>AdminTask.mapUsersToAdminRole ('[-authorizationGroupName groupName -roleName administ rator -userids user1]')</pre> • Using Jython list: <pre>AdminTask.mapUsersToAdminRole (['-authorizationGroupName', 'groupName', '-roleName', 'administrator', '-userids', 'user1'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask mapUsersToAdmin Role {-interactive}</pre> • Using Jython string: <pre>AdminTask.mapUsersToAdmin Role ('[-interactive]')</pre> • Using Jython list: <pre>AdminTask.mapUsersToAdmin Role (['-interactive'])</pre>
----------------------------	---	-------------	--	--

<p>removeGroupsFromAdminRole</p>	<p>The removeGroupsFromAdminRole command removes previously mapped group IDs from administrative roles in the authorization group. The name of the authorization group that you provide determines which authorization table is involved. If you do not specify an authorization group name, the group IDs are removed from the cell level authorization table. The group ID can be a short name or fully qualified domain name if a LDAP user registry is used.</p>	<p>None</p>	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - authorizationGroupName The name of the authorization group. If you do not specify this parameter, the cell level authorization group is assumed. (String, optional) - roleName The name of the administrative role. (String, required) - userids A list of group IDs that you want to remove from the administrative role. (String[], required) • Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask removeGroupsFromAdminRole {-authorizationGroupName groupName - roleName administrator -groupids group1}</pre> • Using Jython string: <pre>AdminTask.removeGroupsFromAdminRole('[-authorizationGroupName groupName -roleName administrator -groupids group1]')</pre> • Using Jython list: <pre>AdminTask.removeGroupsFromAdminRole(['-authorizationGroupName', 'groupName', '-roleName', 'administrator', '-groupids', 'group1'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask removeGroupsFromAdminRole {-interactive}</pre> • Using Jython string: <pre>AdminTask.removeGroupsFromAdminRole ('[-interactive]')</pre> • Using Jython list: <pre>AdminTask.removeGroupsFromAdminRole (['-interactive'])</pre>
----------------------------------	---	-------------	--	---

<p>remove Resource From AuthorizationGroup</p>	<p>The removeResourceFromAuthorizationGroup command removes resources from an existing authorization group. If you do not specify the authorization group, it will be determined and the resource will be removed from that authorization group.</p>	<p>None</p>	<ul style="list-style-type: none"> Parameters: <ul style="list-style-type: none"> - authorizationGroup Name The name of the authorization group. (String, optional) - resourceName The name of the resource instance that you want to remove from the authorization group. (String, required) <p>The resourceName parameter must be in the following format:</p> <pre>ResourceType= ResourceName</pre> <p>where the ResourceType can be any of the following: Application, Server, ServerCluster, Node, or NodeGroup.</p> <p>The ResourceName is the name of the resource instance, for example, server1.</p> <p>The following are examples of the resourceName parameter:</p> <pre>Node=node1: Server=server1</pre> <p>This example uniquely identifies server1. node1 is required if the name of the server exists on multiple nodes.</p> <pre>Application=app1</pre> Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: <pre>\$AdminTask removeResourceFromAuthorizationGroup {-authorizationGroupName groupName -resourceName Application=app1}</pre> Using Jython string: <pre>AdminTask.removeResourceFromAuthorizationGroup(['-authorizationGroupName groupName -resourceName Application=app1'])</pre> Using Jython list: <pre>AdminTask.removeResourceFromAuthorizationGroup(['-authorizationGroupName', 'groupName', '-resourceName', 'Application=app1'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: <pre>\$AdminTask removeResourceFromAuthorizationGroup {-interactive}</pre> Using Jython string: <pre>AdminTask.removeResourceFromAuthorizationGroup (['-interactive'])</pre> Using Jython list: <pre>AdminTask.removeResourceFromAuthorizationGroup (['-interactive'])</pre>
--	---	-------------	--	--

removeUsersFromAdminRole	The removeUsersFromAdminRole command removes previously mapped user IDs from administrative roles in the authorization group. The name of the authorization group that you provide determines which authorization table is involved. If you do not specify an authorization group name, the user ID from the cell level authorization table will be used. The user ID can be a short name or a fully qualified domain name if a LDAP user registry is used.	None	<ul style="list-style-type: none"> Parameters: <ul style="list-style-type: none"> - authorizationGroupName The name of the authorization group. If you do not specify this parameter, the cell level authorization group is assumed. (String, optional) - roleName The name of the administrative role. (String, required) - userids A list of user IDs that you want to remove from the administrative role. (String[], required) Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: <code>\$AdminTask removeUsersFromAdminRole {-authorizationGroupName groupName - roleName administrator -userids user1}</code> Using Jython string: <code>AdminTask.removeUsersFromAdminRole('[-authorizationGroupName groupName -roleName administrator -userids user1]')</code> Using Jython list: <code>AdminTask.removeUsersFromAdminRole(['-authorizationGroupName', 'groupName', '-roleName', 'administrator', '-userids', 'user1'])</code> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: <code>\$AdminTask removeUsersFromAdminRole {-interactive}</code> Using Jython string: <code>AdminTask.removeUsersFromAdminRole ('[-interactive]')</code> Using Jython list: <code>AdminTask.removeUsersFromAdminRole (['-interactive'])</code>
--------------------------	--	------	---	---

Commands for the ChannelFrameworkManagement group of the AdminTask object

For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the ChannelFrameworkManagement group of the AdminTask object:

Command name:	Description:	Target object:	Parameters and return values:	Examples:
---------------	--------------	----------------	-------------------------------	-----------

createChain	The createChain command creates a new chain of transport channels that are based on a chain template.	The instance of the transport channel service under which the new chain is created. (ObjectName, required)	<ul style="list-style-type: none"> Parameters: <ul style="list-style-type: none"> - template The chain template on which to base the new chain. (ObjectName, required) - name The name of the new chain. (String, required) - endPoint The name of the end point to be used by the instance of the TCP inbound channel in the new chain if the chain is an inbound chain. (ObjectName, optional) Returns: The object name of the channel chain that was created. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: <pre>\$AdminTask createChain (cells/rohitbuildCell01/nodes/rohitbuildCellManager01/servers/dmgr server.xml#TransportChannelService_1) {-template WebContainer(templates/chains webcontainer-chains.xml#Chain_1) -name trialChain1} \$AdminTask createChain (cells/rohitbuildCell01/nodes/rohitbuildCellManager01/servers/dmgr server.xml#TransportChannelService_1) {-template WebContainer(templates/chains webcontainer-chains.xml#Chain_1) -name trialChain1 -endPoint (cells/rohitbuildCell01/nodes/rohitbuildCellManager01 serverindex.xml#EndPoint_3) }</pre> Using Jython string: <pre>AdminTask.createChain('cells/rohitbuildCell01/nodes/rohitbuildCellManager01/servers/dmgr server.xml#TransportChannelService_1', '[-template "WebContainer(templates/chains webcontainer-chains.xml#Chain_1)" -name trialChain]')</pre> <pre>AdminTask.createChain('cells/rohitbuildCell01/nodes/rohitbuildCellManager01/servers/dmgr server.xml#TransportChannelService_1', '[-template "WebContainer(templates/chains webcontainer-chains.xml#Chain_1)" -name trialChain -endPoint "(cells/rohitbuildCell01/nodes/rohitbuildCellManager01 serverindex.xml#EndPoint_3)"]')</pre> Using Jython list: <pre>AdminTask.createChain('cells/rohitbuildCell01/nodes/rohitbuildCellManager01/servers/dmgr server.xml#TransportChannelService_1', ['-template', "WebContainer(templates/chains webcontainer-chains.xml#Chain_1)", '-name', 'trialChain']) AdminTask.createChain('cells/rohitbuildCell01/nodes/rohitbuildCellManager01/servers/dmgr server.xml#TransportChannelService_1', ['-template', "WebContainer(templates/chains webcontainer-chains.xml#Chain_1)", '-name', 'trialChain', '-endPoint', "(cells/rohitbuildCell01/nodes/rohitbuildCellManager01 serverindex.xml#EndPoint_3)"])</pre>
-------------	--	--	---	---

				<p>Interactive mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: \$AdminTask createChain {-interactive} Using Jython string: AdminTask.createChain ('[-interactive]') Using Jython list: AdminTask.createChain (['-interactive'])
deleteChain	The deleteChain command deletes an existing chain and, optionally, the transport channels in the chain.	The chain to be deleted. (Object name, required)	<ul style="list-style-type: none"> Parameters: <ul style="list-style-type: none"> - deleteChannels If the value of this attribute is true, non-shared transport channels used by the specified chain will be deleted. (Boolean, optional) Returns: None 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: \$AdminTask deleteChain trialChain1 (cells/rohitbuildCell101/nodes/rohitbuildCellManager01/servers/dmgr server.xml#Chain_1093554462922) \$AdminTask deleteChain trialChain (cells/rohitbuildCell101/nodes/rohitbuildCellManager01/servers/dmgr server.xml#Chain_1093554378078) {-deleteChannels true} Using Jython string: AdminTask.deleteChain('trialChain1(cells/rohitbuildCell101/nodes/rohitbuildCellManager01/servers/dmgr server.xml#TransportChannelService_1)') AdminTask.deleteChain('trialChain1(cells/rohitbuildCell101/nodes/rohitbuildCellManager01/servers/dmgr server.xml#TransportChannelService_1)', '[-deleteChannels true]') Using Jython list: AdminTask.deleteChain('trialChain1(cells/rohitbuildCell101/nodes/rohitbuildCellManager01/servers/dmgr server.xml#TransportChannelService_1)') AdminTask.deleteChain('trialChain1(cells/rohitbuildCell101/nodes/rohitbuildCellManager01/servers/dmgr server.xml#TransportChannelService_1)', ['-deleteChannels', 'true']) <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: \$AdminTask deleteChain {-interactive} Using Jython string: AdminTask.deleteChain ('[-interactive]') Using Jython list: AdminTask.deleteChain (['-interactive'])

listChain Templates	The listChain Templates command displays a list of templates that you can use to create chains in this configuration. All templates have a certain type of transport channel as the last transport channel in the chain.	None	<ul style="list-style-type: none"> • Parameters: <ul style="list-style-type: none"> - acceptorFilter The templates returned by this method all have a transport channel instance of the specified type as the last transport channel in the chain. (String, optional) • Returns: A list of all the chain template object names. If you specify the <code>acceptorFilter</code> parameter, the list that returns is filtered to match the filter that you specified. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask listChainTemplates {} \$AdminTask listChainTemplates "-acceptorFilter WebContainer InboundChannel"</pre> • Using Jython string: <pre>AdminTask.listChainTemplates() AdminTask.listChainTemplates (['-acceptorFilter WebContainerInboundChannel'])</pre> • Using Jython list: <pre>AdminTask.listChainTemplates() AdminTask.listChainTemplates (['-acceptorFilter', 'WebContainerInboundChannel'])</pre> <p>Interactive mode example usage:</p> <ul style="list-style-type: none"> • Using Jacl: <pre>\$AdminTask listChainTemplates {-interactive}</pre> • Using Jython string: <pre>AdminTask.listChainTemplates ('[-interactive]')</pre> • Using Jython list: <pre>AdminTask.listChainTemplates (['-interactive'])</pre>
---------------------	---	------	--	---

listChains	The listChains command lists all the chains that are configured under a particular instance of the transport channel service.	The instance of the transport channel service under which the chains are configured. (ObjectName, required)	<ul style="list-style-type: none"> Parameters: <ul style="list-style-type: none"> - acceptorFilter The chains that are returned by this parameter will have a transport channel instance of the type that you specify as the last transport channel in the chain. (String, optional) - endPointFilter: The chains returned by this parameter will have a TCP inbound channel using an end point with the name that you specify.(String, optional) Returns: A list of all the channel chain object names that match the specified filters. If no you do not specify any parameters, all of the channel chains that are configured under the particular instance of transport channel service are returned. 	<p>Batch mode example usage:</p> <ul style="list-style-type: none"> Using Jacl: <pre>\$AdminTask listChains (cells/rohitbuildCell101/nodes/rohitbuildNode01/servers/server2 server.xml#TransportChannelService_1093445762328) \$AdminTask listChains (cells/rohitbuildCell101/nodes/rohitbuildNode01/servers/server2 server.xml#TransportChannelService_1093445762328) {-acceptorFilter WebContainerInboundChannel} \$AdminTask listChains (cells/rohitbuildCell101/nodes/rohitbuildNode01/servers/server2 server.xml#TransportChannelService_1093445762328) {-endPointFilter WC_adminhost}</pre> Using Jython string: <pre>AdminTask.listChains('(cells/rohitbuildCell101/nodes/rohitbuildNode01/servers/server2 server.xml#TransportChannelService_1093445762328)') AdminTask.listChains('(cells/rohitbuildCell101/nodes/rohitbuildNode01/servers/server2 server.xml#TransportChannelService_1093445762328)', '[-acceptorFilter WebContainerInboundChannel]') AdminTask.listChains('(cells/rohitbuildCell101/nodes/rohitbuildNode01/servers/server2 server.xml#TransportChannelService_1093445762328)', '[-endPointFilter WC_adminhost]')</pre> Using Jython list: <pre>AdminTask.listChains('(cells/rohitbuildCell101/nodes/rohitbuildNode01/servers/server2 server.xml#TransportChannelService_1093445762328)') AdminTask.listChains('(cells/rohitbuildCell101/nodes/rohitbuildNode01/servers/server2 server.xml#TransportChannelService_1093445762328)', ['-acceptorFilter', 'WebContainerInboundChannel']) AdminTask.listChains('(cells/rohitbuildCell101/nodes/rohitbuildNode01/servers/server2 server.xml#TransportChannelService_1093445762328)', ['-endPointFilter', 'WC_adminhost'])</pre>
------------	--	---	--	--

				<p>Interactive mode example usage:</p> <ul style="list-style-type: none">• Using Jacl: <code>\$AdminTask listChains {-interactive}</code>• Using Jython string: <code>AdminTask.listChains ('[-interactive]')</code>• Using Jython list: <code>AdminTask.listChains (['-interactive'])</code>
--	--	--	--	--

Chapter 10. Web applications

Securing Web applications using an assembly tool

You can use three types of Web login authentication mechanisms to configure 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.

Note: This procedure might not match the steps that are required when using your assembly tool, or match the version of the assembly tool that you are using. You should follow the instructions for the tool and version that you are using.

The following steps detail securing a Web application using an assembly tool:

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 "Importing Web archive (WAR) files" and "Importing an enterprise application EAR file" in the Application Server Toolkit documentation.
2. In the Project Explorer folder, locate your Web application.
3. Right-click the deployment descriptor and click **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 select a Web archive (WAR) file, a Web deployment descriptor editor opens. If you select an enterprise application (EAR) file, an application deployment descriptor editor opens.
4. Create security roles either at the application level or at the 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 of 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, click 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, click the Security tab. Under the list of security roles, click **Add**. In the Add Security Role wizard, name and describe the security role and 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, you can do the following actions:
 - 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** The application does not require transport guarantees.
 - Integral**
Data cannot be changed in transit between the client and the server.
 - Confidential**
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. The security runtime uses the exact match first to map the incoming URL with URL patterns. If the exact match is not present, the security runtime 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. The role-name field contains the name of the role that is referenced in the servlet or JavaServer Pages (JSP) code to determine if the caller is in a specified role. Because 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. The Security-role-ref element is at the servlet level. A servlet or JavaServer Pages (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. You can define five types of references on this tab:
 - EJB reference
 - Service reference
 - Resource reference
 - Message destination reference
 - Security role reference
 - Resource environment reference
 - b. Under the list of Enterprise JavaBeans (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 that is used during development to the role using the previous steps. Every role name that is 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 JSP**, click **Add**. The Add Servlet or JSP wizard opens.
 - b. Specify the servlet or JavaServer Pages (JSP) file 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. Click **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 that is used by your Web application.
 8. Configure the login mechanism for the Web module. This configured login mechanism applies to all the servlets, JavaServer Pages (JSP) files and HTML resources in the Web module.

- a. Click the **Pages** tab of a Web deployment descriptor editor and click **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 Web address. For example, you might use `/login.jsp` or `/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 Web archive (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 “Deploying secured applications” on page 913 to finish securing the Web application.

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.
- 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 Uniform Resource Identifier (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).

Security settings

Use the administrative console to modify the security settings for all applications. You can enable security for applications by enabling the **Enable application security** option on the Secure administration, applications, and infrastructure panel.

Note that:

- Global settings apply to existing and future applications and cannot be customized.
- Default settings apply only to future applications and can be customized.

The default settings are used as a template or starting point for configuring individual applications. The administrator should still explicitly configure security settings for each application.

The following security settings are specified during application assembly:

Security role settings

When using the Assembly Toolkit at an application level (Enterprise Archive (EAR) file), security roles are synchronized with the security roles defined for the embedded modules of the application.

If a security role is manually added to the EAR file, it can be automatically removed when the file is saved if an embedded module does not reference the role, or the role is in conflict with an existing role. In this case, remove the manually added role, but then all roles with the same name are removed.

The role is automatically added again when the file is saved if it is still referenced in an embedded module file. If a duplicate role is added in an embedded module file, delete all roles with the same name and manually read the correct role.

Security constraints

Security constraints declare how to protect Web content. 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.

Security constraints are set when configuring a Web application in the Assembly Toolkit.

Security role references

Web application developers or Enterprise JavaBeans (EJB) providers must use a role-name in the code when using the available programmatic security Java 2 Platform, Enterprise Edition (J2EE) application programming interfaces (APIs) `isUserInRole(String roleName)` and `isCallerInRole(String roleName)`.

The roles used in the deployed run-time environment might not be known until the Web application and EJB components (for example, Web archive (WAR) files and `ejb-jar.xml` files) are assembled into an enterprise archive (EAR) file. Therefore, the role names used in the Web application or EJB component code are logical role names which the application assembler maps to the actual run-time environment roles during application assembly. The security role references provide a level of indirection that insulate Web application component and EJB developers from having to know the actual roles in the run-time environment.

The definition of the logical roles and the mapping to the actual run-time environment roles are specified in the `security-role-ref` element of both the Web application and the EJB JAR file deployment descriptors, `web.xml` and `ejb-jar.xml` respectively. Use the assembly tools to define the role names and map them to the actual run-time roles in the environment with the `role-link` element.

The following code sample is an example of a `security-role-ref` from an EJB `ejb-jar.xml` deployment descriptor.

```
... <enterprise-beans>
... <entity>
<ejb-name>AardvarkPayroll</ejb-name>
<ejb-class>com.aardvark.payroll.PayrollBean</ejb-class>
...
<security-role-ref>
<description>
```

This role should be assigned to the employees of the payroll department. Members of this role have access to the payroll record of everyone. The role has been linked to the payroll-department role. This role should be assigned to the employees of the payroll department. Members of this role have access to all payroll records. The role has been linked to the payroll-department role.

```
</description> <role-name>payroll</role-name>
<role-link>payroll-department</role-link>
</security-role-ref>
...
</entity>
...
</enterprise-beans>
```

In the previous example, the string `payroll`, which appears in the `<role-name>` element, is what the EJB provider uses as the argument to the `isCallerInRole()` API. The `<role-link>` element is what ties the logical role to the actual role used in the run-time environment.

Note that for enterprise beans, the `security-role-ref` element must appear in the deployment descriptor even if the logical role name is the same as the actual role name in the environment.

The rules Web application components are slightly different. If no `security-role-ref` element matching a `security-role` element is declared, the container must default to checking the `role-name` element argument against the list of `security-role` elements for the Web application. The `isUserInRole` method references the list to determine whether the caller is mapped to a security role. The developer must be aware that the use of this default mechanism can limit the flexibility in changing role names in the application without having to recompile the servlet making the call.

See the EJB Version 2.0 and Servlet Version 2.3 specification in the Security: Resources for Learning article for complete details on this specification.

Securing applications during assembly and deployment

Several assembly tools exist that are graphical user interfaces for assembling enterprise or Java 2 Platform, Enterprise Edition (J2EE) applications. You can use these tools to assemble an application and secure Enterprise JavaBeans (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 JavaServer Pages (JSP) file, or a servlet. You can also enforce security for each Web resource.

Note: For information about the tools that WebSphere Application Server supports, see Assembly tools.

To secure an EJB module, a Java archive (JAR) file, a Web module, a Web archive (WAR) file, or an application enterprise archive (EAR) file, you can use an assembly tool. You can create an application, an EJB module, or a Web module and secure them using an assembly tool or development tools such as the IBM Rational Application Developer.

1. Secure EJB applications using an assembly tool. For more information, see “Securing enterprise bean applications” on page 919.
2. Secure Web applications using an assembly tool. For more information, see “Securing Web applications using an assembly tool” on page 899.
3. Add users and groups-to-roles while assembling a secured application using an assembly tool. For more information, see “Adding users and groups to roles using an assembly tool” on page 905.
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 909.
5. “Adding the `was.policy` file to applications” on page 525.
6. Assemble the application components that you 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 that are 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 protected by Java 2 security.

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 “Deploying secured applications” on page 913 to complete this task.

Assigning users and groups to roles

This topic describes how to assign users and groups to roles if you are using WebSphere Application Server authorization for Java 2 Platform, Enterprise Edition (J2EE) roles.

Before you perform this task:

- Secure the Web applications and Enterprise JavaBeans (EJB) applications where new roles are created and assigned to Web and enterprise bean 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 you save the configuration and restart the server before the changes become effective. For example, enable security or change the user registry.

These steps are common for both installing an application and modifying an existing application. If the application contains roles, you see the Security role to user/group mapping link during application installation and also during application management, as a link in the Additional properties section.

1. Access the administrative console.
Type `http://localhost:port_number/ibm/console` in a Web browser.
2. Click **Applications > Enterprise applications > application_name**.
3. Under Detail properties, click **Security role to user/group mapping**. A list of all the roles that belong to this application is displayed. If the roles already have users or All Authentication or Everyone special subjects assigned, they display here.
4. To assign the special subjects, select either the **Everyone** or the **All Authenticated** option for the appropriate roles.
5. 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.
6. Click **Look up users** or **Look up groups**.
7. Get the appropriate users and groups from the user registry by completing the Limit and the Search string fields and by clicking **Search**. The Limit field limits the number of users that are obtained and displayed from the user 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 user registry. When you use large user registries such as 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 slow and can make it fail. When more entries exist than requests for entries, a message displays on top of the panel. You can refine your search until you have the required list.
8. Select the users and groups to include as members of these roles from the **Available** field and click **>>** to add them to the roles.
9. 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 the user1 user is assigned to the role1 RunAs role and you try to remove the user1 user from the role1 role, the administrative console validation does not delete the user. A user can only be part of a RunAs role if the user is already in a role either directly or indirectly through a group. In this case, the user1 user is in the role1 role. For more information on the validation checks that are performed between RunAs role mapping and user and group mapping to roles, see “Assigning users to RunAs roles” on page 908.

10. Click **OK**. If any validation problems exist between the role assignments and the RunAs role assignments, the changes are not committed and an error message that indicates the problem displays at the top of the panel. If a problem exists, make sure that the user in the RunAs role is also a member of the regular role. If the regular role contains a group that 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. After the application is installed and running you can access your resources according to the user and group mapping that you did in this task. If you manage applications and modify 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.

Adding users and groups to roles using an assembly tool

After creating new roles and assigning them to enterprise bean and Web resources, use this task to add users and groups to roles with an assembly tool.

Before you perform this task, you already completed the steps in “Securing Web applications using an assembly tool” on page 899 and “Securing enterprise bean applications” on page 919 where you created new roles and assigned those roles to enterprise bean and Web resources. Complete these steps during application installation. 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, you can use an assembly tool to assign users and groups to roles. Using the administrative console to assign users and groups to roles is recommended.

Note: This procedure might not match the steps that are required when using your assembly tool, or match the version of the assembly tool that you are using. You should follow the instructions for the tool and version that you are using.

To add users and groups to roles using an assembly tool, follow these steps:

1. In the Project Explorer view of an assembly tool, right-click an enterprise application project, or Enterprise Archive (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 panel, click **Add**.
3. In the Add Security Role wizard, name and describe the security role. 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 added all the 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.xml` file in the application contains the users and groups-to-roles mapping table, which is the *authorization table*.

After securing an application, install the application using the administrative console.

Mapping users to roles

Use this page to specify the users and groups that are mapped to the security roles that are used with the enterprise application.

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, click **Security role to user/group mapping** under Detailed properties to change user and group mappings to a role.

Look up users:

Enables the server to locate the users that you can define for a particular security role.

Select the check box beside the role and click **Look up users**. Complete the Limit and the Search string fields. The Limit field contains the number of entries that the search function returns. The Search string 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:

Enables the server to locate the groups that you can define for a particular security role.

Select the check box beside the role and click **Look up groups**. Complete the Limit and the Search string fields. The Limit field contains the number of entries that the search function returns. The Search string 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:

Maps specific capabilities to a user. 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.

For example, you might map the user Joe to the administrator role, which enables user Joe to perform all of the tasks associated with the administrator role.

The authorization policy is only enforced when global security is enabled.

Everyone:

Specifies whether to map everyone to a specified role. When you map everyone to a role, anyone can access the resources that are protected by this role and, essentially, there is no security.

All authenticated:

Specifies whether to map all of the authenticated users to a specified role. When you map all authenticated users to a specified role, all of the valid users in the current registry who have been authenticated can access resources that are protected by this role.

Mapped users:

Lists the users that are mapped to the specified role within this application.

Mapped groups:

Lists the groups that are mapped to this specified role within this application.

Look up users and groups settings

Use this page to select users and groups for security roles.

To view this administrative console page, complete the following steps:

1. Click **Applications > Enterprise applications > *application_name***.
2. Under Detail properties, click **Security role to user/group mapping**.
3. Select the role and click either **Look up users** or **Look up groups**.

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 and groups are associated with roles defined in an application when the application is installed or configured. Use the Search pattern field to display users in the Available list. Click >> to add users from the Available list to the Selected list.

Limit:

Specifies the maximum number of users or groups that can be returned when assigning users/groups to roles.

A value of 0 implies a return of all users or 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

Search string:

Indicates the search pattern used to search for the entries in a user registry.

The Search string field contains the search pattern that is used to search for the user or group entries. For example, bob* will search all users or groups starting with bob. A limit of zero (0) retrieves all of the entries that match the pattern. Use a limit of zero (0) only when a small number users or groups match that pattern in the user registry. If the user registry contains more entries that match the pattern than requested for, a message shows in the administrative console to indicate that there are more entries in the user registry.

Data type	String
Units	Number of users
Default	20
Range	A-Z with *

Assigning users to RunAs roles

This article explains how to assign users to the RunAs roles for your application.

Complete the following tasks:

- Secure the Web applications and the EJB applications where new RunAs roles are 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 904 for more information.
- Verify that the user registry requirements are met. These requirements are the same as those discussed in “Assigning users and groups to roles” on page 904. For example, if the role1 role is a role that is also used as a RunAs role, then the user1 user can be added to the RunAs role. 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 is displayed at the top of the panel.

When a user ID and password is assigned to a RunAs role, validation occurs using the current active user registry that is configured. By default, the local operating system registry is set as the active user registry. Therefore, when an application is installed and security is disabled on the server, the local operating system registry is used to validate the user ID and password that is assigned to the RunAs Role. If the intended registry for the application is not local operative system, the validation fails. Therefore, map RunAs roles to users when the security is enabled on the server. However, if the active user registry and the intended registry after enabling security are the same, you can assign the user to a RunAs role when security is disabled.

If the Everyone or All Authenticated special subjects are assigned to a role, validation does not occur for that role.

Validation is done every time you click **Apply** in this panel or when you click **OK** in the Security role to user/group mapping 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 Security role to user/group mappings panel. If a role is assigned both a user and a group to which that user belongs, you can delete either the user or the group from the Security role to user/group mapping 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 the administrative console and not through an assembly tool or 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 user registry. Because the list of groups that are obtained from the user 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, 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 User RunAs roles link during application installation and also during managing applications as a link in the Additional properties section.

1. Click **Applications > Enterprise Applications > *application_name***.
2. Under Additional properties, click **User RunAs roles**. A list of all the RunAs roles that belong to this application display. If the roles already have 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, which is preferred, or the full name, as seen when getting users and groups from the user 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. After the application is installed and running, you can access your resources according to the RunAS role mapping. Save the configuration.

If you manage applications and modify User RunAs roles, make sure you save, stop, and restart the application so that the changes become effective. Try accessing your Java 2 Platform, Enterprise Edition (J2EE) resources to verify that the new changes are in effect.

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 899 and “Securing enterprise bean applications” on page 919.
- Assign users and groups to roles. For more information, see “Adding users and groups to roles using an assembly tool” on page 905. 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.

Note: This procedure might not match the steps that are required when using your assembly tool, or match the version of the assembly tool that you are using. You should follow the instructions for the tool and version that you are using.

To define RunAs roles when a servlet or an enterprise bean in an application is configured with RunAs settings, perform these steps:

1. In the Project Explorer view of an assembly tool, right-click an enterprise application project or Enterprise Archive (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.xml` 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 “User RunAs collection” on page 912.

Ensure all unprotected 1.x methods have the correct level of protection:

Use this page to verify that the unprotected Enterprise JavaBeans (EJB) Version 1.x methods have the correct level of protection before you map users to roles.

This administrative console panel is displayed during the application deployment process. To access the administrative console panel, click **Application > Install new application > application_name**. The panel is displayed as *Ensure all unprotected 1.x methods have the correct level of protection* in the application deployment steps. On this administrative console panel, you can specify whether users can access specific EJB modules.

EJB module:

Specifies the EJB module name.

URI:

Specifies the Uniform Resource Identifier (URI) that is used to locate the Java archive (JAR) file for the EJB module.

Deny all access:

Select this option to protect this EJB module by making it inaccessible to users regardless of their access permissions.

Default: Cleared

Ensure all unprotected 2.x methods have the correct level of protection:

Use this page to verify that the unprotected Enterprise JavaBeans (EJB) Version 2.x methods have the correct level of protection before you map users to roles.

This administrative console panel is displayed during the application deployment process. To access the administrative console panel, click **Application > Install new application > application_name**. The panel is displayed as *Ensure all unprotected 2.x methods have the correct level of protection* in the application deployment steps. On this administrative console panel, you can specify whether users can access specific EJB modules.

To use this administrative console page, select the **Uncheck**, **Exclude**, or **Role** option, the check box next to the EJB module, and click **Apply**. If you select **Role** option, select the appropriate role for the EJB module before you click **Apply**.

Uncheck:

Select this option if you do not want the application server to verify the access permissions for the EJB module. Everyone can access the EJB module.

Default: Selected

Exclude:

Select this option to protect this EJB module by making it inaccessible to users regardless of their access permissions.

Default: Deselected

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.

Default: Deselected

EJB module:

Specifies the name of the module.

If a module name appears in this list, then the module contains unprotected EJB methods.

URI:

Specifies the Uniform Resource Identifier (URI) that is used to locate the Java archive (JAR) file for the EJB module.

Protection type:

Specifies the level of protection that is assigned to a particular module name.

After you select the **Uncheck**, **Exclude**, or **Role** option and click **Apply**, the selected protection option is displayed in this column.

Correct use of the system identity:

Use this page to manage the system identity properties for the Enterprise JavaBeans (EJB) method in your application.

This administrative console panel is displayed during the application deployment process. To access the administrative console panel, click **Applications > Install new application > application_name**. The panel is displayed as *Correct use of System Identity* in the application deployment steps.

To use this panel, complete the following steps:

1. Select the check box next to the EJB method.
2. Select a role that is defined for this enterprise bean.
3. Specify a user name and password for the RunAs role. The user name must be defined in your user registry.
4. Click **Apply**.

The specified user will be assigned to the specified RunAs role for the EJB method that you selected.

Role:

Specifies the RunAs role that is used for this EJB method.

Username:

Specifies the user name that is assigned to the RunAs role for this EJB method.

The user name is used in conjunction with the RunAs role that you select for the Role.

Password:

Specifies the password that is associated with the user name in the user registry.

User RunAs collection:

Use this page to map a specified user identity and password to a RunAs role. This panel enables you to specify application-specific privileges for individual users to run specific tasks using another user identity.

To view this administrative console page, complete the following steps:

1. Click **Applications > Enterprise applications > application_name**.
2. Under Detail properties, click **User RunAs roles**.

The enterprise beans that you install 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.

Username:

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. After you specify the user name and password for the user and select a RunAs role, click **Apply**.

Data type: String

Password:

Specifies the password for the RunAs user.

Data type: String

Role:

Maps specific capabilities to a user.

The authorization policy is only enforced when global security is enabled.

Updating and redeploying secured applications

This section addresses the way to update existing applications.

Before you perform this task, secure Web applications, secure Enterprise JavaBeans (EJB) applications, and deploy them in WebSphere Application Server.

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 904.
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 908.
3. Complete and save the changes.
4. Stop and restart the application for the changes to become effective.
5. Use 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 that you update the user and group information using the administrative console. After the secured applications are modified and either restarted or redeployed, verify that the changes are effective by accessing the resources in the application.

Deploying secured applications

Deploying applications that have security constraints (secured applications) is not much different than deploying applications that do not contain any security constraints. The only difference is that you might need to assign users and groups to roles for a secured application. The secured application requires that you have the correct active user registry.

Before you perform this task, verify that you already designed, developed, and assembled an application with all the relevant security configurations. For more information on these tasks refer to “Developing applications that use programmatic security” on page 510 and “Securing applications during assembly and deployment” on page 903. In this context, deploying and installing an application are considered the same task.

To deploy a newly secured application click **Applications > Install New Application** and follow the prompts to complete the installation steps. One of the required steps to deploy secured applications is to assign users and groups to roles that are defined in the application.

- If you are installing a secured application, roles will be defined in the application.
- If delegation is required in the application, you will be defining RunAs roles also.

During the installation of a new application, the role definition is completed as part of the step that maps security roles to users and groups. If this assignment has already been completed by using an assembly tool, you can still confirm the mapping by going through this installation step. You can add new users and groups and modify existing information during this step.

If the application supports delegation, a RunAs role will already be defined in the application. If the delegation policy is set to `Specified Identity` during assembly, the intermediary invokes a method by 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 that calls the enterprise beans, the method on the enterprise beans is invoked with `bob` as the identity.

As part of the new application installation and deployment process, one of the steps is to map or modify users to the RunAs roles. Use this step to assign new users or modify existing users to RunAs roles when the delegation policy is set to Specified Identity.

Note that the steps are common whether you are installing an application or modifying an existing application.

To install and deploy the application, complete the following steps.

1. Click **Applications > Install New Application**. Complete the required steps until you see the step for mapping security roles to users and groups.
2. If the application contains roles, assign users and groups to roles. At this step during the installation, under Additional Properties, click **Map security roles to users and groups**. For more information, see “Assigning users and groups to roles” on page 904.
3. If RunAs roles exist in the application, assign users to RunAs roles. At this step during the installation, under Additional Properties, click **Map RunAs roles to users**. For more information, see “Assigning users to RunAs roles” on page 908.
4. **Optional:** 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, which is applicable to enterprise beans only. System identity uses the WebSphere Application Server security server ID to invoke downstream methods. Using system identity is not recommended as this ID has more privileges than other identities in 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. When the internalServerId feature is used, runAs with system identity is not supported; you must specify RunAs roles here.
5. Complete the remaining non-security related steps to finish installing and deploying the application.

After 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 can use the application.

Chapter 11. SIP applications

Securing SIP applications

You can apply digest authentication and Trust Association Interceptor (TAI) for a SIP application by applying Lightweight Directory Access Protocol (LDAP) security to the application.

Before you can apply security, you must first deploy an application that has been developed to support security (with the web.xml file configured for security) and roles. The following software must also be installed:

1. Install Tivoli Directory Server version 5.2
2. Set up and activate Lightweight Third Party Authentication. For more information, see the Lightweight Third Party Authentication section.

To apply LDAP security to a SIP application, click **Applications** → **Enterprise Applications** → **applicationName** and complete the following steps:

1. Click **Detail Properties** → **Security role to user/group mapping**.
2. Check **All Authenticated**.
3. Save all changes.
4. Restart the server.

Configuring security for the SIP container

This section provides instructions specific to security for the SIP container, employed with software such as Tivoli Directory Server or Oracle Internet Directory.

Before you can configure security for your SIP container, you will need to:

1. Set up and activate Lightweight Third Party Authentication. For more information, see the Lightweight Third Party Authentication section.
2. Install Oracle Internet Directory or Tivoli Directory Server.

You may also need to:

- Adjust key group settings. Refer to Lightweight Third Party Authentication key sets and key set groups for LTPA key information.
- Establish and configure Trust Association Interceptor (TAI) settings. Refer to Trust association interceptor settings.

You must know the name of the key set group and the management scope where the key set group is defined in order to activate and secure LTPA with keys. Refer to Activating Lightweight Third Party Authentication key versions for the setup and activation procedures.

To configure security based on the Lightweight Directory Access Protocol (LDAP), you can configure digest authentication by using either Tivoli Directory Server or Oracle Internet Directory.

- To configure digest authentication and TAI on WebSphere Application Server for Tivoli, select “Configuring digest authentication and TAI for Tivoli Directory Server” on page 916.
- For configuring digest authentication on WebSphere Application Server for Oracle Internet Directory, select “Configuring digest authentication for Oracle Internet Directory” on page 917.

If setting up a TAI you will need to specify the trust information for any reverse security proxy servers. See “Trust association interceptor settings” on page 228 to configure TAI settings.

Configuring digest authentication and TAI for Tivoli Directory Server

You can configure digest authentication and Trust Association Interceptor (TAI) for IBM Tivoli Directory Server.

To configure digest authentication and TAI on WebSphere Application Server, you will need to:

- Install Tivoli Directory Server version 5.2
- Set up and activate Lightweight Third Party Authentication. For more information, see the Lightweight Third Party Authentication section.
- You also may want to refer to the section Configuring a custom trust association interceptor for more TAI information.

Complete the following procedure to configure digest authentication and TAI on WebSphere Application Server:

1. To set up digest authentication, verify that **Lightweight Third Party Authentication (LTPA)** is configured for use on your server by selecting **Security** → **Secure administration, applications, and infrastructure** → **Authentication mechanisms**. In the **Configuration** tab on the **Authentication mechanisms and expiration** page you should see the **Password** field already filled in.
2. In the administrative console, click **Security** → **Secure administration, applications, and infrastructure**.
 - a. Under **Authentication**, expand **Web security** and click on **Trust association**.
 - b. On the **Configuration** tab, under **General properties**, make sure the **Enable trust association** box is checked. Then click **Apply**.
3. On the **Interceptors** page of the administration console look for `com.ibm.ws.sip.security.digest.DigestTAI` in the **Interceptor class name** list:
 - a. If this class name is not present, click **New** to open the Configuration tab and enter `com.ibm.ws.sip.security.digest.DigestTAI` in the **Interceptor class name** field and click **Apply**. Then proceed to the following steps.
 - b. If this interceptor class is present, you may proceed to set up a realm in digest authentication. To do this, click **com.ibm.ws.sip.security.digest.DigestTAI** → **Custom Properties**:
 - c. Click **OK**.
4. Navigate through **Security** → **Secure administration, applications, and infrastructure** → **Authentication mechanisms and expiration** to the **Configuration** tab.
 - a. In the **Key generation** section, click **Generate Keys**. (No import or export of the key is necessary.)
 - b. Under the Cross-cell single sign-on section fill in the **Password** fields.
 - c. Fill in the **Internal server ID** field.
 - d. Click **OK**.
5. Click to **Security** → **Secure administration, applications, and infrastructure**.
 - a. If the box **Use Java 2 security to restrict application access to local resources** is checked, click to deselect it.
 - b. In the **User account repository** section of the page, select your LDAP registry from the **Available realm definitions** drop-down box.
 - c. Click **Set as current** and then click **Apply**.
6. Save all changes.
7. Restart the server.
8. Be sure you see the following message appear in the SystemOut.log after the server has restarted:
`SECJ0121I: Trust Association Init class com.ibm.ws.sip.security.digest.DigestTAI loaded successfully`

If this message does not appear in the log, digest authentication has not been activated.

Configuring digest authentication for Oracle Internet Directory

You can configure digest authentication for Oracle Internet Directory, an implementation of the Lightweight Directory Access Protocol (LDAP) that uses the Oracle database as a repository for directory entries.

To configure digest authentication for Oracle Internet Directory, you will need to:

- Install Oracle Internet Directory version 9.0.2.
- Set up and activate Lightweight Third Party Authentication. For more information, see the Lightweight Third Party Authentication section.
- You also may want to refer to the section Configuring a custom trust association interceptor for more TAI information.

Complete the following procedure to configure digest authentication for Oracle Internet Directory on WebSphere Application Server:

1. To set up digest authentication, verify that **Lightweight Third Party Authentication (LTPA)** is configured for use on your server by selecting **Security** → **Secure administration, applications, and infrastructure** → **Authentication mechanisms**. In the **Configuration** tab on the **Authentication mechanisms and expiration** page you should see the **Password** field already filled in.
2. In the administrative console, click **Security** → **Secure administration, applications, and infrastructure**.
 - a. Under **Authentication**, expand **Web security** and click on **Trust association**.
 - b. On the **Configuration** tab, under **General properties**, make sure the **Enable trust association** box is checked. Then click **Apply**.
3. On the **Interceptors** page of the administration console look for `com.ibm.ws.sip.security.digest.SIPDigestOID` in the **Interceptor class name** list:
 - a. If this class name is not present, click **New** to open the Configuration tab and enter `com.ibm.ws.sip.security.digest.SIPDigestOID` in the **Interceptor class name** field and click **Apply**. Then proceed to the following steps.
 - b. If this interceptor class is present, you may set up custom properties for it. To do this, click **com.ibm.ws.sip.security.digest.SIPDigestOID** → **Custom Properties**:
 - c. Click **OK**.
4. Navigate through **Security** → **Secure administration, applications, and infrastructure** → **Authentication mechanisms and expiration** to the **Configuration** tab.
 - a. In the **Key generation** section, click **Generate Keys**. (No import or export of the key is necessary.)
 - b. Under the Cross-cell single sign-on section fill in the **Password** fields.
 - c. Fill in the **Internal server ID** field.
 - d. Click **OK**.
5. Click to **Security** → **Secure administration, applications, and infrastructure**.
 - a. If the box **Use Java 2 security to restrict application access to local resources** is checked, click to deselect it.
 - b. In the **User account repository** section of the page, select your LDAP registry from the **Available realm definitions** drop-down box.
 - c. Click **Set as current** and then click **Apply**.
6. Save all changes.
7. Restart the server.
8. Be sure you see the following message appear in the SystemOut.log after the server has restarted:

```
SECJ0121I: Trust Association Init class  
com.ibm.ws.sip.security.digest.SIPDigestOID loaded successfully
```

If this message does not appear in the log, digest authentication has not been activated.

Configuring a custom trust association interceptor

How to configure a custom trust association interceptor (TAI).

Before you can configure a custom TAI, you must enable global security after you install the following software:

1. WebSphere Application Server version 6.1
2. Tivoli Directory Server version 5.2
3. Verify that **Lightweight Third Party Authentication (LTPA)** is configured for use on your server by selecting **Security** → **Secure administration, applications, and infrastructure** → **Authentication mechanisms**. In the **Configuration** tab on the **Authentication mechanisms and expiration** page you should see the **Password** field already filled in.

To configure a custom TAI, you may want to familiarize yourself with the general TAI information contained in the Trust Associations documentation. You also may want to refer to the Developing a custom trust association interceptor topic for information about the Java class extensions.

The JAR file that contains your custom TAI should be deployed in the application server environment in a location that is accessible by the security portions of the application server runtime. They reside in the *WASProductDir*/plugins directory for the application server nodes. You may encounter problems if you try to place your TAI under a shared library for just the application server.

To configure your custom TAI, complete the following steps (for more details, see the “TAI usage” section of IBM WebSphere Developer Technical Journal: Advanced authentication in WebSphere Application Server):

1. Install your TAI JAR file in the *WASProductDir*/plugins directory.
2. From the WebSphere Application Server administrative console, navigate to **Security** → **Secure administration, applications, and infrastructure**
3. Under **Authentication**, expand **Web security** and click on **Trust association**.
4. On the **Configuration** tab, under **General properties**, make sure the **Enable trust association** box is checked. Then click **Apply**.
5. To create the new custom class, navigate to **Trust association** → **Configuration** tab (see step 4 above) and under **Additional properties**, click **Interceptors**.
6. Click **New** and enter the fully qualified class name to your custom TAI class, and click **Apply**.
7. If your TAI depends on custom properties, navigate to **Trust association** → **Configuration** tab (see step 4 above) and under **Additional properties**, click **Interceptors**. Select your Interceptor class name list. In the **Configuration** tab that opens, click on **Custom properties** Type the name and value pairs for the properties on which your TAI depends, and click **Apply**.
8. Save your configuration, and then restart your server to make your TAI fully operational.

Chapter 12. EJB applications

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 Enterprise JavaBeans (EJB) methods need protecting and how to protect them.

Note: This procedure might not match the steps that are required when using your assembly tool, or match the version of the assembly tool that you are using. You should follow the instructions for the tool and version that you are using.

To secure enterprise bean applications, follow these steps:

1. In an assembly tool, import your Enterprise JavaBean (EJB) Java Archive (JAR) file or an application archive (EAR) file that contains one or more Web modules.
See the information about importing an EJB JAR file or importing an enterprise application EAR file in the Application Server Toolkit (AST) documentation.
2. In the Project Explorer, click **EJB Projects** directory and click the name of your application.
3. Right-click the deployment descriptor and click **Open with > Deployment Descriptor Editor**. If you selected an enterprise bean .jar file, an EJB deployment descriptor editor opens. If you select 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 display 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, click the **Assembly** tab. Under Security Roles, click **Add**. In the Add Security Role wizard, name and 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 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, an unselected option specifies that the selected methods from the selected beans do not require authorization to run. 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 is opened.
 - 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 clear 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 the methods that are associated with the enterprise beans. You can click **Apply to All** or **Deselect All** to quickly select or clear multiple methods. The selection affects the default (*) method for each bean only. Creating a method permission for the exact method signature overrides the default (*) method permission setting. The default (*) method represents all the methods within the bean. There are default (*) methods 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 the methods that are 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 that is not excluded, is cleared 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 is opened.
 - 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()) role and contains the name of the role that is referenced in the code. Because you create security roles during the assembly stage, the developer uses a *logical role name* 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 is opened.
 - 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 that is used during development to the role (role-link) using the previous steps.
8. Specify the RunAs identity for enterprise bean 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 identity 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 is opened.
 - 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 is 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 “Deploying secured applications” on page 913 article to complete the task of securing the EJB application.

Configuring security for EJB 2.1 message-driven beans

Use this task to configure resource security and security permissions for Enterprise JavaBeans (EJB) Version 2.1 message-driven beans.

The association between connection factories, destinations, and message-driven beans is provided by listener ports. A listener port allows a deployed message-driven bean associated with the port to retrieve messages from the associated destination. You create listener ports by specifying their administrative name, the connection factory JNDI name, and the destination name (other optional properties are also configurable). Listener ports provide simplified administration of the associations between connection factories, destinations and message-driven beans, and are managed by a listener manager. The listener manager is provided by the message listener service to control and monitor the JMS listeners that are monitoring JMS destinations on behalf of deployed message-driven beans. For more information about listener ports, see [Message-driven beans - listener port components](#)

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 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.

Chapter 13. Client applications

Accessing secure resources using SSL and applet clients

By default, the applet client is configured to have security enabled. If you have administrative 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` and `ssl.client.props` files.

1. Make a copy of the following files so that you can use them for an applet:
 - `<app_client_root>\properties\sas.client.props`
 - `<app_client_root>\properties\ssl.client.props`
2. Edit the copies of the `sas.client.props` and `ssl.client.props` files 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 files you created in step 1, modify the following values:
 - `-Dcom.ibm.CORBA.ConfigURL=file:<app_client_root>\properties\sas.client.props`
 - `-Dcom.ibm.SSL.ConfigURL=file:<app_client_root>\properties\ssl.client.props`

For more information on the `sas.client.props` and `ssl.client.props` files 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 `<app_server_root>/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";
```

Chapter 14. Web services

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 several ways that you can configure the HTTP outbound transport level security for a Web service acting as a client to another Web service server. You can also configure the HTTP outbound transport level security with an assembly tool or by using the Java properties. If you do not configure the HTTP outbound transport level security, the Web services runtime defers to the Java 2 Platform, Enterprise Edition (J2EE) security runtime in the WebSphere product for an effective Secure Sockets Layer (SSL) configuration. If there is no SSL configuration with the J2EE security runtime in the WebSphere product, the Java Secure Socket Extension (JSSE) system properties are used.

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 the standard Java properties for JSSE, the properties are configured as system properties. The configuration specified in the binding takes precedence over the Java properties. However, the configurations that are specified by the J2EE security programming model, or that are associated the Dynamic selection, have higher precedence.

Review the topic Secure communications using Secure Sockets Layer for more information.

Configure the HTTP outbound transport level security with the following steps provided in this task section.

1. Open the administrative console.
2. Click **Applications > Enterprise Applications > *application_instance* > Manage Modules > *module_instance* > Web Services Client Security Bindings.**
3. Click **HTTP SSL Configuration** to access the HTTP SSL configuration panel. Select the **Centrally-managed** radio button so that the system runtime chooses the SSL configuration that is based on the current context. Select the **Specific to this Web service port** radio button if you want to choose the SSL configuration in the HTTP SSL configuration drop down box.

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_instance*.**
2. Click **Manage modules > *URI_file_name* > Web Services: Client Security Bindings.**

3. Under HTTP SSL Configuration, click **Edit**.

SSL configuration

Select the **Centrally-managed** radio button so that the system runtime chooses the SSL configuration that is based on the current context. Select the **Specific to this Web service port** radio button if you want to choose the SSL configuration in the **HTTP SSL configuration** drop down box.

HTTP SSL configuration

The **HTTP SSL configuration** drop down box lists the SSL configurations used with the HTTP transport for a port. Use this drop down box if you want to select the SSL configuration rather than using the SSL configuration that the runtime automatically selects. To use the drop down box, select the **Specific to the Web service port** radio button that is located in the **SSL configuration** field. After you select the radio button, you can click the drop down box to view and select an SSL configuration.

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.

This task is one of several ways that you can configure the HTTP outbound transport level security for a Web Service acting as a client to another Web service server. You can also configure the HTTP outbound transport level security with the administrative console or by using the Java properties. If you do not configure the HTTP outbound transport level security, the Web services runtime defers to the Java 2 Platform, Enterprise Edition (J2EE) security runtime in the WebSphere product for an effective Secure Sockets Layer (SSL) configuration. If there is no SSL configuration with the J2EE security runtime in the WebSphere product, the Java Secure Socket Extension (JSSE) system properties are used.

If you 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 standard Java properties for JSSE, the properties are configured as system properties. The configuration that is specified in the binding takes precedence over the Java properties. However, the configurations that are specified by the J2EE security programming model, or are associated with the Dynamic selection, have a higher precedence.

Review the topic [Secure communications using Secure Sockets Layer](#) for more information.

Configure the HTTP outbound transport level security with the following steps provided in this task section.

1. Start an assembly tool. See ["Starting WebSphere Application Server Toolkit"](#) in the Application Server Toolkit documentation for more information.
2. If you have not done so already, configure the assembly tool to work on J2EE modules. Make sure that the **J2EE** and **Web** categories are enabled. See ["Configuring WebSphere Application Server Toolkit"](#) in the Application Server Toolkit documentation for more information.
3. Migrate the Web archive (WAR) files that are created with the Assembly Toolkit, Application Assembly Tool (AAT) or a different tool to an AST or Rational Web Developer assembly tool. To migrate files, import your WAR files to the assembly tool. See ["Importing Web archive \(WAR\) files"](#) in the Application Server Toolkit documentation for more information.
4. Configure the HTTP outbound transport level security. See ["Enabling Web service endpoints"](#) in the Application Server Toolkit documentation for more information.

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 “Secure transports with JSSE and JCE programming interfaces” on page 593 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.

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. However, transport-level security functionality is independent from functionality that is 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.

Web services applications can also use Federal Information Processing Standard (FIPS) approved ciphers for more secure TLS connections.

WebSphere Application Server uses the Java Secure Sockets Extension (JSSE) package to support SSL and TLS.

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.

Open the administrative console.

1. Click **Applications > Enterprise Applications > *application_instance* > Manage Modules > *module_instance* > Web services: Client security bindings.**

2. Click **HTTP Basic Authentication** to access the HTTP basic authentication panel. Enter the values in the HTTP Basic Authentication panel.

You have configured the HTTP basic authentication.

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_instance***.
2. Click **Manage modules > *URI_file_name* > Web Services: Client Security Bindings**.
3. Under HTTP basic authentication, click **Edit**.

Basic authentication ID

The user name for the HTTP basic authentication for this port is set in this field.

Basic authentication password

The password for the HTTP basic authentication for this port is set in this field.

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.

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. See "Starting WebSphere Application Server Toolkit" in the Application Server Toolkit documentation for more information.

2. If you have not done so already, configure the assembly tool to work on J2EE modules. Make sure that the **J2EE** and **Web** categories are enabled. See "Configuring WebSphere Application Server Toolkit" in the Application Server Toolkit documentation for more information.
3. Migrate the Web archive (WAR) files that are created with the Assembly Toolkit, Application Assembly Tool (AAT) or a different tool to an AST or Rational Web Developer assembly tool. To migrate files, import your WAR files to the assembly tool. See "Importing Web archive (WAR) files" in the Application Server Toolkit documentation for more information.
4. 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.

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
```

Configuring additional HTTP transport properties using the JVM custom property panel in 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 > Java and Process Management > Process Definition > Java Virtual Machine > Custom Properties**.
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.

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:

- Configuring additional HTTP transport properties using 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. 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. Start an assembly tool. See "Starting WebSphere Application Server Toolkit" in the Application Server Toolkit documentation for more information.
3. If you have not done so already, configure the assembly tool to work on J2EE modules. Make sure that the **J2EE** and **Web** categories are enabled. See "Configuring WebSphere Application Server Toolkit" in the Application Server Toolkit documentation for more information.
4. Migrate the Web archive (WAR) files that are created with the Assembly Toolkit, Application Assembly Tool (AAT) or a different tool to an AST or Rational Web Developer assembly tool. To migrate files, import your WAR files to the assembly tool. See "Importing Web archive (WAR) files" in the Application Server Toolkit documentation for more information.
5. Configure the additional HTTP transport properties. 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 the wsadmin command-line tool

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
- Configuring additional HTTP transport properties using the JVM custom property panel in 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 AddressBook http.proxyHost+http.proxyPort myhost+80]]]' )
AdminConfig.save()
```

The following illustrates an example of the Jacly 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.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: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 and later 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 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 inter-operate 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 Security Assertion Markup Language (SAML) profile, the Kerberos profile, the eXtensible rights Markup Language (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.

Compatibility between WebSphere Application Server Version 6.1 and Version 5.x

In WebSphere Application Server Version 6.1, you can run a Version 5.x Web services-secured application on a Version 6.1 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 6.1 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 6.1. This issue occurs because the SOAP message format is different between a Version 5.x application and a Version 6 or later 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. You can configure Web services security on the application level, server level, and the cell level. The following table shows an example of the relationships between each of the configurations that apply to just the application, to an entire server, or to the entire cell. 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. Within the table, the configurations in the Referenced configurations column are referenced by the configuration listed in the Configuration name column. For example, the token generator on the application-level for the request generator references the collection certificate store, the nonce, time stamp, and callback handler configurations.

Table 36. The relationship between the configurations.

Configuration level	Configuration name	Referenced configurations
Application-level request generator	Token generator	<ul style="list-style-type: none"> • Collection certificate store • Nonce • Timestamp • Callback handler
Application-level request generator	Key information	<ul style="list-style-type: none"> • Key locator • Key name • Token
Application-level request generator	Signing information	<ul style="list-style-type: none"> • Key information
Application-level request generator	Encryption information	<ul style="list-style-type: none"> • Key information
Application-level request consumer	Token consumer	<ul style="list-style-type: none"> • Trust anchor • Collection certificate store • Trusted ID evaluators • Java Authentication and Authorization Service (JAAS) configuration
Application-level request consumer	Key information	<ul style="list-style-type: none"> • Key locator • Token
Application-level request consumer	Signing information	<ul style="list-style-type: none"> • Key information
Application-level request consumer	Encryption information	<ul style="list-style-type: none"> • Key information
Application-level response generator	Token generator	<ul style="list-style-type: none"> • Collection certificate store • Callback handler
Application-level response generator	Key information	<ul style="list-style-type: none"> • Key locator • Token
Application-level response generator	Signing information	<ul style="list-style-type: none"> • Key information
Application-level response generator	Encryption information	<ul style="list-style-type: none"> • Key information
Application-level response consumer	Token consumer	<ul style="list-style-type: none"> • Trust anchor • Collection certificate store • JAAS configuration
Application-level response consumer	Key information	<ul style="list-style-type: none"> • Key locator • Key name • Token
Application-level response consumer	Signing information	<ul style="list-style-type: none"> • Key information
Application-level response consumer	Encryption information	<ul style="list-style-type: none"> • Key information
Server-level default generator bindings	Token generator	<ul style="list-style-type: none"> • Collection certificate store • Callback handler
Server-level default generator bindings	Key information	<ul style="list-style-type: none"> • Key locator • Token
Server-level default generator bindings	Signing information	<ul style="list-style-type: none"> • Key information

Table 36. The relationship between the configurations. (continued)

Configuration level	Configuration name	Referenced configurations
Server-level default generator bindings	Encryption information	<ul style="list-style-type: none"> Key information
Server-level default consumer bindings	Token consumer	<ul style="list-style-type: none"> Trust anchor Collection certificate store Trusted ID evaluator JAAS configuration
Server-level default consumer bindings	Key information	<ul style="list-style-type: none"> Key locator Token
Server-level default consumer bindings	Signing information	<ul style="list-style-type: none"> Key information
Server-level default consumer bindings	Encryption information	<ul style="list-style-type: none"> Key information
Cell-level default generator bindings	Token generator	<ul style="list-style-type: none"> Collection certificate store Callback handler
Cell-level default generator bindings	Key information	<ul style="list-style-type: none"> Key locator Token

When 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. Configuration information for the application-level precedes similar configuration information on the server-level.

Because of the relationship between the different Web services security configurations, it is recommended that you specify the configurations on each level of the configuration in following order. You can choose to configure Web services security for the application level or the server level as it depends upon your environment and security needs.

- Assemble your Web services security-enabled application using an assembly tool. Prior to modifying a 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*. 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 989. Return to this article after you have assembled your application and imported it into the administrative console.
- Modify the application-level configurations in the administrative console.
 - 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 1083.
 - 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 1087.
 - Configure the token for the generator binding. For more information, see “Configuring the token generator on the application level” on page 1098.
 - 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 1115.
 - 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 1121.

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 1132.
 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 1146.
 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 1157.
 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 1159.
 10. Configure the token for the consumer binding. For more information, see “Configuring token consumer on the application level” on page 1161.
 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 1170.
 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 1171.
 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 1173.
 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 1178.
- 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 1184
 2. Configure the collection certificate store for the server level. For more information, see “Configuring the collection certificate store for the server or cell-level bindings” on page 1185
 3. Configure a token generator. For more information, see “Configuring token generators on the server or cell level” on page 1188.
 4. Configure a nonce for the server level. For more information, see “Configuring a nonce on the server or cell level” on page 1187.
 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 1198.
 6. Configure the key information for the generator binding. For more information, see “Configuring the key information for the generator binding on the server or cell level” on page 1200.
 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 1202.
 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 1205.
 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 1206.
 10. Configure a token consumer. For more information, see “Configuring token consumers on the server or cell level” on page 1210.
 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 1218.
 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 1219.
 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 1221.

After completing these steps on the appropriate level of WebSphere Application Server, you have secured Web services.

What is new for securing Web services

In WebSphere Application Server Version 6.0.x, and later 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, which is a working draft. See Basic Security Profile for more information. 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 6.0.x and later support 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
- New in Version 6.1: Basic Security Profile (WS-I BSP)

For details on what parts of the previous specifications are supported in WebSphere Application Server Version 6.0.x and later, see “Supported functionality from OASIS specifications” on page 949.

High level features overview in WebSphere Application Server Version 6.0.x and later

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 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 6.0.x and later, 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 and later include 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.
- Support for the acceleration of hardware cryptographic devices.
- Support for secure keys.
- Support for the Basic Security Profile (WS-I BSP).

For more information on some of these enhancements, see “Web services security enhancements” on page 954.

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 6.0.x

You can use Rational Application Developer Version 6.0.x 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 Enterprise JavaBeans™ (EJB) modules.

Rational Web Developer Version 6.0.x

You can use Rational Web Developer Version 6.0.x to develop Web services and configure the deployment descriptor and the binding file for Web services security. However, you cannot assemble Enterprise JavaBeans 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 6.0.x and later, 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 that are 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 6.0.x and later 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 Version 6 and later. 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 6.0.x and later 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
- Re-configure 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 6.0.x and later format using the Rational Web Developer and Application Server Toolkit. You must migrate the configuration manually.

Important: The Web services security support in WebSphere Application Server Version 6.0 is based in part on the OASIS specification titled *Web Services Security: X.509 Token Profile 1.0* plus the first errata (*Errata 1.0*).

In the first errata, the URIs for the X.509 token type and the X.509 Subject Key Identifier value type were modified. WebSphere Application Server Version 6.0 was based on these modified URIs. After WebSphere Application Server Version 6.0 shipped, the OASIS Technical Committee reversed those changes, reverting back to the original 1.0 profile URIs.

There could be interoperability problems between WebSphere Application Server Version 6.0 and other vendor's Web services products that are based on the current version of the profile. WebSphere Application Server was fixed in Versions 6.0.2 and 6.0.1.2 to comply with the latest version of the profile. If WebSphere Application Server Version 6.0 is used in a heterogeneous environment with other vendor's Web services products, it is recommended that the server be upgraded to Version 6.0.1.2, 6.0.2, or later, or to install a service fix that includes APAR PK03507.

FIPS support in WebSphere Application Server

In WebSphere Application Server, Federal Information Processing Standard (FIPS) compliant algorithms for key encryption, data encryption, signature and digest are supported. To enable this mode, select **Use the Federal Information Processing Standard (FIPS)** on the Global security panel of the WebSphere administrative console.

After this option has been selected, and the WebSphere Application Server has been restarted, the lists of available algorithms that are displayed in the Web services security binding configuration panels of the administrative console are then FIPS compliant algorithms.

If a previously deployed application was configured to use a noncompliant algorithm, that application no longer starts after the FIPS mode has been enabled in WebSphere Application Server. The error message Unauthorized data encryption method appears in the case of a noncompliant data encryption algorithm. Similar errors are displayed for unauthorized key encryption, digest and signature methods.

FIPS support in WebSphere Application Server

In WebSphere Application Server, Federal Information Processing Standard (FIPS) compliant algorithms for key encryption, data encryption, signature and digest are supported. To enable this mode, select **Use the United States Federal Information Processing Standard (FIPS) algorithms** on the SSL certificate and key management panel of the WebSphere administrative console.

After this option has been selected, and the WebSphere Application Server has been restarted, the lists of available algorithms that are displayed in the Web services security binding configuration panels of the administrative console are then FIPS compliant algorithms.

If a previously deployed application was configured to use a noncompliant algorithm, that application no longer starts after the FIPS mode has been enabled in WebSphere Application Server. The error message Unauthorized data encryption method appears in the case of a noncompliant data encryption algorithm. Similar errors are displayed for unauthorized key encryption, digest and signature methods.

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 Versions 6.0.x and later:

- Application programming interfaces (API) do not exist for Web services security in WebSphere Application Server Versions 6.0.x and later. 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 Version 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 6.0.x and later, see “Supported functionality from OASIS specifications” on page 949.

Web services security specification for Version 6 and later- 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 and later.

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 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.

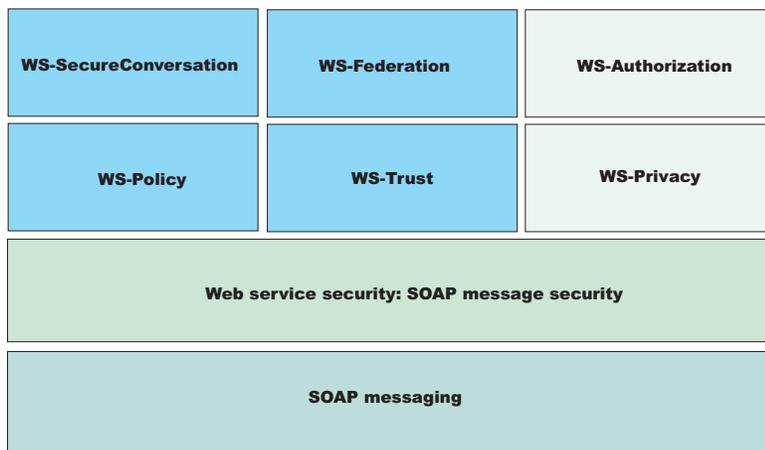
WS-Federation

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. However, the WS-Authorization specification has not been published. The existing implementation of Web services security is based upon the Web Services for Java 2 Platform, Enterprise Edition (J2EE) or Java Specification Requirements (JSR) 109 specification. The implementation of Web services security leverages the J2EE role-based authorization checks. For conceptual information on role-based authorization, see “Role-based authorization” on page 324. If you develop a Web service that requires method-level authorization checks, then you must use stateless session beans to implement your Web service. For more information on using stateless session beans to implement a Web service, see *Developing a Web service from an enterprise bean* and “Securing enterprise bean applications” on page 919. If you develop a Web service that is implemented as a servlet, you can use coarse-grained or URL-based authorization in the Web container. However, in this situation, you cannot use the identity from Web services security for authorization checks. Instead, you can use the identity from the transport. If you use SOAP over HTTP, then the identity is in the HTTP transport.

This following figure shows the relationship between these specifications:



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:

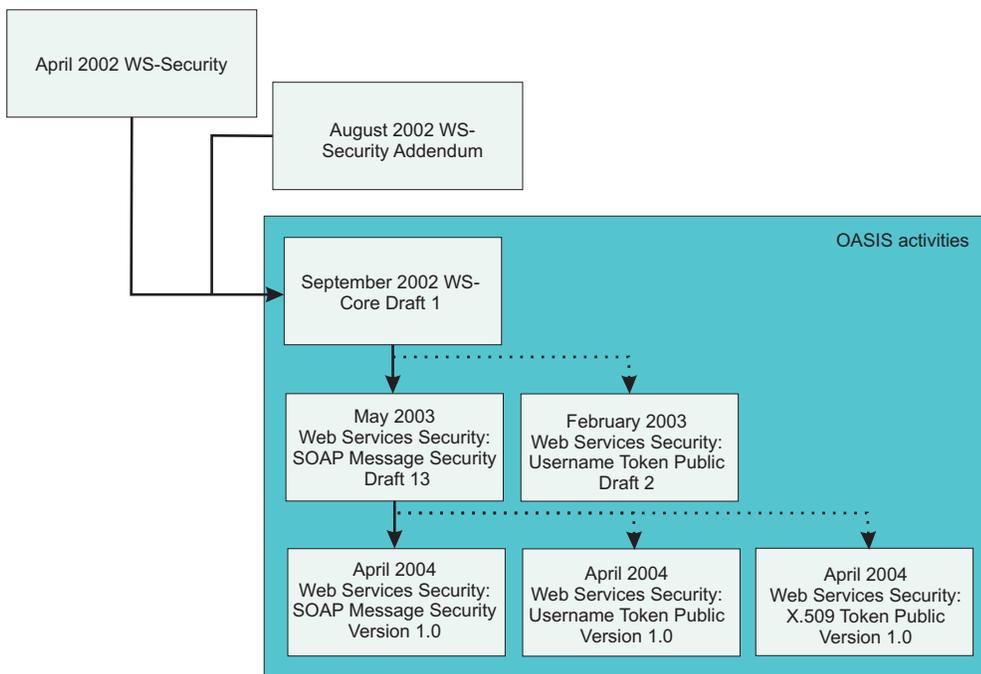
- Web Services Security: SOAP Message Security Draft 13 (formerly Web Services Security Core Specification)
- Web Services Security: Username Token Profile Draft 2

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 and later support 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 and later also extend and provide 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 979.

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 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 Version 6 and later 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 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 and later

OASIS is working on various profiles. For more information, see Organization for the Advancement of Structured Information Standards Committees. WebSphere Application Server Versions 6 and later do 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 and later support 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 and later. 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 and later support both the OASIS Web services security draft 13 and the OASIS Web services security 1.0 specification. But in WebSphere Application Server Version 6 and later, 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 and later. 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 6.1 applications” on page 966.

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.

Specifically, see Basic Security Profile Version 1.0 for details about the BSP. WebSphere Application Server supports compliance with the BSP. See “Basic Security Profile compliance” on page 988 for the details to configure your application in compliance with the BSP.

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 Versions 6 and later, you can plug in your own implementation. By 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="elliottw1"
      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](#).

Supported functionality from OASIS specifications

WebSphere Application Server Version 6 and later support 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
- Basic Security Profile Version 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 Versions 6 and later.

Supported topic	Specific aspect that is supported
Security header	<ul style="list-style-type: none">• @S11 :actor (for an intermediary)• @S11:mustUnderstand
Security tokens	<ul style="list-style-type: none">• Username token (user name and password)• Binary security token (X.509 and Lightweight Third Party Authentication (LTPA))• Custom token<ul style="list-style-type: none">– Other binary security token– XML token<p>Note: WebSphere Application Server does not provide an implementation, but you can use an XML token with plug-in point.</p>
Token references	<ul style="list-style-type: none">• Direct reference• Key identifier• Key name• Embedded reference

Supported topic	Specific aspect that is supported
Signature algorithms	<ul style="list-style-type: none"> • Digest <ul style="list-style-type: none"> SHA1 http://www.w3.org/2000/09/xmlsig#sha1 SHA256 http://www.w3.org/2001/04/xmlenc#sha256 SHA512 http://www.w3.org/2001/04/xmlenc#sha512 • MAC <ul style="list-style-type: none"> HMAC-SHA1 http://www.w3.org/2000/09/xmlsig#hmac-sha1 • Signature <ul style="list-style-type: none"> DSA with SHA1 http://www.w3.org/2000/09/xmlsig#dsa-sha1 Do not use this algorithm if you want your configured application to be in compliance with the Basic Security Profile (BSP) RSA with SHA1 http://www.w3.org/2000/09/xmlsig#rsa-sha1 • Canonicalization <ul style="list-style-type: none"> Canonical XML (with comments) http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments Canonical XML (without comments) http://www.w3.org/TR/2001/REC-xml-c14n-20010315 Exclusive XML canonicalization (with comments) http://www.w3.org/2001/10/xml-exc-c14n#WithComments Exclusive XML canonicalization (without comments) http://www.w3.org/2001/10/xml-exc-c14n# • Transform <ul style="list-style-type: none"> STR transform http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soapmessage-security-1.0#STR-Transform XPath http://www.w3.org/TR/1999/REC-xpath-19991116 Do not use the original XPATH transform if you want your configured application to be in compliance with the Basic Security Profile (BSP). Note: When referring to an element in a SECURE_ENVELOPE that does not carry an attribute of type ID from a ds:Reference in a SIGNATURE, you must use the XPath Filter 2.0 Transform, http://www.w3.org/2002/06/xmlsig-filter2 Enveloped signature http://www.w3.org/2000/09/xmlsig#enveloped-signature XPath Filter2 http://www.w3.org/2002/06/xmlsig-filter2 Note: When referring to an element in a SECURE_ENVELOPE that does not carry an ID attribute type from a ds:Reference in a SIGNATURE, you must use the XPath Filter 2.0 Transform, http://www.w3.org/2002/06/xmlsig-filter2 Decryption transform http://www.w3.org/2002/07/decrypt#XML

Supported topic	Specific aspect that is supported
Signature signed parts	<ul style="list-style-type: none"> • WebSphere Application Server key words: <ul style="list-style-type: none"> – body, which signs the SOAP message body – timestamp, which signs all of the time stamps – securitytoken, which signs all of the security tokens – dsigkey, which signs the signing key – enckey, which signs the encryption key – messageid, which signs the wsa :MessageID element in WS-Addressing. – to, which signs the wsa:To element in WS-Addressing – action, which signs the wsa:Action element in WS-Addressing – relatesto, which signs the wsa:RelatesTo element in WS-Addressing wsa is the namespace prefix of http://schemas.xmlsoap.org/ws/2004/08/addressing – wscontext, which specifies the WS-Context header for the SOAP header. For more information, see Propagating work area context over Web services. – wsafrom, which specifies the <wsa:From> WS-Addressing From element in the SOAP header. – wsareplyto, which specifies the <wsa:ReplyTo> WS-Addressing ReplyTo element in the SOAP header. – wsafaultto, which specifies the <wsa:FaultTo> WS-Addressing FaultTo element in the SOAP header. – wsaall, which specifies all of the WS-Addressing elements in the SOAP header. <ul style="list-style-type: none"> • XPath expression to select an XML element in a SOAP message. For more information, see http://www.w3.org/TR/1999/REC-xpath-19991116.

Supported topic	Specific aspect that is supported
Encryption algorithms	<ul style="list-style-type: none"> • Data encryption <ul style="list-style-type: none"> – Triple DES in CBC: http://www.w3.org/2001/04/xmlenc#tripleDES-cbc – AES128 in CBC: http://www.w3.org/2001/04/xmlenc#aes128-cbc – AES192 in CBC: http://www.w3.org/2001/04/xmlenc#aes192-cbc 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 1149. Do not use the 192-bit data encryption algorithm if you want your configured application to be in compliance with the Basic Security Profile (BSP). – AES256 in CBC: http://www.w3.org/2001/04/xmlenc#aes256-cbc 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 1149. • Key encryption <ul style="list-style-type: none"> – Key transport (public key cryptography) <ul style="list-style-type: none"> - http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p. Note: <ul style="list-style-type: none"> • When running with Software Development Kit (SDK) Version 1.4, the list of supported key transport algorithms does not include this one. This algorithm appears in the list of supported key transport algorithms when running with SDK Version 1.5. • Use of the Federal Information Processing Standard (FIPS)-compliant Java cryptography engine does not support this transport algorithm. - RSA Version 1.5: http://www.w3.org/2001/04/xmlenc#rsa-1_5 – Symmetric key wrap (private key cryptography) <ul style="list-style-type: none"> - Triple DES key wrap: http://www.w3.org/2001/04/xmlenc#kw-tripleDES - AES key wrap (aes128): http://www.w3.org/2001/04/xmlenc#kw-aes128 - AES key wrap (aes192): http://www.w3.org/2001/04/xmlenc#kw-aes192 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 1149. Do not use the 192-bit data encryption algorithm if you want your configured application to be in compliance with the Basic Security Profile (BSP). - AES key wrap (aes256): http://www.w3.org/2001/04/xmlenc#kw-aes256 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 1149. • Manifests-xenc is the namespace prefix of http://www.w3.org/TR/xmlenc-core <ul style="list-style-type: none"> – xenc:ReferenceList – xenc:EncryptedKey <p>Advanced Encryption Standard (AES) is designed to provide stronger and better performance for symmetric key encryption over Triple-DES (data encryption standard). Therefore, it is recommended that you use AES, if possible, for symmetric key encryption.</p>

Supported topic	Specific aspect that is supported
Encryption message parts	<ul style="list-style-type: none"> • WebSphere Application Server keywords <ul style="list-style-type: none"> – bodycontent, which is used to encrypt the SOAP body content – usernametoken, which is used to encrypt the username token – digestvalue, which is used to encrypt the digest value of the digital signature – signature, which is used to encrypt the entire digital signature – wscontextcontent, which encrypts the content in the WS-Context header for the SOAP header. For more information, see Propagating work area context over Web services. • XPath expression to select the XML element in the SOAP message <ul style="list-style-type: none"> – XML elements – XML element contents
Time stamp	<ul style="list-style-type: none"> • Within Web services security header • 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 Versions 6 and later.

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 Versions 6 and later.

Supported topic	Specific aspect that is supported
Token types	<ul style="list-style-type: none"> • X.509 Version 3: Single certificate http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3 • X.509 Version 3: X509PKIPathv1 without certificate revocation lists (CRL) http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509PKIPathv1 • 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	<ul style="list-style-type: none"> • Key identifier – subject key identifier • Direct reference • Custom reference – issuer name and serial number

Functionality that is not supported

The following list shows the functionality that is supported in the Organization for the Advancement of Structured Information (OASIS) specifications, OASIS drafts, and other recommendations, but is not supported by WebSphere Application Server Versions 6 and later:

- 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
- Security Assertion Markup Language (SAML) token profile, WS-SecurityKerberos token profile, and XrML token 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 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.

Web services security enhancements

WebSphere Application Server Version 6 include 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](#). 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 and later support 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)

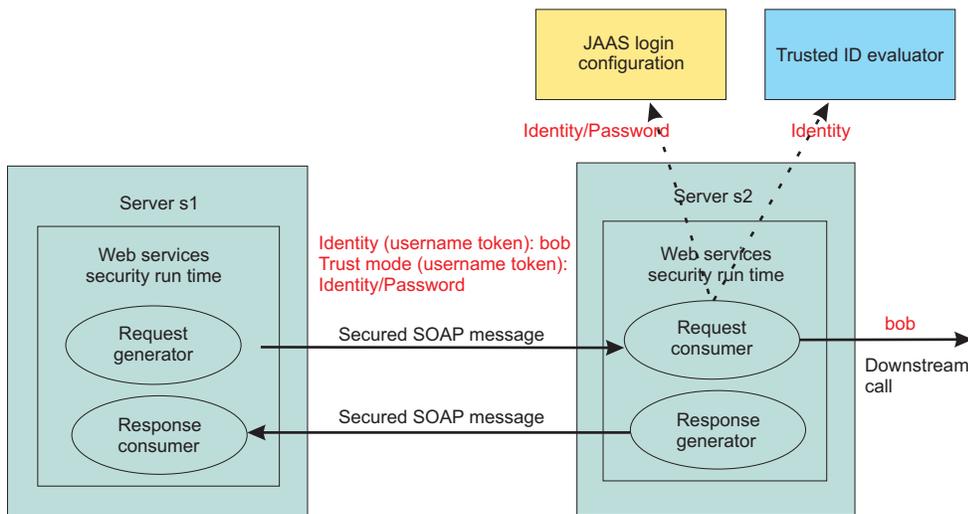
The upstream server sends a username token with a user name and password to a downstream server. The consumer or receiver of the message authenticates the username token and validates

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 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.

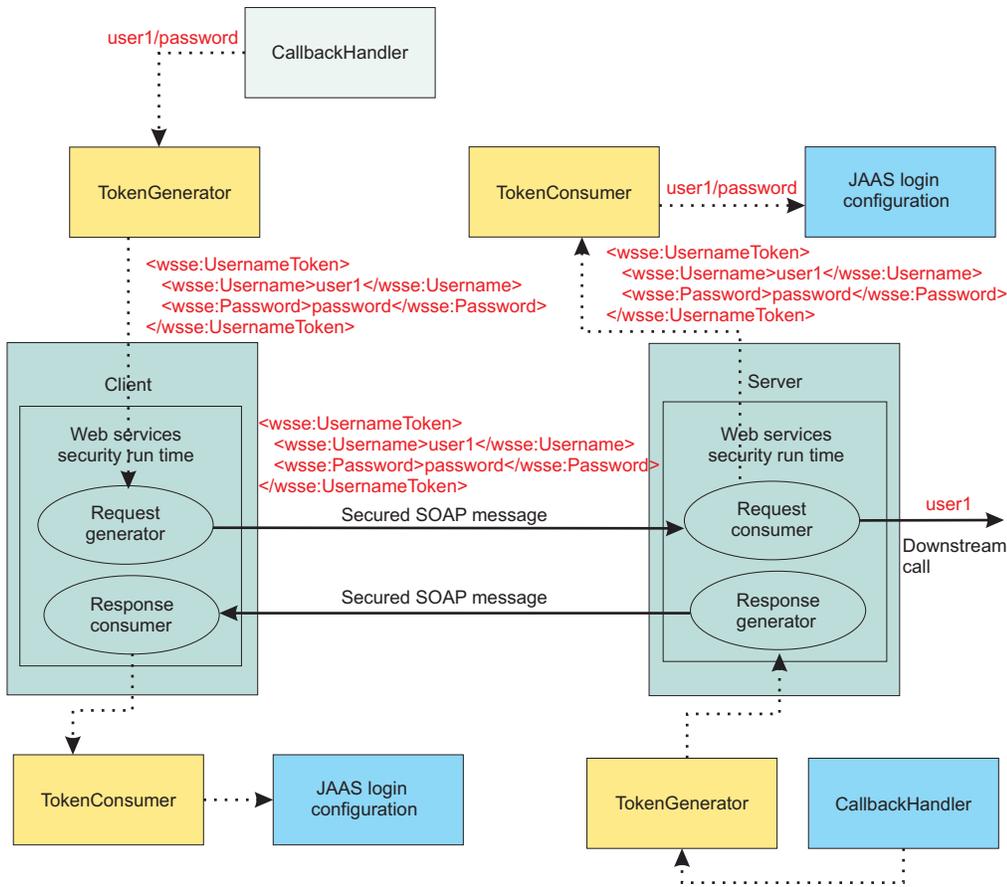


In this figure, server s1 is the upstream server and identity assertion is set up between server s1 and server s2. The s1 server authenticates the identity called *bob*. Server s1 wants to send *bob* to the s2 server with a password. The trust mode is an s1 credential that contains the identity and a 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 and later, 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 Versions 6 and later, 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:

- Unmarshalling the token from the XML format within the SOAP message. In this case, *unmarshalling* 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 or later. You must migrate the source code for the Version 5.x key locator implementation to the key locator programming model for Version 6 and later.

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 Versions 6 and later. 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 Versions 6 and later. 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 Versions 6 and later 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 creates 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 1234. 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 and later cache 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 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 and later 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 and later do not support X509PKIPathv1 CRL encoding in a X.509 token.

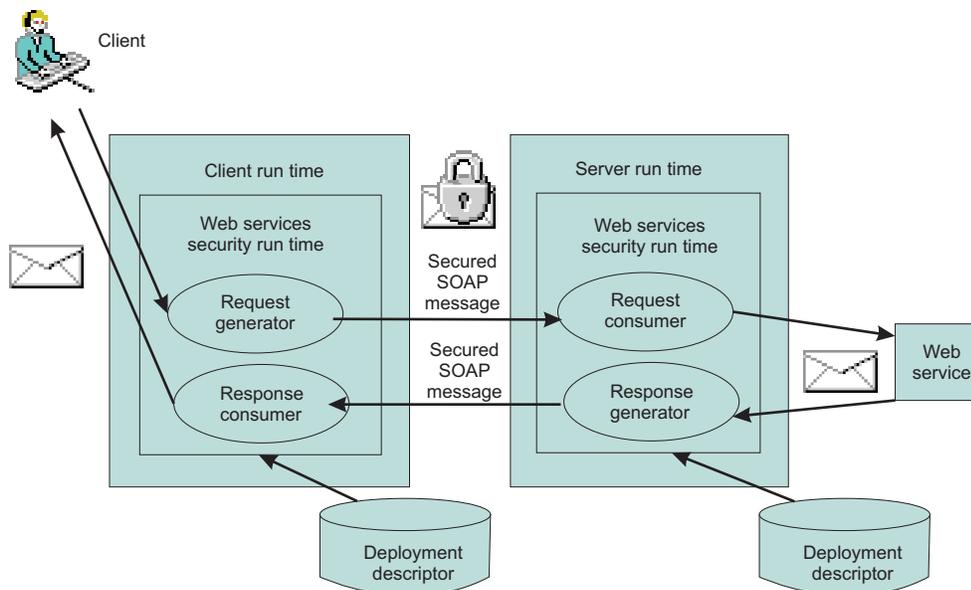
Important: The PKCS#7 encoding was tested with the IBM certificate path (IBM CertPath) provider only. The encoding is not supported for other certificate path providers.

High-level architecture for 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.

WebSphere Application Server Version 6 and later use the Java 2 Platform, Enterprise Edition (J2EE) Version 1.4 Web services deployment model to implement Web services security. 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 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 Versions 6 and later do 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 and later.

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 6.0.x

Use this tool to edit the Web services security deployment descriptor and binding. You can use this tool to assemble both Web and Enterprise JavaBeans (EJB) modules.

Rational Web Developer Version 6.0.x

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 Administrative Console Version 6 and later

Use this tool to edit the Web services security binding of a deployed application.

Overview of platform configuration and default bindings

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.

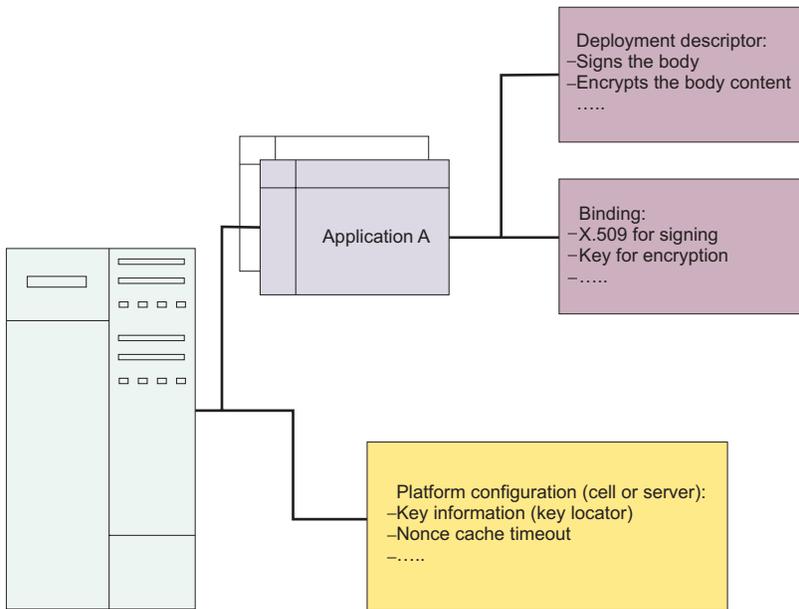
The following table provides the names of the deployment descriptor and binding files for the client and the server.

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 Versions 6 and later have a server level Web Security Services configuration. These configurations are global for all applications. Because WebSphere Application Server Version 6 and later support 5.x applications, some of the configurations are valid for Version 5.x applications only and some are valid for Version 6 and later applications only.

The following figure represents the relationship of the application deployment descriptor and binding files to the cell or server level configuration.



WebSphere Application Server

Platform configuration

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 6.0.x.

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 Versions 6 and later 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 and later 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.

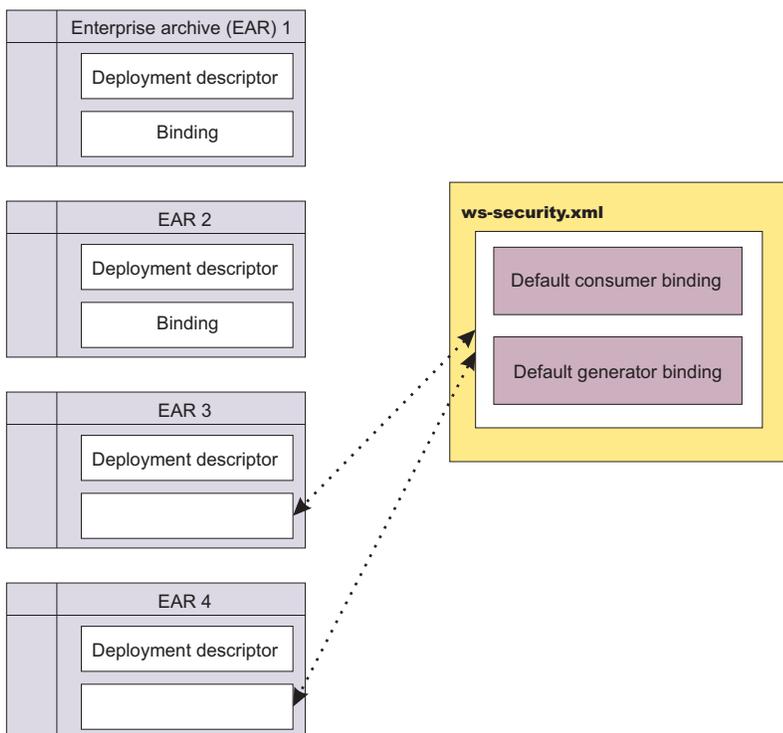
Login mappings

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

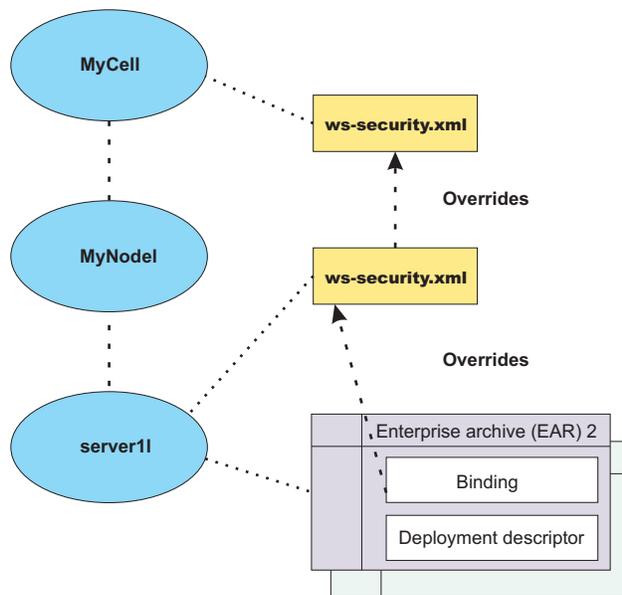
There is only one set of default bindings and they can be shared by multiple applications. This feature is available for WebSphere Application Server Version 6 and later applications only.

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, it must be referenced to use the default bindings 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.



Security model mixture

There can be multiple protocols and channels in the WebSphere Application Server Version 6 and later programming environments. Each of these applications serve different business needs.

For example, you might access:

- A Web-based application through the HTTP transport such as a servlet, JavaServer Pages (JSP) file, HTML and so on.
- An enterprise application through the Remote Method Invocation (RMI) over the Internet Inter-ORB (RMI/IIOP) protocol.
- A Web service application through the SOAP over HTTP, SOAP over the Java Message Service (JMS), or SOAP over the RMI/IIOP protocol.

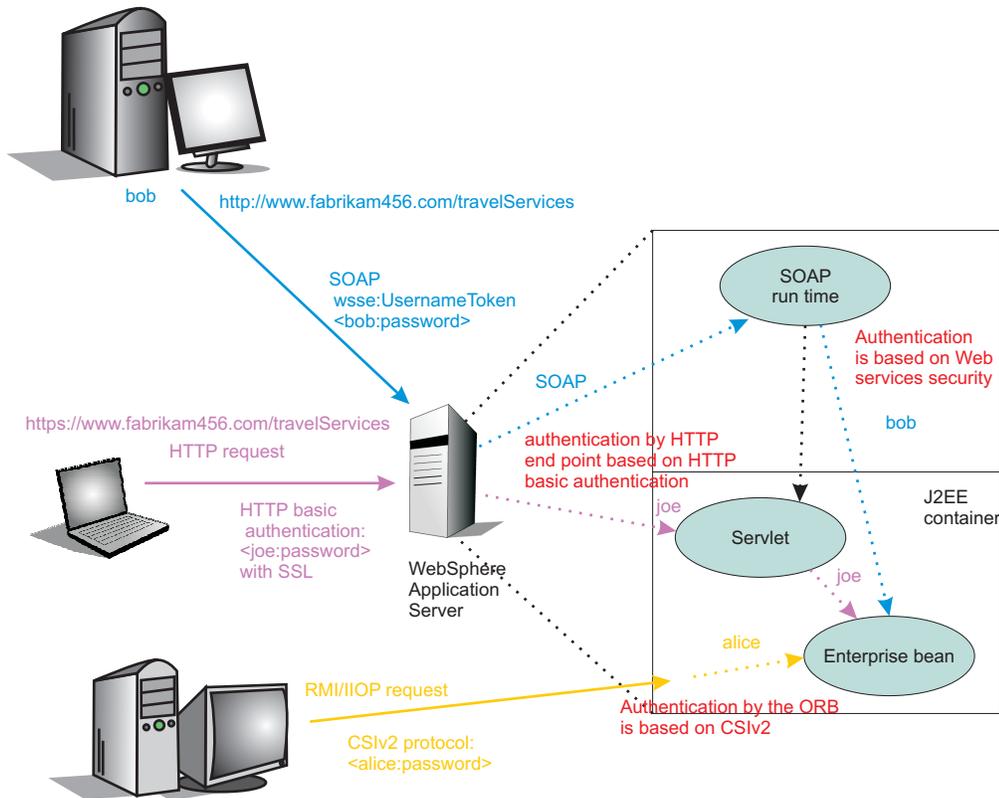
More importantly, Web services are often implemented as servlets with a Enterprise JavaBeans (EJB) file. Therefore, you can mix and match the Web services security model with the Java 2 Platform, Enterprise Edition (J2EE) security model for Web and EJB components. It is intended that Web service security compliment the J2EE role-based security and the security run time for WebSphere Application Server Version 6 and later.

Web services security also can take advantage of the security features in J2EE and the security run time for WebSphere Application Server Version 6 and later. 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 JavaBeans 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

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.

In WebSphere Application Server Version 6 and later, when you enable integrity, confidentiality, and the associated tokens within a SOAP message, security is not guaranteed. 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 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 and later, 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). See Basic Security Profile Version 1.0 for more information.

Migrating Version 5.x applications with Web services security to Version 6.1 applications

Migration of a Java 2 Platform, Enterprise Edition (J2EE) Version 1.3 application that uses Web services security to a Java 2 Platform, Enterprise Edition Version 1.4 application is possible.

You can install Java 2 Platform, Enterprise Edition Version 1.3 applications that use Web services security on a WebSphere Application Server Version 6.1 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 6.1, you must migrate the J2EE Version 1.3 applications to J2EE Version 1.4.

Complete the following steps to migrate a Version 5.x application, along with the Web services security configuration information, to a Version 6.1 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 6.1 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.xml` server-side extensions file. For more information, see “Migrating the server-side extensions configuration.”
2. Configure the `ibm-webservicesclient-ext.xml` client-side extensions file. For more information, see “Migrating the client-side extensions configuration” on page 969.
3. Configure the `ibm-webservices-bnd.xml` server-side bindings file. For more information, see “Migrating the server-side bindings file” on page 970.
4. Configure the `ibm-webservicesclient-bnd.xml` client-side bindings file. For more information, see “Migrating the client-side bindings file” on page 972.

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 37. 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 6.0.x, see Assembly tools.

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 38. 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 39. 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 6.1 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.”

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 40. 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 38 on page 968. 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 967.

- 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.”

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 41. 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
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>
  <wss:SecurityTokenReference>
    <wss:Reference URI="#mytoken" />
  </wss: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#">
  <wss:SecurityTokenReference>
    <wss:KeyIdentifier ValueType="wss:X509v3">/62wX0...</wss:KeyIdentifier>
  </wss: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.wsssecurity.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 describe 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 client-side bindings file.”

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 42. 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

Table 42. The mapping of the configuration sections (continued)

J2EE Version 1.3 binding configuration for Web services security	J2EE Version 1.4 binding configuration for Web services security
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 970.
- 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 970.
- 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 970.

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 6.1 applications” on page 966.

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 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***.
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. Click **Manage 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**.

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 > Secure administration, applications, and infrastructure**.
2. Under Authentication, click **Java Authentication and Authorization Service > 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**.

Default implementations of the Web services security service provider programming interfaces

This information describes the default implementations of the service provider interfaces (SPI) for Web services security within WebSphere Application Server Version 6 and later.

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 1224 for the Version 5.x implementations that are deprecated in Version 6.0.x and later.

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 the following callback handlers:

```
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 user name 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 keystore 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.

Default configuration

WebSphere Application Server Version 6 and later provide 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 and later, 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 and later provide 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 985.
- Uses the `gen_sigtgen` 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_sigtgen_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 985.
 - Uses the `gen_sigtgen` 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 and later provide 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/xmlsig#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 985.
 - References the `con_sigtcon` token consumer configuration. The following information pertains to the `con_sigtcon` configuration:
 - Uses the X.509 Token Consumer, which is configured as the consumer for the default signing information.
 - Contains the `sigtconsumer_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 985.
 - 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 directory: `app_server_root/bin/ikeyman`

The key tool is located in the following directory: *app_server_root/java/jre/bin/keytool*

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 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 1115
- “Configuring the key locator for the consumer binding on the application level” on page 1170
- “Configuring the key locator on the server or cell level” on page 1198

Version 5.x application

SimpleClientSignerKey

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.

Version 5.x application

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.

Version 5.x application

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.

Version 5.x application

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`.

Version 5.x application

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`.

Version 6 and later applications

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.

Version 6 and later applications

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. 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.

Version 6 and later applications

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.

Version 6 and later applications

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.

Version 6 and later applications

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 1087
- “Configuring the collection certificate store for the consumer binding on the application level” on page 1159
- “Configuring the collection certificate store for the server or cell-level bindings” on page 1185

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 1083
- “Configuring trust anchors for the consumer binding on the application level” on page 1157
- “Configuring trust anchors on the server or cell level” on page 1184

Version 5.x application

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.

Version 5.x application

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 1206.

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, which is used for identity assertion, defines the key

name and value pair for the trusted identity. The key name is in the form `trustedId_*` and the value is the trusted identity. For more information, see the example in “Configuring trusted ID evaluators on the server or cell level” on page 1206.

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**.

Basic Security Profile compliance

The Web Services Interoperability Organization (WS-I) Basic Security Profile (BSP) 1.0 promotes interoperability by providing clarifications and amplifications to a set of non-proprietary Web services specifications. WebSphere Application Server Web Services Security provides configuration options to ensure that the BSP recommendations and security considerations can be enabled to ensure interoperability. The degree to which you follow these recommendations is then a measure of how well the application you are configuring complies with the Basic Security Profile (BSP).

Support for applications to comply to the Basic Security Profile (BSP) is new in WebSphere Application Server Version 6.1. For more information on the Basic Security Profile, see Web Services Interoperability Organization (WS-I) Basic Security Profile (BSP), Basic Security Profile Version 1.0.

You can use either a predefined list of keywords or XPath expressions to comply to the BSP. Both the keywords and the XPath expressions are specified in the deployment descriptor configuration file and are configured using an assembly tool.

Basic Security Profile recommendations

Follow these recommendations to ensure that your configured applications are Basic Security Profile (BSP) compliant.

- Do not use the original XPath transform, <http://www.w3.org/TR/1999/REC-xpath-19991116>
When you refer to an element in a `SECURE_ENVELOPE` that does not carry an ID attribute type from a `ds:Reference` in a `SIGNATURE` element, you must use the XPath Filter 2.0 transform, <http://www.w3.org/2002/06/xmldsig-filter2> to refer to that element.
Any `ds:Transform/@Algorithm` attribute in a `SIGNATURE` element must have one of these values:
 - <http://www.w3.org/2001/10/xml-exc-c14n#>
 - <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/2000/09/xmldsig#enveloped-signature>
 - <http://docs.oasis-open.org/wss/2004/XX/oasis-2004XX-wss-swa-profile-1.0#Attachment-Content-Only-Transform>
 - <http://docs.oasis-open.org/wss/2004/XX/oasis-2004XX-wss-swa-profile-1.0#Attachment-Complete-Transform>
- Do not use the <http://www.w3.org/2000/09/xmldsig#dsa-sha1> signature algorithm.
Any `ds:SignatureMethod/@Algorithm` element in a `SIGNATURE` that is based on a symmetric key must have one of the following values:
 - <http://www.w3.org/2000/09/xmldsig#rsa-sha1>
 - <http://www.w3.org/2000/09/xmldsig#hmac-sha1>
- Do not specify the `digestvalue` keyword for the message part to encrypt. Instead, use the `signature` keyword.

If the value of a `ds:DigestValue` element in a `SIGNATURE` element requires encryption, the entire parent `ds:Signature` element must be encrypted. A `SIGNATURE` must not have any `xenc:EncryptedData` elements among its descendants.

- Do not use the `KEYNAME` key information type
`KEYNAME` references can be ambiguous and compliance with the BSP disallows the use of `KEYNAME`. A `SECURITY_TOKEN_REFERENCE` must not use a key name to reference a `SECURITY_TOKEN`. The child element of a `ds:KeyInfo` element in an `ENCRYPTED_KEY` must be either a `SECURITY_TOKEN_REFERENCE` or a `ds:MgmtData` element. Using a `KEYNAME` key information type for an encryption key results in a `KeyName` child element of a `ds:KeyInfo` element and is disallowed for BSP compliance.
- Do not use the `http://www.w3.org/2001/04/xmlenc#aes192-cbc` bit data encryption algorithm.
Any `xenc:EncryptionMethod/@Algorithm` attribute in an `ENCRYPTED_DATA` element must have one of these values:
 - `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`
- Do not use the advanced encryption standard (AES) key wrap (aes192): `http://www.w3.org/2001/04/xmlenc#kw-aes192` key encryption algorithm.
When used for key wrap, any `xenc:EncryptionMethod/@Algorithm` attribute in an `ENCRYPTED_KEY` element must have one of these values:
 - `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`

Configuration Options for BSP Compliance

You achieve BSP compliance when certain configuration choices are made. The assembly tool assists you in using appropriate choices when configuring the application by issuing warning messages. The following configuration descriptions comprise these warnings:

- When configuring the `ds:Transforms` element in a signature, the list of transforms must include as its last child element `http://www.w3.org/2001/10/xml-exc-c14n#` or `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform`
- Add a `wsse:Nonce` or `wsse:Created` element to a Username token to prevent replay. After the element is added, sign the Username token to prevent undetected alteration of these fields; otherwise, replay can occur.

Configuring an application for Web services security with an assembly tool

There are eight parts of Web services security that you must configure to secure your 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.

Prior to completing this task, you must import your application into an assembly tool.

For information on how to import your application, see *Importing an enterprise application EAR file*.

Decide whether or not you are to configure your application to comply with the Basic Security Profile. For more information on how to ensure compliance, see “Basic Security Profile compliance” on page 988

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 43. 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 Versions 6 and later, *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 standalone 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 1074 and “Adding a stand-alone time stamp in consumer security constraints” on page 1075.
- Configure the security token in the generator and consumer constraints. For more information, see “Configuring the security token in generator security constraints” on page 1077 and “Configuring the security token requirement in consumer security constraints” on page 1078.
- Configure a caller part for the consumer security constraints. For more information, see “Configuring the caller in consumer security constraints” on page 1079.
- Configure identity assertion. For more information, see “Configuring identity assertion” on page 1081.
- 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 or an XPath expression. For more information, see either “Signing message elements in generator security constraints with keywords” on page 993 or “Signing message elements in generator security constraints with an XPath expression” on page 1002. 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 996

- “Adding time stamps for integrity to generator security constraints with an XPath expression” on page 1003
 - “Adding a nonce for integrity in generator security constraints with keywords” on page 999
 - “Adding a nonce for integrity to generator security constraints with an XPath expression” on page 1005
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 1008.
 3. Configure the token generator. For more information, see “Configuring token generators with an assembly tool” on page 1009.
 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 1015.
 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 1017.
 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 1019.
 7. Specify which message elements to sign in the consumer security constraints using either keywords or an XPath expression. For more information, see either “Signing message elements in consumer security constraints with keywords” on page 1023 or “Signing message elements in consumer security constraints with an XPath expression” on page 1029. 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 1025
 - “Adding a nonce for integrity in consumer security constraints with keywords” on page 1027
 - “Adding time stamps for integrity in consumer security constraints with an XPath expression” on page 1030
 - “Adding a nonce for integrity in consumer security constraints with an XPath expression” on page 1032
 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 1033.
 9. Configure a token consumer. For more information, see “Configuring token consumers with an assembly tool” on page 1035.
 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 1038.
 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 1039.
 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 1042.
- 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 or an XPath expression. For more information, see either “Encrypting the message elements in generator security constraints with keywords” on page 1045 or “Encrypting the message elements in generator security constraints with an XPath expression” on page 1052. 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 1047

- “Adding the nonce for confidentiality to generator security constraints with keywords” on page 1049
 - “Adding time stamps for confidentiality to generator security constraints with an XPath expression” on page 1053
 - “Adding the nonce for confidentiality to generator security constraints with an XPath expression” on page 1055
2. Configure the token generator. For more information, see “Configuring token generators with an assembly tool” on page 1009.
 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 1015.
 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 1017.
 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 1060.
 6. Specify which message elements to encrypt in the consumer security constraints using either keywords or an XPath expression. For more information, see either “Encrypting message elements in consumer security constraints with keywords” on page 1062 or “Encrypting message elements in consumer security constraints with an XPath expression” on page 1067. 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 1063
 - “Adding a nonce for confidentiality in consumer security constraints with keywords” on page 1065
 - “Adding time stamps for confidentiality in consumer security constraints with an XPath expression” on page 1068
 - “Adding the nonce for confidentiality in consumer security constraints with an XPath expression” on page 1070
 7. Configure a token consumer. For more information, see “Configuring token consumers with an assembly tool” on page 1035.
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 1038.
 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 1039.
 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 1072.

By completing the previous steps, you have configured your application for either digital signature (integrity) or encryption (confidentiality).

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.

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 an enterprise application EAR file" in the Application Server Toolkit documentation.

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.
2. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > J2EE**.
3. **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.

4. **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.
5. 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 992.
6. 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.
7. 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:

action Specifies that the `<wsa:Action>` element is signed.

body Specifies the user data portion of the message. If you select this keyword, the body 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.

relatesto

Specifies that the `<wsa:RelatesTo>` element within the message is signed.

securitytoken

Specifies that the UsernameToken in the SOAP message 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 1074.

to

Specifies that the `<wsa:To>` element within the message is signed.

wsaall

Specifies all of the WS-Addressing elements in the SOAP header.

wsafaultto

Specifies the `<wsa:FaultTo>` WS-Addressing FaultTo element in the SOAP header.

wsafrom

Specifies the `<wsa:From>` WS-Addressing From element in the SOAP header.

wsareplyto

Specifies the `<wsa:ReplyTo>` WS-Addressing ReplyTo element in the SOAP header.

wsccontext

Specifies the WS-Context header for the SOAP header. For more information, see Propagating work area context over Web services.

8. 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 996
- “Adding time stamps for integrity to generator security constraints with an XPath expression” on page 1003
- “Adding a nonce for integrity in generator security constraints with keywords” on page 999
- “Adding a nonce for integrity to generator security constraints with an XPath expression” on page 1005

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/vXUo74xpF0Qd/rr1owBmMdb1TWd0yZwb0HC71kUlnKrK17SofwSLSDUP571iiMXUx3tRdmAVCoDMMFuDXh9V72121uXccx0s1S5KN0D3xw97LLNegQC0/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>vyyu0JwXXSAvRCUCi6TPkeH8yUTU=</ds:DigestValue>
          </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue>dtbYA609wwAUA8BmDmJlVhrWShy60LJB3n4A6ToU01I9tJrNhBksGqks17cykf+uHTJcg0Y18XrRDN4wHTW4zm/tmD5WqQd8K1WpYaGpbw1FoiwKVFNyfqN2K/WbZ2JccmZvJGFa0tqStg6TqSUGLQSA5MCSpZUhck545IY2F4=
        </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>
  <!-- Body content -->
</soapenv:Envelope>
```

```

    </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 1019.

Adding time stamps for integrity to generator security constraints with keywords:

You can specify that a time stamp be embedded in a particular element and that the element be signed.

Prior to completing this task, you must import your application into an assembly tool.

For information on how to import your application, see the article about importing an enterprise application EAR file in the Application Server Toolkit documentation.

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.
2. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > J2EE**.
3. **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.
4. **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.
5. 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 SOAP messages, see “XML digital signature” on page 992.
6. 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.
7. 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 SOAP body and the body is signed.

timestamp

Specifies that the time stamp is embedded in the standalone timestamp element within the message and that it is signed. If you select the timestamp option, make sure that there is a standalone timestamp element in the message. If the element does not exist, see “Adding a stand-alone time stamp to generator security constraints” on page 1074.

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.

wscontext

Specifies the WS-Context header for the SOAP header.

For more information, see Propagating work area context over Web services.

wsafrom

Specifies the <wsa:From> WS-Addressing From element in the SOAP header.

wsareplyto

Specifies the <wsa:ReplyTo> WS-Addressing ReplyTo element in the SOAP header

wsafaultto

Specifies the <wsa:FaultTo> WS-Addressing FaultTo element in the SOAP header.

wsaall Specifies all of the WS-Addressing elements in the SOAP header.

- 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.

9. In the Message Parts section, click **Add** and select `http://www.ibm.com/websphere/webservices/wssecurity/dialect-was` in the Message parts dialect field.
10. 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.

11. 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 999
- “Adding a nonce for integrity to generator security constraints with an XPath expression” on page 1005

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"> AgBgkqhkiG9w0BCQEW21hcnV5Y
W1hQGpwLmlibS5jb22CAgEBMA0GCsqGSib3DQEBBQUAA4GBAHkthdGDgCvdIL9/vXUo74xf0Qd/rrlowB
mMdb1TWdOyzwb0HC71kUlnKrK17SofwSLSDUP571iIMXUx3tRdmAVCoDMMFuDXh9V7212luXccx0s1S5KN
0D3xW97LLNegQC0/b+aFD8XKw2U5ZtwnbFTRgs097dmz09RosDKkLlM</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:Signature>
  </wsse:Security>
</soapenv:Header>
</soapenv:Envelope>
```

```

        </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue>XBpPju5+qH4bBFod01kbb054kEdBD0Pr5ohnXa3TPrDwXqmr67zDP3ZTk7iBS
    ADnH+d1fKupFhx+Nzu2h5/j1/KYwaR2HTTv/KYE6IdqXVz3EFglUIBLzQnJ2Zbn62eBx5Th285Cn2Vr
    xtdb5BvUaldt6M6k61CvR1z3/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 1019.

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.

See the information about Importing an enterprise application EAR file in the Application Server Toolkit documentation.

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 SOAP messages, see “XML digital signature” on page 992.
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 SOAP body element and the body element is signed.

timestamp

Specifies that the nonce is embedded in the standalone `timestamp` element within the message and the element is signed. If `timestamp` is selected, make sure that there is a standalone `timestamp` element in the message. If not, see “Adding a stand-alone time stamp to generator security constraints” on page 1074.

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.

wscontext

Specifies the WS-Context header for the SOAP header. For more information, see Propagating work area context over Web services.

wsafrom

Specifies the `<wsa:From>` WS-Addressing From element in the SOAP header.

wsareplyto

Specifies the `<wsa:ReplyTo>` WS-Addressing ReplyTo element in the SOAP header.

wsafaultto

Specifies the `<wsa:FaultTo>` WS-Addressing FaultTo element in the SOAP header.

wsaa11 Specifies all of the WS-Addressing elements in the SOAP header.

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 1025
- “Adding time stamps for integrity in consumer security constraints with an XPath expression” on page 1030

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"> E21hcnV5YW1hQGpwLmliS5jb22CAgEBMA0GCSqGS1b3DQEBBQUAA4GBAHkthdGDgCvdIL9/vXUo74xpF0Qd/rr1owBmMdb1TWd0yzwb0HC71kUlnKrkI7SofwSLSDUP571iMXUx3tRdmAVCoDMMFuDXh9V72121uXccx0s1S5KN0D3xw97LLNegQC0/b+aFD8XKw2U5ZtwnbFTRgs097dmz09RosDKkL1M
</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>iE2G53VMwCFBI6Bw0Wia0LYvemZUJTXJocXpy81oyW1LiR8bBQcFioD0uD
XXzVj3K+ZD2pYhc0krVYqkYY0IZoRx7xpWt+9qn7aSbxKjuH1MFNCdB1Uxp608zCzCswvuoCffj1
001tUQ8JTEBnmMB0cfaoiG5bFkU0EpkFo2P9c</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:Body>
</soapenv:Envelope>
```

```

        </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/r3VPSbQ1oZTs0K
s1GE/iDL4YbKDTwdL+e2Hb7nNZn397nRJQ9ePGgf7PRdEuqATFbfq0/T+6j6Fk/MbSHmZnHHoBscFX8W/dY
ssyCmWdp99447kRhnJbNg5JxarkFmMLqpxKfm1iP3hKP5DpJY=</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 1042.

Signing message elements in generator security constraints with an XPath expression

You can specify which message parts to digitally sign when you configure the generator security constraints for either the request generator or the response generator.

Prior to completing this task, you must import your application into an assembly tool.

For information on how to import your application, see “Importing an enterprise application EAR file” in the Application Server Toolkit documentation.

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.
2. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > J2EE**.
3. **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 must 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.
4. **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 must 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.
5. Expand the Integrity section. Integrity refers to digital signature and confidentiality refers to encryption. Integrity decreases the risk of data modification when you transmit data across a network. For more information on digitally signing SOAP messages, see “XML digital signature” on page 992.
6. 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.
7. 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.
Do not use this transform algorithm if you want your configured application to be compliant with the Basic Security Profile (BSP). Instead use `http://www.w3.org/2002/06/xmldsig-filter2` to ensure compliance.
 - 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']
```
8. 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 996
- “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 999
- “Adding a nonce for integrity to generator security constraints with an XPath expression” on page 1005

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 1019.

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 an enterprise application EAR file” in the Application Server Toolkit documentation.

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.
2. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > J2EE**.
3. **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.
4. **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.
5. 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 992.
6. 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_tmstamp`.
 - 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.
7. 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.

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. This message part is signed as well as the parent element of the time stamp.

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 999
- “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 1019.

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 an enterprise application EAR file” in the Application Server Toolkit documentation.

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 992.
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 996
- “Adding time stamps for integrity to generator security constraints with an XPath expression” on page 1003

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 1019.

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 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*.

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.

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 an enterprise application EAR file" in the Application Server Toolkit documentation.

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.
2. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > J2EE**.
3. **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.
4. **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.
5. Expand the Certificate Store List > Collection Certificate Store section and click **Add**.
6. 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.
7. Specify a certificate store provider in the Provider field. The IBM CertPath 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).
8. 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.

9. 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.

10. Click **OK** to save your configuration.

Trust anchor

A *trust anchor* specifies the keystores that contain trusted root certificates. These certificates are used to validate the X.509 certificate that is embedded in the SOAP message.

These keystores 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 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 compromised. 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 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 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 keystore object. The type, path, and password of the keystore are passed to the implementation through the administrative console or by scripting.

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 an enterprise application EAR file" in the Application Server Toolkit documentation.
- 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 1077.
- 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 1008.

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 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.
2. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > J2EE**.
3. **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.
4. **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.
5. **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.
6. 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.
7. Specify a unique name in the Token generator name field. For example, you might specify `gen_sigtgen`. 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.
8. 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`
9. 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.
10. 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
11. 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 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.

12. 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`
13. Specify the basic authentication password in the Password field. This password is passed to the constructors of the callback handler implementation.
14. **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.

JCERACFKS

Use JCERACFKS if the certificates are stored in a SAF key ring (z/OS only).

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.
15. **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 1008.
16. **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.

17. 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 1017.

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 Versions 6 and later support a secret key-based signature called HMAC-SHA1. If you use HMAC-SHA1, the 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 Versions 6 and later 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.

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 an enterprise application EAR file" in the Application Server Toolkit documentation.

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 and later provide 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 and later 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) policy file and if your keystore file uses the Java Keystore (JKS) format.

JCEKS

Use this option if you are using Java Cryptography Extensions policy file.

JCERACFKS

Use JCERACFKS if the certificates are stored in a SAF key ring (z/OS only).

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.”

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 an enterprise application EAR file” in the Application Server Toolkit documentation.
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 1015.
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 1009

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 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.
2. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > J2EE**.
3. **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.
4. **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.
5. Expand the Key Information section and click **Add** to add a new entry or click **Edit** to edit a selected entry.
6. 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 1019 and “Configuring encryption information for the generator binding with an assembly tool” on page 1072.
7. 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">/62wX0...</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>
```

Note: To be compliant with the Basic Security Profile (BSP), do not use the KEYNAME information type. A SECURITY_TOKEN_REFERENCE must not use a KEYNAME to reference a SECURITY_TOKEN.

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>
```

8. 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 1015.
 - 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.
9. **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 1009.

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” or “Configuring encryption information for the generator binding with an assembly tool” on page 1072.

Configuring signing information for the generator binding with an assembly tool

You can configure the signing information for the server-side and client-side generator bindings by using 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 an enterprise application EAR file” in the Application Server Toolkit documentation.
2. Specify which message parts to digitally sign. For more information, see “Signing message elements in generator security constraints with keywords” on page 993 or “Signing message elements in generator security constraints with an XPath expression” on page 1002.
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 1017.

Complete the following steps to configure the signing information for the server-side and client-side bindings by 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.
2. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > J2EE**.
3. **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 must 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.
4. **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 must 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.
5. 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 1042.
 - c. **Optional:** Select **Show only FIPS Compliant Algorithms** if you want only the FIPS compliant algorithms to show in the encryption method algorithm drop-down lists. Use this option if you expect this application to run on a WebSphere Application Server that has set the **Use the United States Federal Information Processing Standard (FIPS) algorithms** option in the SSL certificate and key management panel of the administrative console for WebSphere Application Server.
 - d. 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#hmac-sha1>
 - <http://www.w3.org/2000/09/xmldsig#dsa-sha1>

Do not use this algorithm if you want the configured application to be compliant with the Basic Security Profile (BSP). Any `ds:SignatureMethod/@Algorithm` element in a SIGNATURE based on a symmetric key must have a value of <http://www.w3.org/2000/09/xmldsig#rsa-sha1> or <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 1042.
6. 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 1017.
 - c. **Optional:** Select the **Use key information signature** option if you want to sign the key information within the 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.
7. **Optional:** Determine whether to disable the Inclusive namespace prefix list. The Exclusive XML Canonicalization Version 1.0 specification recommends that you include all of the namespace declarations that correspond to the namespace prefix in the canonicalization form. For security reasons, WebSphere Application Server, by default, includes the prefix in the digital signature for Web services security. However, some implementations of Web services security cannot handle this prefix list. WebSphere Application Server can handle digitally signed messages that either contain or do not contain the prefix list. If you experience a signature validation failure when a signed SOAP message is sent and you are using another vendor in your environment, it is highly recommended that you check with their Web site for a possible fix to their implementation before you disable this property. To disable this property, complete the following steps:
 - a. Under Properties, click **Add**.
 - b. In the Name field, enter the `com.ibm.wsspi.wssecurity.dsig.inclusiveNamespaces` property.
 - c. In the Value field, enter the `false` value.
 - d. Click **OK**.
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.
 - 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 993 or “Signing message elements in generator security constraints with an XPath expression” on page 1002.
 - c. **Optional:** Select **Show only FIPS Compliant Algorithms** if you want only the FIPS compliant algorithms to show in the encryption method algorithm drop-down lists. Use this option if you expect this application to run on a WebSphere Application Server that has set the **Use the United States Federal Information Processing Standard (FIPS) algorithms** option in the SSL certificate and key management panel of the administrative console for WebSphere Application Server.
 - d. Select a digest method algorithm in the Digest method algorithm field.
 WebSphere Application Server supports the following digest method algorithms:
 - <http://www.w3.org/2000/09/xmldsig#sha1>
 - <http://www.w3.org/2001/04/xmlenc#sha256>
 - <http://www.w3.org/2001/04/xmlenc#sha512>
 This digest method algorithm is used to create the digest for each message part that is specified by this part reference.

- e. Expand the Transforms subsection and the part reference configuration from the Part reference subsection.
- f. 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.
- g. Specify a transform name in the Name field. For example, you might specify `reqint_body_transform1`.
- h. 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/xmlnsig#' and local-name()='Signature'])`

Note: Do not use this transform algorithm if you want your configured application to be compliant with the Basic Security Profile (BSP). Instead use <http://www.w3.org/2002/06/xmlsig-filter2> to ensure compliance.

<http://www.w3.org/2002/06/xmlsig-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/xmlsig#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.

11. Click **OK**.

After you complete this task for the generator binding, you must configure the signing information for consumer binding.

Signing message elements in consumer security constraints with keywords

You can 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.

Prior to completing this task, you must import your application into an assembly tool.

See information about importing an enterprise application EAR file in the Application Server Toolkit documentation.

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 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.
2. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > J2EE**.
3. **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.
4. **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.
5. 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 SOAP messages, see “XML digital signature” on page 992.
6. 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.

7. 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 standalone `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 standalone `timestamp` element in the message. If the element does not exist, see “Adding a stand-alone time stamp in consumer security constraints” on page 1075.

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.

wscontext

Specifies the WS-Context header for the SOAP header. For more information, see Propagating work area context over Web services.

wsafrom

Specifies the `<wsa:From>` WS-Addressing From element in the SOAP header.

wsareplyto

Specifies the `<wsa:ReplyTo>` WS-Addressing ReplyTo element in the SOAP header.

wsafaultto

Specifies the `<wsa:FaultTo>` WS-Addressing FaultTo element in the SOAP header.

wsaa11 Specifies all of the WS-Addressing elements in the SOAP header.

8. 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 1030
- “Adding a nonce for integrity in consumer security constraints with keywords” on page 1027
- “Adding a nonce for integrity in consumer security constraints with an XPath expression” on page 1032

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 1042.

Adding time stamps for integrity in consumer security constraints with keywords:

You can 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.

Prior to completing this task, you must import your application into an assembly tool.

For information on how to import your application, see “Importing an enterprise application EAR file” in the Application Server Toolkit documentation.

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.
2. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > J2EE**.
3. **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.
4. **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.
5. 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 SOAP messages, see “XML digital signature” on page 992.
6. 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.

7. 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 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 time stamp is inserted into the <wsa:RelatesTo> element within the message and the <wsa:RelatesTo> element is signed.

wscontext

Specifies the WS-Context header for the SOAP header. For more information, see Propagating work area context over Web services.

wsafrom

Specifies the <wsa:From> WS-Addressing From element in the SOAP header.

wsareplyto

Specifies the <wsa:ReplyTo> WS-Addressing ReplyTo element in the SOAP header.

wsafaultto

Specifies the <wsa:FaultTo> WS-Addressing FaultTo element in the SOAP header.

wsaa11 Specifies all of the WS-Addressing elements in the SOAP header.

8. 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 1032

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 1042.

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 an enterprise application EAR file” in the Application Server Toolkit documentation.

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 992.
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 1075 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.

wscontext

Specifies the WS-Context header for the SOAP header. For more information, see Propagating work area context over Web services.

wsafrom

Specifies the <wsa:From> WS-Addressing From element in the SOAP header.

wsareplyto

Specifies the <wsa:ReplyTo> WS-Addressing ReplyTo element in the SOAP header.

wsafaultto

Specifies the <wsa:FaultTo> WS-Addressing FaultTo element in the SOAP header.

wsaall Specifies all of the WS-Addressing elements in the SOAP header.

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 1025
- “Adding time stamps for integrity in consumer security constraints with an XPath expression” on page 1030

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 1042.

Signing message elements in consumer security constraints with an XPath expression

You can specify which message parts are expected to be signed using 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 an enterprise application EAR file” in the Application Server Toolkit documentation.

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.
2. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > J2EE**.
3. **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 must 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.
4. **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 must 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.
5. Expand the Required Integrity section. Integrity refers to digital signature and confidentiality refers to encryption. Integrity decreases the risk of data modification when you transmit data across a network. For more information on digitally signing SOAP messages, see “XML digital signature” on page 992.
6. 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, 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.

7. 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.
Do not use this transform algorithm if you want your configured application to be compliant with the Basic Security Profile (BSP). Instead use `http://www.w3.org/2002/06/xmldsig-filter2` to ensure compliance.
 - 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.

8. 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 1025
- “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 1027
- “Adding a nonce for integrity in consumer security constraints with an XPath expression” on page 1032

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 1042.

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 an enterprise application EAR file" in the Application Server Toolkit documentation.

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.
2. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > J2EE**.
3. **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.
4. **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.
5. 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 992.
6. 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.
7. 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.

8. 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.
9. In the Message Parts section, select the message parts keyword.
10. 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 1027
- “Adding a nonce for integrity in consumer security constraints with an XPath expression”

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 1042.

Adding a nonce for integrity in consumer security constraints with an XPath expression:

You can 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.

Prior to completing this task, you must import your application into an assembly tool.

See information about Importing an enterprise application EAR file in the Application Server Toolkit documentation.

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 SOAP messages, see “XML digital signature” on page 992.
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 1025
- “Adding time stamps for integrity in consumer security constraints with an XPath expression” on page 1030

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 1042.

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 an enterprise application EAR file” in the Application Server Toolkit documentation.

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.
2. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > J2EE**.
3. **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.
4. **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.
5. Expand the Certificate Store List > Collection Certificate Store section and click **Add**.
6. 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.
7. Specify a certificate store provider in the Provider field. The IBM CertPath 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).
8. 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.

9. 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/mycr1`. 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.

10. Click **OK** to save your configuration.

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.

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 an enterprise application EAR file" in the Application Server Toolkit documentation.
- 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 1078
- 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 1033.

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 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.
2. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > J2EE**.

3. **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.
4. **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.
5. **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.
6. 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.
7. 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.
8. 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.
9. 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.
10. 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.
11. **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 44. JAAS configuration names and the corresponding value type

<code>jaas.config</code> name	Value type
<code>system.wssecurity.UsernameToken</code>	Username Token

Table 44. 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

12. **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:
- 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 6.0.x 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.
 - 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.
 - 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.

13. **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 45. Nonce and time stamp properties

Name	Value
<code>com.ibm.wsspi.wssecurity.token.username.verifyNonce</code>	true
<code>com.ibm.wsspi.wssecurity.token.username.verifyTimestamp</code>	true

14. **Optional:** Select the **Use certificate path settings** option if you are configuring this token consumer for an X.509 security token.
15. 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:

- 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).
16. 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 1039.

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 an enterprise application EAR file” in the Application Server Toolkit documentation.

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 6.0.x and later 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.
2. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > J2EE**.
3. **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.
4. **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.
5. Expand the Key locators section and click **Add** to add a new entry or click **Edit** to edit a selected entry.
6. 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.
7. 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.

8. 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) policy file and if your keystore file uses the Java Keystore (JKS) format.
 - JCEKS** Use this option if you are using Java Cryptography Extensions policy file.
 - JCERACFKS** Use JCERACFKS if the certificates are stored in a SAF key ring (z/OS only).
 - 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.
9. 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.
10. 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.”

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 an enterprise application EAR file” in the Application Server Toolkit documentation.

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 1038.
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 1035

Complete the following steps using an assembly tool to configure the key information for the server-side and client-side bindings. 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 must 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 must 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 1042 and “Configuring encryption information for the consumer binding with an assembly tool” on page 1060.
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">/62wX0...</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>
```

Note: To be compliant with the Basic Security Profile (BSP), do not use the KEYNAME information type. A SECURITY_TOKEN_REFERENCE must not use a KEYNAME to reference a SECURITY_TOKEN.

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 1038.
 - 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 1035.

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 1060.

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 an enterprise application EAR file” in the Application Server Toolkit documentation.
2. Specify which message parts to digitally sign. For more information, see “Signing message elements in consumer security constraints with keywords” on page 1023 or “Signing message elements in consumer security constraints with an XPath expression” on page 1029.
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 1039.

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.
2. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > J2EE**.
3. **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 must 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.
4. **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 must 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.
5. 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 1019.

- c. **Optional:** Select **Show only FIPS Compliant Algorithms** if you want only the FIPS compliant algorithms to show in the **Digest method algorithm** drop-down list. Use this option if you expect this application to run on a WebSphere Application Server that has set the **Use the United States Federal Information Processing Standard (FIPS) algorithms** option in the SSL certificate and key management panel of the administrative console for WebSphere Application Server.
 - d. 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#hmac-sha1>
 - <http://www.w3.org/2000/09/xmldsig#dsa-sha1>

Do not use this algorithm if you want the configured application to be compliant with the Basic Security Profile (BSP). Any ds:SignatureMethod/@Algorithm element in a SIGNATURE based on a symmetric key must have a value of <http://www.w3.org/2000/09/xmldsig#rsa-sha1> or <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 1019.
6. 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 1039.
 7. **Optional:** Select the **Use key information signature** option if you want to sign the key information within the SOAP message.
 8. **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.
 9. Click **OK** to save your signing information configuration.
 10. Expand the Part References subsection and select the signing information configuration from the Signing Information section.
 11. 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 1023 or “Signing message elements in consumer security constraints with an XPath expression” on page 1029.
 - c. **Optional:** Select **Show only FIPS Compliant Algorithms** if you want only the FIPS compliant algorithms to show in the **Digest method algorithm** drop-down list. Use this option if you expect

this application to run on a WebSphere Application Server that has set the **Use the United States Federal Information Processing Standard (FIPS) algorithms** option in the SSL certificate and key management panel of the administrative console for WebSphere Application Server.

- d. Select a digest method algorithm in the Digest method algorithm field.

WebSphere Application Server supports the following digest method algorithms:

- <http://www.w3.org/2000/09/xmldsig#sha1>
- <http://www.w3.org/2001/04/xmllenc#sha256>
- <http://www.w3.org/2001/04/xmllenc#sha512>

This digest method algorithm is used to create the digest for each message part that is specified by this part reference.

- e. Click **OK** to save your part reference configuration.

12. Expand the Transforms subsection and the part reference configuration from the Part reference subsection.

13. 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 click **Add** to specify the property name and value, 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'])

Note: Do not use this transform algorithm if you want your configured application to be compliant with the Basic Security Profile (BSP). Instead use <http://www.w3.org/2002/06/xmldsig-filter2> to ensure compliance.

<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.dsign.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/xmlsig#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.

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 an enterprise application EAR file" in the Application Server Toolkit documentation.

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.
2. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > J2EE**.
3. **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 must 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.
4. **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 must 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.
5. 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 1057.

6. 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.
7. 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.

usnametoken

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 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.

signature

Specifies an entire signature. You can encrypt the signature element, `ds:Signature`, by selecting this message part.

Note: If the value of a `ds:DigestValue` element in a signature needs to be encrypted, the entire parent `ds:Signature` element must be encrypted to ensure that the application complies to the Basic Security Profile (BSP). You can use the signature keyword to perform the encryption.

wscontextcontent

Encrypts the content in the WS-Context header for the SOAP header. For more information, see Propagating work area context over Web services.

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” on page 1047
- “Adding time stamps for confidentiality to generator security constraints with an XPath expression” on page 1053
- “Adding the nonce for confidentiality to generator security constraints with keywords” on page 1049
- “Adding the nonce for confidentiality to generator security constraints with an XPath expression” on page 1055

8. Click **OK** to save your configuration.

The following sample is a part of a Simple Object Access Protocol (SOAP) message; the 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>n1lF+Uthee0H96HbtRro1J/tBm0azyryNYRwr/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 1072.

Adding time stamps for confidentiality to generator security constraints with keywords:

You can specify that a time stamp is embedded in a particular element and that the element is encrypted.

Prior to completing this task, you must import your application into an assembly tool.

For information on how to import your application, see “Importing an enterprise application EAR file” in the Application Server Toolkit documentation.

Complete the following steps to specify the time stamp for confidentiality by 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.
2. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > J2EE**.
3. **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 must 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.
4. **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 must 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.
5. Expand the Confidentiality section. Confidentiality refers to encryption and 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 1057.
6. 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_tmstamp`.

- 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.
7. 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 SOAP message body and the body is encrypted.

username token

Specifies a user name 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 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.

signature

Specifies an entire signature. You can encrypt the signature element, `ds:Signature`, by selecting this message part.

Note: If the value of a `ds:DigestValue` element in a signature needs to be encrypted, the entire parent `ds:Signature` element must be encrypted to ensure that the application complies to the Basic Security Profile (BSP). You can use the signature keyword to perform the encryption.

wscontextcontent

Encrypts the content in the WS-Context header for the SOAP header. For more information, see *Propagating work area context over Web services*.

8. 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 arbitrary decimal digits. 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.

9. In the Message Parts section, click **Add** and select `http://www.ibm.com/websphere/webservices/wssecurity/ dialect-was` in the Message parts dialect field.
10. 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.

11. 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”
- “Adding the nonce for confidentiality to generator security constraints with an XPath expression” on page 1055

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/0fV0sF
        un0uxg/h/Y+1erE+NaysREuL+E9AQm01xALNEdBX9zpeVf+ZffUCSzfXXe9iosQ1Pe9jG
        7yTp+rhZGdp/KOp26c3DZXCNDr0Wgz31wn3KNm6bG06RmBzahEOSW8d0wR999DeqSp0Y12
        d8iWJa3HZ8gnGnineCiZ3wrHy9r0C58iijcsNv1fP31ExuA5WkHra6rndhbi8P7jDMhkzf
        40dj2yy1M3XURWa1OLNYhNJ9YaWACsaYCY2ukcKtzw==</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 1072.

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 an enterprise application EAR file” in the Application Server Toolkit documentation.

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.
2. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > J2EE**.
3. **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 must 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.
4. **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 must 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.
5. Expand the Confidentiality section. Confidentiality refers to encryption and 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 1057.
6. 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.
7. 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.

username token

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.

signature

Specifies an entire signature. You can encrypt the signature element, ds:Signature, by selecting this message part.

Note: If the value of a ds:DigestValue element in a signature needs to be encrypted, the entire parent ds:Signature element must be encrypted to ensure that the application complies to the Basic Security Profile (BSP). You can use the signature keyword to perform the encryption.

wscontextcontent

Encrypts the content in the WS-Context header for the SOAP header. For more information, see Propagating work area context over Web services.

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 nonce for confidentiality. When you select the message part, you are encrypting the message part in addition to the parent element of the nonce.

10. 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 1047
- “Adding time stamps for confidentiality to generator security constraints with an XPath expression” on page 1053

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>pZpVL6Rs6zhvu8UrC7TH3BA2zv0dpPpLeHnwH0dCpmdc7ETz1tUHDdXLFxy143
      nYu91MxpzspWt1rWx2Lx9vFGRIfb1RSX51EpV8+0LvezvhJYY/cbTA04mTMUzCfv28v2TI09AZ
      Q4TjII4u+cPeh5f0prBVK1E5hLTq14QMcf/rq9h+tttrJbR7ub3AUgIVo42ucQs5HZbaDiJxm
      dSuFboBq141v1Ep24ZfeoB/p7aHzyeWy7pYt00bshpks/oBw0/78vxSk1VJKu4sUseFvZa+B7s
      ciFneeNnNuRCqB2JXc/vtH8313AELUZg60ehd4vqvXkyuvSLoHZ/kKnF/A5c+BP5Bo1pgvwmDE
      eJItQ5a7L0KkTavLuc2WgtVo1947fnNGm2TN4C6U/cp9ERT7jAB9Lr/1v/8ZqPZYmssyME4pGe
      SWLy232WrPvk6HEu96GHfRt+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 1072.

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 an enterprise application EAR file" in the Application Server Toolkit documentation.

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.
2. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > J2EE**.
3. **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 must 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.
4. **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 must 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.
5. Expand the Confidentiality section. Confidentiality refers to encryption and 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 1057.
6. 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.
7. 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.

Do not use this transform algorithm if you want your configured application to be compliant with the Basic Security Profile (BSP). Instead use <http://www.w3.org/2002/06/xmldsig-filter2> to ensure compliance.

- 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 1047
- “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 1049
- “Adding the nonce for confidentiality to generator security constraints with an XPath expression” on page 1055

8. 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 1072.

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 an enterprise application EAR file” in the Application Server Toolkit documentation.

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.

2. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > J2EE**.
3. **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 must 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.
4. **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 must 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.
5. Expand the Confidentiality section. Confidentiality refers to encryption and 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 1057.
6. 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.
7. 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-xml-19991116` dialect specifies which message part is encrypted using an XPath expression. Do not use this transform algorithm if you want your configured application to be compliant with the Basic Security Profile (BSP). Instead use `http://www.w3.org/2002/06/xmldsig-filter2` to ensure compliance.
 - 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 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.

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 Timestamp for Confidentiality. The selection of message part here is for encrypting a message part in addition to the parent element of the timestamp

10. 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 1049
- “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 1072.

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 an enterprise application EAR file” in the Application Server Toolkit documentation.

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.
2. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > J2EE**.
3. **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 must 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.
4. **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 must 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.
5. Expand the Confidentiality section. Confidentiality refers to encryption and 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 1057.
6. 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.
7. 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.
Do not use this transform algorithm if you want your configured application to be compliant with the Basic Security Profile (BSP). Instead use `http://www.w3.org/2002/06/xmldsig-filter2` to ensure compliance.
 - 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']
```
8. In the Message Parts section, click **Add** and select `http://www.w3.org/TR/1999/REC-xpath-19991116` in the Message parts dialect field. Do not use this transform algorithm if you want your configured application to be compliant with the Basic Security Profile (BSP). Instead use `http://www.w3.org/2002/06/xmldsig-filter2` to ensure compliance.
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 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.

10. 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 1047
- “Adding time stamps for confidentiality to generator security constraints with an XPath expression” on page 1053

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 1072.

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>yMTEy0TA1M...</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 substructures, and attachments of a SOAP message. The specification also requires that when you encrypt parts of a SOAP message, you prepend 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>
```

```

    </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>yMTEy0TA1M...</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 an enterprise application EAR file" in the Application Server Toolkit documentation.
2. Specify which message parts to encrypt. For more information, see "Encrypting message elements in consumer security constraints with keywords" on page 1062 or "Encrypting message elements in consumer security constraints with an XPath expression" on page 1067.
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 1039.

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 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.
2. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > J2EE**.
3. **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 must 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.
4. **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 must 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.

5. 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. **Optional:** Select **Show only FIPS Compliant Algorithms** if you only want the FIPS compliant algorithms to be shown in the encryption method algorithm drop-down lists. Use this option if you expect this application to run on a WebSphere Application Server that has set the **Use the United States Federal Information Processing Standard (FIPS) algorithms** option in the SSL certificate and key management panel of the administrative console for WebSphere Application Server.
- c. 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>.

Do not use the 192-bit key encryption algorithm if you want your configured application to be in compliance with the Basic Security Profile (BSP).

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 1072.

- d. 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-oaep-mgf1p>.

When running with Software Development Kit (SDK) Version 1.4, the list of supported key transport algorithms will not include this one. This algorithm will appear in the list of supported key transport algorithms when running with SDK Version 1.5.

Restriction: This algorithm is not supported when the WebSphere Application Server is running in Federal Information Processing Standard (FIPS) mode.

- 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>.

Do not use the 192-bit key encryption algorithm if you want your configured application to be in compliance with the Basic Security Profile (BSP).

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 1072.

6. 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 1039.
7. 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.
8. 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 1072.

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 an enterprise application EAR file” in the Application Server Toolkit documentation.

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 must 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 must 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 and 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 1057.

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.

username token

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.

signature

Specifies an entire signature. You can encrypt the signature element, ds:Signature, by selecting this message part.

Note: If the value of a ds:DigestValue element in a signature needs to be encrypted, the entire parent ds:Signature element must be encrypted to ensure that the application complies to the Basic Security Profile (BSP). You can use the signature keyword to perform the encryption.

wscontextcontent

Encrypts the content in the WS-Context header for the SOAP header. For more information, see Propagating work area context over Web services.

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 1068
- “Adding a nonce for confidentiality in consumer security constraints with keywords” on page 1065
- “Adding the nonce for confidentiality in consumer security constraints with an XPath expression” on page 1070

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 1060.

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 an enterprise application EAR file" in the Application Server Toolkit documentation.

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.
2. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > J2EE**.
3. **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.
4. **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.
5. 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 1057.
6. 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.
7. 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.

username token

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.

signature

Specifies an entire signature. You can encrypt the signature element, ds:Signature, by selecting this message part.

Note: If the value of a ds:DigestValue element in a signature needs to be encrypted, the entire parent ds:Signature element must be encrypted to ensure that the application complies to the Basic Security Profile (BSP). You can use the signature keyword to perform the encryption.

wscontextcontent

Encrypts the content in the WS-Context header for the SOAP header. For more information, see Propagating work area context over Web services.

8. 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."
9. In the Message Parts section, select the message parts keyword.
10. 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 1027
- "Adding a nonce for integrity in consumer security constraints with an XPath expression" on page 1032

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 1060.

Adding a nonce for confidentiality in consumer security constraints with keywords:

You can 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.

Prior to completing this task, you must import your application into an assembly tool.

For information on how to import your application, see "Importing an enterprise application EAR file" in the Application Server Toolkit documentation.

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.
2. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > J2EE**.

3. **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 must 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.
4. **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 must 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.
5. Expand the Required Confidentiality section. Confidentiality refers to encryption and 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 1057.
6. 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.
7. Under Nonce, click **Add** and select the Nonce dialect. The <http://www.ibm.com/websphere/webservices/wssecurity/dialect-was> dialect specifies the message part that 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 SOAP message body and the body along with the embedded nonce is checked for confidentiality.

username token

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 user name token element along with the embedded nonce is checked for confidentiality.

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 `digestvalue` element along with the embedded nonce is checked for confidentiality.

signature

Specifies an entire signature. You can encrypt the signature element, `ds:Signature`, by selecting this message part.

Note: If the value of a `ds:DigestValue` element in a signature needs to be encrypted, the entire parent `ds:Signature` element must be encrypted to ensure that the application complies to the Basic Security Profile (BSP). You can use the signature keyword to perform the encryption.

wscontextcontent

Encrypts the content in the WS-Context header for the SOAP header. For more information, see Propagating work area context over Web services.

8. 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.
9. In the Message Parts section, select the message parts keyword.
10. 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 1063
- “Adding time stamps for confidentiality in consumer security constraints with an XPath expression” on page 1068

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 1060.

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 an enterprise application EAR file” in the Application Server Toolkit documentation.

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.
2. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > J2EE**.
3. **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 must 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.

4. **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 must 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.
5. Expand the Required Confidentiality section. Confidentiality refers to encryption and 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 1057.
6. 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.
7. 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.
Do not use this transform algorithm if you want your configured application to be compliant with the Basic Security Profile (BSP). Instead use `http://www.w3.org/2002/06/xmldsig-filter2` to ensure compliance.
 - 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 1063
- “Adding time stamps for confidentiality in consumer security constraints with an XPath expression”
- “Adding a nonce for confidentiality in consumer security constraints with keywords” on page 1065
- “Adding the nonce for confidentiality in consumer security constraints with an XPath expression” on page 1070

8. 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 1060.

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 an enterprise application EAR file" in the Application Server Toolkit documentation.

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.
2. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > J2EE**.
3. **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 must 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.
4. **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 must 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.
5. Expand the Required Confidentiality section. Confidentiality refers to encryption and 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 1057.
6. 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.
7. 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.

Do not use this transform algorithm if you want your configured application to be compliant with the Basic Security Profile (BSP). Instead use <http://www.w3.org/2002/06/xmlsig-filter2> to ensure compliance.

- b. Specify the message part that 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.

8. 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 1027
- “Adding a nonce for integrity in consumer security constraints with an XPath expression” on page 1032

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 1060.

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 an enterprise application EAR file” in the Application Server Toolkit documentation.

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.
2. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > J2EE**.
3. **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 must 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.
4. **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 must 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.
5. Expand the Required Confidentiality section. Confidentiality refers to encryption and 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 1057.
6. 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.
7. Click **Add** under the Nonce section of the Required Confidentiality Dialog window. Complete the following steps to specify a nonce dialect and its associated 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.
Do not use this transform algorithm if you want your configured application to be compliant with the Basic Security Profile (BSP). Instead use `http://www.w3.org/2002/06/xmldsig-filter2` to ensure compliance.
 - b. Specify the message part that has an embedded nonce is unencryption 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']
```
8. If you have not specified message parts in Required Confidentiality, 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.
9. In the Message Parts section, select the message parts keyword.
10. 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 1063
- “Adding time stamps for confidentiality in consumer security constraints with an XPath expression” on page 1068

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 1060.

Configuring encryption information for the generator binding with an assembly tool

The encryption information on the generator side is used for encrypting an outgoing SOAP message. You can configure the encryption information for the generator binding by using 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 an enterprise application EAR file" in the Application Server Toolkit documentation.
2. Specify which message parts to encrypt. For more information, see "Encrypting the message elements in generator security constraints with keywords" on page 1045 or "Encrypting the message elements in generator security constraints with an XPath expression" on page 1052.
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 1017.

Complete the following steps to configure the encryption 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.
2. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > J2EE**.
3. **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 must 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.
4. **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 must 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.
5. 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. For example, you might specify gen_encinfo.
 - b. **Optional:** Select **Show only FIPS Compliant Algorithms** if you want only the FIPS compliant algorithms to be shown in the encryption method algorithm drop-down lists. Use this option if you expect this application to run on a WebSphere Application Server that has set the **Use the United States Federal Information Processing Standard (FIPS) algorithms** option in the SSL certificate and key management panel of the administrative console for WebSphere Application Server.
 - c. 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 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 JCE policy file from the following Web site: <http://www.ibm.com/developerworks/java/jdk/security/index.html>.

Do not use the 192-bit key encryption algorithm if you want your configured application to be in compliance with the Basic Security Profile (BSP).

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 1060.

- d. 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-oaep-mgf1p>.

When running with Software Development Kit (SDK) Version 1.4, the list of supported key transport algorithms will not include this one. This algorithm appears in the list of supported key transport algorithms when running with SDK Version 1.5.

Restriction: This algorithm is not supported when the WebSphere Application Server is running in Federal Information Processing Standard (FIPS) mode.

By default, the RSA-OAEP algorithm uses the SHA1 message digest algorithm to compute a message digest as part of the encryption operation. Optionally, you can use the SHA256 or SHA512 message digest algorithm by specifying a key encryption algorithm property. The property name is: `com.ibm.wsspi.wssecurity.enc.rsaoep.DigestMethod`. The property value is one of the following URIs of the digest method:

- <http://www.w3.org/2001/04/xmlenc#sha256>
- <http://www.w3.org/2001/04/xmlenc#sha512>

By default, the RSA-OAEP algorithm uses a null string for the optional encoding octet string for the OAEPParams. You can provide an explicit encoding octet string by specifying a key encryption algorithm property. For the property name, you can specify `com.ibm.wsspi.wssecurity.enc.rsaoep.OAEPparams`. The property value is the base 64-encoded value of the octet string.

Important: You can set these digest method and OAEPParams properties on the generator side only. On the consumer side, these properties are read from the incoming SOAP message.

- 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>.

Do not use the 192-bit key encryption algorithm if you want your configured application to be in compliance with the Basic Security Profile (BSP).

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 1072.

- e. Specify a name in the Key information name field. For example, you might specify `gen_ekeyinfo`.
 - f. 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 1017.
 - g. Select a confidentiality part in the Confidentiality part field. The value in this field specifies the name of the confidentiality element that is encrypted.
6. 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 1060.

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 an enterprise application EAR file” in the Application Server Toolkit documentation.

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 standalone 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.
2. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > J2EE**.
3. **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.
4. **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.
5. 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.

6. 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 PnYnMnDnHnMnS, 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 an enterprise application EAR file" in the Application Server Toolkit documentation.

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 standalone 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.
2. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > J2EE**.
3. **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.
4. **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.
5. 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.
6. 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 PnYnMnDnHnMnS, 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.

Security token

Web services security provides a general-purpose mechanism to associate security tokens with messages for single message authentication. 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.

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 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 6.0.x 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.

Configuring the security token in generator security constraints

You can specify the security token when you configure the request generator or the response generator.

Prior to completing this task, you must import your application into an assembly tool.

See the information about Importing an enterprise application EAR file in the Application Server Toolkit documentation.

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 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.
2. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > J2EE**.
3. **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.
4. **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.
5. Expand the **Security Token** section. The Security Token Dialog window is displayed.
6. 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.
7. 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 1009.

Configuring the security token requirement in consumer security constraints

You can specify the security token when you configure the response consumer or the request consumer.

Prior to completing this task, you must import your application into an assembly tool.

For information on how to import your application, see “Importing an enterprise application EAR file” in the Application Server Toolkit documentation.

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 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.
2. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > J2EE**.
3. **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.
4. **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.
5. Expand the Required Security Token section. The Required Security Token Dialog window is displayed.
6. 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.
7. 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 1035.

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 an enterprise application EAR file” in the Application Server Toolkit documentation.

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.
2. Switch to the J2EE perspective. Click **Window > Open Perspective > J2EE**.
3. **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.
4. **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.
5. Expand the Caller Part section.
6. 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 1023
 - “Signing message elements in consumer security constraints with an XPath expression” on page 1029
 - “Encrypting message elements in consumer security constraints with keywords” on page 1062
 - “Encrypting message elements in consumer security constraints with an XPath expression” on page 1067

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 standalone security token as the caller token.

If a standalone 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 46. URI and Local name combinations

URI	Local name	Description
A namespace URI is not applicable.	Specify <code>http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3</code> as the local name value.	Specifies the name of an X.509 certificate token
A namespace URI is not applicable.	Specify <code>http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509PKIPathv1</code> as the local name value.	Specifies the name of the X.509 certificates in a PKI path
A namespace URI is not applicable.	Specify <code>http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#PKCS7</code> as the local name value.	Specifies a list of X509 certificates and certificate revocation lists (CRL) in a PKCS#7
Specify <code>http://www.ibm.com/websphere/appserver/tokentype/5.0.2</code> as the URI value.	Specify LTPA as the local name value.	Specifies a binary security token that contains an embedded Lightweight Third Party Authentication (LTPA) token.
Specify <code>http://www.ibm.com/websphere/appserver/tokentype</code> as the URI value.	Specify LTPA_PROPAGATION as the local name value.	Specifies a binary security token that contains an embedded propagation token.
Specify the namespace URI value as indicated by the provider.	Specify <code>http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#UsernameToken</code> as the local name value.	Specifies the token type that is configured to perform token validation. This local name is used to remap an incoming security token to a different security token. You can use this local name value in a situation that is similar to the following scenario: A client sends a username token to the server. The custom token consumer on the server uses the security token service to authenticate the user name information. The username token is used to create a new token type such as a Security Assertion Markup Language (SAML) token. You can use the identity from the SAML token for authentication and authorization verification in WebSphere Application Server.

The custom token requires that you specify both the URI and the Local name.

7. **Optional:** Configure identity assertion. For more information, see “Configuring identity assertion”
8. **Optional:** Click **Add** and specify a Trust method property in the Trust method property section, if necessary.
9. **Optional:** Click **Add** and specify an additional property in the Property section, if necessary.
10. Click **OK** to save the configuration changes.

Note: These configurations on the consumer side and the generator side must match.

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 an enterprise application EAR file” in the Application Server Toolkit documentation.

Identity assertion is one of the WebSphere Application Server Version 6.0.x and later 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.
2. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > J2EE**.
3. Expand the Web Services > Services section in the Project Explorer and double-click the name of the Web service.
4. Click the **Extensions** tab and expand the Response Consumer Service Configuration Details > Caller Part section to configure the caller token.
5. 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.
6. 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.
7. **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.

8. Expand the Web Services > Client section in the Project Explorer and double-click the name of the Web service.
9. Click the **WS Extension** tab and expand the Request Generator Configuration > Security Token section.
10. 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.
11. 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.

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 1184.

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 Manage modules, click *URI_name*.
 - c. Under Web Services Security 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.
 - JCERACFKS** Use JCERACFKS if the certificates are stored in a SAF key ring (z/OS only).
 - 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 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. Click **Manage modules > *URI_name***.

3. **Version 6 and later applications** Under Web Services Security 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. **Version 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.

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**, **JCERACFKS** (z/OS only), **JCE4758RACFKS** (z/OS only), **PKCS11KS (PKCS11)**, or **PKCS12KS (PKCS12)**.

The value for this field is **JKS**, **JCEKS**, **JCERACFKS** (z/OS only), **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 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. Click **Manage modules > URI_name**.
3. Under Web Services Security 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. **Version 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.

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 configuration name:

Specifies the name of the key store configuration defined in the keystore settings in secure communications.

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:

Version 5.x or 6 application JKS

Use this option if you are not using Java Cryptography Extensions (JCE).

Version 5.x or 6 application JCEKS

Use this option if you are using Java Cryptography Extensions.

Version 6 and later applications JCERACFKS

Use JCERACFKS if the certificates are stored in a SAF key ring (z/OS only).

Version 6 and later applications 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.

Version 6 and later applications 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

You can configure a collection certificate for the generator bindings 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 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 Manage modules, click *URL_name*.
 - c. Under Web Services Security 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 IBM CertPath 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 Manage modules, click *URL_name*.
 - c. Under Web Services Security 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.

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. Click **Manage modules > *URI_name***.
3. Under Web Services Security 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. **Version 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.

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 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.

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. Click **Manage modules > *URI_name***
3. Under Web Services Security 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. **Version 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.

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. The 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.

This product supports the IBM CertPath 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	IBM CertPath

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. Click **Manage modules > *URI_name***.
3. Under Web Services Security 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. **Version 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**.

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. Click **Manage modules > *URI_name***.
3. Under Web Services Security 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. **Version 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.

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 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 the 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**.

Version 6 and later applications

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. Click **Manage modules > *URI_name***.
3. Under Web Services Security 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.

Version 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 **Manage modules > *URI_name***.

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. Click **Manage modules > *URI_name***.
3. Under Web Services Security 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.

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 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>

```

Nonce, a randomly generated token

Nonce is a randomly generated, cryptographic token that is 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 nonsecure transport, such as HTTP, the token might be intercepted and used in a replay attack. The same password might be reused when the user name 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 user name token in the received 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 user name token, you can specify it in the token generator for the user name token. When the token generator for the user name token is specified, you can select the **Add nonce** option if you want to include nonce in the user name token.

Custom security token propagation

Web services security has the ability to send security tokens in the security header of a SOAP message. These security tokens can be used to sign, verify, encrypt or decrypt message parts. They can also be sent as stand-alone security tokens and set as the caller on the request consumer. *Custom security token propagation* is a feature that is used to propagate these custom security tokens using Web services security.

Web services security supports the Username, X.509 and Lightweight Third-Party Authentication (LTPA) security token types. When you use security token propagation, the propagation token is sent in the wsse:BinarySecurityToken element in the security header of the SOAP message. Web services security uses the same propagation token format as used by the Security attribute propagation feature.

Configuring this option is similar to the configuration for sending and receiving LTPA tokens. The same token generator and token consumer implementations are used, for example:

- com.ibm.wsspi.wssecurity.token.LTPATokenGenerator
- com.ibm.wsspi.wssecurity.token.LTPATokenConsumer

But, the token type Uniform Resource Identifier (URI) and local name for the token generator and token consumer are different. For custom token properties, use the following values:

- **Token type URI:** http://www.ibm.com/websphere/appserver/tokentype
- **Token type local name:** LTPA_PROPAGATION

By default, the custom token propagation uses the following JAAS login configuration entries:

- **Inbound:** WSS_INBOUND
- **Outbound:** WSS_OUTBOUND

You can use the `com.ibm.ws.webservices.wssecurity.constants.jaasConfig` custom property to specify a different JAAS login configuration for the generator. You can do this configuration on the CallbackHandler configuration panel. To specify a different JAAS login configuration on the consumer side, use the JAAS configuration name field in the Token consumer panel.

rrdSecurity.props file

Remote request dispatcher (RRD) supports LTPA and security attribute propagation for Web services security (WS-Security). You can enable token propagation in the `was_install/profiles/profileName/properties/rrdSecurity.props` file.

The `rrdSecurity.props` file contains comments to describe the security attributes.

The following is the format of the `rrdSecurity.props` file. The default values are in bold face type.

- `LTPAPropagation= (True | False)`
- `SecurityAttributePropagation= (True | False)`
- `SSLRequired= (True | False)`

The WS-Security runtime inspects the run as (invocation) subject and propagates the security tokens in the subject. The default setting is to only propagate the LTPA tokens.

Custom security tokens can be passed as attributes of the LTPA tokens. The security attribute propagation support uses the same pluggable JAAS login module as the CSv2 support. The security attribute is not signed or encrypted, therefore, you should not send the attribute in clear text form. You must require SSL to ensure integrity and confidentiality. If SSL is not required, RRD uses the same scheme, such as HTTP or HTTPS, to make the Web services call that the original request used. See Custom security token propagation for more information.

You must also configure the target Web service to validate the LTPA tokens and security attributes.

These links are provided for convenience. Often, the information is not specific to an 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.

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 Manage modules, click *URI_name*.
 - c. Under Web Services Security 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_sigtgen`.
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 1087.

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 47. Additional nonce properties

Property name	Default value	Explanation
<code>com.ibm.ws.wssecurity.config.token.BasicAuth.Nonce.cacheTimeout</code>	600 seconds	Specifies the timeout value, in seconds, for the nonce value that is cached on the server.

Table 47. Additional nonce properties (continued)

Property name	Default value	Explanation
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 panel, 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). For LTPA token propagation, you can use LTPA_PROPAGATION for the value type local name and <http://www.ibm.com/websphere/appserver/tokentype> for the value type 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 one 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.

JCERACFKS

Use JCERACFKS if the certificates are stored in a SAF key ring (z/OS only).

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.

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. Click **Manage modules**.
3. Click the Uniform Resource Identifier (URI).
4. Under Web Services Security 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.

This product 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 provided by this product 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.

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. Click **Manage modules**.
3. Click the Uniform Resource Identifier (URI).
4. Under Web Services Security 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.

This product 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 this product 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.

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. Click **Manage 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**.

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.

The application server provides the following default callback handler implementations:

Version 5.x or 6 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 the 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.

Version 5.x or 6 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.

Version 5.x or 6 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, the 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.

Version 5.x or 6 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, the 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.

Version 6 and later applications

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.

Version 6 and later applications

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.

Version 6 and later applications

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 the 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, the application server sends only 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 this product:

- 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 this product:

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 configuration name:

Specifies the name of the key store configuration defined in the keystore settings in secure communications.

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 product 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

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 the application server. This option provides stronger protection for stored private keys by using Triple DES encryption.

JCERACFKS

Use JCERACFKS if the certificates are stored in a SAF key ring (z/OS only).

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. Click **Manage modules > *URI_name***.
3. Under Web Services Security 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. **Version 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**.

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. Click **Manage modules > *URI_name***.
3. **Version 6 and later applications** 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. Under Web Services Security 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.

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, the 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. Click **Manage modules > URI_file_name**.
3. Under Web Services Security Properties, click **Web services: Client security bindings**.

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. Click **Manage modules > *URI_file_name***.
3. Under Web Services Security Properties, click **Web services: Server security bindings**.

Port:

Specifies the port in which messages are received from the request generator or the request sender. The request generator is the term that is used for Version 6.x applications and the request sender is the term that is used for Version 5.x applications.

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 to be used for signature and encryption information. The key locator information for the generator specifies which key locator implementation is used to locate the key to be 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 Manage modules, click *URI_name*.
 - c. Under Web Services Security 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 6.0.x and later 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 6.0.x and later 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. Keystore 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 keystore password, location, and type.

- a. Specify a password in the keystore **Password** field. This password is used to access the keystore file.
- b. Specify the location of the keystore file in the keystore **Path** field.
- c. Select a keystore type from the **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.

JCERACFKS

Use JCERACFKS if the certificates are stored in a SAF key ring (z/OS only).

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 **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.

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. Click **Manage modules > *URI_name***.
3. Under Web Services Security 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. **Version 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 6.0.x. or later 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**.

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. This product 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. Click **Manage modules > *URI_name***.
3. Under Web Services Security 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. **Version 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.

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.

Version 6 and later applications

Key locators that are associated with Version 6 and later applications must implement the `com.ibm.wsspi.wssecurity.keyinfo.KeyLocator` interface. This product provides the following default key locator class implementations for Version 6 and later 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.

Version 5.x application

Key locators that are associated with Version 5.x applications must implement the `com.ibm.wsspi.wssecurity.config.KeyLocator` interface. This product 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 configuration name:

Specifies the name of the key store configuration defined in the keystore settings in secure communications.

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.

JCERACFKS

Use JCERACFKS if the certificates are stored in a SAF key ring (z/OS only).

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 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 delete a property that you specified previously.

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 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 > New**.

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.
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/xml#dsig#' and local-name()='Signature'])	This property is used with this algorithm: http://www.w3.org/TR/1999/REC-xpath-19991116

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 Manage modules, click *URI_name*.

- c. Under Web Services Security 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#">
  <wss:SecurityTokenReference>
    <wss:KeyIdentifier ValueType="wss:X509v3">/62wX0...</wss:KeyIdentifier>
  </wss: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>
  <wss:SecurityTokenReference>
    <wss:Reference URI="#mytoken" />
  </wss: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>
  <wss:SecurityTokenReference>
    <wss:Embedded wsu:Id="tok1" />
    ...
  </wss:Embedded>
  </wss: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 1115
 - “Configuring the key locator for the consumer binding on the application level” on page 1170
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 1098.
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 Lightweight Third-Party Authentication token. When you specify a value type local name of LTPA, you must also specify a namespace URI of <http://www.ibm.com/websphere/appserver/tokentype/5.0.2>.

LTPA_PROPAGATION

Lightweight Third-Party Authentication propagation token. When you specify a value type local name of LTPA_PROPAGATION, you must also specify a namespace URI of <http://www.ibm.com/websphere/appserver/tokentype>.

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.

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 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.

Note: This option is available on the application level for Version 6 and later applications.

1. Click **Applications > Enterprise applications > *application_name***.
2. Click **Manage modules > *URI_name***.
3. Under Web Services Security 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**.

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 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.

Note: This option is available on the application level for Version 6.0.x applications.

1. Click **Applications > Enterprise applications > application_name**.
2. Under Related items, click **Manage 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.

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.

This product supports the following types of key information. Each type of key information is described in Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)

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: X.509 Certificate Token Profile Version 1.0. The other types are described in Web Services Security: SOAP Message Security 1.0 (WS-Security 2004).

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	<ol style="list-style-type: none">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 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
Default consumer binding	Server level	<ol style="list-style-type: none"> 1. Click Servers > Application servers > <i>server_name</i>. 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 key locator or click the name of a configured key locator to modify its configuration.
Request sender binding	Application level	<ol style="list-style-type: none"> 1. Click Applications > Enterprise applications > <i>application_name</i>. 2. Under Related items, click Manage modules > <i>URI_name</i>. 3. Click Web services: Client security bindings. Under Request sender binding, click Edit. 4. Under Additional properties, click Key locators. 5. Click New to create a new key locator or click the name of a configured key locator to modify its configuration.
Response receiver binding	Application level	<ol style="list-style-type: none"> 1. Click Applications > Enterprise applications > <i>application_name</i>. 2. Under Related items, click Manage modules > <i>URI_name</i>. 3. Click Web services: Client security bindings. Under Response receiver binding, click Edit. 4. Under Additional properties, click Key locators. 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	<ol style="list-style-type: none"> 1. Click Applications > Enterprise applications > <i>application_name</i>. 2. Under Related items, click Manage modules > <i>URI_name</i>. 3. Click Web services: Server security bindings. Under Request receiver binding, click Edit. 4. Under Additional properties, click Key locators. 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	<ol style="list-style-type: none"> 1. Click Applications > Enterprise applications > <i>application_name</i>. 2. Under Related items, click Manage modules > <i>URI_name</i>. 3. Click Web services: Server security bindings. Under Response sender binding, click Edit. 4. Under Additional properties, click Key locators. 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	<ol style="list-style-type: none"> 1. Click Applications > Enterprise applications > <i>application_name</i>. 2. Under Related items, click Manage modules > <i>URI_name</i>. 3. Click Web services: Client security bindings. Under Request generator (sender) binding, click Edit. 4. Under Additional properties, click Key locators. 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	<ol style="list-style-type: none"> 1. Click Applications > Enterprise applications > <i>application_name</i>. 2. Under Related items, click Manage modules > <i>URI_name</i>. 3. Click Web services: Client security bindings. Under Response consumer (receiver) binding, click Edit custom. 4. Under Additional properties, click Key locators. 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	<ol style="list-style-type: none"> 1. Click Applications > Enterprise applications > <i>application_name</i>. 2. Under Related items, click Manage modules > <i>URI_name</i>. 3. Click Web services: Server security bindings. Under Request consumer (receiver) binding, click Edit custom. 4. Under Additional properties, click Key locators. 5. Click New to create a new key locator or click the name of a configured key locator to modify its configuration.
Response generator (sender) binding	Application level	<ol style="list-style-type: none"> 1. Click Applications > Enterprise applications > <i>application_name</i>. 2. Under Related items, click Manage modules > <i>URI_name</i>. 3. Click Web services: Server security bindings. Under Response generator (sender) binding, click Edit custom. 4. Under Additional properties, click Key locators. 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 6.0.x applications.

Binding name	Cell level, server level, or application level	Path
Default generator binding	Server level	<ol style="list-style-type: none"> 1. Click Servers > Application servers > <i>server_name</i>. 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 key locator or click the name of a configured key locator to modify its configuration.
Request generator (sender) binding	Application level	<ol style="list-style-type: none"> 1. Click Applications > Enterprise applications > <i>application_name</i>. 2. Under Related items, click Manage modules > <i>URI_name</i>. 3. Click Web services: Client security bindings. Under Request generator (sender) binding, click Edit. 4. Under Additional properties, click Key locators. 5. Click New to create a new key locator or click the name of a configured key locator to modify its configuration.
Response generator (sender) binding	Application level	<ol style="list-style-type: none"> 1. Click Applications > Enterprise applications > <i>application_name</i>. 2. Under Related items, click Manage modules > <i>URI_name</i>. 3. Click Web services: Server security bindings. Under Response generator (sender) binding, click Edit custom. 4. Under Additional properties, click Key locators. 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.

The 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	<ol style="list-style-type: none"> 1. Click Servers > Application servers > <i>server_name</i>. 2. Under Security, click Web services: Default bindings for Web services security. 3. Under Default generator bindings, click Token generator. 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	<ol style="list-style-type: none"> 1. Click Servers > Application servers > <i>server_name</i>. 2. Under Security, click Web services: Default bindings for Web services security. 3. Under Default consumer bindings, click Token consumer. 4. Click New to create a new token consumer or click the name of a configured token consumer to modify its configuration.
Request generator (sender) binding	Application level	<ol style="list-style-type: none"> 1. Click Applications > Enterprise applications > <i>application_name</i>. 2. Under Related items, click Manage modules > <i>URI_name</i>. 3. Click Web services: Client security bindings. Under Request generator (sender) binding, click Edit custom. 4. Under Additional properties, click Token generators. 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	<ol style="list-style-type: none"> 1. Click Applications > Enterprise applications > <i>application_name</i>. 2. Under Related items, click Manage modules > <i>URI_name</i>. 3. Click Web services: Client security bindings. Under Response consumer (receiver) binding, click Edit custom. 4. Under Required properties, click Token consumers. 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
Request consumer (receiver) binding	Application level	<ol style="list-style-type: none"> 1. Click Applications > Enterprise applications > <i>application_name</i>. 2. Under Related items, click Manage modules > <i>URI_name</i>. 3. Click Web services: Server security bindings. Under Request consumer (receiver) binding, click Edit custom. 4. Under Required properties, click Token consumers. 5. Click New to create a new token consumer or click the name of a configured token consumer to modify its configuration.
Response generator (sender) binding	Application level	<ol style="list-style-type: none"> 1. Click Applications > Enterprise applications > <i>application_name</i>. 2. Under Related items, click Manage modules > <i>URI_name</i>. 3. Click Web services: Server security bindings. Under Response generator (sender) binding, click Edit custom. 4. Under Additional properties, click Token generators. 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. This product 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. This product 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.

This product provides the following predefined value type URIs for the Lightweight Third Party Authentication (LTPA) token:

- <http://www.ibm.com/websphere/appserver/tokentype>
- <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. This product 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>

Lightweight Third Party Authentication (LTPA)

LTPA_PROPAGATION

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 LTPA token propagation, the value type local name is LTPA_PROPAGATION. If you enter LTPA_PROPAGATION for the local name, you must specify the <http://www.ibm.com/websphere/appserver/tokentype> 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 **Value type 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 must 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 Manage modules, click *URL_name*.
 - c. Under Web Services Security 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#hmac-sha1>
 - <http://www.w3.org/2000/09/xmldsig#dsa-sha1>Do not use this algorithm if you want the configured application to be compliant with the Basic Security Profile (BSP). Any `ds:SignatureMethod/@Algorithm` element in a SIGNATURE based on a symmetric key must have a value of <http://www.w3.org/2000/09/xmldsig#rsa-sha1> or <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 following algorithms:

 - <http://www.w3.org/2000/09/xmlldsig#sha1>
 - <http://www.w3.org/2001/04/xmlenc#sha256>
 - <http://www.w3.org/2001/04/xmlenc#sha512>
 - 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>

Do not use this transform algorithm if you want your configured application to be compliant with the Basic Security Profile (BSP). Instead use <http://www.w3.org/2002/06/xmlldsig-filter2> to ensure compliance.
 - <http://www.w3.org/2002/06/xmlldsig-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 **Apply**.
10. **Optional:** Determine whether to disable the Inclusive namespace prefix list. The Exclusive XML Canonicalization Version 1.0 specification recommends that you include all of the namespace declarations that correspond to the namespace prefix in the canonicalization form. For security reasons, WebSphere Application Server, by default, includes the prefix in the digital signature for Web services security. However, some implementations of Web services security cannot handle this prefix list. WebSphere Application Server can handle digitally signed messages that either contain or do not contain the prefix list. If you experience a signature validation failure when a signed Simple Object Access Protocol (SOAP) message is sent and you are using another vendor in your environment, it is highly recommended that you check with their Web site for a possible fix to their implementation before you disable this property. To disable this property, complete the following steps:
 - a. Under Additional properties, click **Canonicalization method properties > New**.
 - b. In the Property name field, enter the `com.ibm.wsspi.wssecurity.dsig.inclusiveNamespaces` property.
 - c. In the Property value field, enter the `false` value.
 - d. Click **OK**.

You can set this property for both the request generator and the response generator configurations.
11. 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.

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. Click **Manage modules** > *URI_name*.
3. **Version 6 and later applications** Under Web Services Security 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. **Version 6 and later applications** Under Required properties, click **Signing information**.
5. **Version 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. **Version 5.x application** Under Additional properties, click **Signing information**.
7. Click **New** to create a signing parameter. Click **Delete** to delete a signing parameter.

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 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. Click **Manage modules > *URI_name***.
3. Under Web Services Security 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. **Version 6 and later applications** Under Required properties, click **Signing information**.
5. **Version 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. **Version 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.

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:

- **Version 5.x or 6 application** <http://www.w3.org/2000/09/xmlnsig#rsa-sha1>
- **Version 5.x or 6 application** <http://www.w3.org/2000/09/xmlnsig#dsa-sha1>

Do not use this algorithm if you want the configured application to be compliant with the Basic Security Profile (BSP). Any ds:SignatureMethod/@Algorithm element in a signature based on a symmetric key must have a value of <http://www.w3.org/2000/09/xmlnsig#rsa-sha1> or <http://www.w3.org/2000/09/xmlnsig#hmac-sha1>.

- **Version 6 and later applications** <http://www.w3.org/2000/09/xmlnsig#hmac-sha1>

For Version 6.0.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 6.0.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/xmlsig#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 6.0.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.

This product 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, the application server uses the `keyinfo` value, by default.

The Key information signature type field is available for the token consumer binding.

For Version 6.0.x.x applications, this 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 the 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	<ol style="list-style-type: none">1. Click Servers > Application Servers > <i>server_name</i>.2. Under Security, click Web services: Default bindings for Web services security.3. Under Default generator binding, click Key information.
Default consumer binding	Server level	<ol style="list-style-type: none">1. Click Servers > Application Servers > <i>server_name</i>.2. Under Security, click Web services: Default bindings for Web services security.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

The application server searches for trust anchor configurations on the application and server levels and lists the configurations in this menu.

Version 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	<ol style="list-style-type: none"> 1. Click Servers > Application servers > <i>server_name</i>. 2. Under Security, click Web services: Default bindings for Web services security. 3. Under Additional properties, click Trust anchors > New.
Default consumer binding	Server level	<ol style="list-style-type: none"> 1. Click Servers > Application servers > <i>server_name</i>. 2. Under Security, click Web services: Default bindings for Web services security. 3. Under Additional properties, click Trust anchors > New.
Response receiver	Application level for Version 5.x applications	<ol style="list-style-type: none"> 1. Click Applications > Enterprise applications > <i>application_name</i>. 2. Click Manage modules > <i>URI_name</i>. 3. Click Web services: Client security bindings. 4. Under the Response receiver binding, click Edit. 5. Under Additional properties, click Trust anchors > New.
Request receiver	Application level for Version 5.x applications	<ol style="list-style-type: none"> 1. Click Applications > Enterprise applications > <i>application_name</i>. 2. Click Manage modules > <i>URI_name</i>. 3. Click Web services: Server security bindings. 4. Under the Request receiver binding, click Edit. 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 1086.

Certificate store

The 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	<ol style="list-style-type: none"> 1. Click Servers > Application servers > <i>server_name</i>. 2. Under Security, click Web services: Default bindings for Web services security. 3. Under Additional properties, click Collection certificate store > New.
Default consumer binding	Server level	<ol style="list-style-type: none"> 1. Click Servers > Application servers > <i>server_name</i>. 2. Under Security, click Web services: Default bindings for Web services security. 3. Under Additional properties, click Collection certificate store > New.
Response receiver	Application level for Version 5.x applications	<ol style="list-style-type: none"> 1. Click Applications > Enterprise applications > <i>application_name</i>. 2. Click Manage modules > <i>URI_name</i>. 3. Click Web services: Client security bindings. 4. Under the Response receiver binding, click Edit. 5. Under Additional properties, click Collection certificate store > New.
Request receiver	Application level for Version 5.x applications	<ol style="list-style-type: none"> 1. Click Applications > Enterprise applications > <i>application_name</i>. 2. Click Manage modules > <i>URI_name</i>. 3. Click Web services: Server security bindings. 4. Under the Request receiver binding, click Edit. 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 1090.

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 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. Click **Manage modules** > *URI_name*.
3. Under Web Services Security 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**.

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 Uniform Resource Identifier (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 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.

Note: Part references are available through the administrative console using Version 6.0.x.x applications only.

1. Click **Applications > Enterprise applications** > *application_name*.

2. Click **Manage modules** > *URI_name*.
3. Under Web Services Security 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.

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 Uniform Resource Identifier (URI) of the digest method that is used for the signed part that is specified by the part reference.

This product provides the following predefined algorithm URIs:

- <http://www.w3.org/2000/09/xmlsig#sha1>
- <http://www.w3.org/2001/04/xmlenc#sha256>
- <http://www.w3.org/2001/04/xmlenc#sha512>

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 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_name***.
5. Under Additional properties, click **Transforms**.

Version 6 and later applications

To view this administrative console page for the application level, complete the following steps.

Note: This option is available for Version 6 and later applications only.

1. Click **Applications > Enterprise applications > *application_name***.
2. Click **Manage Modules > *URI_name***.
3. Under Web Services Security 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**.

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.

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 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_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.

Version 6 and later applications

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. Click **Manage modules** > *URI_name*.
3. Under Web Services Security 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.

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.

This product 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/xmlsig#' and local-name()='Signature'])`

Note: Do not use this transform algorithm if you want your configured application to be compliant with the Basic Security Profile (BSP). Instead use <http://www.w3.org/2002/06/xmldsig-filter2> to ensure compliance.

<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.dsig.XPath2Expression_2 = [XPath expression#1]
com.ibm.wsspi.wssecurity.dsig.XPath2Filter_1 = [intersect]
com.ibm.wsspi.wssecurity.dsig.XPath2Order_1 = [1]
com.ibm.wsspi.wssecurity.dsig.XPath2Expression_2 = [XPath expression#2]
com.ibm.wsspi.wssecurity.dsig.XPath2Filter_2 = [subtract]
com.ibm.wsspi.wssecurity.dsig.XPath2Order_2 = [2]
```

<http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform>

This algorithm specifies the enhancements to SOAP messaging that provide message integrity and confidentiality.

<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

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 Manage modules, click *URI_name*.
 - c. Under Web Services Security 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>.

Do not use the 192-bit key encryption algorithm if you want your configured application to be in compliance with the Basic Security Profile (BSP).

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-oaep-mgf1p>.

When running with Software Development Kit (SDK) Version 1.4, the list of supported key transport algorithms does not include this one. This algorithm appears in the list of supported key transport algorithms when running with SDK Version 1.5.

Restriction: This algorithm is not supported when the WebSphere Application Server is running in Federal Information Processing Standard (FIPS) mode.

By default, the RSA-OAEP algorithm uses the SHA1 message digest algorithm to compute a message digest as part of the encryption operation. Optionally, you can use the SHA256 or SHA512 message digest algorithm by specifying a key encryption algorithm property. For the property name, you can specify `com.ibm.wsspi.wssecurity.enc.rsaoaep.DigestMethod`. The property value is one of the following URIs of the digest method:

- <http://www.w3.org/2001/04/xmlenc#sha256>
- <http://www.w3.org/2001/04/xmlenc#sha512>

By default, the RSA-OAEP algorithm uses a null string for the optional encoding octet string for the OAEPParams. You can provide an explicit encoding octet string by specifying a key encryption algorithm property. For the property name, you can specify `com.ibm.wsspi.wssecurity.enc.rsaoaep.OAEPparams`. The property value is the base 64-encoded value of the octet string.

Important: You can set these digest method and OAEPParams properties on the generator side only. On the consumer side, these properties are read from the incoming Simple Object Access Protocol (SOAP) message.

- 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>.

Do not use the 192-bit key encryption algorithm if you want your configured application to be in compliance with the Basic Security Profile (BSP).

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 1121.
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.

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. Click **Manage modules > *URI_name***.
3. Under Web Services Security 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. **Version 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**.

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 **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 Modules, click **Module update > module_name**.
3. Under Web Services Security 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. **Version 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.

Encryption information name:

Specifies the name for the encryption information.

Data type String

Data encryption algorithm:

Specifies the algorithm Uniform Resource Identifier (URI) of the data encryption method.

The following algorithms are supported:

- <http://www.w3.org/2001/04/xmlenc#tripleledes-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 1155.
- <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 1155.

Do not use the 192-bit data encryption algorithm if you want your configured application to be in compliance with the Basic Security Profile (BSP).

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	<ol style="list-style-type: none"> 1. Click Servers > Application servers > <i>server_name</i>. 2. Under Security, click Web services: Default bindings for Web services security. 3. Under Additional properties, click Key locators.
Default consumer binding	Server level	<ol style="list-style-type: none"> 1. Click Servers > Application servers > <i>server_name</i>. 2. Under Security, click Web services: Default bindings for Web services security. 3. Under Additional properties, click Key locators.
Request sender	Application level	<ol style="list-style-type: none"> 1. Click Applications > Enterprise applications > <i>application_name</i>. 2. Click Manage modules > <i>URI_name</i>. 3. Click Web services: Client security bindings. Under Request sender binding, click Edit. 4. Under Additional properties, click Key locators.
Request receiver	Application level	<ol style="list-style-type: none"> 1. Click Applications > Enterprise applications > <i>application_name</i>. 2. Click Manage modules > <i>URI_name</i>. 3. Click Web services: Server security bindings. Under Request receiver binding, click Edit. 4. Under Additional properties, click Key locators.
Response sender	Application level	<ol style="list-style-type: none"> 1. Click Applications > Enterprise applications > <i>application_name</i>. 2. Click Manage modules > <i>URI_name</i>. 3. Click Web services: Server security bindings. Under Response sender binding, click Edit. 4. Under Additional properties, click Key locators.
Response receiver	Application level	<ol style="list-style-type: none"> 1. Click Applications > Enterprise applications > <i>application_name</i>. 2. Click Manage modules > <i>URI_name</i>. 3. Click Web services: Client security bindings. Under Response receiver binding, click Edit. 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 the application server:

- <http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p>.

When running with Software Development Kit (SDK) Version 1.4, the list of supported key transport algorithms does not include this one. This algorithm appears in the list of supported key transport algorithms when running with Software Development Kit (SDK) Version 1.5 or later.

By default, the RSA-OAEP algorithm uses the SHA1 message digest algorithm to compute a message digest as part of the encryption operation. Optionally, you can use the SHA256 or SHA512 message digest algorithm by specifying a key encryption algorithm property. The property name is:

`com.ibm.wsspi.wssecurity.enc.rsaoaep.DigestMethod`. The property value is one of the following URIs of the digest method:

- <http://www.w3.org/2001/04/xmlenc#sha256>
- <http://www.w3.org/2001/04/xmlenc#sha512>

By default, the RSA-OAEP algorithm uses a null string for the optional encoding octet string for the `OAEPparams`. You can provide an explicit encoding octet string by specifying a key encryption algorithm property. For the property name, you can specify `com.ibm.wsspi.wssecurity.enc.rsaoaep.OAEPparams`. The property value is the base 64-encoded value of the octet string.

Important: You can set these digest method and `OAEPparams` properties on the generator side only. On the consumer side, these properties are read from the incoming Simple Object Access Protocol (SOAP) message.

- 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>

Do not use the 192-bit data encryption algorithm if you want your configured application to be in compliance with the Basic Security Profile (BSP).

- <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 one of the following sets of steps:

- For application server platforms using IBM Developer Kit, Java Technology Edition Version 1.4.2, including the AIX, Linux, and Windows platforms, complete the following steps to obtain unlimited jurisdiction policy files:
 1. Go to the following Web site: <http://www.ibm.com/developerworks/java/jdk/security/index.html>
 2. Click **Java 1.4.2**
 3. Click **IBM SDK Policy files**.
The Unrestricted JCE Policy files for SDK 1.4 Web site is displayed.
 4. Enter your user ID and password or register with IBM to download the policy files. The policy files are downloaded onto your machine.
- For application server platforms using the Sun-based Java Development Kit (JDK) Version 1.4.2, including the Solaris environments and the HP-UX platform, complete the following steps to obtain unlimited jurisdiction policy files:

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 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.

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 one of the following sets of steps:

- For application server platforms using IBM Developer Kit, Java Technology Edition Version 5, 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 5**
 3. Click **IBM SDK Policy files**.
The Unrestricted JCE Policy files for SDK 5 Web site is displayed.
 4. Enter your user ID and password or register with IBM to download the policy files. The policy files are downloaded onto your machine.
- For application server platforms using the Sun-based Java Development Kit (JDK) Version 5, 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/1.5.0/download.jsp>
 2. Click **Archive area**.
 3. Locate the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 1.5.1 information and click **Download**. The policy 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.

Version 6 and later applications

You must specify either one or no encryption key configurations for the request generator and response generator bindings.

Version 6 and later applications

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	<ol style="list-style-type: none">1. Click Servers > Application servers > server_name.2. Under Security, click Web services: Default bindings for Web services security.3. Under Default generator binding, click Key information.
Default consumer binding	Server level	<ol style="list-style-type: none">1. Click Servers > Application servers > server_name.2. Under Security, click Web services: Default bindings for Web services security.3. Under Default consumer binding, click Key information.
Request generator (sender) binding	Application level	<ol style="list-style-type: none">1. Click Applications > Enterprise applications > application_name.2. Click Manage modules > URI_name.3. Under Web Services Security Properties, click Web services: Client security bindings.4. Under Request generator (sender) binding, click Edit custom.5. Under Required properties, click Key information.
Response generator (sender) binding	Application level	<ol style="list-style-type: none">1. Click Applications > Enterprise applications > application_name.2. Click Manage modules > URI_name.3. Under Web Services Security Properties, click Web services: Server security bindings.4. Under Response generator (sender) binding, click Edit custom.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:
 - Click **Manage modules > *URI_file_name* > Web Services: Client Security Bindings**. Under Request sender binding, click **Edit**. Under Web Service Security Properties, click **Encryption Information**.
 - Click **Manage modules > *URI_file_name* > Web Services: Server Security Bindings**. Under Response sender binding, click **Edit**. Under Web Service Security Properties, click **Encryption Information**.
2. Select **None** or **Dedicated encryption information**. The 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 topic.

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. Click **Manage modules > *URI_name***.
3. Under Web Service Security 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-oaep-mgf1p>.

When running with Software Development Kit (SDK) Version 1.4, the list of supported key transport algorithms does not include this one. This algorithm appears in the list of supported key transport algorithms when running with JDK 1.5 or later.

By default, the RSA-OAEP algorithm uses the SHA1 message digest algorithm to compute a message digest as part of the encryption operation. Optionally, you can use the SHA256 or SHA512 message digest algorithm by specifying a key encryption algorithm property. The property name is: `com.ibm.wsspi.wssecurity.enc.rsaoaep.DigestMethod`. The property value is one of the following URIs of the digest method:

- <http://www.w3.org/2001/04/xmlenc#sha256>
- <http://www.w3.org/2001/04/xmlenc#sha512>

By default, the RSA-OAEP algorithm uses a null string for the optional encoding octet string for the `OAEPParams`. You can provide an explicit encoding octet string by specifying a key encryption algorithm property. For the property name, you can specify `com.ibm.wsspi.wssecurity.enc.rsaoaep.OAEPParams`. The property value is the base 64-encoded value of the octet string.

Important: You can set these digest method and `OAEPParams` properties on the generator side only. On the consumer side, these properties are read from the incoming Simple Object Access Protocol (SOAP) message.

- 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.

Do not use the 192-bit key encryption algorithm if you want your configured application to be in compliance with the Basic Security Profile (BSP).

- <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 one of the following sets of steps:

- For application server platforms using IBM Developer Kit, Java Technology Edition Version 1.4.2, including the AIX, Linux, and Windows platforms, complete the following steps to obtain unlimited jurisdiction policy files:
 1. Go to the following Web site: <http://www.ibm.com/developerworks/java/jdk/security/index.html>
 2. Click **Java 1.4.2**
 3. Click **IBM SDK Policy files**.

The Unrestricted JCE Policy files for SDK 1.4 Web site is displayed.

4. Enter your user ID and password or register with IBM to download the policy files. The policy files are downloaded onto your machine.
- For application server platforms using the Sun-based Java Development Kit (JDK) Version 1.4.2, including the Solaris environments and the HP-UX platform, complete the following steps to obtain unlimited jurisdiction policy files:
 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>

Do not use the 192-bit data encryption algorithm if you want your configured application to be in compliance with the Basic Security Profile (BSP).

- <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

You can configure trust anchors for the consumer binding at the application level.

This article 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 1184.

You can configure a trust anchor for the trust anchor using an assembly tool or the administrative console. This article 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 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 Manage modules, click *URI_name*.
 - c. Under Web Services Security 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.

JCERACFKS Use JCERACFKS if the certificates are stored in a SAF key ring (z/OS only).

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.

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 Manage modules, click *module_name*.
 - c. Under Web services security properties, you can access the collection certificate store information for the response consumer and request consumer 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 Response consumer (receiver) binding, click **Edit custom**.
 - d. 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 IBM CertPath 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.

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 and later 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
  8Xg26havepvmSJ8XxiACMihTJuh1t3ufsrjbfQJ0qh5VcRvI+AKEaNmEgEV65jUYAC9
  C/iwBBWk5U/6DIk7LfXcTT0ZPAd+3D3nCS0f+6tnqMou8EG9mtMeTKccz/pJVTZjaRSo
  msu0sewsOKf1/WPsjW0bR/2g3NaVvBy18V1TFBpUbGFVGgzHRjBKAGo+ctk180n1VLIk
  TUjt/XdYvEp0r6QoddGi4okjDGPpyoDxcvKZnReXww5Usoq1pfXwN4KG9as=
</wsse:BinarySecurityToken>
</wsse:Security>
</soapenv:Header>
```

As shown in the example, the token is Base64Binary encoded.

Configuring token consumer on the application level

You can specify the token consumer on the application level. The token consumer 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 Manage modules, click *URI_name*.
 - c. Under Web Services Security 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

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 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 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 979.

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.

Table 48. Uniform Resource Identifier (URI) and Local name combinations

URI	Local name	Description
A namespace URI is not applicable.	Specify <code>http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3</code> as the local name value.	Specifies the name of an X.509 certificate token

Table 48. Uniform Resource Identifier (URI) and Local name combinations (continued)

URI	Local name	Description
A namespace URI is not applicable.	Specify <code>http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509PKIPathv1</code> as the local name value.	Specifies the name of the X.509 certificates in a PKI path
A namespace URI is not applicable.	Specify <code>http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#PKCS7</code> as the local name value.	Specifies a list of X509 certificates and certificate revocation lists (CRL) in a PKCS#7
Specify <code>http://www.ibm.com/websphere/appserver/tokentype/5.0.2</code> as the URI value.	Specify LTPA as the local name value.	Specifies a binary security token that contains an embedded Lightweight Third Party Authentication (LTPA) token.
Specify <code>http://www.ibm.com/websphere/appserver/tokentype</code> as the URI value.	Specify LTPA_PROPAGATION as the local name value.	Specifies a binary security token that contains an embedded propagation token.
Specify the namespace URI value as indicated by the provider.	Specify <code>http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#UsernameToken</code> as the local name value.	<p>Specifies the token type that is configured to perform token validation. This local name is used to remap an incoming security token to a different security token. You can use this local name value in a situation that is similar to the following scenario:</p> <p>A client sends a username token to the server. The custom token consumer on the server uses the security token service to authenticate the user name information. The username token is used to create a new token type such as a Security Assertion Markup Language (SAML) token. You can use the identity from the SAML token for authentication and authorization verification in WebSphere Application Server.</p>

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 > Secure administrative, applications, and infrastructure**. Under Authentication, click **Java Authentication and Authorization Service** and click 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 the `com.ibm.CSIOutboundPropagationEnabled` property is true. This property is set in the CSiv2 authentication panel.

To access the panel, click **Security > Secure administrative, applications, and infrastructure**. Under Authentication, click **RMI/IIOP security > CSiv2 Outbound authentication**. To set the `com.ibm.CSIOutboundPropagationEnabled` property, select **Security attribute propagation**. To modify this JAAS login configuration, see the JAAS - System logins panel.

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 6.0.x 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.

system.WSS_INBOUND

This selection specifies the login configuration for inbound or consumer requests for security token propagation using Web services security. To modify this configuration, see the JAAS configuration panel for system logins.

system.WSS_OUTBOUND

This selection specifies the login configuration for outbound or generator requests for security token propagation using Web services security. 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.

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. Click **Manage modules**.
3. Click the Uniform Resource Identifier (URI).
4. Under Web Services Security 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.

This product 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 this product 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

Information type	Required or 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.

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. Click **Manage modules**.
3. Click the Uniform Resource Identifier (URI).
4. Under Web Services Security 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.

This product 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 this product 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
Key locators	Optional
Collection certificate store	Optional

Information type	Required or 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.

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. The key locator implementation locates the key to be 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 Manage modules, click *URI_name*.
 - c. Under Web Services Security 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 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 6.0.x 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 6.0.x 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 keystore **Password** field. This password is used to access the keystore file.
 - b. Specify the location of the keystore file in the keystore **Path** field.
 - c. Select a keystore type from the keystore **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.

JCERACFKS

Use JCERACFKS if the certificates are stored in a SAF key ring (z/OS only).

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.

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 Manage modules, click *URI_name*.
 - c. Under Web Services Security 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">
      /62wX0...
    </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: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 1170.
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 1161.
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.

Configuring the signing information for the consumer binding on the application level

You can configure the signing information for the server-side request consumer and the client-side response consumer bindings at 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 must configure the key information that is referenced by the key information references on the signing information panel within the administrative console.

WebSphere Application Server uses the signing information on the consumer side to verify the integrity of the received 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:port_number/ibm/console` in your Web browser unless you have changed the port number.
2. Click **Applications > Enterprise applications > application_name**.
3. Under Manage modules, click *URI_name*.
4. Under Web Services Security Properties you can access the signing information for the request generator and response generator 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#hmac-sha1`
 - `http://www.w3.org/2000/09/xmldsig#dsa-sha1`Do not use this algorithm if you want the configured application to be compliant with the Basic Security Profile (BSP). Any `ds:SignatureMethod/@Algorithm` element in a signature based on a symmetric key must have a value of `http://www.w3.org/2000/09/xmldsig#rsa-sha1` or `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 algorithms:

 - <http://www.w3.org/2000/09/xmlenc#sha1>
 - <http://www.w3.org/2001/04/xmlenc#sha256>
 - <http://www.w3.org/2001/04/xmlenc#sha512>

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.
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>

Do not use this transform algorithm if you want your configured application to be compliant with the Basic Security Profile (BSP). Instead use <http://www.w3.org/2002/06/xmlenc-filter2> to ensure compliance.
 - <http://www.w3.org/2002/06/xmlenc-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/xmlenc#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.

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 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. Click **Manage modules > *URI_name***.
3. Under Web Services Security 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.

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 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. Click **Manage modules > URI_name**.
3. Under Web Services Security 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.

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	<ol style="list-style-type: none"> 1. Click Servers > Application Servers > server_name. 2. Under Security, click Web services: Default bindings for Web services security. 3. Under Default consumer binding, click Key information.
Response consumer (receiver) binding	Application level	<ol style="list-style-type: none"> 1. Click Applications > Enterprise applications > application_name. 2. Click Manage modules > URI_name. 3. Under Web Services Security Properties, click Web services: Client security bindings. 4. Under Response consumer (receiver) binding, click Edit custom. 5. Under Required properties, click Key information.

Binding name	Cell level, Server level, or application level	Path
Request consumer (receiver) binding	Application level	<ol style="list-style-type: none"> 1. Click Applications > Enterprise applications > application_name. 2. Click Manage modules > URI_name. 3. Under Web Services Security Properties, click Web services: Server security bindings. 4. Under Request consumer (receiver) binding, click Edit custom. 5. Under Required properties, click Key information.

Configuring the encryption information for the consumer binding on the application level

Configure the key information that is referenced in the encryption information panel. For more information, see “Configuring the key information for the consumer binding on the application level” on page 1171.

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 Manage modules, click *URI_name*.
 - c. Under Web Services Security 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>.

Do not use the 192-bit key encryption algorithm if you want your configured application to be in compliance with the Basic Security Profile (BSP).

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-oaep-mgf1p>.

When running with Software Development Kit (SDK) Version 1.4, the list of supported key transport algorithms does not include this one. This algorithm appears in the list of supported key transport algorithms when running with SDK Version 1.5.

Restriction: This algorithm is not supported when the WebSphere Application Server is running in Federal Information Processing Standard (FIPS) mode.

- 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>.

Do not use the 192-bit key encryption algorithm if you want your configured application to be in compliance with the Basic Security Profile (BSP).

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.

Hardware cryptographic device support for Web Services Security

In IBM WebSphere Application Server Version 6.1 or later, Web services security supports the use of cryptographic hardware devices. There are two ways in which to use hardware cryptographic devices with Web services security as described in this article.

Enabling cryptographic operations on hardware devices

You can enable cryptographic operations on hardware devices. The keys that are used can be stored in a Java keystore file; it is not necessary to store them on the hardware device. The decision to use enable cryptographic operations on hardware devices is made at the server level only, not at the application level.

If cryptographic operations on hardware device is enabled, the Web service security run time first attempts to use the hardware device for cryptographic operations. If the attempt to use the hardware device fails, or the algorithm is not supported by the hardware device, the run time uses a software provider from the security providers list.

Enabling this feature might improve the performance depending on the hardware device.

For more information on how to enable cryptographic operations on hardware devices, see “Configuring hardware cryptographic devices for Web Services Security” on page 1181.

Secure keys

Cryptographic keys can be stored on the hardware cryptographic device and never leave the device. These secure keys are confined to the hardware cryptographic device for security considerations rather than performance considerations. The option to select whether to use keys stored in a hardware cryptographic device or a Java keystore file can be made at the application level.

If the keystore reference is specified to be a hardware device configuration, the Web services security run time first attempts to obtain the cryptographic algorithm from the hardware device. If the algorithm is not supported or fails, the run time uses a software provider from the security providers list.

For information on how to enable secure keys, see “Enabling cryptographic keys stored in hardware devices in Web Services Security” on page 1182.

Limitations

The hardware cryptographic device support for Web Services Security currently has the following limitations:

- There is no support for a Web services client running as a J2EE Application Client.
- There is no support for hardware cryptographic devices on iSeries.
- Versions 5.x and 6.0 Web services security applications can run in a Version 6.1 WebSphere Application Server, but they cannot take advantage of the hardware cryptographic support. Only Version 6.1 and later Web services security applications can take advantage of the hardware cryptographic support.

Enabling hardware cryptographic devices for Web Services Security

A cryptographic token is a hardware or software device with a built-in keystore implementation. Cryptographic devices are used to manage certificates stored on the cryptographic tokens. These devices are also called smartcards.

You can enable Web Services Security using cryptographic hardware devices for both Web service clients and Web service providers running in the WebSphere Application Server environment.

Web services security using cryptographic hardware devices is supported for both Web (JavaServer Pages (JSP) or servlet) and Enterprise JavaBeans (EJB) Web service clients. Individual applications have the option to select whether to use keys stored in hardware devices or in a Java keystore file.

There are two ways to enable hardware cryptographic devices for Web service security:

- Enable cryptographic operations on hardware devices for Web services security. For more information, see “Configuring hardware cryptographic devices for Web Services Security.”
- Enable cryptographic keys that are stored in hardware devices for Web services security. For more information, see “Enabling cryptographic keys stored in hardware devices in Web Services Security” on page 1182

Note: Hardware cryptographic devices for Web Services Security are not supported on the Java 2 Platform, Enterprise Edition (J2EE) Application Client on distributed platform.

Related concepts

“Hardware cryptographic device support for Web Services Security” on page 1180

In IBM WebSphere Application Server Version 6.1 or later, Web services security supports the use of cryptographic hardware devices. There are two ways in which to use hardware cryptographic devices with Web services security as described in this article.

Related tasks

“Configuring hardware cryptographic devices for Web Services Security”

You can configure a hardware cryptographic device in Web Service Security. The key for the cryptographic operation can be stored in an ordinary Java key store file, and does not need to be stored on the hardware devices.

“Enabling cryptographic keys stored in hardware devices in Web Services Security” on page 1182

You can enable individual Web service applications to use cryptographic keys stored in hardware devices in Web Services Security.

Configuring hardware cryptographic devices for Web Services Security

You can configure a hardware cryptographic device in Web Service Security. The key for the cryptographic operation can be stored in an ordinary Java key store file, and does not need to be stored on the hardware devices.

You must first configure a hardware cryptographic device using the Secure Sockets Layer (SSL) certificate and key management panels in the administrative console.

1. In the administrative console, click **Servers > Application servers** and then select the server name.
2. Under **Security**, select **Web services: default bindings for Web services security**.
3. Under Cryptographic Hardware, select **Enable cryptographic operations on hardware device** and then specify the name of the hardware cryptographic device configuration name. For more information, see “Configuring a hardware cryptographic keystore” on page 453.
4. Click **OK**.

This procedure configures a hardware cryptographic device for all Web services security applications running on the WebSphere Application Server.

Related concepts

“Hardware cryptographic device support for Web Services Security” on page 1180

In IBM WebSphere Application Server Version 6.1 or later, Web services security supports the use of cryptographic hardware devices. There are two ways in which to use hardware cryptographic devices with Web services security as described in this article.

Related tasks

“Enabling hardware cryptographic devices for Web Services Security” on page 1181

“Enabling cryptographic keys stored in hardware devices in Web Services Security”

You can enable individual Web service applications to use cryptographic keys stored in hardware devices in Web Services Security.

“Configuring a hardware cryptographic keystore” on page 453

You can create a hardware cryptographic keystore that WebSphere Application Server can use to provide cryptographic token support in the server configuration.

Enabling cryptographic keys stored in hardware devices in Web Services Security

You can enable individual Web service applications to use cryptographic keys stored in hardware devices in Web Services Security.

You must first configure the hardware acceleration device using the key management panels in the administrative console. See “Configuring hardware cryptographic devices for Web Services Security” on page 1181

1. In the administrative console, click **Servers > Application servers** and then select the server name.
2. Under **Security**, click **Web services: default bindings for Web services security**.
3. Under **Additional properties**, click **key locators**.
4. Select the key locator name.
5. Under **Key store**, specify the name of the keystore configuration.

If the keystore reference is specified to a hardware device configuration, the Web Services Security run time first attempts to obtain the cryptographic algorithm from the hardware device. If the hardware device is not supported or if it fails, the run time for Web services security obtains the cryptographic algorithm from the security providers list. See “Creating a keystore configuration” on page 452 for more information about how to create the name of a keystore configuration.

If hardware acceleration is enabled, the Web service security run time first attempts to use the hardware device for cryptographic operations. If the attempt to use the hardware device fails or if the algorithm is not supported by the hardware device, the run time will use a software provider from the security providers list. The run time displays a warning message that you failed to use hardware cryptographic provider but the process will continue using the software that is provided.

6. Click **OK**.

If the name of the keystore reference is a Java keystore file, a hardware acceleration device configured at the application server level (`ws-security.xml`) is used for cryptographic operations.

Related tasks

“Enabling hardware cryptographic devices for Web Services Security” on page 1181

“Configuring hardware cryptographic devices for Web Services Security” on page 1181

You can configure a hardware cryptographic device in Web Service Security. The key for the cryptographic operation can be stored in an ordinary Java key store file, and does not need to be stored on the hardware devices.

Retrieving tokens from the JAAS Subject in a server application

In WebSphere Application Server Version 6.1, the security handlers are responsible for propagating security tokens. These security tokens are embedded in the 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 Secure administration, applications, and infrastructure 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

Retrieving tokens from the JAAS Subject in an application

The security handlers are responsible for propagating security tokens. These security tokens are embedded in the 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.tokenProperagation` 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.wssecurity.auth.token.UsernameToken)t.get("...");
}
```

After completing these steps, you have retrieved the security tokens from the JAAS Subject in a client application

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.
 - JCERACFKS** Use JCERACFKS if the certificates are stored in a SAF key ring (z/OS only).
 - 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 for the server or cell-level bindings

Collection certificate stores contain untrusted, intermediary certificate files awaiting validation. You can configure the collection certificate store on the server level and the cell level.

Validation might consist of checking for a valid signature in a digitally signed 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 store 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 IBM CertPath 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 1007.
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.
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.

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 954.

Configuring a nonce on the server or cell level

You can configure nonce for the cell level using the WebSphere Application Server administrative console.

Nonce is a randomly generated, cryptographic token that is used to prevent replay attacks of user name tokens that are used with SOAP messages. Typically, nonce is used with the user name token.

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 for the server or cell-level bindings” on page 1185.

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 1200. 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> uniform resource identifier (URI) value in the **Value type URI** field as well.

LTPA_PROPAGATION

For LTPA token propagation, the value type local name is LTPA_PROPAGATION. If you enter LTPA_PROPAGATION for the local name, you must specify the <http://www.ibm.com/websphere/appserver/tokentype> 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 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 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 plug in 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 for the server or cell-level bindings” on page 1185.

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 Keystore (JKS) format.
- JCEKS**
Use this option if you are using Java Cryptography Extensions.
- JCERACFKS**
Use JCERACFKS if the certificates are stored in a SAF key ring (z/OS only).

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. Use the alias 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.

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 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:

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 **Manage 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. Click **Manage modules > URI_name**.
3. Under Web Services Security 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.

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	<ol style="list-style-type: none"> 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.

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 49. 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 the 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, this product provides predefined value types. When you specify the following local names, you do not need to specify the Uniform Resource Identifier (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>

Lightweight Third Party Authentication (LTPA)

LTPA_PROPAGATION

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 LTPA token propagation, the value type local name is

LTPA_PROPAGATION. If you enter LTPA_PROPAGATION for the local name, you must specify the `http://www.ibm.com/websphere/appserver/tokentype` 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.

The application server provides the following predefined value type URIs:

- For the LTPA token: `http://www.ibm.com/websphere/appserver/tokentype/5.0.2`
- For the LTPA token propagation: `http://www.ibm.com/websphere/appserver/tokentype`

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**.

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.

This product 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**.

Note: The Algorithm mappings feature is not supported when the **Use the Federal Information Processing Standard (FIPS)** option has been selected on the SSL certificate and key management panel of the administrative console. When this option is selected, the **New** button in the Algorithm mappings panel is not available.

4. Click **New**.
5. Under Additional properties, click **Algorithm URI > *algorithm_URI_name***

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 this product. 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 1136.
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 1142.

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 1149.
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 1149.

The actual implementation of the algorithm is done in the implementation class for the engine factory.

Algorithm mapping collection

You can view a list of custom uniform resource identifier (URI) algorithms for digest value calculation, signature, key encryption, and data encryption. The 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**.

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).

Note: The Algorithm mappings feature is not supported when the **Use the Federal Information Processing Standard (FIPS) algorithms** option has been selected on the Global security panel of the administrative console. When this option is selected, the **New** button in the Algorithm mappings panel is not available.

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. The 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 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*.

Note: The Algorithm mappings feature is not supported when the **Use the Federal Information Processing Standard (FIPS)** option has been selected on the Global security panel of the administrative console. When this option is selected, the **New** button in the Algorithm mappings panel is not available.

Note: The Algorithm mappings feature is not supported when the **Use the Federal Information Processing Standard (FIPS)** option has been selected on the SSL certificate and key management panel of the administrative console. When this option is selected, the **New** button in the Algorithm mappings panel is not available.

4. Click **New**.

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. This product 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 6.0.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 6.0.x 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.

JCERACFKS

Use JCERACFKS if the certificates are stored in a SAF key ring (z/OS only).

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 1198.
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 signer key 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 1188. 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 identifier** 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 Uniform Resource Identifier (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 and <http://www.ibm.com/websphere/appserver/tokentype> for the LTPA_PROPAGATION 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

For LTPA_PROPAGATION

LTPA_PROPAGATION

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.

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#hmac-sha1>
 - <http://www.w3.org/2000/09/xmldsig#dsa-sha1>

Do not use this algorithm if you want the configured application to be compliant with the Basic Security Profile (BSP). Any ds:SignatureMethod/@Algorithm element in a SIGNATURE based on a symmetric key must have a value of <http://www.w3.org/2000/09/xmlsig#rsa-sha1> or <http://www.w3.org/2000/09/xmlsig#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 1200.
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 following algorithms:

- <http://www.w3.org/2000/09/xmldsig#sha1>
- <http://www.w3.org/2001/04/xmlenc#sha256>
- <http://www.w3.org/2001/04/xmlenc#sha512>

- 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>

Do not use this transform algorithm if you want your configured application to be compliant with the Basic Security Profile (BSP). Instead use <http://www.w3.org/2002/06/xmldsig-filter2> to ensure compliance.

- <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 **Apply**.
12. **Optional:** Determine whether to disable the Inclusive namespace prefix list. The Exclusive XML Canonicalization Version 1.0 specification recommends that you include all of the namespace declarations that correspond to the namespace prefix in the canonicalization form. For security reasons, WebSphere Application Server, by default, includes the prefix in the digital signature for Web services security. However, some implementations of Web services security cannot handle this prefix list. WebSphere Application Server can handle digitally signed messages that either contain or do not contain the prefix list. If you experience a signature validation failure when a signed Simple Object Access Protocol (SOAP) message is sent and you are using another vendor in your environment, it is highly recommended that you check with their Web site for a possible fix to their implementation before you disable this property. To disable this property, complete the following steps:
 - a. Under Additional properties, click **Canonicalization method properties > New**.
 - b. In the Property name field, enter the `com.ibm.wsspi.wssecurity.dsig.inclusiveNamespaces` property.

- c. In the Property value field, enter the false value.
 - d. Click **OK**.
13. 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.

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>.

Do not use this algorithm, the 192-bit key encryption algorithm, if you want your configured application to be in compliance with the Basic Security Profile (BSP).

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-oaep-mgf1p>When running with JDK 1.4, the list of supported key transport algorithms will not include this one. This algorithm will appear in the list of supported key transport algorithms when running with JDK 1.5.

Restriction: This algorithm is not supported when the WebSphere Application Server is running in Federal Information Processing Standard (FIPS) mode.

By default, the RSA-OAEP algorithm uses the SHA1 message digest algorithm to compute a message digest as part of the encryption operation. Optionally, you can use the SHA256 or SHA512 message digest algorithm by specifying a key encryption algorithm property. The property name is: `com.ibm.wsspi.wssecurity.enc.rsaoaep.DigestMethod`. The property value is one of the following URIs of the digest method:

- <http://www.w3.org/2001/04/xmlenc#sha256>
- <http://www.w3.org/2001/04/xmlenc#sha512>

By default, the RSA-OAEP algorithm uses a null string for the optional encoding octet string for the OAEPParams. You can provide an explicit encoding octet string by specifying a key encryption algorithm property. For the property name, you can specify `com.ibm.wsspi.wssecurity.enc.rsaoaep.OAEPparams`. The property value is the base 64-encoded value of the octet string.

Important: You can set these digest method and OAEPParams properties on the generator side only. On the consumer side, these properties are read from the incoming Simple Object Access Protocol (SOAP) message.

- 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>.

Do not use this algorithm, the 192-bit key encryption algorithm, if you want your configured application to be in compliance with the Basic Security Profile (BSP).

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 an 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 1200.
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.

Configuring trusted ID evaluators on the server or cell level

You can configure trusted identity (ID) evaluators. The trusted ID evaluator determines whether or not to trust the identity-asserting authority.

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, the 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 delete a trusted ID evaluator.

Version 6 and later applications

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.

Version 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. Click **Manage modules > *URI_name***.
3. Under Web Service Security 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 6.0.x.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.

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.

Version 6 and later applications

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. Click **Manage modules**.
3. Click the *URI_name*.
4. Under Web Services Security 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.

Version 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.

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 for the server or cell-level bindings” on page 1185.

- 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 1184.
 - 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 1206.
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 1206.
 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 1218. 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 Lightweight Third Party Authentication (LTPA) or token propagation (LTPA_PROPAGATION), you must specify both the value type local name and the Uniform Resource Identifier (URI). For LTPA, specify LTPA for the local name and `http://www.ibm.com/websphere/appserver/tokentype/5.0.2` for the URI. For LTPA token propagation, specify LTPA_PROPAGATION for the local name and `http://www.ibm.com/websphere/appserver/tokentype` 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 the `com.ibm.CSIOutboundPropagationEnabled` property is true. This property is set in the CSiv2 authentication panel. To access the panel, click **Security > Secure administrative, applications, and infrastructure**. Under Authentication, click **RMI/IOP security** and click **CSiv2 outbound authentication**. To set the `com.ibm.CSIOutboundPropagationEnabled` property, select **Security attribute propagation**. To modify this JAAS login configuration, see the JAAS - System logins panel.

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 and later 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.

system.WSS_INBOUND

This selection specifies the login configuration for inbound or consumer requests for security token propagation using Web services security. To modify this configuration, see the JAAS configuration panel for system logins.

system.WSS_OUTBOUND

This selection specifies the login configuration for outbound or generator requests for security token propagation using Web services security. 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.

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 and later applications on the application level, complete the following steps:

1. Click **Applications > Enterprise applications > *application_name***.
2. Click **Manage modules > *URI_name***.
3. Under Web Services Security 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**.

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 and later applications on the application level, complete the following steps:

1. Click **Applications > Enterprise applications > *application_name***.
2. Click **Manage modules > *URI_name***.
3. Under Web Services Security 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.

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 . Under Security, click Web services: Default bindings for Web services security . 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	<ol style="list-style-type: none">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.

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	<ol style="list-style-type: none">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.

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.

This product 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`

Lightweight Third Party Authentication (LTPA)

LTPA_PROPAGATION

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 LTPA token propagation, the value type local name is LTPA_PROPAGATION. If you enter LTPA_PROPAGATION for the local name, you must specify the `http://www.ibm.com/websphere/appserver/tokentype` 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.

The application server provides the following predefined value type URIs:

- For the LTPA token: `http://www.ibm.com/websphere/appserver/tokentype/5.0.2`
- For the LTPA token propagation: `http://www.ibm.com/websphere/appserver/tokentype`

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 1198.
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 1210.
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

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/xmlsig#rsa-sha1>
 - <http://www.w3.org/2000/09/xmlsig#hmac-sha1>
 - <http://www.w3.org/2000/09/xmlsig#dsa-sha1>

Do not use this algorithm if you want the configured application to be compliant with the Basic Security Profile (BSP). Any `ds:SignatureMethod/@Algorithm` element in a SIGNATURE based on a symmetric key must have a value of <http://www.w3.org/2000/09/xmlsig#rsa-sha1> or <http://www.w3.org/2000/09/xmlsig#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_keyinfo.
 - 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 1171.
 - 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 following algorithms:
 - <http://www.w3.org/2000/09/xmldsig#sha1>
 - <http://www.w3.org/2001/04/xmlenc#sha256>
 - <http://www.w3.org/2001/04/xmlenc#sha512>
 - 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>

Do not use this transform algorithm if you want your configured application to be compliant with the Basic Security Profile (BSP). Instead use <http://www.w3.org/2002/06/xmldsig-filter2> to ensure compliance.

- <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.

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>.

Do not use the 192-bit key encryption algorithm if you want your configured application to be in compliance with the Basic Security Profile (BSP).

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-oaep-mgf1p>.

When running with Software Development Kit (SDK) Version 1.4, the list of supported key transport algorithms does not include this one. This algorithm appears in the list of supported key transport algorithms when running with SDK Version 1.5.

Restriction: This algorithm is not supported when the WebSphere Application Server is running in Federal Information Processing Standard (FIPS) mode.

- 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>.

Do not use the 192-bit key encryption algorithm if you want your configured application to be in compliance with the Basic Security Profile (BSP).

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 1218.
9. Click **OK** and **Save** to save the configuration.

You have configured the encryption information for the consumer binding at the server level.

You must specify a similar encryption information configuration for the generator.

Tuning Web services security for Version 6.1 applications

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, the default Java Cryptography Extension (JCE) jurisdiction policy file shipped with the SDK enables you to use cryptography to enforce this default policy.

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.

For WebSphere Application Server platforms using IBM Developer Kit, Java Technology Edition Version 5, 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 **J2SE 5.0**
3. Scroll down and click **IBM SDK Policy files**.
The Unrestricted JCE Policy files for the SDK Web site is displayed.
4. Click **Sign in** and provide your IBM intranet ID and password.
5. Select the appropriate Unrestricted JCE Policy files and then click **Continue**.
6. View the license agreement and then click **I Agree**.
7. Click **Download Now**.

For WebSphere Application Server platforms using the Sun-based Java Development Kit (JDK) Version 5, 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/1.5.0/download.jsp>
2. Click **Other Downloads**.
3. Locate the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 1.5.1 information and click **Download**. The policy files are downloaded onto your machine.

In IBM WebSphere Application Server Version 6.1, Web services security supports the use of cryptographic hardware devices. There are two ways in which to use hardware cryptographic devices with Web services security.

See “Hardware cryptographic device support for Web Services Security” on page 1180 for more information.

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

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.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6 and later applications.

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 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 inter-operate between formerly incompatible security technologies, such as public key infrastructure and Kerberos in heterogeneous environments like Microsoft .NET and environments that are based on the Java 2 Platform, Enterprise Edition (J2EE) standards. 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.

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 1253
- "Securing Web services for Version 5.x applications using XML encryption" on page 1306
- "Securing Web services for Version 5.x applications using basicauth authentication" on page 1325
- "Securing Web services for Version 5.x applications using identity assertion authentication" on page 1333
- "Securing Web services for version 5.x applications using signature authentication" on page 1339
- "Securing Web services for version 5.x applications using a pluggable token" on page 1351

Web services security specification—a chronology

This chronology describes the process that has been used to develop the Web services security specifications. The chronology includes both the Organization for the Advancement of Structured Information Standards (OASIS) and non-OASIS activities.

Non-OASIS activities

Important: There is an important distinction between Version 5.x and Version 6.0.x applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x 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 user name tokens and encoded binary security tokens. After some discussion and an interoperability test that was 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 timestamp and nonce (randomly generated token).

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. 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.

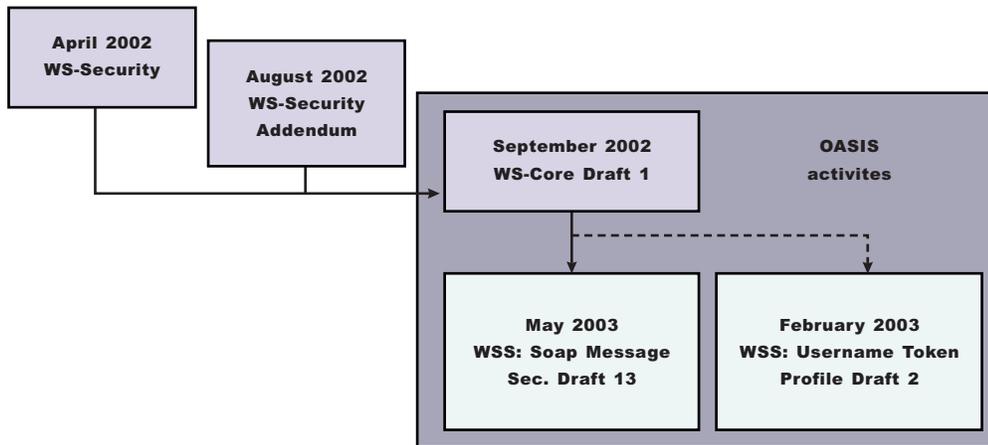


Figure 8. Web services security specification support

Web services security support

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later applications.

WebSphere Application Server, Versions 4.x, 5, and 5.0.1 support digital signature for Apache 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.

The WebSphere Application Server security infrastructure fully integrates Web services security with the Java 2 Platform, Enterprise Edition (J2EE) security specification. Web services security is not supported in a pure Java client or a nonmanaged 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 50. 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	<ul style="list-style-type: none"> • Body • Securitytoken • Timestamp
XML encryption	<ul style="list-style-type: none"> • Bodycontent • Usenametoken

Method	Element
AuthMethod	<ul style="list-style-type: none"> • BasicAuth • IDAssertion (From WebSphere Application Server to another WebSphere Application Server) • Signature • Lightweight Third Party Authentication (LTPA) on the server side • Other customer tokens

WebSphere Application Server signs and encrypts the following elements for the response message:

Method	Element
XML digital signature	<ul style="list-style-type: none"> • Body • Timestamp
XML encryption	<ul style="list-style-type: none"> • Bodycontent

The namespaces used for sending a message were published by OASIS in draft 13 (<http://schemas.xmlsoap.org/ws/2003/06/secext>). However, the Web services security run time in WebSphere Application Server can accept any of the following namespaces:

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 article describes the relationship between Web services security (message level security) model and the Java 2 Platform, Enterprise Edition (J2EE) security model. It also includes information on J2EE role-based authorization checks.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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 J2EE architecture so that you can develop and run Web services on the J2EE component architecture.

Important: *Web services security* refers to the Web services security: SOAP Message Security specification. For more information, see “Web services security support” on page 1226.

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.

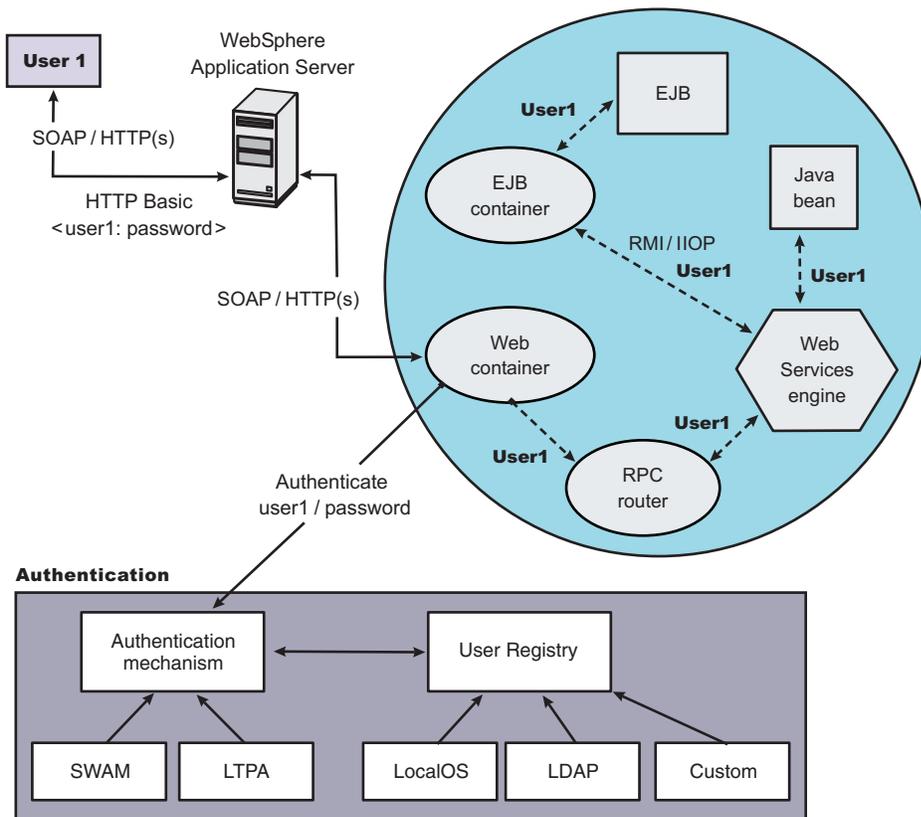


Figure 9. SOAP message flow using existing security infrastructure of WebSphere Application Server

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.

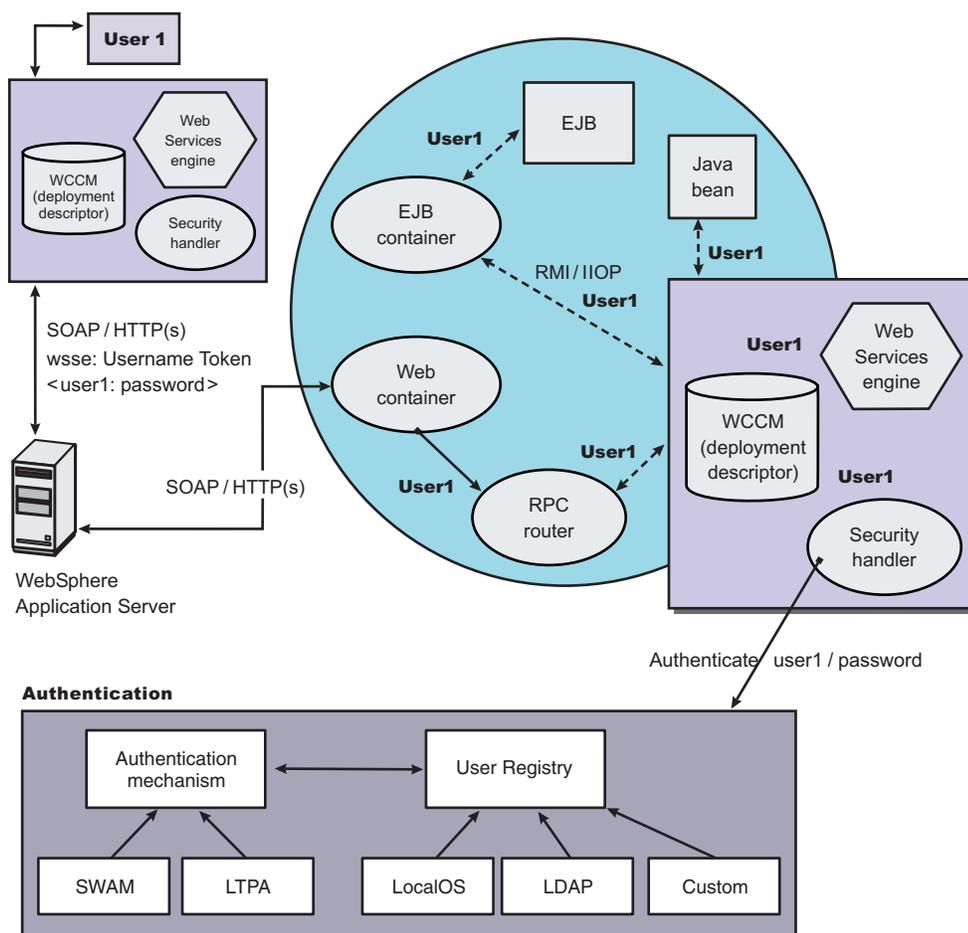


Figure 10. SOAP message flow using Web services security

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 Enterprise JavaBeans (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.

J2EE role-based authorization checks

A standard for authorization does not exist for Web services. However, IBM, in conjunction with Microsoft Corporation, jointly published a security white paper road map for Web services called *Security in a Web Services World: A Proposed Architecture and Roadmap* in which a proposal exists for the WS-Authorization specification. However, the WS-Authorization specification has not been published.

The existing implementation of Web services security is based upon the Web Services for J2EE specification or the Java Specification Requirements (JSR) 109 specification. The implementation of Web services security leverages the J2EE role-based authorization checks. For conceptual information on role-based authorization, see “Role-based authorization” on page 324. If you develop a Web service that requires method-level authorization checks, then you must use stateless session beans to implement your Web service. For more information on using stateless session beans to implement a Web service, see *Developing a Web service from an enterprise bean* and “Securing enterprise bean applications” on page 919. If you develop a Web service that is implemented as a servlet, you can use coarse-grained or URL-based authorization in the Web container. However, in this situation, you cannot use the identity from Web services security for authorization checks. Instead, you can use the identity from the transport. If you use SOAP over HTTP, then the identity is in the HTTP 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: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6 and later applications.

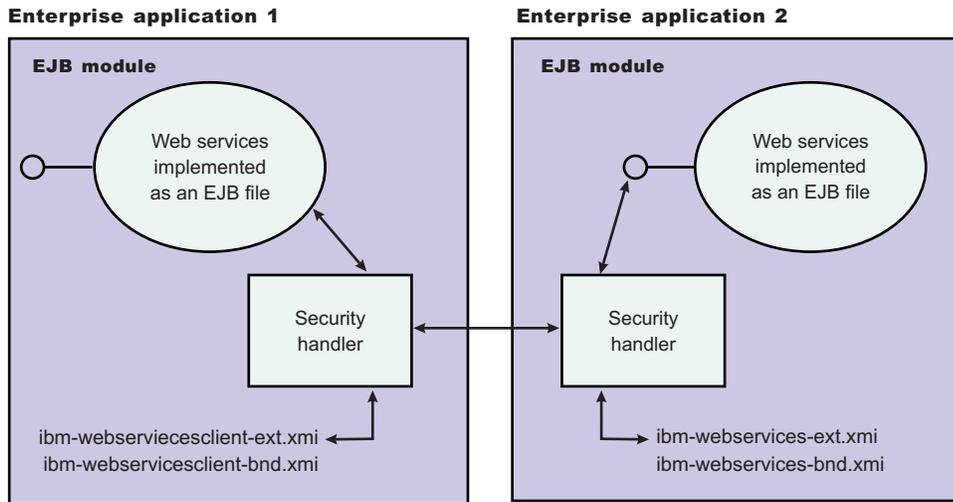


Figure 11. Web services security model

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 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 Java 2 Platform, Enterprise Edition (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: 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.

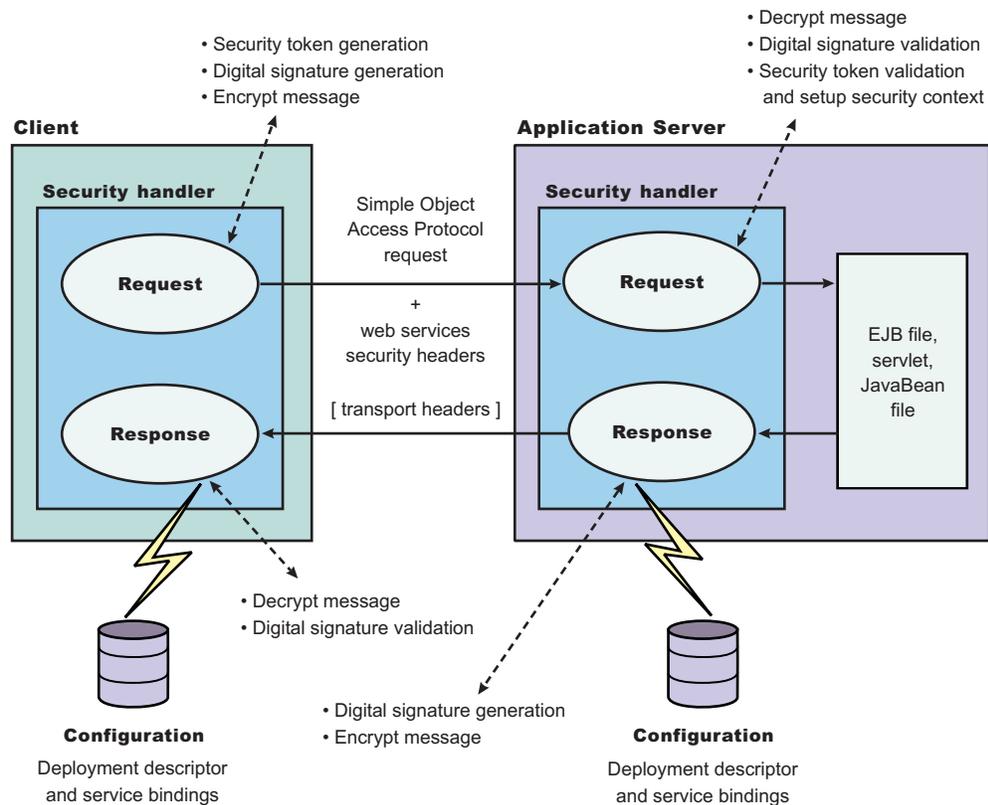


Figure 12. Web services security message interpretation

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 <code>ibm-webservicesclient-ext.xmi</code> and <code>ibm-webservicesclient-bnd.xmi</code> files)	<ul style="list-style-type: none"> 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 <code>ibm-webservices-ext.xmi</code> and <code>ibm-webservices-bnd.xmi</code> files)	<ul style="list-style-type: none"> Verifies that the Web services security constraints are met. 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. Verifies the required signature. Verifies that the message is encrypted and decrypts the message if encrypted. Validates the security tokens and sets up the security context for the downstream call.
Response sender (defined in the <code>ibm-webservices-ext.xmi</code> and <code>ibm-webservices-bnd.xmi</code> files)	<ul style="list-style-type: none"> 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 <code>ibm-webservicesclient-ext.xmi</code> or <code>ibm-webservicesclient-bnd.xmi</code> files)	<ul style="list-style-type: none"> • Verifies that the Web services security constraints are met. • 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. • Verifies the required signature. • 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.

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 the 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 the 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).

transition: In previous releases of WebSphere Application Server, the nonce value was 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.

Enable cryptographic operations on hardware device

Enables cryptographic operations on hardware devices. Enabling this feature might improve the performance, depending on the hardware device.

Cryptographic hardware configuration name

Specifies the name of the hardware device configuration name defined in the keystore settings in the secure communications.

This value is necessary only if **Hardware acceleration** has been selected.

Usage scenario for propagating security tokens

This sample shows propagating security tokens using Web services security, the security infrastructure of the WebSphere Application Server, and Java 2 Platform, Enterprise Edition (J2EE) security.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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 the Enterprise JavaBeans (EJB) file 2. The EJB file 2 calls Web services 3 and Web services 3 calls Web services 4.

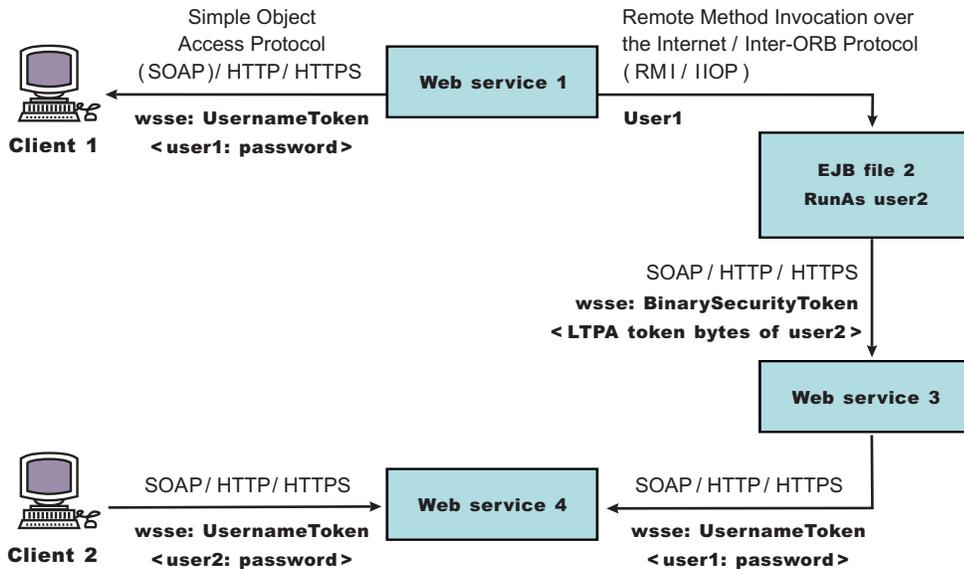


Figure 13. Propagating security tokens

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.

Web services security constraints

The Web services security model that is used by WebSphere Application Server is the declarative model. A version 5.x application must be secured with Web services security by defining the security constraints in the IBM extension deployment descriptors and in IBM extension bindings.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6 and later 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) programming model. See the following figure for more details.

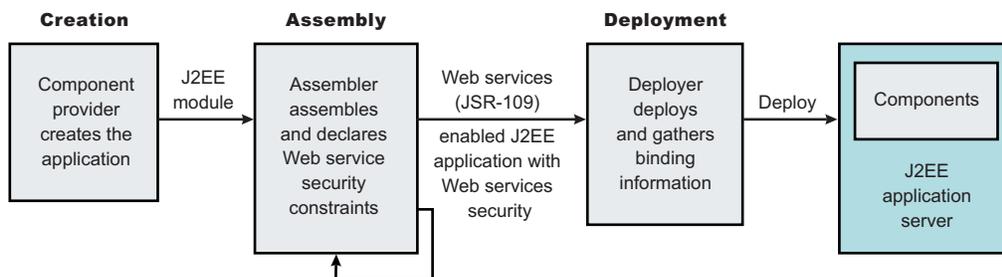


Figure 14. Application development life cycle

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 Enterprise JavaBeans (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.

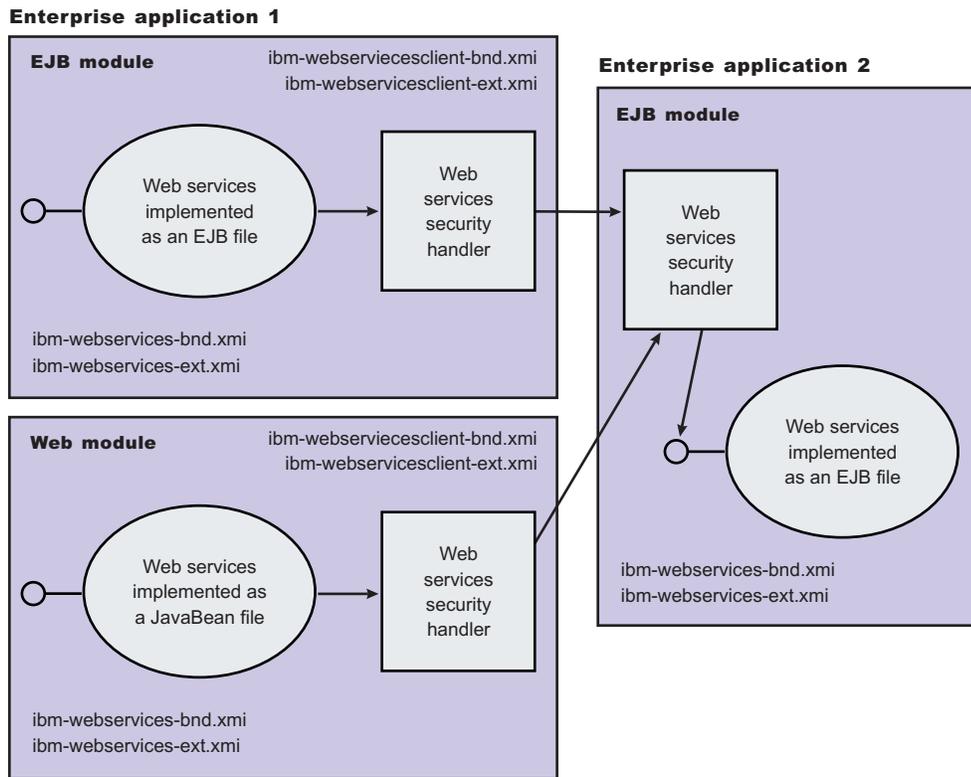


Figure 15. IBM extension deployment descriptors and bindings

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.

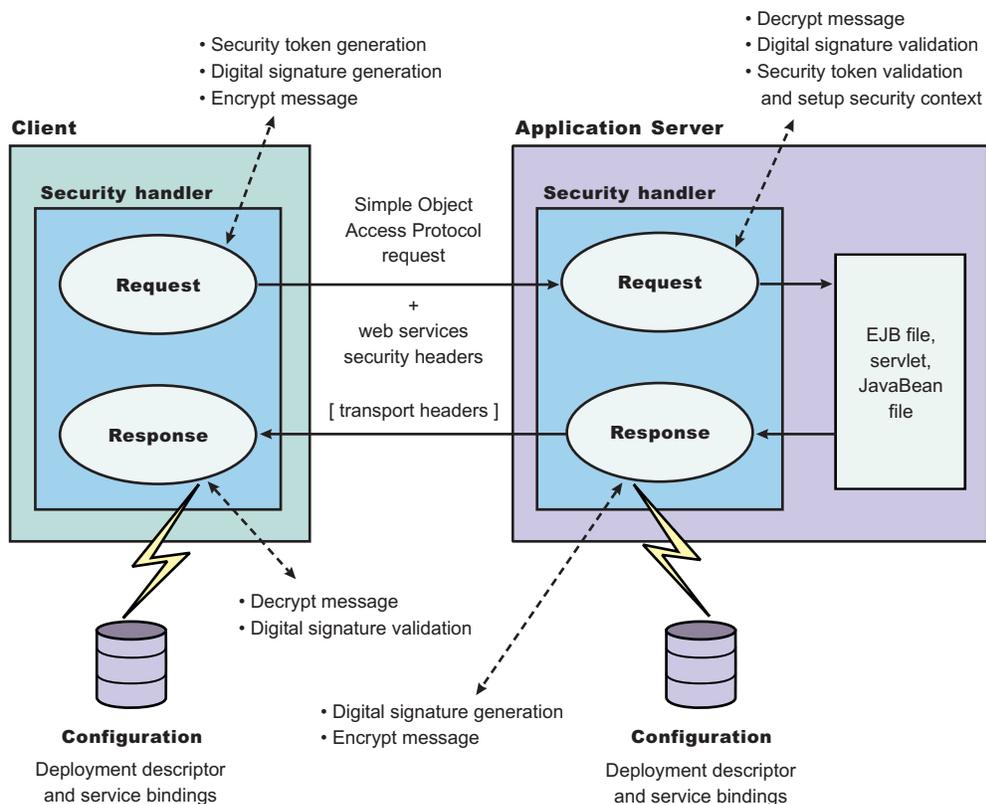


Figure 16. Web services security constraints

Sample configuration for Web services security for a version 5.x application

To secure a version 5.x application with Web services security, you must define the security constraints in the IBM extension deployment descriptors and in IBM extension bindings. Sample keystore files and default binding information are provided for a sample configuration to demonstrate what IBM deployment descriptor extensions and bindings can do.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6 and later applications.

WebSphere Application Server provides the following sample keystores for sample configurations. These sample keystores are for testing and sample purposes only. Do not use them in a 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 Simple Object Access Protocol (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.wscontext:WsClientExtension xmi:version="2.0"
  xmlns:xmi="http://www.omg.org/XMI" xmlns:com.ibm.etools.webservice.wscontext=
  http://www.ibm.com/websphere/appserver/schemas/5.0.2/wscontext.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>
        </securityRequestSenderServiceConfig>
      </clientServiceConfig>
    </portQnameBindings>
  </serviceRefs>
</com.ibm.etools.webservice.wscontext:WsClientExtension>
```

```

    <confidentiality>
      <confidentialParts part="bodycontent"/>
      <confidentialParts part="usertoken"/>
    </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.wsclient: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 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 portQnameLocalName="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}PDM20jEr" path=
            "${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-sender.ks" type="JKS"/>
          <keys alias="soaprequester" keypass="{xor}PDM20jEr" name="clientsignerkey"/>
        </keyLocators>
      </securityRequestSenderBindingConfig>
    </portQnameBindings>
  </serviceRefs>
</com.ibm.etools.webservice.wscbnd:ClientBinding>

```

```

<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=
    "${USER_INSTALL_ROOT}/etc/ws-security/samples/enc-sender.jceks" type="JCEKS"/>
  <keys alias="Group1" keypass="{xor}NDomLz4sLA==" name="CN=Group1"/>
</keyLocators>
<loginBinding authMethod="BasicAuth" callbackHandler=
"com.ibm.wsspi.wssecurity.auth.callback.StdinPromptCallbackHandler"/>
</securityRequestSenderBindingConfig>
<securityResponseReceiverBindingConfig>
  <signingInfos>
    <signatureMethod algorithm="http://www.w3.org/2000/09/xmlsig#rsa-sha1"/>
    <certPathSettings>
      <trustAnchorRef ref="SampleClientTrustAnchor"/>
      <certStoreRef ref="SampleCollectionCertStore"/>
    </certPathSettings>
    <canonicalizationMethod algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
    <digestMethod algorithm="http://www.w3.org/2000/09/xmlsig#sha1"/>
  </signingInfos>
  <trustAnchors name="SampleClientTrustAnchor">
    <keyStore storepass="{xor}PDM20jEr" path=
      "${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-sender.ks" type="JKS"/>
  </trustAnchors>
  <certStoreList>
    <collectionCertStores provider="IBMCertPath" name="SampleCollectionCertStore">
      <x509Certificates path="${USER_INSTALL_ROOT}/etc/ws-security/samples/intca2.cer"/>
    </collectionCertStores>
  </certStoreList>
  <encryptionInfos name="EncInfo2">
    <encryptionKey locatorRef="SampleReceiverEncryptionKeyLocator"/>
    <encryptionMethod algorithm="http://www.w3.org/2001/04/xmlenc#tripleDES-cbc"/>
    <keyEncryptionMethod algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
  </encryptionInfos>
  <keyLocators name="SampleReceiverEncryptionKeyLocator" classname=
"com.ibm.wsspi.wssecurity.config.KeyStoreKeyLocator">
    <keyStore storepass="{xor}PDM20jEr" path=
      "${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-sender.ks" type="JKS"/>
    <keys alias="soaprequester" keypass="{xor}PDM20jEr" 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="usernetoken"/>
          </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.wsbind:WSBinding xmi:version="2.0"
  xmlns:xmi="http://www.omg.org/XMI"
  xmlns:com.ibm.etools.webservice.wsbind=
  http://www.ibm.com/websphere/appserver/schemas/5.0.2/wsbind.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.wsbind:WSBinding>
```

Overview of authentication methods

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: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6 and later applications.

When the WebSphere Application Server is configured to use the BasicAuth authentication method, the sender attaches the Lightweight Third Party Authentication (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 51. 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>.

Table 51. Authentication methods and their security tokens (continued)

Authentication method	Security token
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>: <ul style="list-style-type: none"> • If the <trustMode> is BasicAuth, IDAssertion requires <wsse:UsernameToken> with <wsse:Username> and <wsse:Password>. • 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.

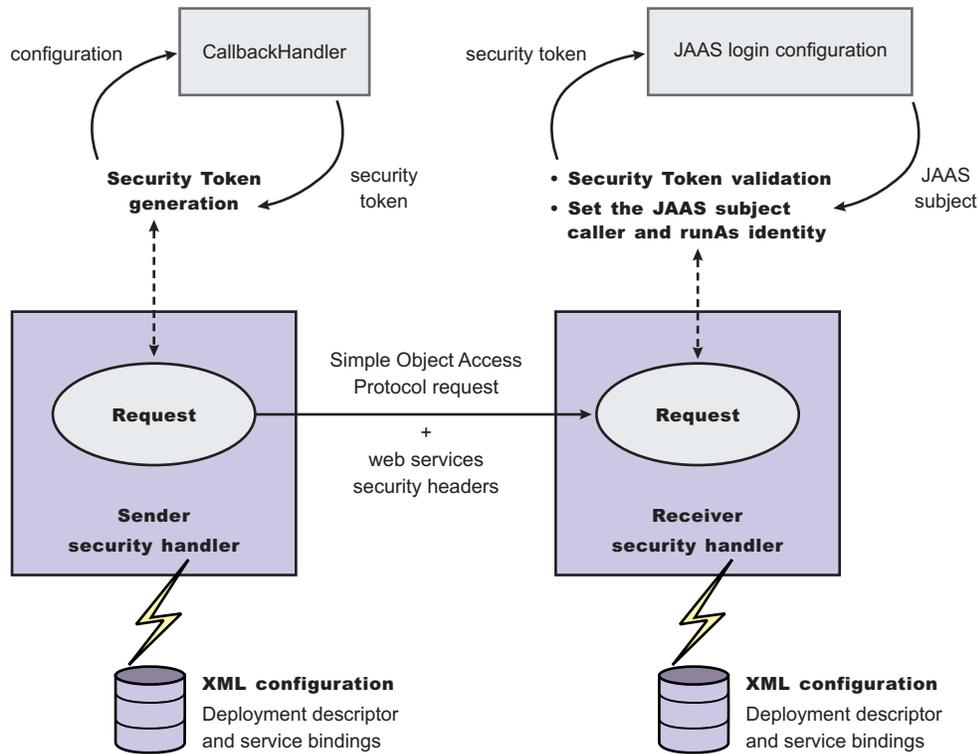


Figure 17. Security token generation and validation

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 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: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6 and later applications.

Signing parameter configuration settings

Use this page to configure new signing parameters.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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. Click **Manage modules** > *URI_file_name*
3. **Version 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/xmlsig#rsa-sha1>
- <http://www.w3.org/2000/09/xmlsig#dsa-sha1>
- <http://www.w3.org/2000/09/xmlsig#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/xmlsig#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	<ol style="list-style-type: none">1. Click Servers > Application servers > <i>server_name</i>.2. Under Security, click Web services: Default bindings for Web services security.3. Under Additional properties, click Key locators.
Request sender	Application level	<ol style="list-style-type: none">1. Click Applications > Enterprise applications > <i>application_name</i>.2. Click Manage modules > <i>URI_name</i>.3. Click Web services: Client security bindings.4. Under Request sender binding, click Edit.5. Under Additional properties, click Key locators.
Request receiver	Application level	<ol style="list-style-type: none">1. Click Applications > Enterprise applications > <i>application_name</i>.2. Click Manage modules > <i>URI_name</i>.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	<ol style="list-style-type: none">1. Click Applications > Enterprise applications > <i>application_name</i>.2. Click Manage modules > <i>URI_name</i>.3. Click Web services: Server security bindings.4. Under Response sender binding, click Edit.5. Under Additional properties, click Key locators.

Binding name	Cell level, server level, or application level	Path
Response receiver	Application level	<ol style="list-style-type: none"> 1. Click Applications > Enterprise applications > <i>application_name</i>. 2. Click Manage modules > <i>URL_name</i>. 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

XML digital signature is one of the methods WebSphere Application Server provides to secure your Web services. It provides message integrity and authentication capabilities when used with SOAP messages

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later applications.

WebSphere Application Server provides several different methods to secure your Web services; 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 an 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

Each application server, in WebSphere Application Server, uses a copy of the `ws-security.xml` file to define the default binding information for Web services security.

Important: There is an important distinction between Version 5.x and Version 6.0.x applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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.

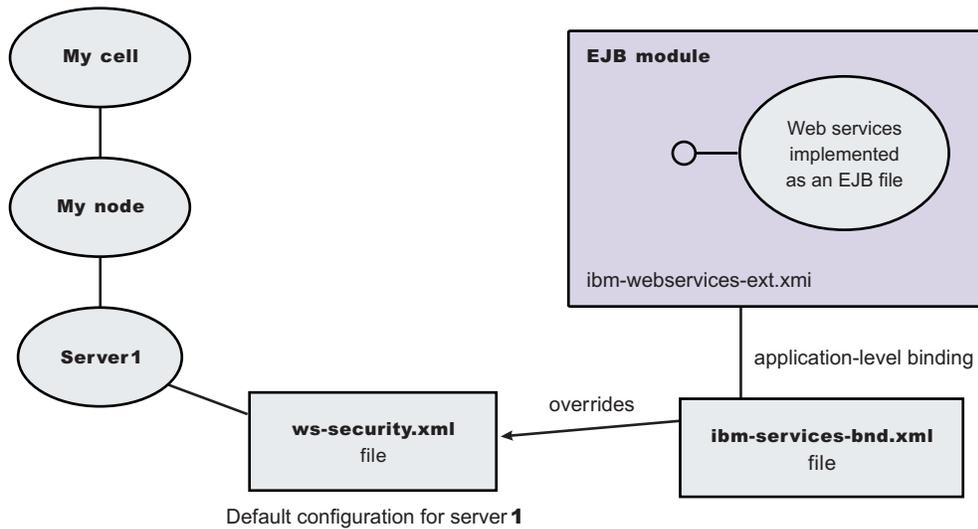


Figure 18. Configuration when using the **addNode** command

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: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6 and later 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 SOAP message header. The JAAS login configuration is defined in the administrative console under **Security > Secure administration, applications, and infrastructure > Java Authentication and Authorization Service > 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.

Trust anchors

A trust anchor specifies keystores that contain trusted root certificates that validate the signer certificate. The request receiver and the response receiver use these keystores to validate the signer certificate of the digital signature.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later applications.

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) use these keystores 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 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 keystore object. The type, path, and password of the keystore are passed to the implementation through the administrative console or by scripting.

Configuring trust anchors using an assembly tool

Use an assembly tool to configure trust anchors (that specify key stores which contain trusted root certificates to validate the signer certificate) or trust stores at the application level.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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 1299 or “Configuring the server security bindings using the administrative console” on page 1302.

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.xml` 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.xml` 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 1299 or “Configuring the server security bindings using the administrative console” on page 1302 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.xml` 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 the Java Cryptography Extension (JCE) and Java Cryptography Extension Key Stores (JCEKS) types.
- g. Save the changes.
4. Configure the server-side request receiver, which is defined in the `ibm-webservices-bnd.xml` bindings extensions file.
 - a. Click **Windows > Open perspective > J2EE**.
 - b. Select the Web services enabled Enterprise JavaBeans (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 1284
2. “Configuring the server for request digital signature verification: choosing the verification method” on page 1285

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 1291
2. “Configuring the client for response digital signature verification: choosing the verification method” on page 1293

Configuring trust anchors using the administrative console

Use the WebSphere Application Server administrative console to configure trust anchors that specify key stores which contain trusted root certificates to validate the signer certificate.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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 1299 or “Configuring the server security bindings using the administrative console” on page 1302.

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 1299 or “Configuring the server security bindings using the administrative console” on page 1302 for an introduction on how to manage Web services security binding information on the server.
3. Click **Applications > Enterprise applications > *enterprise_application***.
4. Under Manage modules, click *URI_name*.
5. Under Web Services Security 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 Web Services Security 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 1284
2. “Configuring the server for request digital signature verification: choosing the verification method” on page 1285

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 1291
2. “Configuring the client for response digital signature verification: choosing the verification method” on page 1293

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 SOAP message.

Important: There is an important distinction between Version 5.x and Version 6.0.x applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x 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

You can configure the client-side collection certificate store using the assembly tool.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6 and later 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 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. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > 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

You can configure the client-side collection certificate store by using the administrative console.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6 and later 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 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:port_number/ibm/console` in your Web browser unless you have changed the port number.

2. Click **Applications > Enterprise applications > application_name**.
3. Under Manage modules, click *URI_name*.
4. 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 1291
 - “Configuring the client for response digital signature verification: choosing the verification method” on page 1293
5. Under Response receiver binding, click **Edit** to edit the client security bindings.
6. Click **Collection certificate store**.
7. Click a Certificate store name to edit an existing certificate store or click **New** to add a new certificate store name.
8. 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.
9. Leave the Certificate store provider field value as IBM CertPath.
10. Click **Apply**.
11. Under Additional properties, click **X.509 certificates > New**.
12. 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.
13. Click **OK**.

Configuring the server-side collection certificate store using an assembly tool

A *collection certificate store* is a collection of non-root, certificate authority (CA) certificates and certificate revocation lists (CRLs). This collections of CA certificates and CRLs are used to check the signature of a digitally signed SOAP message. You can configure the server-side collection certificate store by using an assembly tool.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later applications.

You can configure the collection certificate either by using an assembly tool or by using 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 “Starting WebSphere Application Server Toolkit” in the Application Server Toolkit documentation.
2. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > 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 IBM CertPath.
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

You can configure the collection certificate either by using an assembly tool or the WebSphere Application Server administrative console.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6 and later 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 is used to check the signature of a digitally signed SOAP message.

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:port_number/ibm/console` in your Web browser unless you have changed the port number.
2. Click **Applications > Enterprise applications > application_name**.
3. Under Manage modules, click *URI_name*
4. Under Web Services Security 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 1284
 - “Configuring the server for request digital signature verification: choosing the verification method” on page 1285
5. Click **Edit** under Request Receiver Binding to edit the server security bindings.
6. Click **Collection certificate store**.
7. Click a Certificate store name to edit an existing certificate store or click **New** to add a new certificate store name.
8. 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.
9. Leave the Certificate store provider field as IBM CertPath.
10. Click **Apply**.
11. Under Additional Properties, click **X.509 Certificates > New**.
12. 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.

13. Click **OK**.

Configuring default collection certificate stores at the server level in the WebSphere Application Server administrative console

You can define a single collection certificate store for all of the applications that need to use the same certificates. Use the WebSphere Application Server administrative console to configure the default collection certificate store at the server level.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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 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:port_number/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: There is an important distinction between Version 5.x and Version 6.0.x applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x 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: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6 and later applications.

Web services security service provider programming interfaces

Several Service Provider Interfaces (SPIs) are provided to extend the capability of the Web services security runtime.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later applications.

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 (RunAs 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 runtime. 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 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: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > 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 the Java Cryptography Extensions (JCE) policy. 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 WSIKeyStoreMapKeyLocator 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

You can configure binding information and key locators using the WebSphere Application Server administrative console.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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. Open the administrative console.
Type `http://localhost:port_number/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 the Java Cryptography Extensions (JCE) policy and use **JCEKS** if you are using the JCE policy.

Trusted ID evaluator

The trusted ID evaluator is an abstraction of the mechanism that evaluates whether the given ID name is to be trusted. The trusted ID evaluator is typically used by the eventual receiver in a multi-hop environment.

Important: There is an important distinction between Version 5.x and Version 6.0.x applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x applications.

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 Web services security implementation (`com.ibm.wsspi.wssecurity.id.TrustedIDEvaluator`) 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.

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 Simple Object Access Protocol (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.

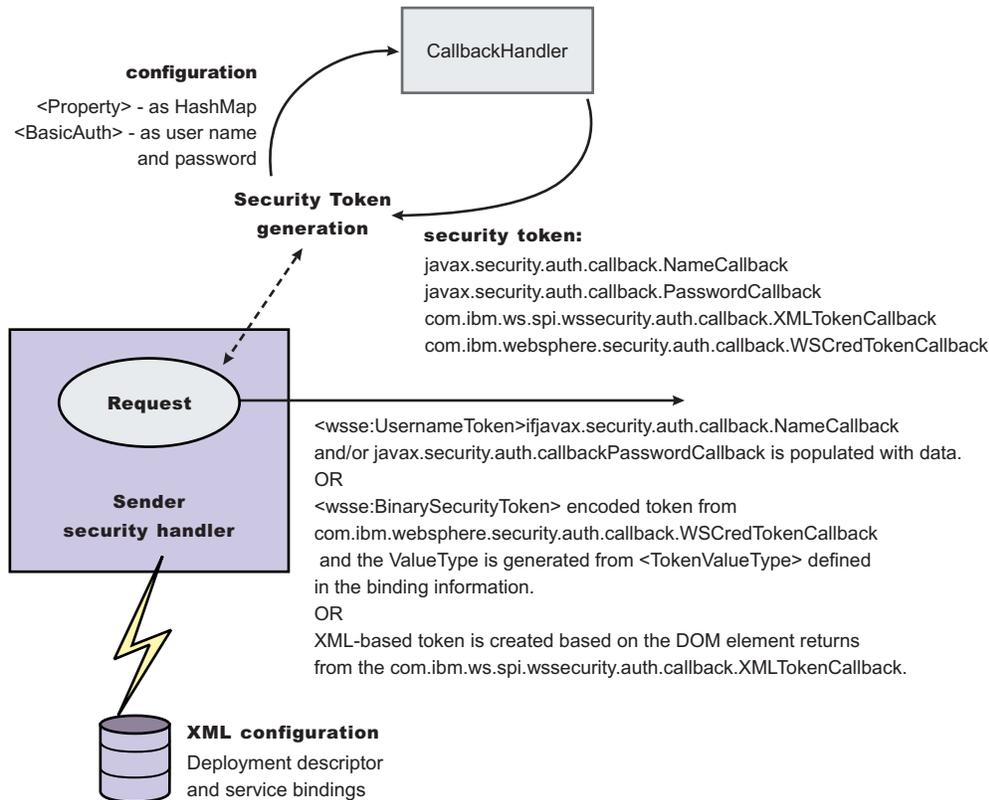


Figure 19. Request sender SOAP message process

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.xml` 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.

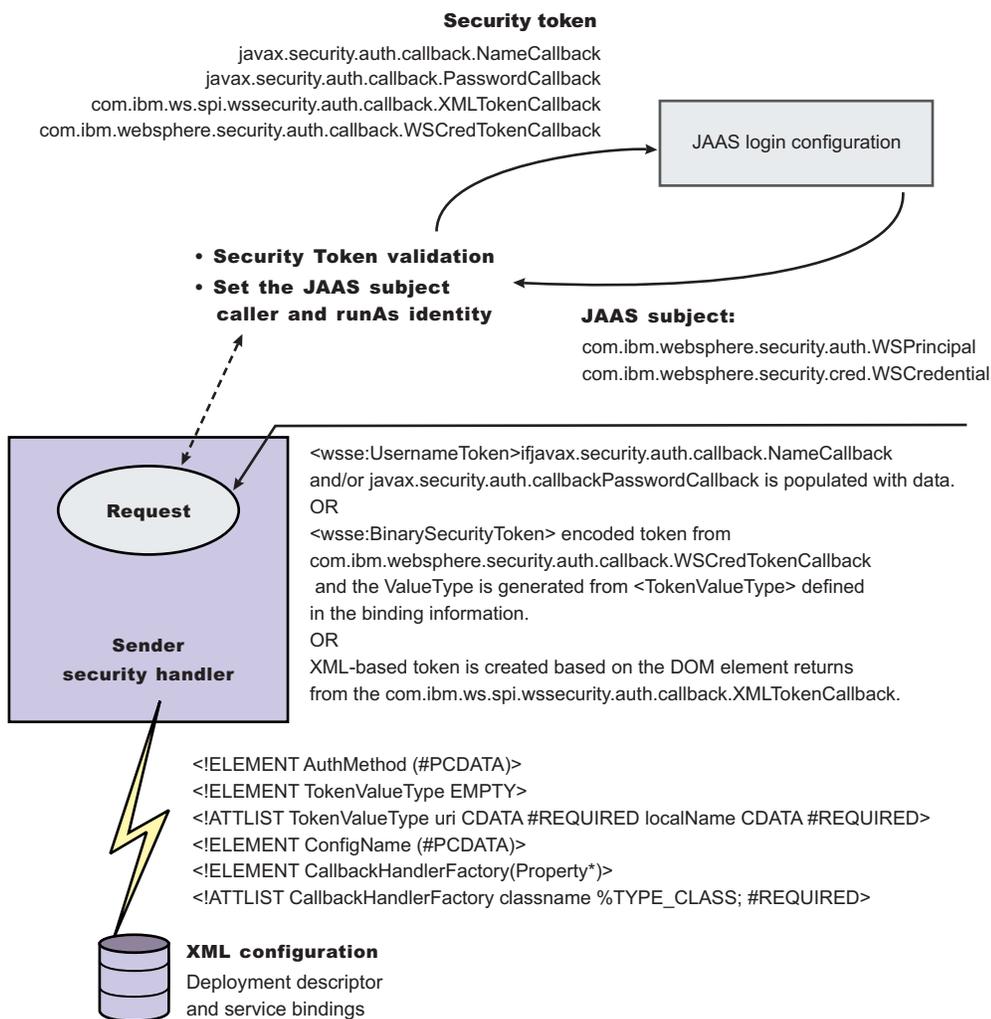


Figure 20. Request receiver SOAP message process

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: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6 and later 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: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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. Click **Manage modules > *URI_name***.
3. Under Web Services Security 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 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: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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. Click **Manage modules > *URI_name***.
3. Under Web Services Security 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 the `com.ibm.CSIOutboundPropagationEnabled` property is `true`. This property is set in the CSlv2 authentication panel. To access the panel, click **Security > Global security > Authentication protocol > CSlv2 Outbound authentication**. To set the `com.ibm.CSIOutboundPropagationEnabled` property, select **Security attribute propagation**.

Version 6 and later applications

system.wssecurity.X509BST

Verifies an X.509 binary security token (BST) by checking the validity of the certificate and the certificate path.

Version 6 and later applications

system.wssecurity.PKCS7

Verifies an X.509 certificate with a certificate revocation list in a PKCS7 object.

Version 6 and later applications

system.wssecurity.PkiPath

Verifies an X.509 certificate with a public key infrastructure (PKI) path.

Version 6 and later applications

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 > Secure administration, applications, and infrastructure**.
2. Under Java Authentication and Authorization Service, click **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 Java Authentication and Authorization Service (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 > Secure administration, applications, and infrastructure**.
2. Under Java Authentication and Authorization Service, click **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

To configure the client for request signing, specify which message parts to digitally sign when configuring the client.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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 1295
- “Configuring the security bindings on a server acting as a client using the administrative console” on page 1298

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 1249.
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 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

To configure the client for request signing, specify which message parts to digitally sign when configuring the client.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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 1295
- “Configuring the security bindings on a server acting as a client using the administrative console” on page 1298

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 1280 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. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. 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/xmlsig-core>.

Name	Purpose
Canonicalization method algorithm	Canonicalizes the <SignedInfo> element before the information is digested as part of the signature operation.
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 key store where the alias and the certificate exist.

7. **Optional:** Select **Show only FIPS Compliant Algorithms** if you only want the FIPS compliant algorithms to be shown in the **Digest method algorithm** and **Signature method algorithm** drop-down lists. Use this option if you expect this application to be run on a WebSphere Application Server that has set the **Use the United States Federal Information Processing Standard (FIPS) algorithms** option in the SSL certificate and key management panel of the WebSphere administrative console.

Important: If you configure the client and server signing information correctly, but receive a Soap body not signed error when running 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

Configure the server for request digital signature verification by modifying the extensions to indicate which parts of the request to verify.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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 1299
- “Configuring the server security bindings using the administrative console” on page 1302

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 1280 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. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. 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 running 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

To configure the server for request digital signature verification, use an assembly tool to modify the extensions and indicate which digital signature method the server will use during verification.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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 1299
- “Configuring the server security bindings using the administrative console” on page 1302

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 1284. 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. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. 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 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/xmlsig-core>.

Name	Purpose
Canonicalization method algorithm	Canonicalizes the <code><SignedInfo></code> 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 <code><DigestValue></code> element. The signing of the <code><DigestValue></code> 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.

Name	Purpose
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.
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.

8. **Optional:** Select **Show only FIPS Compliant Algorithms** if you only want the FIPS compliant algorithms to be shown in the Signature method algorithm and Digest method algorithm drop-down lists. Use this option if you expect this application to be run on a WebSphere Application Server that has set the **Use the United States Federal Information Processing Standard (FIPS) algorithms** option in the SSL certificate and key management panel of the administrative console for WebSphere Application Server.

Important: If you configure the client and server signing information correctly, but receive a Soap body not signed error when running 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

Use an assembly tool to specify which message parts to digitally sign when configuring the server for response signing.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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 1299
- “Configuring the server security bindings using the administrative console” on page 1302

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. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. 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** is 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 running 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

Use an assembly tool to specify which digital signature method to use when configuring the server for response signing.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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 1299
- “Configuring the server security bindings using the administrative console” on page 1302

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. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > 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/xmlsig-core>.

Name	Purpose
Canonicalization method algorithm	Canonicalizes the <code><SignedInfo></code> 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 <code><DigestValue></code> element. Signing the <code><DigestValue></code> 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 <code><SignedInfo></code> element into the <code><SignatureValue></code> element. The algorithm selected for the server response sender configuration must match the algorithm selected in the client response receiver configuration.

Name	Purpose
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: <ul style="list-style-type: none"> • “Configuring key locators using an assembly tool” on page 1268 • “Configuring key locators using the administrative console” on page 1269

8. **Optional:** Select **Show only FIPS Compliant Algorithms** if you only want the FIPS compliant algorithms to be shown in the Signature method algorithm and Digest method algorithm drop-down lists. Use this option if you expect this application to be run on a WebSphere Application Server that has set the **Use the United States Federal Information Processing Standard (FIPS) algorithms** option in the SSL certificate and key management panel of the administrative console for WebSphere Application Server.

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

To configure the Web services security extensions and the Web services security bindings, use the WS Extension tab and the WS Binding tab in the Client Deployment Descriptor within an assembly tool such as the Application Server Toolkit or Rational Web Developer.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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 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 1295
- “Configuring the security bindings on a server acting as a client using the administrative console” on page 1298

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. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. 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 running 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

You can configure the Web services security extensions and Web services security bindings using 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.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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 1299
- “Configuring the server security bindings using the administrative console” on page 1302

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 1291 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. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. 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/xmlsig-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.

8. **Optional:** Select **Show only FIPS Compliant Algorithms** if you only want the FIPS compliant algorithms to be shown in the Signature method algorithm and Digest method algorithm drop-down lists. Use this option if you expect this application to be run on a WebSphere Application Server that has set the **Use the United States Federal Information Processing Standard (FIPS) algorithms** option in the SSL certificate and key management panel of the administrative console for WebSphere Application Server.

Important: If you configure the client and server signing information correctly, but receive a Soap body not signed error when running 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

Use the Web services client editor within an assembly tool to include the binding information, that describes how to run the security specifications found in the extensions, in the client enterprise archive (EAR) file.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > 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 1280

Integrity

“Configuring the client for request signing: digitally signing message parts” on page 1280

Confidentiality

“Configuring the client for request encryption: Encrypting the message parts” on page 1311

Login Config**BasicAuth**

“Configuring the client for basic authentication: specifying the method” on page 1325

IDAssertion

“Configuring the client for identity assertion: specifying the method” on page 1333

Signature

“Configuring the client for signature authentication: specifying the method” on page 1339

LTPA

“Configuring the client for LTPA token authentication: specifying LTPA token authentication” on page 1357

ID assertion

“Configuring the client for identity assertion: specifying the method” on page 1333

Add created time stamp

“Configuring the client for request signing: digitally signing message parts” on page 1280

Response receiver configuration details**Required integrity**

“Configuring the client for response digital signature verification: verifying the message parts” on page 1291

Required confidentiality

“Configuring the client for response decryption: decrypting the message parts” on page 1322

Add received time stamp

“Configuring the client for response digital signature verification: verifying the message parts” on page 1291

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 1282

Encryption information

“Configuring the client for request encryption: choosing the encryption method” on page 1311

Key locators

“Configuring key locators using an assembly tool” on page 1268

Login binding**Basic auth**

“Configuring the client for basic authentication: collecting the authentication information” on page 1327

ID assertion

“Configuring the client for identity assertion: collecting the authentication method” on page 1334

Signature

“Configuring the client for signature authentication: collecting the authentication information” on page 1341

LTPA

“Configuring the client for LTPA token authentication: collecting the authentication method information” on page 1358

Security response receiver binding configuration**Signing information**

“Configuring the client for response digital signature verification: choosing the verification method” on page 1293

Encryption information

“Configuring the client for response decryption: choosing a decryption method” on page 1323

Trust anchor

“Configuring trust anchors using an assembly tool” on page 1257

Certificate store list

“Configuring the client-side collection certificate store using an assembly tool” on page 1261

Key locators

“Configuring key locators using an assembly tool” on page 1268

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 52. 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

Use the Web services client editor within an assembly tool to include the binding information, that describes how to run the security specifications found in the extensions, in the client enterprise archive (EAR) file.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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. Click **Applications > Install New Application**.

You can access the administrative console by typing `http://localhost:port_number/ibm/console` in your Web browser unless you have changed the port number.

See also *Installing a new application*.

2. Click **Applications > Enterprise applications > application_name**.
3. Under Manage modules, click *URI_name*.
4. Under Web Services Security 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.

5. 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 1251
 - “Encryption information configuration settings” on page 1155
 - “Key locator configuration settings” on page 1118
 - “Login bindings configuration settings” on page 1307
 - Response receiver binding

- “Signing information configuration settings” on page 1136
- “Encryption information configuration settings” on page 1149
- “Trust anchor configuration settings” on page 1086
- “Collection certificate store configuration settings” on page 1090
- “Key locator configuration settings” on page 1118

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 53. Related configurations

Client configuration	Server configuration
Request sender	Request receiver
Response receiver	Response sender

Configuring the server security bindings using an assembly tool

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later applications.

Create an Enterprise JavaBeans (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 Java 2 Platform, Enterprise Edition (J2EE) application. You must import the Web services 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. Switch to the J2EE perspective. Click **Window > Open Perspective > 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 1284

Required confidentiality

“Configuring the server for request decryption: decrypting the message parts” on page 1315

Login config

Basic auth

“Configuring the server to handle BasicAuth authentication information” on page 1330

ID assertion

“Configuring the server to handle identity assertion authentication” on page 1335

Signature

“Configuring the server to support signature authentication” on page 1342

LTPA

“Configuring the server to handle LTPA token authentication information” on page 1359

Add received time stamp

“Configuring the server for request digital signature verification: Verifying the message parts” on page 1284

Response sender service configuration details

Details

“Configuring the server for response signing: digitally signing message parts” on page 1288

Integrity

“Configuring the server for response signing: digitally signing message parts” on page 1288

Confidentiality

“Configuring the server for response encryption: encrypting the message parts” on page 1318

Add created time stamp

“Configuring the server for response signing: digitally signing message parts” on page 1288

- 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 1285

Encryption Information

“Configuring the server for request decryption: choosing the decryption method” on page 1316

Trust Anchor

“Configuring trust anchors using an assembly tool” on page 1257

Certificate Store List

“Configuring the server-side collection certificate store using an assembly tool” on page 1263

Key Locators

“Configuring key locators using an assembly tool” on page 1268

Login Mapping**Basic auth**

“Configuring the server to validate BasicAuth authentication information” on page 1330

ID assertion

“Configuring the server to validate identity assertion authentication information” on page 1337

Signature

“Configuring the server to validate signature authentication information” on page 1343

LTPA

“Configuring the server to validate LTPA token authentication information” on page 1359

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 1289

Encryption information

“Configuring the server for response encryption: choosing the encryption method” on page 1319

Key locators

“Configuring key locators using an assembly tool” on page 1268

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 1295.

Configuring the server security bindings using the administrative console

Use the WebSphere Application Server administrative console to edit bindings for a Web service after these bindings are deployed on a server.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later applications.

Create an Enterprise JavaBeans (EJB) file Java archive (JAR) file or Web archive (WAR) file containing the security binding file (`ibm-webservices-bnd.xml`) and the security extension file (`ibm-webservices-ext.xml`). If this archive is acting as a client to a downstream service, you also need the client-side binding file (`ibm-webservicesclient-bnd.xml`) and the client-side extension file (`ibm-webservicesclient-ext.xml`). 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.
Type `http://localhost:port_number/ibm/console` in your Web browser unless you have changed the port number.
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 Manage modules, click *URI_name*.
4. Under Web Services Security 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

Extensible Markup Language (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 syntax to represent XML encrypted data, the information used to decrypt the data, and a list of encryption algorithms such as triple Data Encryption Standard (DES), Advanced Encryption Standard (AES), and Rivest-Shamir-Adleman algorithm (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>yMTEy0TA1M...</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 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 prepend 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: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6 and later applications.

Securing Web services for Version 5.x applications using XML encryption

XML encryption is one method that WebSphere Application Server provides to secure your Web services. It enables you to encrypt an XML element, the content of an XML element, or arbitrary data such as an XML document.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later applications.

WebSphere Application Server provides several different methods to secure your Web services. 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 1311.
 - b. “Configuring the client for request encryption: choosing the encryption method” on page 1311.
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 1315.
- b. “Configuring the server for request decryption: choosing the decryption method” on page 1316.

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 1318.
 - b. “Configuring the server for response encryption: choosing the encryption method” on page 1319.
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 1322.
- b. “Configuring the client for response decryption: choosing a decryption method” on page 1323.

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: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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 **Manage 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 use 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 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 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 Java 2 Platform, Enterprise Edition (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 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: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6 and later applications.

Request sender binding collection:

Use this page to specify the binding configuration to send request messages for Web services security.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later applications.

To view this administrative console page, complete the following steps:

1. Click **Applications > Enterprise applications > *application_name***.
2. Click **Manage modules > *URI_file_name***
3. Under Web Services Security 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 > Secure administration, applications, and infrastructure**.
2. Under the Java Authentication and Authorization Service field, select **Application logins** or **System logins**.

Configuring the client for request encryption: Encrypting the message parts

To configure the client for request encryption, specify which message parts to encrypt when configuring the client.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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 1295
- “Configuring the security bindings on a server acting as a client using the administrative console” on page 1298

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. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > 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

Usenametoken

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

To configure the client for request encryption, specify which encryption method to use when configuring the client.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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 1295
- “Configuring the security bindings on a server acting as a client using the administrative console” on page 1298

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. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > 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 Client Deployment Descriptor editor within the assembly tool.
6. Expand **Security request sender binding configuration > Encryption information**.
7. 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 1268 and “Configuring key locators using the administrative console” on page 1269.

8. **Optional:** Select **Show only FIPS Compliant Algorithms** if you only want the FIPS compliant algorithms to be shown in the **Data Encryption method algorithm** and **Key Encryption method algorithm** drop-down lists. Use this option if you expect this application to be run on a WebSphere Application Server that has set the **Use the United States Federal Information Processing Standard (FIPS) algorithms** option in the SSL certificate and key management panel of the WebSphere administrative console.

For more information, see “Configuring key locators using an assembly tool” on page 1268 and “Configuring key locators using the administrative console” on page 1269.

You must specify which parts of the request message to encrypt. See “Configuring the client for request encryption: Encrypting the message parts” on page 1311 if you have not previously specified this information.

Request receiver

The request receiver defines the security requirement of the SOAP message. The security handler on the request receiver side of the SOAP message enforces the security specifications that are defined in the IBM extension deployment descriptor (`ibm-webservices-ext.xmi`) and bindings (`ibm-webservices-bnd.xmi`).

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6 and later applications.

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: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later applications.

To view this administrative console page, complete the following steps:

1. Click **Applications > Enterprise applications > application_name**.
2. Click **Manage modules > URI_file_name**
3. Under Web Services Security 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 > Secure administration, applications, and infrastructure**.
2. Under the Java Authentication and Authorization Service field, select **Application logins** or **System logins**.

Configuring the server for request decryption: decrypting the message parts

Use the WS Extensions tab and the WS Binding configurations tab to specify which parts of the request message must be decrypted by the server.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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 1299
- “Configuring the server security bindings using the administrative console” on page 1302

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. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > 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.

username token

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 to 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

Use the WS Extensions tab and the WS Bindings tab to configure the Web services security extensions and Web services security bindings.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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 1299
- “Configuring the server security bindings using the administrative console” on page 1302

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. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > 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 1268 and “Configuring key locators using the administrative console” on page 1269.

8. **Optional:** Select **Show only FIPS Compliant Algorithms** if you only want the FIPS compliant algorithms to be shown in the Data Encryption method algorithm and Key Encryption method algorithm drop-down lists. Use this option if you expect this application to be run on a WebSphere Application Server that has set the **Use the United States Federal Information Processing Standard (FIPS) algorithms** option in the SSL certificate and key management panel of the administrative console for WebSphere Application Server.

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 1268 and “Configuring key locators using the administrative console” on page 1269 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 1315 if you have not previously specified this information.

Response sender

The response sender defines the security requirements of the SOAP response message. The security handler acts on the security constraints that are defined for the response in the IBM extension deployment descriptors.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6 and later applications.

The IBM extension deployment descriptors are 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 sent 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: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later applications.

To view this administrative console page, complete the following steps:

1. Click **Applications > Enterprise applications > *application_name***.
2. Click **Manage modules > *URI_file_name***
3. Under Web Services Security 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: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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 1299
- “Configuring the server security bindings using the administrative console” on page 1302

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. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > 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 1303.
7. Select the parts of the response that you want to encrypt by clicking **Add** and selecting **Bodytoken** or **Usenametoken**. The following information describes the message parts:

Bodycontent

User data portion of the message.

Usenametoken

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: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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 1299
- “Configuring the server security bindings using the administrative console” on page 1302

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. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > 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 1268 and “Configuring key locators using the administrative console” on page 1269.

8. Select **Show only FIPS Compliant Algorithms** if you only want the FIPS compliant algorithms to be shown in the Data Encryption method algorithm and Key Encryption method algorithm drop-down lists. Use this option if you expect this application to be run on a WebSphere Application Server that has set the **Use the United States Federal Information Processing Standard (FIPS) algorithms** option in the SSL certificate and key management panel of the administrative console for WebSphere Application Server.

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 1268 and “Configuring key locators using the administrative console” on page 1269.

You must specify which parts of the response message to encrypt. See “Configuring the server for response encryption: encrypting the message parts” on page 1318 if you have not previously specified this information.

Response receiver

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.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6 and later applications.

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-webservicesclient-bnd.xmi` file.

For example, the security requirement might have the response 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: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later applications.

To view this administrative console page, complete the following steps:

1. Click **Applications > Enterprise applications > *application_name***.
2. Click **Manage 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

To configure the client for response decryption, specify which response message parts to decrypt when configuring the client. The server response encryption and client response decryption configurations must match.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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 1295
- “Configuring the security bindings on a server acting as a client using the administrative console” on page 1298

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. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > 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 **Usenametoken**. The following information describes these message parts:

Bodycontent

The user data portion of the message.

Usenametoken

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

To configure the client for response decryption, specify which decryption method to use when the client decrypts the response message. The server response encryption and client response decryption configurations must match.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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 1295
- “Configuring the security bindings on a server acting as a client using the administrative console” on page 1298

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. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > 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 SOAP messages, see “XML encryption” on page 1303.
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 that is 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 by using the public key for CN=Alice, O=IBM, C=US. The client decrypts the secret key by using the private key for CN=Alice, O=IBM, C=US.

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 1268 and “Configuring key locators using the administrative console” on page 1269.

8. **Optional:** Select **Show only FIPS Compliant Algorithms** if you only want the FIPS compliant algorithms to be shown in the Data Encryption method algorithm and Key Encryption method algorithm drop-down lists. Use this option if you expect this application to be run on a WebSphere Application Server that has set the **Use the United States Federal Information Processing Standard (FIPS) algorithms** option in the SSL certificate and key management panel of the administrative console for WebSphere Application Server.

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 1268 and “Configuring key locators using the administrative console” on page 1269.

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 1322 if you have not previously specified this information.

Securing Web services for Version 5.x applications using basicauth authentication

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later applications.

WebSphere Application Server provides several different methods to secure your Web services. 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: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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 1326.

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. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > 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 1295
- “Configuring the security bindings on a server acting as a client using the administrative console” on page 1298

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 1327.

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 SOAP message. The callback handler that is used is specified in the `<LoginBinding>` element of the bindings file, `ibm-webservicesclient-bnd.xml`. 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.xml` 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: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6 and later applications.

Configuring the client for basic authentication: collecting the authentication information

The `BasicAuth` authentication method 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.

Note: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later applications.

For more information on `BasicAuth` authentication, see “`BasicAuth` authentication method” on page 1326.

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. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > 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 blank.

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.NonPromptCallbackHandler` 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 1325 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. 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: There is an important distinction between Version 5.x and Version 6.0.x applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x applications.

Configuring the server to handle BasicAuth authentication information

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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 1326.

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. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > 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: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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 1326.

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. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > 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 1330 if you have not previously specified this information.

Identity assertion

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6 and later 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 not 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: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later applications.

WebSphere Application Server provides several different methods to secure your Web services. 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 1334
2. Secure the server for identity assertion authentication.
 - a. “Configuring the server to handle identity assertion authentication” on page 1335
 - b. “Configuring the server to validate identity assertion authentication information” on page 1337

After completing these steps, you have secured your Web services using identity assertion authentication.

Configuring the client for identity assertion: specifying the method

You can 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.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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 1271.

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. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > 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 1332.
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 1341 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 1295 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

You can 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.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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 1271.

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. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > 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 1333 if you have not previously specified this information.

Configuring the server to handle identity assertion authentication

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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 1271. 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. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > 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 Authentication (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 or repository, which is configured on the **Security > Secure administration, applications, and infrastructure** panel in the administrative console for WebSphere Application Server. 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 1299.

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

The purpose of identity assertion is to assert the authenticated identity of the originating client from a Web service to a downstream Web service.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6 and later applications.

Use this task to configure identity assertion authentication. 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 1271. 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. After 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. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > 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

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`

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 implementation 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 1335 if you have not previously specified this information.

Securing Web services for version 5.x applications using signature authentication

WebSphere Application Server provides several different methods to secure your Web services. XML digital signature is one of these methods.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6 and later applications.

You can secure your Web services by 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 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 1341.
2. Secure the server for signature authentication.
 - a. “Configuring the server to support signature authentication” on page 1342.
 - b. “Configuring the server to validate signature authentication information” on page 1343.

After completing these steps, you have secured your Web services using signature authentication.

Configuring the client for signature authentication: specifying the method

Signature authentication, the use of an X.509 certificate to login on the target server, can be configured.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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 1340.

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. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > 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 1295.

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 1341 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 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 6.0.x and later. The information does not apply to Version 6 and later applications.

Configuring the client for signature authentication: collecting the authentication information

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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 1340.

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. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > 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 1265 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/xmlsig-core>

Canonicalization method algorithm

Canonicalizes the `SignedInfo` element before it is digested as part of the signature operation.

Digest method 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

Represents a reference to a key locator implementation. For more information on configuring key locators, see “Key locator” on page 1265.

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 token value type Uniform Resource Identifier (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 1339 if you have not previously specified this information.

Configuring the server to support signature authentication

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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 1340.

1. Launch an assembly tool. For more information on the assembly tools, see Assembly tools.
2. Switch to the Java 2 Platform, Enterprise Edition (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 1299.

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: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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 1340.

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. Switch to the Java 2 Platform, Enterprise Edition (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 1342 if you have not previously specified this information.

Overview of token types

Web services security defines two types of security tokens. The deployment descriptor extension file defines the types of tokens that the message can accept.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.1x and later applications.

The two types of security tokens that are defined by Web services security are:

- User name 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"/>
      </serverServiceConfig>
    </pcBinding>
  </wsDescExt>
</com.ibm.etools.webservice.wsext:WsExtension>
```

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.wsclientextension xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:com.ibm.etools.webservice.wsclientextension=
"http://www.ibm.com/websphere/appserver/schemas/5.0.2/wsclientextension.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"/>
</SecurityRequestSenderServiceConfig>
</ClientServiceConfig>
</PortQnameBinding>
</ServiceRefs>
</WsClientExtension>
```

User name 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` element that contains the user name is used in identity assertion. Identity assertion 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 6.0.x and later. The information does not apply to Version 6.0.x and later 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:

You can configure nonce for the application level using the WebSphere Application Server administrative console.

Important: The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6 and later applications.

Nonce is a randomly generated, cryptographic token used to thwart the hijacking of username tokens used with SOAP messages. Nonce is used in conjunction with the BasicAuth authentication method.

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.
Type `http://localhost:port_number/ibm/console` in your Web browser unless you have changed the port number.
2. Click **Applications > Enterprise applications > application_name**.
3. Under Manage modules, click *URI_name*.
4. Under Web Services Security 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 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 6.0.x and later. The information does not apply to Version 6.0.x and later 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.
Type `http://localhost:port_number/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, a Lightweight Third Party Authentication (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  
  8Xg26havepvmSJ8XxiACMihTJuh1t3uofsrbFQJ0qh5VcRvI+AKEaNmEgEV65jUYAC9  
  C/iwBBWk5U/6DiK7LfXcTT0ZPAd+3D3nCS0f+6tnqMou8EG9mtMeTKccz/pJVTZjaRS0
```

```
msu0sewsOKf1/WPsjW0bR/2g3NaVvBy18V1TFBpUbGFVGgzHRjBKAGo+ctk180n1VLIk
TUjt/XdYvEp0r6QoddGi4okjDGPyyoDxcvKZnReXww5Usoq1pfXwN4KG9as=
</wsse:BinarySecurityToken></wsse:Security></soapenv:Header>
```

As shown in the example, the token is Base64Binary encoded.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6 and later 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="elliottw1"
      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: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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 6.0.x and later. The information does not apply to Version 6 and later applications.

Securing Web services for version 5.x applications using a pluggable token

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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 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 1354

Configuring pluggable tokens using an assembly tool

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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 service that is based on the Java 2 Platform, Enterprise Edition (J2EE) specification. If not, see *Developing Web services applications* to create a Web service that is based on the J2EE specification. 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 1299
- “Configuring the server security bindings using the administrative console” on page 1302

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. Switch to the J2EE perspective. Click **Window > Open Perspective > 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 existing 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. This is a mandatory field. You must define the JAAS login configuration name in the WebSphere Application Server administrative console under **Security > Secure administration, applications, and infrastructure**. Under Authentication, click **Java Authentication and Authorization Service > Application logins**. For more information, see “Configuring programmatic logins for Java Authentication and Authorization Service” on page 552.
 - 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 the `com.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: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later applications.

Prior to completing these steps, it is assumed that you have already created a Web service that is based on the Java 2 Platform, Enterprise Edition (J2EE) specification. If not, see Developing Web services applications to create Web services that is based on the J2EE specification. 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 1299
- “Configuring the server security bindings using the administrative console” on page 1302

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 Manage modules, click *URI_name*. The *URI* is the Web services-enabled module.
 - a. Under Web Services Security 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 Web Services Security 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 programmatic logins for Java Authentication and Authorization Service” on page 552.
 - 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 1357
- “Configuring the client for LTPA token authentication: collecting the authentication method information” on page 1358
- “Configuring the server to handle LTPA token authentication information” on page 1359
- “Configuring the server to validate LTPA token authentication information” on page 1359

Pluggable token support

Important: There is an important distinction between Version 5.x and Version 6.0.x applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x 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

To configure Lightweight Third-Party Authentication (LTPA) token authentication, specify LTPA token authentication. Only configure the client for LTPA token authentication if the authentication mechanism configured in WebSphere Application Server is LTPA.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > 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 1361.

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” on page 1358 for more information.

Configuring the client for LTPA token authentication: collecting the authentication method information

To configure Lightweight Third-Party Authentication (LTPA) token authentication, collect the LTPA token authentication information. Do not configure the client for LTPA token authentication unless the authentication mechanism configured in WebSphere Application Server is LTPA.

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > 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** (Uniform Resource Identifier) 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 blank.

See “Configuring the client for LTPA token authentication: specifying LTPA token authentication” on page 1357 if you have not previously specified this information.

Configuring the server to handle LTPA token authentication information

Important: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > 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: There is an important distinction between Version 5.x and Version 6 and later applications. The information in this article supports Version 5.x applications only that are used with WebSphere Application Server Version 6.0.x and later. The information does not apply to Version 6.0.x and later 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. Switch to the Java 2 Platform, Enterprise Edition (J2EE) perspective. Click **Window > Open Perspective > 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 (Uniform Resource Identifier) 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 1359 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 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 6.0.x and later. The information does not apply to Version 6 and later applications.

Tuning Web services security for Version 5.x applications

The Java Cryptography Extension (JCE) policy 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 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.

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**
3. Click **IBM SDK Policy files**.

The Unrestricted JCE Policy files for SDK 1.4 Web site is displayed.

4. 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 **Other Downloads**.
3. Locate the 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.

Enabling security for WSIF

Here are the steps to be taken to enable the Web Services Invocation Framework (WSIF) to interact with a security manager.

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, complete the following step:

To load the WSDL file, enable the **FilePermission** attribute in the `was.policy` file. This permission is required when a WSDL file is referred to using the `file://` protocol.

Security API for the UDDI Version 3 registry

In UDDI Version 1 and Version 2 the security API was part of the Publish API. In UDDI Version 3 the security API is independent.

Use the UDDI Version 3 Client for Java (see UDDI Version 3 Client) to access programmatically all API calls and arguments supported by the UDDI Version 3 registry. You can also access the API functions graphically by using the UDDI user interface, however not all of the functions are available with this method.

The UDDI Version 3 registry supports the following Security 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 UDDI Version 3 API specification at http://www.uddi.org/pubs/uddi_v3.htm.

Chapter 15. 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.

Chapter 16. 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 [Enabling security for all application servers](#).

1. Configure authorization settings to access JMS resources owned by the embedded WebSphere JMS provider.

Authorization to access JMS resources owned by the embedded messaging subsystem is controlled by settings in the `app_server_root\config\cells\your_cell_name\integral-jms-authorizations.xml` file. The settings grant or deny authenticated users access to messaging 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 Java Message Service (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.

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.

1. Before installing WebSphere MQ, create and mount a file system called /var/mqm. 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 directory used to hold messaging logging files as well as the directories used to hold the messaging data. During the installation process these directories are secured with a default set of security attributes to prevent

unauthorised access. If you change these permissions you should ensure that the permissions specified give WebSphere MQ messaging the required access.

Configuring security for EJB 2.1 message-driven beans

Use this task to configure resource security and security permissions for Enterprise JavaBeans (EJB) Version 2.1 message-driven beans.

The association between connection factories, destinations, and message-driven beans is provided by listener ports. A listener port allows a deployed message-driven bean associated with the port to retrieve messages from the associated destination. You create listener ports by specifying their administrative name, the connection factory JNDI name, and the destination name (other optional properties are also configurable). Listener ports provide simplified administration of the associations between connection factories, destinations and message-driven beans, and are managed by a listener manager. The listener manager is provided by the message listener service to control and monitor the JMS listeners that are monitoring JMS destinations on behalf of deployed message-driven beans. For more information about listener ports, see [Message-driven beans - listener port components](#)

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 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.

Chapter 17. 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-impl.jar` and `activation-impl.jar` files. WebSphere Application Server uses the general-purpose JavaMail configuration files contained in the `mail-impl.jar` and `activation-impl.jar` files and does not put any mail configuration files in `<user.home>` and `<java.home>/lib`. File read permission for both the `mail-impl.jar` and `activation-impl.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}${/}lib${/}mail-impl.jar", "read";
    permission java.io.FilePermission "${was.install.root}${/}lib${/}activation-impl.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}${/}lib${/}mail-impl.jar", "read";
    permission java.io.FilePermission "${was.install.root}${/}lib${/}activation-impl.jar", "read";
    java.io.FilePermission "${java.home}${/}lib${/}mailcap", "read";
    permission java.io.FilePermission "${user.home}${/}lib${/}mailcap", "read";
};
```

Chapter 18. 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 a brief description of each WebSphere extension.

See the *Developing and deploying applications* PDF book for a brief description of each WebSphere extension.

In addition, now your applications can use the Eclipse extension framework. Your applications are extensible as soon as you define an extension point and provide the extension processing code for the extensible area of the application. You can also plug an application into another extensible application by defining an extension that adheres to the target extension point requirements. The extension point can find the newly added extension dynamically and the new function is seamlessly integrated in the existing application. It works on a cross Java 2 Platform, Enterprise Edition (J2EE) module basis.

The application extension registry uses the Eclipse plug-in descriptor format and application programming interfaces (APIs) as the standard extensibility mechanism for WebSphere applications. Developers that build WebSphere application modules can use WebSphere Application Server extensions to implement Eclipse tools and to provide plug-in modules to contribute functionality such as actions, tasks, menu items, and links at predefined extension points in the WebSphere application. Read the information about Application extension registry in the *Developing and deploying applications* PDF book.

Scheduler

Securing scheduler tasks

Scheduled tasks are protected using application isolation and administrative roles. This topic describes how to secure scheduler tasks.

If a task is created using a Java 2 Platform, Enterprise Edition (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 administrative 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. Enable security for all application servers.
2. Manage schedulers.

Chapter 19. Tuning, hardening, and maintaining

After you have installed WebSphere Application Server, there are several considerations for tuning, strengthening, and maintaining your security configuration.

The following topics are covered in this section:

- Tuning security configurations
- Hardening security configurations
- Changing keys and passwords
- Securing passwords in files

Tuning security configurations

Performance issues typically involve trade-offs between function and speed. Usually, the more function and the more processing that are 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 can disable Secure Sockets Layer (SSL). If you have a lot of users, can they be mapped to groups and then associated to your Java 2 Platform, Enterprise Edition (J2EE) roles? These questions are things to consider when designing your security infrastructure.

- Consider the following recommendations for tuning general security.
 - 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.
 - Consider increasing the cache and token timeout if you feel your environment is secure enough. By increasing these values, 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 timeout 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 269 for a list of these properties.
 - 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.
 - Use the wsadmin script to complete the access IDs for all the users and 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. WebSphere Application Server maps user and group names to unique access IDs in the authorization table. The exact format of the access ID depends on the repository. The access ID can only be determined during and after application deployment. Authorization tables created during assembly time do not have the proper access IDs. See Commands for the AdminApp object for more information about how to update access IDs.
 - Consider tuning the Object Request Broker (ORB) because it is a factor in enterprise bean performance with or without security enabled. Refer to the ORB tuning guidelines topic.
 - If using SSL, enable the SSL session tracking mechanism option as described in the article, Session management settings.
 - In some cases, using the unrestricted Java Cryptography Extension (JCE) policy file can improve performance. Refer to the article, Tuning Web services security.
- Consider the following steps to tune Common Secure Interoperability version 2 (CSIv2).

- Consider using Secure Sockets Layer (SSL) client certificates instead of a user ID and password to authenticate Java clients. Because you are already making the SSL connection, using mutual authentication adds little overhead while it removes the service context that contains the user ID and password completely.
- 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.
- Consider putting only 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, the identity token is trusted. The SSL connection trusts the server through client certificate authentication.
- Ensure that stateful sessions are enabled for CSIv2. This is the default, but requires authentication only on the first request and on any subsequent token expirations.
- **V6.0.x** If you are communicating only with WebSphere Application Server Version 5 or higher 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.

Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

- Consider the following steps to tune Lightweight Directory Access Protocol (LDAP) authentication.
 1. In the administration console, click **Security > Secure administration, applications, and infrastructure**.
 2. Under User account repository, click the **Available realm definitions** drop-down list, select **Standalone LDAP registry** and click **Configure**.
 3. Select the **Ignore case for authorization** option in the standalone LDAP registry configuration, when case-sensitivity is not important.
 4. Select the **Reuse connection** option.
 5. Use the cache features that your LDAP server supports.
 6. Choose either the IBM Tivoli Directory Server or SecureWay directory type, if you are using an IBM Tivoli Directory Server. The IBM Tivoli Directory Server yields improved performance because it is programmed to use the new group membership attributes to improve group membership searches. However, authorization must be case insensitive to use IBM Tivoli Directory Server.
 7. 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 can increase performance in group membership lookup. However, use **Role** only for group mechanisms.
- Consider the following steps to tune Web authentication.
 - Increase the cache and token timeout values 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 that are already created. A disadvantage of increasing the token timeout 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 269 for a list of these properties.
 - Enable single sign-on (SSO). To configure SSO, click **Security > Secure administration, applications, and infrastructure**. Under Web security, click **Single sign-on (SSO)**.
SSO is only available when you configure **LTPA** as the authentication mechanism in the Authentication mechanisms and expiration panel. Although you can select Simple WebSphere Authentication Mechanism (SWAM) as the authentication mechanism on the Authentication mechanisms and expiration panel, SWAM is deprecated in Version 6.1 and does not support SSO. 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. Some situations exist where SSO is not a desirable and you do not want to use it in those situations.

- Disable or enabling the **Web Inbound Security Attribute Propagation** option on the Single sign-on (SSO) panel if the function is not required. In some cases, having the function enabled can improve 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 can improve performance. This improvement is most likely when the user registry calls do not take considerable resources.
- Consider the following steps to tune authorization.
 - Map your users to groups in the user registry. Associate the groups with your Java 2 Platform, Enterprise Edition (J2EE) roles. This association greatly improves performance when the number of users increases.
 - Judiciously assign method-permissions for enterprise beans. For example, you can use an asterisk (*) to indicate all the 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 that is required to load the deployment descriptor. It also reduces the search time during method-permission match for the enterprise beans method.
 - Judiciously assign security-constraints for servlets. For example, you can use the *.jsp URL pattern to apply the same authentication data constraints to indicate all JavaServer Pages (JSP) files. For a given URL, the exact match in the deployment descriptor takes precedence over the longest path match. Use the *.jsp, *.do, *.html extension match if no exact matches exist and longest path matches exist for a given URL in the security constraints.

You always have 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, create a benchmark before making any change to ensure that 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 of configuration for your environment. Continue to run benchmarks if anything changes in your environment, to help determine the impact of these changes.

Secure Sockets Layer performance tips

Use this page to learn about Secure Sockets Layer (SSL) performance tips. Be sure to consider that performance issues typically involve trade-offs between function and speed. Usually, the more function and the more processing that are involved, the slower the performance.

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 Transmission Control Protocol/Internet Protocol (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.

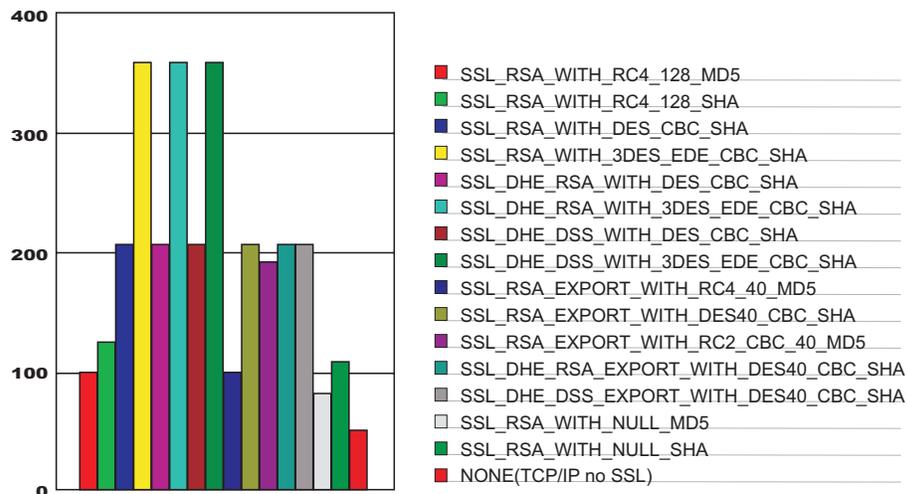
Note: HTTP Transports have been deprecated. For instructions on how to set a maximum keep alive value for channel based configurations, see HTTP transport channel settings.

- 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, Data Encryption Standard (DES) and triple-strength DES (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 cryptographic 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 Secure Hash Algorithm (SHA) are the two hash algorithms used to provide data integrity. MD5 is generally 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

Use the following procedures to tune the performance, without compromising your security settings.

Enabling security decreases performance. The following tuning parameters provide ways to minimize this performance impact.

- Disable security on any application servers that do not need security. You can disable security in the administrative console by clicking **Security > Secure administration, applications, and infrastructure** and deselecting the **Enable administrative security** option.
- Fine-tune the **Authentication cache timeout** value on the Authentication mechanisms and expiration panel in the administrative console. For more information, see the “Secure administration, applications, and infrastructure settings” on page 75 topic.
- Configure the security cache properties. For more information, see the “Security cache properties” on page 269 topic.
- Enable the **Enable SSL ID tracking** option on the Session management panel in the administrative console. For more information, see the Session management settings topic.
- 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. For more information, see the “Tuning Web services security for Version 6.1 applications” on page 1223 topic.

- Read the Secure Sockets Layer performance tips and “Tuning security configurations” on page 1377 topics for more information.

Hardening security configurations

There are several methods that you can use to protect the WebSphere Application Server infrastructure and applications from different forms of attack. Several different techniques can help with multiple forms of attack. Sometimes a single attack can leverage multiple forms of intrusion to achieve the end goal.

For example, in the simplest case, network sniffing can be used to obtain passwords and those passwords can then be used to mount an application-level attack. The following issues are discussed in IBM WebSphere Developer Technical Journal: WebSphere Application Server V5 advanced security and system hardening:

- Take preventative measures to protect the infrastructure.
- Make applications less vulnerable to attack.

Securing passwords in files

Password encoding and encryption deters the casual observation of passwords in server configuration and property files.

The following topics are covered in this section:

- Password encoding and encryption
- Encoding passwords in files
- Enabling custom password encryption

Encoding password in files

Use the **PropFilePasswordEncoder** utility to encode your passwords in the files. WebSphere Application Server does not provide a utility for decoding the passwords.

WebSphere Application Server contains several encoded passwords that are not encrypted. WebSphere Application Server provides the **PropFilePasswordEncoder** utility, which you can use to encode these passwords. However, the utility does not encode passwords that are contained within XML or XMI files. Instead, WebSphere Application Server automatically encodes the passwords in the following XML or XMI files.

Table 54. XML and XMI files that contain encoded passwords

File name	Additional information
<code>profile_root/config/cells/cell_name/security.xml</code>	The following fields contain encoded passwords: <ul style="list-style-type: none"> • LTPA password • JAAS authentication data • User registry server password • LDAP user registry bind password • Keystore password • Truststore password • Cryptographic token device password
<code>war/WEB-INF/ibm_web_bnd.xml</code>	Specifies the passwords for the default basic authentication for the resource-ref bindings within all the descriptors, except in the Java cryptography architecture

Table 54. XML and XMI files that contain encoded passwords (continued)

File name	Additional information
ejb_jar/META-INF/ibm_ejbjar_bnd.xml	Specifies the passwords for the default basic authentication for the resource-ref bindings within all the descriptors, except in the Java cryptography architecture
client_jar/META-INF/ibm-appclient_bnd.xml	Specifies the passwords for the default basic authentication for the resource-ref bindings within all the descriptors, except in the Java cryptography architecture
ear/META-INF/ibm_application_bnd.xml	Specifies the passwords for the default basic authentication for the run as bindings within all the descriptors
<i>profile_root/config/cells/cell_name</i> <i>/nodes/node_name/servers/</i> <i>server_name/security.xml</i>	The following fields contain encoded passwords: <ul style="list-style-type: none"> Keystore password Truststore password Cryptographic token device password Session persistence password DRS client data replication password
<i>profile_root/config/cells/cell_name</i> <i>/nodes/node_name/servers/</i> <i>server_name/resources.xml</i>	The following fields contain encoded passwords: <ul style="list-style-type: none"> WAS40Datasource password mailTransport password mailStore password MQQueue queue mgr password
<ul style="list-style-type: none"> <i>profile_root/config/cells/cell_name</i> <i>/ws-security.xml</i> <i>profile_root/config/cells/cell_name</i> <i>/nodes/node_name/servers/server_name/ws-security</i> 	
ibm-webservices-bnd.xmi	
ibm-webservicesclient-bnd.xmi	

You can use the **PropFilePasswordEncoder** utility to encode the passwords that are located in the following files.

Table 55. Files that you can encode using the PropFilePasswordEncoder utility

File name	Additional information
<i>app_server_root</i> <i>/properties/sas.client.props</i>	Specifies the passwords for the following files: <ul style="list-style-type: none"> com.ibm.ssl.keyStorePassword com.ibm.ssl.trustStorePassword com.ibm.CORBA.loginPassword
<i>app_server_root</i> <i>/properties/soap.client.props</i>	Specifies passwords for: <ul style="list-style-type: none"> com.ibm.ssl.keyStorePassword com.ibm.ssl.trustStorePassword com.ibm.SOAP.loginPassword
<i>app_server_root</i> <i>/properties/sas.tools.properties</i>	Specifies passwords for: <ul style="list-style-type: none"> com.ibm.ssl.keyStorePassword com.ibm.ssl.trustStorePassword com.ibm.CORBA.loginPassword

Table 55. Files that you can encode using the PropFilePasswordEncoder utility (continued)

File name	Additional information
<i>app_server_root</i> /properties/sas.stdclient.properties	Specifies passwords for: <ul style="list-style-type: none"> • com.ibm.ssl.keyStorePassword • com.ibm.ssl.trustStorePassword • com.ibm.CORBA.loginPassword
<i>app_server_root</i> /properties/wssserver.key	

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

1. Access the file using a text editor and type over the encoded password. The new password is shown is no longer encoded and must be re-encoded.
2. Use the PropFilePasswordEncoder.bat or the PropFilePasswordEncode.sh file in the *app_server_root/profiles/profile_name/bin* directory to encode the password again.

V6.0.x If you are encoding the SAS properties files again, type: PropFilePasswordEncoder "*file_name*" -sas and the PropFilePasswordEncoder file encodes the known SAS properties.

Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

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

where:

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

Note: Only the password should be encoded in this file using the **PropFilePasswordEncoder** tool. Use the **PropFilePasswordEncoder** utility to encode WebSphere Application Server password files only. The utility cannot encode passwords that are contained in XML files or other files that contain open and close tags.

If you reopen the affected files, the passwords are encoded. WebSphere Application Server does not provide a utility for decoding the passwords.

PropFilePasswordEncoder command reference

The **PropFilePasswordEncoder** command encodes passwords that are located in plain text property files. This command encodes both Secure Authentication Server (SAS) property files and non-SAS property files. After you encode the passwords, a decoding command does not exist.

To encode passwords, you must run this command from the directory:

- **V6.0.x** *app_server_root/bin*

Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

Syntax

The command syntax is as follows:

```
PropFilePasswordEncoder "file_name"
```

Parameters

The following option is available for the **PropFilePasswordEncoder** command:

V6.0.x -SAS

This parameter is required if you are encoding passwords in the `sas.client.props` file.

-help or -?

If you specify this parameter, the script ignores all other parameters and displays usage text.

V6.0.x The following examples demonstrate the correct syntax:

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

Enabling custom password encryption

After creating the server profile, perform this task to better protect passwords contained in configuration.

Create your custom class for encrypting passwords. For more information, see “Plug point for custom password encryption” on page 639.

Complete the following steps to enable custom password encryption.

1. Add the following system properties for every server and client process. For server processes, update the `server.xml` file for each process. Add these properties as a `genericJvmArgument` argument preceded by a **-D** prefix.

```
com.ibm.wsspi.security.crypto.customPasswordEncryptionClass=
    com.acme.myPasswordEncryptionClass
com.ibm.wsspi.security.crypto.customPasswordEncryptionEnabled=true
```

Tip: If the custom encryption class name is

`com.ibm.wsspi.security.crypto.CustomPasswordEncryptionImpl`, it is automatically enabled when this class is present in the classpath. Do not define the system properties that are listed previously when the custom implementation has this package and class name. To disable encryption for this class, you must specify `com.ibm.wsspi.security.crypto.customPasswordEncryptionEnabled=false` as a system property.

2. Add the Java archive (JAR) file containing the implementation class to the `app_server_root/classes` directory so that the WebSphere Application Server runtime can load the file.
3. Restart all server processes.
4. Edit each configuration document that contains a password and save the configuration. All password fields are then run through the **WSEncoderDecoder** utility, which calls the plug point when it is enabled. The `{custom:alias}` tags are displayed in the configuration documents. The passwords, even though they are encrypted, are still Base64-encoded. They seem similar to encoded passwords, except for the tags difference.
5. Encrypt any passwords that are in client-side property files using the **PropsFilePasswordEncoder** (`.bat` or `.sh`) utility. This utility requires that the properties listed previously are defined as system properties in the script to encrypt new passwords instead of encoding them.
6. To decrypt passwords from client Java virtual machines (JVMs), add the properties listed previously as system properties for each client utility.
7. Ensure that all nodes have the custom encryption classes in their class paths prior to enabling this function.

Custom password encryption is enabled.

If custom password encryption fails or is no longer required, see “Disabling custom password encryption.”

Disabling custom password encryption

If custom password encryption fails or is no longer required, perform this task to disable custom password encryption.

Enable custom password encryption.

Complete the following steps to disable custom password encryption.

1. Change the `com.ibm.wsspi.security.crypto.customPasswordEncryptionEnabled` property to be `false` in the `security.xml` file, but leave the `com.ibm.wsspi.security.crypto.customPasswordEncryptionClass` property configured. Any passwords in the model that still have the `{custom:alias}` tag are decrypted by using the customer password encryption class.
2. If an encryption key is lost, any passwords that are encrypted with that key cannot be retrieved. To recover a password, retype the password in the password field in plaintext and save the document. The new password must be written out using encoding with the `{xor}` tag with scripting or from the administrative console.

```
com.ibm.wsspi.security.crypto.customPasswordEncryptionClass=  
    com.acme.myPasswordEncryptionClass  
com.ibm.wsspi.security.crypto.customPasswordEncryptionEnabled=false
```

3. Restart all processes to make the changes effective.
4. Edit each configuration document that contains an encrypted password and save the configuration. All password fields are then run through the **WSEncoderDecoder** utility, which calls the plug point in the presence of the `{custom:alias}` tag. The `{xor}` tags display in the configuration documents again after the documents are saved.
5. Decrypt and encode any passwords that are in client-side property files using the **PropsFilePasswordEncoder** (.bat or .sh) utility. If the encryption class is specified, but custom encryption is disabled, running this utility converts the encryption to encoding and causes the `{xor}` tags to display again.
6. Disable custom password encryption from the client Java virtual machines (JVMs) by adding the system properties listed previously to all client scripts. This action enables the code to decrypt passwords, but this action is not used to encrypt them again. The `{xor}` algorithm becomes the default for encoding. Leave the custom password encryption class defined for a time in case any encrypted passwords still exist in the configuration.

Custom password encryption is disabled.

Chapter 20. Troubleshooting security configurations

The following topics help to troubleshoot specific problems that are related to configuring and enabling security configurations.

Refer to Security components troubleshooting tips for instructions on how to troubleshoot errors that are related to security.

Refer to SPNEGO TAI troubleshooting tips for instructions on how to troubleshoot errors that are related to diagnosing Simple and Protected GSS-API Negotiation (SPNEGO) trust association interceptor (TAI) problems and exceptions.

- Errors when configuring or enabling security
- Errors after enabling security
- Access problems after enabling security
- Errors after configuring or enabling Secure Sockets Layer
- Errors configuring Secure Sockets Layer encrypted access
- Single sign-on configuration troubleshooting tips
- Authorization provider troubleshooting tips

Security components troubleshooting tips

This document explains basic resources and steps for diagnosing security-related issues in WebSphere Application Server.

Basic resources and steps for diagnosing security-related issues in WebSphere Application Server include:

- What “Log files” on page 1388 to look at and what to look for in them.
- What to look at and what to look for “Using SDSF” on page 1389.
- “General approach for troubleshooting security-related issues” on page 1390 to isolating and resolving security problems.
- When and how to “Trace security” on page 1394.
- An overview and table of “CSIv2 CORBA minor codes” on page 1395.

The following security-related problems are addressed elsewhere in the information center:

- Errors and access problems after enabling security
After enabling security, a degradation in performance is realized. For more information about using unrestricted policy files, see the Enabling security for the realm section of the *Securing applications and their environment* PDF book.
- 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 is 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.

V6.0.x For an overview of WebSphere Application Server security components such as Secure Authentication Services (SAS) and how they work in a distributed or an iSeries environment, refer to the *Securing applications and their environment* PDF book.

Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

Log files

When troubleshooting the security component, browse the Java Virtual Machine (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 CWSA0001I: Security configuration initialized.
SASRas      A CWSA0002I: Authentication protocol: CSIV2/IBM
SASRas      A CWSA0003I: Authentication mechanism: SWAM
SASRas      A CWSA0004I: Principal name: MYHOSTNAME/aServerID
SASRas      A CWSA0005I: SecurityCurrent registered.
SASRas      A CWSA0006I: Security connection interceptor initialized.
SASRas      A CWSA0007I: Client request interceptor registered.
SASRas      A CWSA0008I: Server request interceptor registered.
SASRas      A CWSA0009I: 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 CWSA0001I: Security configuration initialized.
SASRas      A CWSA0002I: Authentication protocol: CSIV2/IBM
SASRas      A CWSA0003I: Authentication mechanism: SWAM
SASRas      A CWSA0004I: Principal name: MYHOSTNAME/aServerID
SASRas      A CWSA0005I: SecurityCurrent registered.
SASRas      A CWSA0006I: Security connection interceptor initialized.
SASRas      A CWSA0007I: Client request interceptor registered.
SASRas      A CWSA0008I: Server request interceptor registered.
SASRas      A CWSA0009I: 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 Lightweight Directory Access Protocol (LDAP) has been specified as the security mechanism, but the LDAP keys have not been properly configured:

```
SASRas      A CWSA0001I: Security configuration initialized.
SASRas      A CWSA0002I: Authentication protocol: CSIV2/IBM
SASRas      A CWSA0003I: Authentication mechanism: LTPA
SASRas      A CWSA0004I: Principal name: MYHOSTNAME/anID
SASRas      A CWSA0005I: SecurityCurrent registered.
SASRas      A CWSA0006I: Security connection interceptor initialized.
SASRas      A CWSA0007I: Client request interceptor registered.
SASRas      A CWSA0008I: Server request interceptor registered.
SASRas      A CWSA0009I: 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 Secure Sockets Layer (SSL) configuration might lead to the following message. 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 certificate authority (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
```

Using SDSF

When troubleshooting the security component, use System Display and Search Facility (SDSF) to browse logs for the server that hosts the resource you are trying to access. The following sample of messages helps you see from a server in which the security service has started successfully:

```
+BBOM0001I com_ibm_authMechanisms_type_OID: No OID for this mechanism.
+BBOM0001I com_ibm_security_SAF_unauthenticated: WSGUEST.
+BBOM0001I com_ibm_security_SAF_EJBROLE_Audit_Messages_Suppress: 0.
+BBOM0001I com_ibm_ws_logging_zos_errorlog_format_cbe: NOT SET, 280
DEFAULT=0.
+BBOM0001I com_ibm_CSI_performClientAuthenticationRequired: 0.
+BBOM0001I com_ibm_CSI_performClientAuthenticationSupported: 1.
+BBOM0001I com_ibm_CSI_performTransportAssocSSLTLSRequired: 0.
+BBOM0001I com_ibm_CSI_performTransportAssocSSLTLSSupported: 1.
+BBOM0001I com_ibm_CSI_rmiInboundPropagationEnabled: 1.
+BBOM0001I com_ibm_CSI_rmiOutboundLoginEnabled: 0.
+BBOM0001I com_ibm_CSI_rmiOutboundPropagationEnabled: 1.
+BBOM0001I security_assertedID_IBM_accepted: 0.
+BBOM0001I security_assertedID_IBM_sent: 0.
+BBOM0001I security_disable_daemon_ssl: NOT SET, DEFAULT=0.
+BBOM0001I security_sslClientCerts_allowed: 0.
+BBOM0001I security_sslKeyring: NOT SET.
+BBOM0001I security_zOS_domainName: NOT SET.
+BBOM0001I security_zOS_domainType: 0.
+BBOM0001I security_zSAS_ssl_repertoire: SY1/DefaultIIOPSSL.
+BBOM0001I security_EnableRunAsIdentity: 0.
+BBOM0001I security_EnableSyncToOSThread: 0.
+BBOM0001I server_configured_system_name: SY1.
+BBOM0001I server_generic_short_name: BBOC001.
+BBOM0001I server_generic_uuid: 457
*** Message beginning with BB000222I apply to Java within ***
*** WebSphere Application Server Security ***
+BB000222I: SECJ6004I: Security Auditing is disabled.
+BB000222I: SECJ0215I: Successfully set JAAS login provider 631
configuration class to com.ibm.ws.security.auth.login.Configuration.
```

```
+BB000222I: SECJ0136I: Custom 632
Registry:com.ibm.ws.security.registry.zOS.SAFRegistryImpl has been initialized
+BB000222I: SECJ0157I: Loaded Vendor AuthorizationTable: 633
com.ibm.ws.security.core.SAFAuthorizationTableImpl
```

General approach for troubleshooting security-related issues

When troubleshooting security-related problems, the following questions are very helpful:

Does the problem occur when security is disabled?

This question 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 seem to initialize properly?

A lot of security code is visited during initialization. So you can see problems there first if the problem is configuration related.

The following sequence of messages that are generated in the SystemOut.log indicate normal code initialization of an application server. This sequence varies 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: 2005/05/06 17:27:31.539 01 t=8E96E0 c=UNK key=P8 (13007002)
  ThreadId: 0000000a
  FunctionName: printProperties
  SourceId: com.ibm.ws390.orb.CommonBridge
  Category: AUDIT
  ExtendedMessage: BB0J0077I java.security.policy =
    /WebSphere/V6R1M0/AppServer/profiles/default/pr
Trace: 2005/05/06 17:27:31.779 01 t=8E96E0 c=UNK key=P8 (13007002)
  ThreadId: 0000000a
  FunctionName: printProperties
  SourceId: com.ibm.ws390.orb.CommonBridge
  Category: AUDIT
  ExtendedMessage: BB0J0077I java.security.auth.login.config =
    /WebSphere/V6R1M0/AppServer/profiles/default/pr
Trace: 2005/05/06 17:27:40.892 01 t=8E96E0 c=UNK key=P8 (13007002)
  ThreadId: 0000000a
  FunctionName: com.ibm.ws.security.core.SecurityDM
  SourceId: com.ibm.ws.security.core.SecurityDM
  Category: INFO
  ExtendedMessage: BB000222I: SECJ0231I: The Security component's FFDC
    Diagnostic Module com.ibm.ws.security.core.Secur
red successfully: true.
```

Trace: 2005/05/06 17:27:40.892 01 t=8E96E0 c=UNK key=P8 (0000000A)
 Description: Log Boss/390 Error
 from filename: ./bborjtr.cpp
 at line: 932
 error message: BB000222I: SECJ0231I: The Security component's FFDC
 Diagnostic Module com.ibm.ws.security.core.Securit
 d successfully: true.

Trace: 2005/05/06 17:27:41.054 01 t=8E96E0 c=UNK key=P8 (13007002)
 ThreadId: 0000000a
 FunctionName: com.ibm.ws.security.audit.AuditServiceImpl
 SourceId: com.ibm.ws.security.audit.AuditServiceImpl
 Category: AUDIT
 ExtendedMessage: BB000222I: SECJ6004I: Security Auditing is disabled.

Trace: 2005/05/06 17:27:41.282 01 t=8E96E0 c=UNK key=P8 (13007002)
 ThreadId: 0000000a
 FunctionName: com.ibm.ws.security.core.distSecurityComponentImpl
 SourceId: com.ibm.ws.security.core.distSecurityComponentImpl
 Category: INFO
 ExtendedMessage: BB000222I: SECJ0309I: Java 2 Security is disabled.

Trace: 2005/05/06 17:27:41.282 01 t=8E96E0 c=UNK key=P8 (0000000A)
 Description: Log Boss/390 Error
 from filename: ./bborjtr.cpp
 at line: 932
 error message: BB000222I: SECJ0309I: Java 2 Security is disabled.

Trace: 2005/05/06 17:27:42.239 01 t=8E96E0 c=UNK key=P8 (13007002)
 ThreadId: 0000000a
 FunctionName: com.ibm.ws.security.auth.login.Configuration
 SourceId: com.ibm.ws.security.auth.login.Configuration
 Category: AUDIT
 ExtendedMessage: BB000222I: SECJ0215I: Successfully set JAAS login
 provider configuration class to com.ibm.ws.securit
 Configuration.

Trace: 2005/05/06 17:27:42.253 01 t=8E96E0 c=UNK key=P8 (13007002)
 ThreadId: 0000000a
 FunctionName: com.ibm.ws.security.core.distSecurityComponentImpl
 SourceId: com.ibm.ws.security.core.distSecurityComponentImpl
 Category: INFO
 ExtendedMessage: BB000222I: SECJ0212I: WCCM JAAS configuration information
 successfully pushed to login provider clas

Trace: 2005/05/06 17:27:42.254 01 t=8E96E0 c=UNK key=P8 (0000000A)
 Description: Log Boss/390 Error
 from filename: ./bborjtr.cpp
 at line: 932
 error message: BB000222I: SECJ0212I: WCCM JAAS configuration information
 successfully pushed to login provider class.

Trace: 2005/05/06 17:27:42.306 01 t=8E96E0 c=UNK key=P8 (13007002)
 ThreadId: 0000000a
 FunctionName: com.ibm.ws.security.core.distSecurityComponentImpl
 SourceId: com.ibm.ws.security.core.distSecurityComponentImpl
 Category: INFO
 ExtendedMessage: BB000222I: SECJ0240I: Security service initialization
 completed successfully

Trace: 2005/05/06 17:27:42.306 01 t=8E96E0 c=UNK key=P8 (0000000A)
 Description: Log Boss/390 Error
 from filename: ./bborjtr.cpp
 at line: 932
 error message: BB000222I: SECJ0240I: Security service initialization
 completed successfully

Trace: 2005/05/06 17:27:42.952 01 t=8E96E0 c=UNK key=P8 (13007002)
 ThreadId: 0000000a
 FunctionName: com.ibm.ws.objectpool.ObjectPoolService
 SourceId: com.ibm.ws.objectpool.ObjectPoolService
 Category: INFO
 ExtendedMessage: BB000222I: OBPL0007I: Object Pool Manager service
 is disabled.

Trace: 2005/05/06 17:27:53.512 01 t=8E96E0 c=UNK key=P8 (13007002)
 ThreadId: 0000000a
 FunctionName: com.ibm.ws.security.registry.UserRegistryImpl
 SourceId: com.ibm.ws.security.registry.UserRegistryImpl
 Category: AUDIT
 ExtendedMessage: BB000222I: SECJ0136I: Custom
 Registry:com.ibm.ws.security.registry.zOS.SAFRegistryImpl
 has been init

Trace: 2005/05/06 17:27:55.229 01 t=8E96E0 c=UNK key=P8 (13007002)
 ThreadId: 0000000a
 FunctionName: com.ibm.ws.security.role.PluggableAuthorizationTableProxy
 SourceId: com.ibm.ws.security.role.PluggableAuthorizationTableProxy
 Category: AUDIT

```

ExtendedMessage: BB000222I: SECJ0157I: Loaded Vendor
      AuthorizationTable: com.ibm.ws.security.core.SAFAuthorizationTab
Trace: 2005/05/06 17:27:56.481 01 t=8E96E0 c=UNK key=P8 (13007002)
ThreadId: 0000000a
FunctionName: com.ibm.ws.security.core.distSecurityComponentImpl
SourceId: com.ibm.ws.security.core.distSecurityComponentImpl
Category: INFO
ExtendedMessage: BB000222I: SECJ0243I: Security service started successfully
Trace: 2005/05/06 17:27:56.481 01 t=8E96E0 c=UNK key=P8 (0000000A)
Description: Log Boss/390 Error
from filename: ./bborjtr.cpp
at line: 932
error message: BB000222I: SECJ0243I: Security service started successfully
Trace: 2005/05/06 17:27:56.482 01 t=8E96E0 c=UNK key=P8 (13007002)
ThreadId: 0000000a
FunctionName: com.ibm.ws.security.core.distSecurityComponentImpl
SourceId: com.ibm.ws.security.core.distSecurityComponentImpl
Category: INFO
ExtendedMessage: BB000222I: SECJ0210I: Security enabled true
Trace: 2005/05/06 17:27:56.483 01 t=8E96E0 c=UNK key=P8 (0000000A)
Description: Log Boss/390 Error
from filename: ./bborjtr.cpp
at line: 932
error message: BB000222I: SECJ0210I: Security enabled true

```

Is there a stack trace or exception printed in the system log file?

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 can actually be misleading. When support analyzes a stack trace, they can request additional trace if it is not clear what the problem is. If it seems to be security-related and the solution cannot be determined from the stack trace or problem description, you are 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 because the behavior of the object request broker (ORB), among other components, is different between the two. When a remote method invocation takes place, an entirely different security code path is entered.
- **V6.0.x** When you know that the problem involves two or more servers, the techniques of troubleshooting change. You need to trace all the servers involved simultaneously so that the trace shows the client and server sides of the problem. 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 Secure Authentication Services (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 failure 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 deployment and to the caller's identity who is accessing the method and the roles that are required by the method.

Is this a Web or EJB request?

Web requests have a completely different code path than Enterprise JavaBeans (EJB) requests. Different security features exist for Web requests than for EJB requests, requiring a completely different body of knowledge to resolve. For example, when using the Lightweight Third-Party Authentication (LTPA) authentication mechanism, the single sign-on feature (SSO) is available for Web requests but not for EJB requests. Web requests involve HTTP header information that is 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 can be involved in the problem and all require consideration 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 Workload Manager (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.

V6.0.x 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)?

SSL is a totally distinct separate layer of security. Troubleshooting SSL problems is usually separate from troubleshooting authentication and authorization problems, and you have many considerations. Usually, SSL problems are first-time setup problems because the configuration can be difficult. Each client must contain the signer certificate of the server. During mutual authentication, each server must contain the client's signer certificate. Also, there can be protocol differences (SSLv3 vs. Transport Layer Security (TLS)), and listener port problems related to stale Interoperable Object References (IORs), that is IORs from a server, that reflect the port prior to the server restarting.

For SSL problems, sometimes you get a request for an SSL trace to determine what is happening with the SSL handshake. The SSL handshake is the process that occurs when a client opens a socket to a server. If anything goes wrong with the key exchange, cipher exchange, and so on, the handshake fails and the socket is not valid. Tracing JSSE (the SSL implementation that is 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 the system property to the generic JVM arguments property of the JVM settings page. For more information on this task, refer to Java virtual machine settings section of the *Administering applications and their environment* PDF book.
- Turn on ORB trace as well.
- Recreate the problem.

The `SystemOut.log` of both processes contain the JSSE trace. You can find trace similar to the following example:

```
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.e03ae78375>
>> handshakeV3 type = 20
>> finished.
<< sendChangeCipherSpec.
<< sendFinished.

```

Trace security

The classes that implement WebSphere Application Server security are:

- com.ibm.ws.security.*
- com.ibm.websphere.security.*
- com.ibm.WebSphereSecurityImpl.*
- SASRas
- com.ibm.ws.wim.* for tracing with a Virtual Member Manager (VMM) repository

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 administrative console.
- **V6.0.x** SASRas=all=enabled. This trace statement collects the trace for SAS (low-level authentication logic).
- com.ibm.ws.wim.*=all=enabled:com.ibm.websphere.wim.*=all=enabled. This trace statement collects the trace for VMM.

V6.0.x

Fine tuning SAS traces:

If a subset of classes need to be traced for the SAS/CSlv2 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 CSlv2, or SAS Trace Settings

Situations arise where reviewing trace for the CSlv2 or SAS authentication protocols can assist in troubleshooting difficult problems. This section describes how to enable to CSlv2 and SAS trace.

Enabling Client-Side CSlv2 and SAS Trace

To enable CSlv2 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 output 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

Whenever exceptions occur within the security code on either the client or server, the eventual exception becomes a Common Object Request Broker Architecture (CORBA) exception. Any exception that occurs gets embedded in 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 that a client can expect to receive after running a security-related request such as authentication. It also includes the CORBA exception type that the minor code appears in.

The following exception shows an example of a CORBA exception where the minor code is 49424300 and 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`, never invoke the message; therefore it is always `completed:No`. Other exceptions that are caught on the server side can 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 running a security-related request such as authentication. It also includes the CORBA exception type that the minor code would appear in.

Table 56.

Minor code name	Minor code value (in hex)	Exception type (all in the package of org.omg.CORBA.*)	Minor code description	Retry performed (when authenticationRetryEnabled = true)
AuthenticationFailed	49424300	NO_PERMISSION	This code is a generic authentication failed error. It does not give any details about whether or not the user ID or password is valid. Some user registries can choose to use this type of error code, others can choose to use the next three types that are more specific.	Yes
InvalidUserId	49424301	NO_PERMISSION	This code occurs when the registry returns bad user ID.	Yes
InvalidPassword	49424302	NO_PERMISSION	This code occurs when the registry returns a bad password.	Yes
InvalidSecurityCredentials	49424303	NO_PERMISSION	This is a generic error indicating that the credentials are bad for some reason. It might be that the right attributes are not set.	Yes, if client has BasicAuth credential (token based credential was rejected in the first place).
InvalidRealm	49424304	NO_PERMISSION	This code 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 not valid.	Yes, if client has BasicAuth credential (token based credential was rejected in the first place).
CredentialTokenExpired	49424306	NO_PERMISSION	This code is more specific about why the validation failed. In this case, the token has an absolute lifetime and the 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 the token 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 successfully creates a new session.	Yes
SessionConflictingEvidence	49424309	NO_PERMISSION	This indicates that a session already exists on the server that 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

Table 56. (continued)

Minor code name	Minor code value (in hex)	Exception type (all in the package of org.omg.CORBA.*)	Minor code description	Retry performed (when authenticationRetryEnabled = true)
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 local or remote security server 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 not valid 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 that is 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 does not occur since the LocalOS credential is not a forwardable credential, therefore, validation never needs to be called on this credential.	No
CredentialTokenNotSet	49421095	NO_PERMISSION	This is used to indicate that 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 code is a generic CORBA specific exception in system code.	No
JavaException	494210B1	INTERNAL	This is a generic error that indicated that an unexpected Java exception occurred.	No
ValuesIsNull	494210B2	INTERNAL	This code is used to indicate that a value or parameter that passed in is 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 are specified.	No
NullPointerException	494210B4	INTERNAL	This code is used to indicate that a NullPointerException is caught in the runtime.	No

Table 56. (continued)

Minor code name	Minor code value (in hex)	Exception type (all in the package of org.omg.CORBA.*)	Minor code description	Retry performed (when authenticationRetryEnabled = true)
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	This indicates a duplicate credential attribute that is specified during the set_attributes operation.	No
MethodNotImplemented	494210C0	NO_IMPLEMENT	This indicates that a method invoked is not implemented.	No
GSSFormatError	494210C5	BAD_PARAM	This code indicates that a Generic Security Services (GSS) encoding or decoding routine has created an exception.	No
TagComponentFormatError	494210C6	BAD_PARAM	This code indicates that a tag component cannot be read properly.	No
InvalidSecurityAttributeType	494210C7	BAD_PARAM	This code indicates an attribute type specified during the set_attributes operation is not a valid type.	No
SecurityConfigError	494210CA	INITIALIZE	This code indicates 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.

Errors when trying to configure or enable security

Use this information to troubleshoot problems with configuring or enabling security.

What kind of error are you seeing?

- ““LTPA password not set. validation failed” message displayed as error in the administrative console after saving administrative or application security settings ” on page 1399
- “Errors when trying to configure or enable security”
- “The setupClient.bat or setupClient.sh file is not working correctly” on page 1399
-  “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 1399
- “WebSphere Application Server Version 6 is not working correctly with Enterprise Workload Manager (EWLM)” on page 1399
- If you successfully configured security, but are now having problems accessing Web resources or the administrative console, refer to Errors or access problems after enabling security.
- “NMSV0610I: A NamingException is being thrown from a javax.naming.Context implementation” on page 1400

For general tips on diagnosing and resolving security-related problems, see the topic Troubleshooting the security component.

IBM Support has documents and tools that can save you time gathering information needed to resolve problems as described in Troubleshooting help from IBM. Before opening a problem report, see the Support page:

- <http://www.ibm.com/software/webservers/appserv/was/support/>

"LTPA password not set. validation failed" message displayed as error in the administrative console after saving administrative or application 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 > Secure administration, applications and infrastructure > Authentication mechanisms and expiration**.
- Complete the password and confirm password fields.
- Click **OK**.
- Try setting administrative or application security again.

The setupClient.bat or setupClient.sh file is not working correctly

The setupClient.bat file on Windows operating systems and the setupClient.sh file on Linux and UNIX-based platforms incorrectly specify the location of the SOAP security properties file.

Windows In the setupClient.bat file, the correct location is:

```
set CLIENTSOAP=-Dcom.ibm.SOAP.ConfigURL=file:%WAS_HOME%/properties/soap.client.props
```

AIX **Linux** In the setupClient.sh file, the CLIENTSOAP variable is:

```
CLIENTSOAP=-Dcom.ibm.SOAP.ConfigURL=file:$WAS_HOME/properties/soap.client.props
```

In the setupClient.bat and the setupClient.sh files, complete the following steps:

1. Remove the leading slash (/) 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 **HP-UX**

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 Hewlett Packard 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 EWLM, you must manually update the WebSphere Application Server 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.

For more information on configuring server.policy files, refer to the server.policy file permissions section in the *Developing and deploying applications* PDF book.

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.

NMSV0610I: A NamingException is being thrown from a javax.naming.Context implementation

If you use CSiv2 inbound authentication, basic authentication is required, and Java clients running with `com.ibm.CORBA.validateBasicAuth=true` might fail with the following exception:

If you use CSiv2 inbound authentication, basic authentication is required, and Java™ clients running with `com.ibm.CORBA.validateBasicAuth=true` might fail with the following exception:

```
NMSV0610I: A NamingException is being thrown from a javax.naming.Context
implementation. Details follow:
```

```
Context implementation: com.ibm.ws.naming.jndicos.CNContextImpl
Context method: lookupExt
Context name: TestaburgerNode01Cell/nodes/TestaburgerNode01/servers/server1
Target name: SecurityServer
Other data: ""
Exception stack trace: javax.naming.NoPermissionException: NO_PERMISSION
exception caught. Root exception is org.omg.CORBA.NO_PERMISSION:
vmcid: 0x49421000 minor code: 92 completed: No
...
SECJ0395E: Could not locate the SecurityServer at host/port:9.42.72.27/9100
to validate the userid and password entered. You may need to specify valid
securityServerHost/Port in (WAS_INSTALL_ROOT)/properties/sas.client.props file.
```

To fix this problem, modify the `com.ibm.CORBA.validateBasicAuth=false` property in the `clients.sas.clients.props` file and then run the client.

Errors after enabling security

Use this information if you are experiencing errors after security is enabled.

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*.

IBM Support has documents and tools that can save you time gathering information needed to resolve problems as described in *Troubleshooting help* from IBM. Before opening a problem report, see the Support page:

- <http://www.ibm.com/software/webservers/appserv/was/support/>

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 that they are correct.
- **Security configuration error : User registry type is not set correctly.** Check the user registry property in administrative security settings in the administrative console. Verify that the user registry property 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 is supposed to have access to a resource does not, a configuration step is probably missing. For more information on configuring access to resources, review the chapter *Authorizing access to administrative roles* in the *Securing applications and their environment* PDF book.

Specifically:

- Check the 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 user is required to have one or more of the required roles, use the administrative console to assign that user to required roles, 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. Please refer to Problem Determination Guide for further information.
{0}Permission/:{1}Code/:{2}{3}Stack Trace/:{4}Code Base Location/:{5}
```

The Java security manager `checkPermission` method has reported a `SecurityException` exception .

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 Reliability Availability Serviceability (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 **java.security.debug** run-time flag

- 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, 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 application programming interface (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 Java Message Service (JMS) 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 that 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 Lightweight Third Party Authentication (LTPA) as the authentication mechanism, but not generating the LTPA keys. The LTPA keys encrypt the LTPA token.

To resolve this problem:

1. Click **Security > Secure administration, applications and infrastructure > Authentication > Authentication mechanisms and expiration > 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 **Save**.

The `AccessControlException` exception, 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. An exception similar to the following example displays. The error message and number can vary.

```
CWSRV0020E: [Servlet Error]-[validator]: Failed to load servlet:
java.security.AccessControlException: access denied
(java.io.FilePermission
app_server_root/systemApps/isclite.ear/isclite.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 *Securing applications and their environment* PDF book. 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 created when a client tries to access a resource that is 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 that is 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 `ra.xml` file bundled in a `.rar` file.
- The class path is not set correctly, preventing the permissions for the `resource.xml` file for Service Provider Programming Interface (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` exception in the `SystemOut.log` file which reports the details of the syntax error. For example:

```
CWSCJ0189E: Caught ParserException while creating template for application policy
```

```
profile_root/config/cells/cell_name/nodes/node_name/app.policy
```

Where:

- `cell_name` represents the name of your cell.
- `profile_name` represents the name of your profile.
- `node_name` represents the name of your 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:

```
app_server_root/logs/server1/SystemOut_02.08.20_11.19.53.log :
access denied (java.io.FilePermission
app_server_root/logs/server1/SystemOut_02.08.20_11.19.53.log delete)
```

Code:

```
com.ibm.ejs.ras.RasTestHelper$7 in
{file:app_server_root/installedApps/app1/JrasFVTApp.ear/RasLib.jar
}
```

Stack Trace:

```
java.security.AccessControlException: access denied (java.io.FilePermission
app_server_root/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:/app_server_root/plugins/com.ibm.ws.runtime_6.1.0.jar

ClassLoader: com.ibm.ws.bootstrap.ExtClassLoader
Permissions granted to CodeSource
(file:/app_server_root/plugins/com.ibm.ws.runtime_6.1.0.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.)
```

Where:

- *app1* represents the name of your application.
 - *app_server_root* represents the installation root directory for WebSphere Application Server - Express.
 - *profile_root* represents the location and name of a particular profile in your system.
 - *profile1* or *profile_name* represents the name of your profile.
 - *server1* or *server_name* represents the name of your application server.
- 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 the ra.xml file, correct them with the policy tool. Avoid editing the policy manually, because syntax errors can result. For additional information about using this tool, refer to the section Using PolicyTool to edit policy files in the *Developing and deploying applications* PDF book.
- If a permission is listed as Unresolved, it does not take effect. Verify that the specified permission name is correct.
- If the class path that is 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 the ra.xml file.
- If the permission is not 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 the permission was 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 the Java 2 security chapter in the *Securing applications and their environment* PDF book carefully to determine in which policy file or ra.xml file to specify that permission.

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 that is indicated is not found in the Lightweight Directory Access Protocol (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.

Access problems after enabling security

Use this information if you are experiencing 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 sign-on, I cannot log on to the administrative console
- The following exception displays in the SystemOut.log file after I start the server and enable security: "SECJ0306E: No received or invocation credential exists on the thread."

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 Troubleshooting 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: "CWMN0022E: 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. 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 the ID is not a member, add the ID with at least monitor access privileges, for read-only access.

- Verify that the enable_trusted_application flag is set to true. To check the enable_trusted_application flag value using the administrative console, click **Security > Secure administration, applications and infrastructure**. Under Additional properties, click **Custom properties > EnableTrustedApplications**.

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 is displayed 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 Sockets 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 that 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 the 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 JavaServer Pages (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 that 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 that is read by the sas.client.props properties file that is 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 Common Object Request Broker Architecture (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 not valid:

```
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)
```

A CORBA INITIALIZE exception with CWSA1477W: 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:
CWSA1477W: SECURITY CLIENT/SERVER CONFIG MISMATCH:
The client security configuration (sas.client.props or outbound settings in
administrative console) does not support the server security configuration for
the following reasons:
ERROR 1: CWSA0607E: The client requires SSL Confidentiality but the server does not
support it.
ERROR 2: CWSA0610E: The server requires SSL Integrity but the client does not
support it.
ERROR 3: CWSA0612E: 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)
```

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 CWSA 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 administrative console. To change the configuration for the target server, modify the inbound configuration in the Security administrative console.

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 seems that security is enabled and an enterprise bean is secured, occasions can occur when the client runs the remote method without prompting. If the remote method is protected, an authorization failure results. Otherwise, run 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 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 does not have a valid URL syntax, or the `sas.client.props` that is pointed to cannot be found. Verify that the `com.ibm.CORBA.ConfigURL` property is valid. Check the Java documentation for a description of URL formatting rules. Also, validate that the file exists at the specified path.

 An example of a valid property is `C:/WebSphere/AppServer/properties/sas.client.props`.

- 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, which consists of a 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 -help` command to display the parameters to use.

Use the following command options after enabling security:

- `./stopServer serverName -username name -password password`
- `./stopNode -username name -password password`
- `./stopManager -username name -password password`

If you use the Windows service panel or the `net stop` command to stop the WebSphere Application Server processes and the service could not be stopped, update the existing Application Server service using additional stop arguments. You might need to end the server process from the Task Manager before updating the service. Use the `-stopArgs` and the `-encodeParams` parameters to update the service as described in the "Updating an existing Application Server service" example in the WASService command chapter of the *Administering applications and their environment* PDF book..

After enabling single sign-on, I cannot logon to the administrative console

This problem occurs when single sign-on (SSO) is enabled, and you attempt to access the administrative console using the short name of the server, for example `http://myserver:port_number/ibm/console`. The server accepts your user ID and password, but returns you to the logon 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`.

The following exception displays in the SystemOut.log file after I start the server and enable security: "SECJ0306E: No received or invocation credential exists on the thread."

The following message displays when one or more nodes within the cell was not synchronized during configuration:

```
SECJ0306E: No received or invocation credential exists on the thread. The Role based
authorization check will not have an accessId of the caller to check. The parameters
are: access check method getServerConfig on resource FileTransferServer and module
FileTransferServer. The stack trace is java.lang.Exception: Invocation and received
credentials are both null.
```

Make sure that each of the nodes are synchronized and then restart the deployment manager.

Errors after configuring or enabling Secure Sockets Layer

This topic explains various problems you might encounter after configuring or enabling Secure Sockets Layer (SSL).

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 that indicate SSL problems, verify that your HTTP server is configured correctly for SSL. Browse the welcome page of the HTTP server using SSL by entering the URL:

https://host_name.

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/htpservers/library.html>. Click **Frequently Asked Questions > SSL**.
- If you use the IBM Key Management (IKeyman) tool to create certificates and keys, remember to stash the password to a file when creating the Key Database (KDB) file with the IBM Key Management Tool.
 1. Go to the directory where the KDB file is created, and see if an .sth file exists.
 2. If not, open the KDB file with the IBM Key Management Tool, and click **Key Database File > Stash Password**. The following message is displayed: 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 that is hosted by WebSphere Application Server, or the problem displays 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 1387

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, see Troubleshooting 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.net.ssl.SSLHandshakeException - The client
and server could not negotiate the desired level of security. Reason: handshake
failure:host=MYSERVER,port=1079 minor code: 4942F303 completed: No] at
com.ibm.CORBA.transport.TransportConnectionBase.connect
(TransportConnectionBase.java:NNN)
```

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. In the administrative console, click **Security > SSL certificate and key management**. Under Configuration settings, click **Manage endpoint security configurations > endpoint_configuration_name**. Under Related items, click **SSL configurations > SSL_configuration_name**. You can also browse the file manually by viewing the *install_root/properties/sas.client.props* file.
2. Check the property that is specified by the `com.ibm.ssl.protocol` file to determine which protocol is specified.
3. Check the cipher types that are specified by the `com.ibm.ssl.enabledCipherSuites` interface. You might want to add more cipher types to the list. To see which cipher suites are currently enabled, click **Quality of protection settings (QoP)**, and look for the **Cipher Suites** property.
4. Correct the protocol or cipher problem by using a different client or server protocol and cipher selection. Typical protocols are SSL or SSLv3.
5. Make the cipher selection 40-bit instead of 128-bit. For Common Secure Interoperability Version 2 (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.naming.ServiceUnavailableException: A
communication failure occurred while attempting to obtain an initial context using
the provider url: "corbaloc:iiop:localhost:2809". Make sure that the host and port
information is correct and that the server identified by the provider url is a
running name server. If no port number is specified, the default port number 2809
is used. Other possible causes include the network environment or workstation
network configuration. [Root exception is org.omg.CORBA.TRANSIENT:
CAUGHT_EXCEPTION_WHILE_CONFIGURING_SSL_CLIENT_SOCKET: CWWJE0080E:
javax.net.ssl.SSLHandshakeException - The client and server could not
negotiate the desired level of security. Reason: unknown
certificate:host=MYSERVER,port=1940 minor code: 4942F303 completed: No]
```

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 (CA)-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

A Java exception stack error might display if the following situations occur:

- A personal certificate exists in the client keystore that is used for SSL mutual authentication.
- The signer certificate is not extracted into the server truststore file, and thus the server cannot trust the certificate whenever the SSL handshake is made.

The following message is an example of the Java exception stack error:

```
ERROR: Could not get the initial context or unable to look
up the starting context. Exiting.
Exception received: javax.naming.ServiceUnavailableException:
A communication failure occurred while attempting to obtain an
initial context using the provider url: "corbaloc:iiop:localhost:2809".
```

Make sure that the host and port information is correct and that the server identified by the provider url is a running name server. If no port number is specified, the default port number 2809 is used. Other possible causes include the network environment or workstation network configuration.

```
[Root exception is org.omg.CORBA.TRANSIENT: CAUGHT_EXCEPTION_WHILE_CONFIGURING_SSL_CLIENT_SOCKET: CWWJE0080E: javax.net.ssl.SSLHandshakeException - The client and server could not negotiate the desired level of security. Reason: bad certificate: host=MYSERVER,port=1940 minor code: 4942F303 completed: No]
```

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.CORBA.INTERNAL: Trace from server: 1198777258
at host MYHOST on port 0 >>org.omg.CORBA.INTERNAL: EntryNotFoundException minor
code: 494210B0 completed:
No at com.ibm.ISecurityLocalObjectBaseL13Impl.PrincipalAuthFailReason.
map_auth_fail_to_minor_code(PrincipalAuthFailReason.java:99)
```

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.

Errors configuring Secure Sockets Layer encrypted access

You might have errors returned when you are trying to configure Secure Sockets Layer (SSL) for encrypted access. This article describes some of the common errors you might encounter and makes suggestions on how to fix the problems.

What kind of error are you seeing?

- ““The Java Cryptographic Extension (JCE) files were not found.” error when launching iKeyman” on page 1413
- ““Unable to verify MAC.” error when the wrong keystore password is used” on page 1413
- ““SSL handshake failure” error when no trusted certificate is found” on page 1413
- “The certificate alias cannot be found in the keystore” on page 1414

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 might 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 it points to the Java Developer Kit that is shipped with WebSphere Application Server.

 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`.
The file is located in the `install_dir/java/jre/lib/ext/` directory.

"Unable to verify MAC." error when the wrong keystore password is used

You might receive the following error when the keystore password is not being used correctly.

```
CWPKI0033E: The keystore located at "C:/WebSphere/AppServer/profiles/AppSrv01/etc/trust.p12"  
failed to load due to the following error: Unable to verify MAC.
```

Change the **Password** field that references this keystore by using the correct password. The default password is WebAS. Only use this password in a production environment.

"SSL handshake failure" error when no trusted certificate is found

You might receive the following error when you attempt to add the signer to the local truststore:

```
CWPKI0022E: SSL HANDSHAKE FAILURE: A signer with SubjectDN "CN=BIRKT40.austin.ibm.com,  
O=IBM, C=US" was sent from target host:port "9.65.49.131:9428".
```

The signer might need to be added to the local truststore `C:/WASX_c0602.31/AppServer/profiles/Dmgr09/etc/trust.p12` that is located in the SSL configuration alias `DefaultSSLSettings`. The truststore is loaded from the SSL configuration file `file:C:\WASX_c0602.31\AppServer\profiles\Dmgr09\properties\ssl.client.props`.

The extended error message from the SSL handshake exception is:

```
"No trusted certificate found."
```

This error indicates that the signer certificate from the specified target host and port has not been located in the specified truststore, the SSL settings, and the SSL configuration file. If this occurs in a client process, there are several things that you can do:

- Enable the signer exchange prompt.
- Run the **retrieveSigners** script. For more information about the location of the signer certificate see the `retrieveSigners` command topic in the *Securing applications and their environment PDF* book.
- Manually export the signers from the server and import them to the client.

If this issue occurs in a server process, then complete one of the following procedures:

- In the administrative console, find the target endpoint (host name and port) and determine the certificate that is used by the SSL configuration associated with it. Extract the SSL certificate to a file, and import it into the sending server truststore that is referenced in the error message.

- In the administrative console, find the sending server truststore. Go to signer certificates, add from Port, and connect directly to the target host and port, which are indicated in the message, to retrieve the signer directly into the truststore.
- Manually extract the signer from the target server and host keystore by using the iKeyman utility, and import the signer into the truststore of the server sending the certificate.

The certificate alias cannot be found in the keystore

You might receive the following error when the certificate alias is not found in the referenced keystore:

```
CWPKI0023E: The certificate alias "default" specified by the property
com.ibm.ssl.keyStoreClientAlias is not found in KeyStore
"c:/WebSphere/AppServer/profiles/Dmgr01/config/cells/myCell/key.p12".
```

This error indicates that the certificate alias that was specified cannot be found in the referenced keystore. Either change the certificate alias or make sure that alias exists in the specified keystore.

Single sign-on configuration troubleshooting tips

This topic describes common problems in configuring single sign-on (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, 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 **idsldapsearch** 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.
- Credentials that are not valid: This error indicates that the password is not valid.
- Cannot contact the LDAP server: This error indicates that the host name or the port specified for the server is not valid or that the LDAP server is not running.
- An empty list means that the base directory that is 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 that are specified in the Directory Assistance document must match the values specified for the user registry in the WebSphere Application Server administrative domain:
 - Domain name
 - LDAP host name
 - LDAP port
 - Base DN

Additionally, the rules that are defined in the Directory assistance document must refer to the base distinguished name (DN) of the directory that contains 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 that are 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 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 not valid 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 WebSphere Application Server and the Domino server to use the same LDAP directory. The HTTP cookie that is 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 that are 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. You need the name of the DNS domain in which WebSphere Application Server is configured. For additional information about configuring DNS domains for SSO, refer to the Single sign-on topic in the *Securing applications and their environment* PDF book.
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. ICM 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 is 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`.

Authorization provider troubleshooting tips

This article describes the issues you might encounter using a Java Authorization 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.

Tivoli Access Manager as a Java Authorization 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.
- "Access denied exceptions accessing applications when using JACC" on page 1420
- "An "HPDBA0219E: An error occurred reading data from an SSL connection" might occur" on page 1420

External providers for Java Authorization Contract for Containers authorization provider

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 have problems configuring JACC, check the following items:

- Ensure that the parameters are correct. For example, you do not want a number after TAM_Policy_server_hostname:7135, but you do want 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" is displayed, 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 started.
- Ensure that the password for the sec_master user is correct.
- Check the SystemOut.log file and search for the AMAS string 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 is configured, check the following items:

- Ensure that WebSphere Application Server and Tivoli Access Manager use the same Lightweight Directory Access Protocol (LDAP) server.
- If the message "Policy Director Authentication failed" is displayed, ensure that the:
 - WebSphere Application Server LDAP server ID is the same as the "Administrator user" in the Tivoli Access Manager JACC configuration panel.
 - Verify that the Tivoli Access Manager Administrator distinguished name (DN) is correct.
 - Verify that the password of the Tivoli Access Manager administrator has not expired and is valid.
 - Ensure that the 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) is displayed, take the following actions:
 1. Go to the *profile_root/etc/tam* directory.
 2. Change xxxx to an available port number in the *amwas.commonconfig.properties* file, and the *amwas*cellName_dmgr.properties* file if the deployment manager failed to start. If the node failed to start, change xxx to an available port number in the *amwas*cellName_nodeName_.properties* file. If the Application Server failed to start, change xxxx in the *amwas*cellname_nodeName_serverName.properties* file.

The application might not deploy properly

When you click **Save**, the policy and role information is propagated to the Tivoli Access Manager policy. This process 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 configure 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 is configured, take the following actions:

- 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.

```
install_root/java/jre/PdPerm.properties  
install_root/java/jre/PdPerm.ks  
profile_root/etc/tam/*
```

- Use a utility to clear the security configuration and return the system to the state it was in before you configure the JACC provider for Tivoli Access Manager. 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 JACC provider for Tivoli Access Manager. Backup the security.xml file before running this utility.

Enter the following commands:

```
install_root/java/jre/bin/java -classpath  
"install_root/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 try 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."
```

This problem might be caused by the host name exceeding predefined limits with Tivoli Access Manager when it is configured against MS Active Directory. In WebSphere Application Server, the maximum length of the host name can not exceed 46 characters.

Check that the host name is not fully qualified. Configure the machine so that the host name does not include the host domain.

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 passw0rd
```

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 propagatePolicyToJACCProvider** command:

```
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, that 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 MapRolesToUsers task option is no longer valid when Tivoli Access Manager is used as the authorization server. To correct the error, change MapRolesToUsers to TAMMapRolesToUsers.

Access denied exceptions accessing applications when using JACC

In the case of Tivoli Access Manager, you might see the following error message.

```
AWXJR0044E: The access decision for Permission, {0}, was denied because either the PolicyConfiguration or RoleConfiguration objects did not get created successfully at application installation time. RoleConfiguration exists = {false}, PolicyConfiguration exists = {false}."
```

If the access denied exceptions are not expected for the application, check the SystemOut.log files to see if the security policy information was correctly propagated to the provider.

If the security policy information for the application is successfully propagated to the provider, the audit statements with the message key SECJ0415I appear. However, if there was a problem propagating the security policy information to the provider (for example: network problems, JACC provider is not available), the SystemOut.log files contain the error message with the message keys SECJ0396E (during install) or SECJ0398E (during modification). The installation of the application is not stopped due to a failure to propagate the security policy to the JACC provider. Also, in the case of failure, no exception or error messages appear during the save operation. When the problem causing this failure is fixed, run the propagatePolicyToJaccProvider tool to propagate the security policy information to the provider without reinstalling the application. For more information about this task, see the Propagating security policy of installed applications to a JACC provider using wsadmin scripting topic in the *Securing applications and their environment* PDF book.

An "HPDBA0219E: An error occurred reading data from an SSL connection" might occur

An error message (HPDBA0219E) might appear in dmgr SystemOut.log when you install an application on WebSphere Application Server for Network Deployment (ND) and a managed node with Tivoli Access Manager is enabled.

If the error occurs, then the security policy data of recently deployed applications might not be immediately available. The policy data is available based on the server replicate time of the Tivoli Access Manager. This is defaulted to 30 seconds after all updates have been completed. To ensure that the latest policy data is available, log on to the pdadmin console and type: server replicate.

SPNEGO trust association interceptor (TAI) troubleshooting tips

Presented here is a list of trouble shooting tips useful in diagnosing Simple and Protected GSS-API Negotiation (SPNEGO) TAI problems and exceptions.

The IBM Java Generic Security Service (JGSS) and IBM SPNEGO providers use a Java virtual machine (JVM) custom property to control trace information. The SPNEGO TAI uses the JRas facility to allow an administrator to trace only specific classes. To debug the TAI using tracing, the following important trace specifications or JVM customer should be used:

Table 57. SPNEGO TAI trace specifications

Trace	Use
<code>com.ibm.security.jgss.debug</code>	Set this JVM Custom Property to <code>a11</code> to trace through JGSS code. Messages appear in the <code>trace.log</code> file, and SystemOut.log .
<code>com.ibm.security.krb5.Krb5Debug</code>	Set this JVM Custom Property to <code>a11</code> to trace through the Kerberos5-specific JGSS code. Messages appear in the <code>trace.log</code> file, and SystemOut.log .

Table 57. SPNEGO TAI trace specifications (continued)

Trace	Use
<code>com.ibm.ws.security.spnego.*</code>	Set this trace on using the administrative console > troubleshooting > Logging and Tracing > server1 > Change Log Detail Levels > com.ibm.ws.security.spnego.* . Messages appear in the <code>trace.log</code> file.

Problem: WebSphere Application Server and the Active Directory (AD) Domain Controller's time are not synchronized within 5 minutes

The time is not synchronized between WebSphere Application Server and AD Domain Controller.

```
[2/24/06 13:12:46:093 CST] 00000060 Context      2 com.ibm.ws.security.spnego.Context
begin GSSContext accepted
[2/24/06 13:12:46:093 CST] 00000060 Context      E com.ibm.ws.security.spnego.Context
begin
CWSPN0011E: An invalid SPNEGO token has been encountered while authenticating a
HttpServletRequest:
0000: 60820160 06062b06 01050502 a1820154  `..^ ..+. .... ...T
0010: 30820150 a0030a01 01a10b06 092a8648 0..P .... .... .*H
0020: 82f71201 0202a282 013a0482 01366082 .... .... :... .6`.
0030: 01320609 2a864886 f7120102 0203007e .2.. *.H. .... ...~
0040: 82012130 82011da0 03020105 a1030201 ..!0 .... .... ....
0050: 1ea41118 0f323030 36303232 34313931 .... .200 6022 4191
0060: 3234365a a5050203 016b48a6 03020125 246Z .... .kH. ...%
0070: a9161b14 57535345 432e4155 5354494e .... WSSE C.AU STIN
0080: 2e49424d 2e434f4d aa2d302b a0030201 .IBM .COM .-0+ ....
0090: 00a12430 221b0448 5454501b 1a773230 ..$0 ".H TTP. .w20
00a0: 30337365 63646576 2e617573 74696e2e 03se cdev .aus tin.
00b0: 69626d2e 636f6dab 81aa1b81 a76f7267 ibm. com. .... .org
00c0: 2e696574 662e6a67 73732e47 53534578 .iet f.jg ss.G SSEx
00d0: 63657074 696f6e2c 206d616a 6f722063 cept ion, maj or c
00e0: 6f64653a 2031302c 206d696e 6f722063 ode: 10, min or c
00f0: 6f64653a 2033370a 096d616a 6f722073 ode: 37. .maj or s
0100: 7472696e 673a2044 65666563 74697665 trin g: D efec tive
0110: 20746f6b 656e0a09 6d696e6f 72207374 tok en.. mino r st
0120: 72696e67 3a20436c 69656e74 2074696d ring : Cl ient tim
0130: 65204672 69646179 2c204665 62727561 e Fr iday , Fe brua
0140: 72792032 342c2032 30303620 61742031 ry 2 4, 2 006 at 1
0150: 3a31323a 34352050 4d20746f 6f20736b :12: 45 P M to o sk
0160: 65776564 ewed
```

Solution: You can fix this in one of two ways. The preferred way is to synchronize the WebSphere Application Server system time to within 5 minutes of the AD server's time. A best practice is to use a time server to keep all systems synchronized. Or you can add or adjust the `clockskew` parameter in the Kerberos configuration file.

Note: The default for the `clockskew` parameter is 300 seconds (or 5 minutes).

Problem: Getting exception: No factory available to create a name for mechanism 1.3.6.1.5.5.2

There apparently is no factory available to process the creation of a name for the specific mechanism.

The `systemout.log` file displays something like this:

```
[4/8/05 22:51:24:542 EDT] 5003e481 SystemOut      0 [JGSS_DBG_PROV] Provider
  IBMJGSSProvider version 1.01 does not support mech 1.3.6.1.5.5.2
[4/8/05 22:51:24:582 EDT] 5003e481 ServerCredent >
  com.ibm.ws.security.spnego.ServerCredential initialize ENTRY
```

```
SPNEG0014: Kerberos initialization Failure: org.ietf.jgss.GSSEException, major code: 2,
    minor code: 0
major string: Unsupported mechanism
minor string: No factory available to create name for mechanism 1.3.6.1.5.5.2
at com.ibm.security.jgss.i18n.I18NException.throwGSSEException
    (I18NException.java:30)
at com.ibm.security.jgss.GSSManagerImpl.a(GSSManagerImpl.java:36)
at com.ibm.security.jgss.GSSCredentialImpl.add(GSSCredentialImpl.java:217)
at com.ibm.security.jgss.GSSCredentialImpl.<init>(GSSCredentialImpl.java:264)
```

Solution: Check the `java.security` file to ensure it contains the `IBMSPNego` security provider and that the provider is defined correctly. The `java.security` file should contain a line similar to:

```
security.provider.6=com.ibm.security.jgss.mech.spnego.IBMSPNego
```

Problem: Getting an exception

An exception has occurred when reporting to the client.

You get the following display.

```
Error authenticating request. Reporting to client
Major code = 11, Minor code = 31
org.ietf.jgss.GSSEException, major code: 11, minor code: 31
major string: General failure, unspecified at GSSAPI level
minor string: Kerberos error while decoding and verifying token:
    com.ibm.security.krb5.internal.KrbException, status code: 31
message: Integrity check on decrypted field failed
```

as the JGSS library is trying to process the SPNEGO token.

Cause: This exception is the result of encoding the ticket using one key and attempting to decode it using a different key. There are number of possible reasons for this condition:

1. The Kerberos keytab file has not been copied to the server machine after it has been regenerated.
2. The Kerberos configuration points to the wrong Kerberos keytab file.
3. The Kerberos service principal name (SPN) has been defined to the Active Directory more than once; this can occur because you have another userid with a similarly defined SPN (either exactly the same name, or one having a different name but with a port defined part of the SPN).

Solution: If the problem is with the Kerberos keytab file, then fix it. If the problem is with multiple SPN definitions, then remove the extra or conflicting SPN, confirm that the SPN is no longer registered with the Active Directory, and then add the SPN. The Active Directory may need to be searched for other entries with SPNs defined that clash with the SPN.

To confirm that the SPN is not registered, the command:

```
setspn -l userid
```

should return with the following response:

```
Cannot find account userid
```

Problem: Single sign-on is not occurring.

When trace is turned on, the following message appears:

```
[2/27/06 14:28:04:191 CST] 00000059 SpnegoHandler <
    com.ibm.ws.security.spnego.SpnegoHandler handleRequest: Received a
    non-SPNEGO Authorization Header RETURN
```

Cause: The client is returning an NT LAN manager (NTLM) response to the authorize challenge, not a SPNEGO token. This condition can occur due to any of the following reasons:

- The client has not been configured properly.
- The client is not using a supported browser. For example, when using Microsoft Internet Explorer 5.5, SP1 responds with a non-SPNEGO authentication header.
- The user has not logged into the Active Directory domain, or into a trusted domain, or the client used does not support integrated authentication with Windows – in this case, the SPNEGO TAI is working properly.
- The user is accessing a service defined on the same machine upon which the client is running (local host). Microsoft Internet Explorer resolves the host name of the URL to `http://localhostsomeURL` instead of a fully qualified name.
- The SPN is not found in the Active Directory. The SPN must be of the format `HTTP/server.realm.com`. The command to add the SPN is

```
setspn -a HTTP/server.realm.com userid
```

If the SPN is defined incorrectly as `HTTP/server.realm.com@REALM.COM` with the addition of `@REALM.COM`, then delete the user, redefine the user, and redefine the SPN.

Problem: Credential Delegation is not working

An invalid option is detected. When trace is turned on, the following message is displayed:

```
com.ibm.security.krb5.KrbException, status code: 101 message: Invalid option in
ticket request
```

Cause: The Kerberos configuration file is not properly configured.

Solution: Ensure that neither `renewable`, nor `proxiable` are set to `true`.

Problem: Unable to get SSO working using RC4-HMAC encryption.

When trace is turned on, you get the following message in the trace:

```
com.ibm.security.krb5.internal.crypto.KrbCryptoException, status code: 0
message: Checksum error; received checksum does not match computed checksum
```

Cause: RC4-HMAC encryption is not supported with a Microsoft Windows 2000 Kerberos key distribution center (KDC). To confirm this condition, examine the trace and identify where the exception is thrown. The content of the incoming ticket should be visible in the trace. Although the incoming ticket is encrypted, the SPN for the service is readable. If a Microsoft Windows 2000 KDC is used and the system is configured to use RC4-HMAC, the string representing the ticket for `userid@REALM` (instead of the expected `HTTP/hostname.realm@REALM`) is displayed. For example, this is beginning of the ticket received from a Microsoft Windows 2000 KDC:

```
0000: 01 00 6e 82 04 7f 30 82 04 7b a0 03 02 01 05 a1 ..n...0.....
0010: 03 02 01 0e a2 07 03 05 00 20 00 00 00 a3 82 03 .....
0020: a5 61 82 03 a1 30 82 03 9d a0 03 02 01 05 a1 0a .a...0.....
0030: 1b 08 45 50 46 44 2e 4e 45 54 a2 18 30 16 a0 03 ..REALM.COM.0..
0040: 02 01 01 a1 0f 30 0d 1b 0b 65 70 66 64 77 61 73 .....0...userid
0050: 75 6e 69 74 a3 82 03 6e 30 82 03 6a a0 03 02 01 .a.f...n0..j....
```

The realm is `REALM.COM`. The service name is `userid`. A correctly formed ticket for the same SPN is:

```
0000: 01 00 6e 82 04 56 30 82 04 52 a0 03 02 01 05 a1 ..n..V0..R.....
0010: 03 02 01 0e a2 07 03 05 00 20 00 00 00 a3 82 03 .....
0020: 82 61 82 03 7e 30 82 03 7a a0 03 02 01 05 a1 0a .a...0..z.....
0030: 1b 08 45 50 46 44 2e 4e 45 54 a2 2a 30 28 a0 03 ..REALM.COM.0...
0040: 02 01 02 a1 21 30 1f 1b 04 48 54 54 50 1b 17 75 .....0...HTTP..u
0050: 73 31 30 6b 65 70 66 77 61 73 73 30 31 2e 65 70 serid.realm.com.
0060: 66 64 2e 6e 65 74 a3 82 03 39 30 82 03 35 a0 03 ...n.....90..5..
```

Solution: To correct the problem, either use the Single data encryption standard (DES) or use a Microsoft Windows 2003 Server for a KDC. Remember to regenerate the SPN, and the Kerberos keytab file.

Problem: User receives the following message when accessing a protected URL through the SPNEGO SSO

Bad Request

Your browser sent a request that this server could not understand.
Size of request header field exceeds server limit.

Authorization: Negotiate YII.....

Cause: This message is generated by the Apache/IBM HTTP Server. This server is indicating that the authorization header returned by the user's browser is too large. The long string that follows the word Negotiate (in the error message above) is the SPNEGO token. This SPNEGO token is a wrapper of the Microsoft Windows Kerberos token. Microsoft Windows includes the user's PAC information in the Kerberos token. The more security groups that the user belongs to, the more PAC information is inserted in the Kerberos token, and the larger the SPNEGO becomes. IBM HTTP Server 2.0 (also Apache 2.0 and IBM HTTP Server 6.0) limit the size of any acceptable HTTP header to be 8K. In Microsoft Windows domains having many groups, and with user membership in many groups, the size of the user's SPNEGO token may exceed the 8K limit.

Solution: If possible, reduce the number of security groups the user is a member of. IBM HTTP Server 2.0.47 cumulative fix PK01070 allows for HTTP header sizes up to and beyond the Microsoft limit of 12K. WebSphere Application Server Version 6.0 users can obtain this fix in fixpack 6.0.0.2.

Note: Non-Apache based Web servers may require differing solutions.

Problem: Even with JGSS tracing disabled, some KRB_DBG_KDC messages appear in the SystemOut.log

Cause: While most of the JGSS tracing is controlled by the `com.ibm.security.jgss.debug` property, a small set of messages are controlled by the `com.ibm.security.krb5.Krb5Debug` property. The `com.ibm.security.krb5.Krb5Debug` property has a default value to put some messages to the **SystemOut.log**.

Solution: To remove all KRB_DBG_KDC messages from the **SystemOut.log**, set the JVM property as follows:

```
-Dcom.ibm.security.krb5.Krb5Debug=none
```

Problem: HTTP Post parameters are lost during interaction with the SPNEGO TAI, when stepping down to userid/password login.

Cause: The Microsoft Internet Explorer maintains state during a user's request. If a request was given the response of an "HTTP 401 Authenticate Negotiate", and the browser responds with a NTLM token obtained through a userid/password challenge, the browser resubmits the request. If this second request is given a response of an HTML page containing a redirection to the same URL but with new arguments (via Javascript) then the browser does not resubmit the POST parameters. To avoid this problem, it is critical to NOT perform the automatic redirection. If the user clicks on a link, the problem does not occur. See section 5.2 Client Returns NTLM Token to SPNEGO Challenge for a resolution to the problem,

Solution: The browser responds to the Authenticate/Negotiate challenge with an NTLM token, not an SPNEGO token. The SPNEGO TAI sees the NTLM, and returns back a HTTP 403 response, along with the HTML page. When the browser runs the Javascript `redirTimer` function, any POST or GET parameters that were present on the original request are lost.

By leveraging the `SPN<id>.NTLMTokenReceivedPage` property, an appropriate message page can be returned to the user. The default message that is returned (in the absence of a user defined property) is:

```
"<html><head><title>An NTLM Token was Received.</title></head>"
+ "<body>Your browser configuration is correct, but you have not logged into
  a supported Windows Domain."
+ "<p>Please login to the application using the normal login page.</html>";
```

Using the `SPN<id>.NTLMTokenReceivedPage` property, you can customize the exact response. It is critical that the returned HTML not perform a redirection.

When the SPNEGO TAI has been configured to use the shipped default `HTTPHeaderFilter` class as the `SPN<id>.filterClass`, then the `SPN<id>.filter` can be used to allow the second request to flow directly to the normal WebSphere Application Server security mechanism. In this way, the user experiences the normal authentication mechanism.

An example of such a configuration follows. The required SPNEGO TAI properties necessary and the HTML file content are presented.

Table 58. SPNEGO TAI properties and HTML

SPNEGO TAI Property Name	HTML File Content
<code>com.ibm.ws.security.spnego.SPN1.hostName</code>	<code>server.wasteched30.torolab.ibm.com</code>
<code>com.ibm.ws.security.spnego.SPN1.filterClass</code>	<code>com.ibm.ws.security.spnego.HTTPHeaderFilter</code>
<code>com.ibm.ws.security.spnego.SPN1.filter</code>	<code>request-ur!=noSPNEGO</code>
<code>com.ibm.ws.security.spnego.SPN1.NTLMTokenReceivedPage</code>	<code>File:///C:/temp/NTLM.html</code>

Note: Observe that the filter property instructs the SPNEGO TAI to NOT intercept any HTTP request that contains the string “noSPNEGO”.

Here is an example of a generating a helpful response.

```
<html>
<head>
<title>NTLM Authentication Received </title>
<script language="javascript">
  var purl="" + document.location;
  if (purl.indexOf("noSPNEGO") < 0) {
    if (purl.indexOf('?') >= 0) purl += "&noSPNEGO";
    else purl += "?noSPNEGO";
  }
</script>
</head>
<body>
<p>An NTLM token was retrieved in response to the SPNEGO challenge. It is likely that
you are not logged into a Windows domain.<br>
Click on the following link to get the requested website.
<script language="javascript">
  document.write("<a href='"+purl+"'>");
  document.write("Open the same page using the normal authentication
  mechanism.");
  document.write("</a><br>");
```

```
</script>  
You will not automatically be redirected.  
</body>  
</html>
```

Appendix. Directory conventions

References in product information to *app_server_root*, *profile_root*, and other directories infer specific default directory locations. This topic describes the conventions in use for WebSphere Application Server - Express.

Default product locations when the root user or an administrator user installs the product

The root user or administrator user (on a Windows system) is capable of registering shared products and installing into system-owned directories. The following default directories are system-owned directories.

These file paths are default locations. You can install the product and other components in any directory where you have write access. You can create profiles in any valid directory where you have write access. Multiple installations of WebSphere Application Server - Express products or components, of course, require multiple locations.

app_server_root - the install_root for WebSphere Application Server

The following list shows default installation root directories for WebSphere Application Server - Express:

▶ AIX	/usr/IBM/WebSphere/AppServer
▶ HP-UX	/opt/IBM/WebSphere/AppServer
▶ Linux	/opt/IBM/WebSphere/AppServer
▶ Solaris	/opt/IBM/WebSphere/AppServer
▶ Windows	C:\Program Files\IBM\WebSphere\AppServer

profile_root

The following list shows the default directory for a profile named *profile_name* on each distributed operating system:

▶ AIX	/usr/IBM/WebSphere/AppServer/profiles/ <i>profile_name</i>
▶ HP-UX	/opt/IBM/WebSphere/AppServer/profiles/ <i>profile_name</i>
▶ Linux	/opt/IBM/WebSphere/AppServer/profiles/ <i>profile_name</i>
▶ Solaris	/opt/IBM/WebSphere/AppServer/profiles/ <i>profile_name</i>
▶ Windows	C:\Program Files\IBM\WebSphere\AppServer\profiles\ <i>profile_name</i>

plugins_root

The following default installation root is for the Web server plug-ins for WebSphere Application Server:

▶ AIX	/usr/IBM/HTTPServer/Plugins
▶ HP-UX	/opt/IBM/HTTPServer/Plugins
▶ Linux	/opt/ibm/HTTPServer/Plugins
▶ Solaris	/opt/IBM/HTTPServer/Plugins
▶ Windows	C:\Program Files\IBM\HTTPServer\Plugins

web_server_root

The following default installation root directories are for the IBM HTTP Server:

- ▶ AIX /usr/IBM/HTTPServer
- ▶ HP-UX /opt/IBM/HTTPServer
- ▶ Linux /opt/ibm/HTTPServer
- ▶ Solaris /opt/IBM/HTTPServer
- ▶ Windows C:\Program Files\IBM\HTTPServer

gskit_root

The following list shows the default installation root directories for Version 7 of the IBM Global Security Kit (GSKit):

- ▶ AIX /usr/ibm/gsk7
- ▶ HP-UX /opt/ibm/gsk7
- ▶ Linux /opt/ibm/gsk7
- ▶ Solaris /opt/ibm/gsk7
- ▶ Windows C:\Program Files\IBM\GSK7

app_client_root

The following default installation root directories are for the WebSphere Application Client:

- ▶ AIX /usr/IBM/WebSphere/AppClient (J2EE Application client only)
- ▶ HP-UX /opt/IBM/WebSphere/AppClient (J2EE Application client only)
- ▶ Linux /opt/IBM/WebSphere/AppClient (J2EE Application client only)
- ▶ Solaris /opt/IBM/WebSphere/AppClient (J2EE Application client only)
- ▶ Windows C:\Program Files\IBM\WebSphere\AppClient

updi_root

The following list shows the default installation root directories for the Update Installer for WebSphere Software:

- ▶ AIX /usr/IBM/WebSphere/UpdateInstaller
- ▶ HP-UX /opt/IBM/WebSphere/UpdateInstaller
- ▶ Linux /opt/IBM/WebSphere/UpdateInstaller
- ▶ Solaris /opt/IBM/WebSphere/UpdateInstaller
- ▶ Windows C:\Program Files\IBM\WebSphere\UpdateInstaller

cip_app_server_root

The following list shows the default installation root directories for a customized installation package (CIP) produced by the Installation Factory.

A CIP is a WebSphere Application Server - Express product bundled with one or more maintenance packages, an optional configuration archive, one or more optional enterprise archive files, and other optional files and scripts:

- ▶ AIX /usr/IBM/WebSphere/AppServer/cip/cip_uid
- ▶ HP-UX /opt/IBM/WebSphere/AppServer/cip/cip_uid
- ▶ Linux /opt/IBM/WebSphere/AppServer/cip/cip_uid

Solaris /opt/IBM/WebSphere/AppServer/cip/cip_uid

Windows C:\Program Files\IBM\WebSphere\AppServer\cip\cip_uid

The *cip_uid* variable is the CIP unique ID generated during creation of the build definition file. You can override the generated value in the Build definition wizard. Use a unique value to allow multiple CIPs to install on the system.

component_root

The component installation root directory is any installation root directory described in this topic. Some programs are for use across multiple components. In particular, the Update Installer for WebSphere Software is for use with WebSphere Application Server - Express, Web server plug-ins, the Application Client, and the IBM HTTP Server. All of these components are part of the product package.

Default product locations when a non-root user or a non-administrator user installs the product

The non-root user or non-administrator user (on a Windows system) is not capable of registering shared products and installing into system-owned directories. The following default directories are user-owned directories in the home directory of the non-root installer as opposed to being globally shared resources that are available to all users.

app_server_root

The following list shows the default installation directories for non-root installation of WebSphere Application Server - Express:

AIX *user_home/IBM/WebSphere/AppServer*

HP-UX *user_home/IBM/WebSphere/AppServer*

Linux *user_home/IBM/WebSphere/AppServer*

Solaris *user_home/IBM/WebSphere/AppServer*

Windows C:\IBM\WebSphere\AppServer

profile_root

The following list shows the default directories for creating profiles:

AIX *user_home/IBM/WebSphere/AppServer/profiles/*

HP-UX *user_home/IBM/WebSphere/AppServer/profiles/*

Linux *user_home/IBM/WebSphere/AppServer/profiles/*

Solaris *user_home/IBM/WebSphere/AppServer/profiles/*

Windows C:\IBM\WebSphere\AppServer\profiles\

web_server_root

The following default installation root directories are for the IBM HTTP Server:

AIX *user_home/IBM/HTTPServer*

HP-UX *user_home/IBM/HTTPServer*

Linux *user_home/ibm/HTTPServer*

Solaris *user_home/IBM/HTTPServer*

Windows C:\IBM\HTTPServer

plugins_root

The following list shows the default installation root directories for the Web server plug-ins for WebSphere Application Server:

- ▶ AIX *user_home/IBM/HTTPServer/Plugins*
- ▶ HP-UX *user_home/IBM/HTTPServer/Plugins*
- ▶ Linux *user_home/ibm/HTTPServer/Plugins*
- ▶ Solaris *user_home/IBM/HTTPServer/Plugins*
- ▶ Windows *C:\IBM\HTTPServer\Plugins*

app_client_root

The following list shows the default installation root directories for the WebSphere Application Client:

- ▶ AIX *user_home/IBM/WebSphere/AppServer/AppClient (J2EE Application client only)*
- ▶ HP-UX *user_home/IBM/WebSphere/AppClient (J2EE Application client only)*
- ▶ Linux *user_home/IBM/WebSphere/AppClient (J2EE Application client only)*
- ▶ Solaris *user_home/IBM/WebSphere/AppClient (J2EE Application client only)*
- ▶ Windows *C:\IBM\WebSphere\AppClient*

updi_root

The following list shows the default installation directories for non-root installation of WebSphere Application Server - Express:

- ▶ AIX *user_home/IBM/WebSphere/UpdateInstaller*
- ▶ HP-UX *user_home/IBM/WebSphere/UpdateInstaller*
- ▶ Linux *user_home/IBM/WebSphere/UpdateInstaller*
- ▶ Solaris *user_home/IBM/WebSphere/UpdateInstaller*
- ▶ Windows *C:\Program Files\IBM\WebSphere\UpdateInstaller*

cip_app_server_root

The following list shows the default installation root directories for a WebSphere Application Server - Express product CIP:

- ▶ AIX *user_home/IBM/WebSphere/AppServer/cip/cip_uid*
- ▶ HP-UX *user_home/IBM/WebSphere/AppServer/cip/cip_uid*
- ▶ Linux *user_home/IBM/WebSphere/AppServer/cip/cip_uid*
- ▶ Solaris *user_home/IBM/WebSphere/AppServer/cip/cip_uid*
- ▶ Windows *C:\IBM\WebSphere\AppServer\cip\cip_uid*

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 Intellectual Property & Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
USA

Trademarks and service marks

For trademark attribution, visit the IBM Terms of Use Web site (<http://www.ibm.com/legal/us/>).