



Security

Note

Before using this information, be sure to read the general information under “Notices” on page 373.

Compilation date: March 15, 2004

© Copyright International Business Machines Corporation 2004. All rights reserved.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

How to send your comments v

Chapter 1. Welcome to Security 1

Chapter 2. Securing applications and their environments 9

Planning to secure your environment 9

 Security considerations when adding a Base

 Application Server node to Network Deployment 18

 Security considerations specific to a multi-node or process Network Deployment environment. . 19

 Creating login key files 21

 Preparing truststore files 22

 Configuring the application server for interoperability 22

Implementing security considerations. 23

 Securing your environment before installation. . 23

 Securing your environment after installation . . 24

 Protecting plain text passwords. 25

Migrating security configurations from previous releases. 27

 Migrating custom user registries 28

 Migrating trust association interceptors 31

 Migrating Common Object Request Broker Architecture programmatic login to Java

 Authentication and Authorization Service . . . 34

 Migrating from the CustomLoginServlet class to servlet filters 37

Developing secured applications 38

 Developing with programmatic security APIs for Web applications 39

 Developing form login pages 47

 Developing with programmatic APIs for EJB applications 50

 Developing with the Java Authentication and Authorization Service to log in programmatically. 54

 Developing your own J2C principal mapping module. 59

 Developing custom user registries 61

 Developing a custom interceptor for trust associations 70

Assembling secured applications 74

 Enterprise bean component security 75

 Securing enterprise bean applications using the Assembly Toolkit 75

 Web component security 78

 Securing Web applications using the Assembly Toolkit 78

 Role-based authorization 80

 Adding users and groups to roles using the Assembly Toolkit 85

 Mapping users to RunAs roles using the Assembly Toolkit 86

Deploying secured applications. 86

 Assigning users and groups to roles 87

 Delegations 92

 Assigning users to RunAs roles. 94

 Updating and redeploying secured applications 98

Testing security 99

Managing security. 100

 Global security 100

 Configuring global security. 101

 Global security and server security 108

 Configuring server security. 108

 Administrative console and naming service authorization 113

 Assigning users to administrator roles 115

 Assigning users to naming roles 119

 Authentication mechanisms 120

 Configuring authentication mechanisms . . . 122

 User registries 147

 Configuring user registries 148

 Java Authentication and Authorization Service 196

 Configuring Java Authentication and

 Authorization Service login. 199

 Authentication protocol for EJB security . . . 227

 Configuring Common Secure Interoperability Version 2 and Security Authentication Service authentication protocols 235

 Secure Sockets Layer 265

 Configuring Secure Sockets Layer 271

 Cryptographic token support 311

 Opening a cryptographic token using the key management utility (iKeyman) 312

 Configuring to use cryptographic tokens . . . 313

 Using Java Secure Socket Extension and Java Cryptography Extension with Servlets and

 enterprise bean files 315

 Java 2 security 320

 Configuring Java 2 security. 327

Troubleshooting security configurations 356

Tuning security configurations. 357

 Tuning CSiv2 358

 Tuning LDAP authentication 358

 Tuning Web authentication 359

 Tuning authorization 359

 Security cache properties 360

 Secure Sockets Layer performance tips . . . 360

Tuning security. 362

Chapter 3. Integrating IBM WebSphere Application Server security with existing security systems 363

Interoperability issues for security 367

 Interoperability with C++ common object request broker architecture client support and

 limitations 367

Interoperating with a C++ common object request broker architecture client 368

Interoperating with previous product versions . . 370

Security: Resources for learning 371

Trademarks and service marks 375

Notices 373

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. Welcome to Security

IBM WebSphere Application Server Version 5 provides security infrastructure and mechanisms to protect sensitive 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. IBM WebSphere Application Server security is based on industry standards. Version 5 has an open architecture that processes secure connectivity and interoperability with Enterprise Information Systems including:

- DB2
- CICS
- MQ Series
- Lotus Domino
- IBM Directory

WebSphere Application Server also supports other security providers including:

- IBM Tivoli Access Manager (Policy Director)
- WebSEAL secure proxy server

Based on industry standards

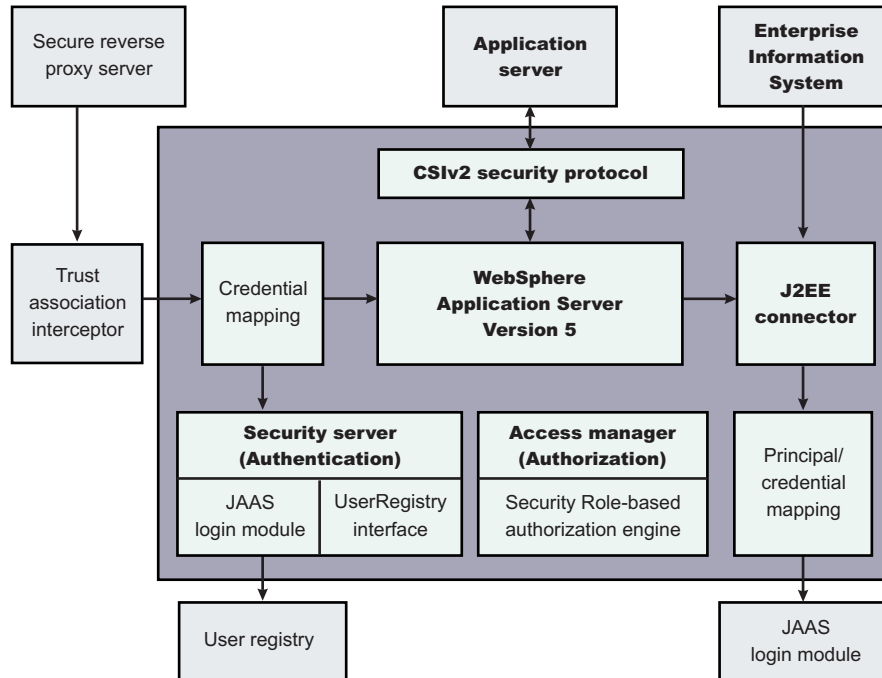
The product provides a unified, policy-based, and permission-based model for securing Web resources and enterprise JavaBeans according to J2EE specifications. Specifically Version 5 complies with J2EE specification Version 1.3.1 and has passed the J2EE Compatibility Test Suite. Product security is a layered architecture built on top of an operating system platform, a Java virtual machine (JVM), and Java 2 security. This security model employs a rich set of security technology including the:

- Java 2 security model, which provides policy-based, fine-grained, and permission-based access control to system resources.
- Common Secure Interoperability Version 2 (CSIv2) security protocol, in addition to the Secure Authentication Services (SAS) security protocol. Both protocols are supported by prior product releases. CSIv2 is an integral part of the J2EE 1.3 Specification and is essential for interoperability among application servers from different vendors and with enterprise CORBA services.
- Java Authentication and Authorization Service (JAAS) programming model for Java applications, servlets, and enterprise beans.
- J2EE Connector architecture for plugging in resource adapters that support access to Enterprise Information Systems.

The standard security model and interface supported include Java Secure Socket Extension (JSSE) and Java Cryptographic Extension (JCE) provider for secure socket communication, message encryption, and data encryption.

Open architecture paradigm

An application server plays an integral part in the multiple-tier enterprise computing framework. IBM WebSphere Application Server adopts the open architecture paradigm and provides many plug-in points to integrate with enterprise software components. Plug-in points are based on standard J2EE specifications wherever applicable.



The light blue shaded background indicates the boundary between the product and other business application components.

The product provides Simple WebSphere Authentication Mechanism (SWAM) and Lightweight Third Party Authentication (LTPA) mechanisms. Exactly one may be configured to be the active authentication mechanism for the security domain of the product. Exactly one user registry implementation may be configured to be the active user registry of the product security domain. The product provides the following user registry implementations: UNIX, Windows, and AS/400 LocalOS and LDAP. It also provides file-based and Java database connectivity (JDBC)-based user registry reference implementations. It supports a flexible combination of authentication mechanisms and user registries. SWAM is simple to configure and is useful for a single application server environment. LTPA generates a security token for authenticated users, which can propagate to downstream servers and is suitable for a distributed environment with multiple application servers. It is possible to use SWAM in a distributed environment if identity assertion is enabled. Note that identity assertion feature is available only on the CSiv2 security protocol.

The LTPA authentication mechanism is designed for distributed security. Downstream servers can validate the security token. It also supports setting up a trust association relationship with reverse secure proxy servers and single signon (SSO), which will be discussed later. Besides the combination of LTPA and LDAP or Custom user registry interface, Version 5 supports LTPA with a LocalOS user registry interface. The new configuration is particularly useful for a single node with multiple application servers. It can function in a distributed environment if the local OS user registry implementation is a centralized user registry (such as Windows Domain Controller) or can be maintained in a consistent state on multiple nodes.

The product supports the J2EE Connector architecture and offers container-managed authentication. It provides a default J2C principal and credential mapping module that maps any authenticated user credential to a

password credential for the specified Enterprise Information Systems (EIS) security domain. The mapping module is a special JAAS login module designed according to the Java 2 Connector and JAAS specifications. Other mapping login modules can be plugged in.

Note: **5.1+** WebSphere Application Server Version 5.1 does not support Windows NT.

Backward compatibility

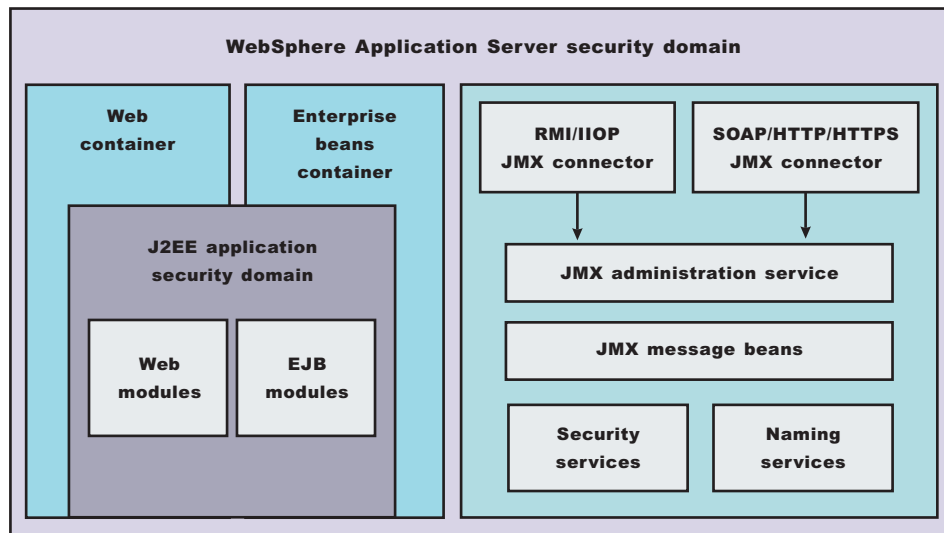
While adding new security functions and moving towards new industry standards, this version maintains backward compatibility with the 4.0.x and 3.5.x releases. Applications created in the Version 4.x development environment can deploy in Version 5. When Java 2 Security is enforced in Version 5, give special consideration to Version 4.0.x applications because Version 4.0 applications might not be Java 2 security compliant. Refer to the Security migration section for steps to port Version 4.0.x to Version 5. See also the Security section of What's new in this release.

Security for J2EE resources is provided by Web containers and EJB containers

Each container provides two kinds of security: *declarative security* and *programmatic security*. In declarative security, the security structure of an application, including data integrity and confidentiality, authentication requirements, security roles, and access control, is expressed in a form external to the application. In particular the deployment descriptor is the primary vehicle for declarative security in the J2EE platform. The product maintains a J2EE security policy, including information derived from the deployment descriptor and specified by deployers and administrators in a set of XML descriptor files. At run time, the container uses the security policy 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, the application code can use programmatic security to make access decisions. The API for programmatic security consists of two methods of the EJB EJBContext interface (`isCallerInRole`, `getCallerPrincipal`) and two methods of the servlet `HttpServletRequest` interface (`isUserInRole`, `getUserPrincipal`).

From a security perspective, every application server process consists of a Web container, an EJB container, and the administrative subsystem. There are many other components that constitute a server process, which are not discussed here. Security services consist of an authentication mechanism, a user registry, and an access control manager. Remote interfaces to the administrative subsystem, including the Admin Service interface through JMX connectors, the user registry interface, and the naming interface are protected by extended security role-based access control. The product supports the Java 2 security model. All the system code, including the administrative subsystem, the Web container, and the EJB container code, are running in the product security domain. The system code, shown in the WebSphere Application Server security domain box in the following diagram, is granted `AllPermission` and can access all system resources. Application code running in the application security domain, which by default is granted with permissions according to J2EE specifications, only can access a restricted set of system resources. The product run-time classes are protected by the product class loader and are kept invisible to application code.

WebSphere Application Server process



All of the application server processes, by default, share a common security configuration, which is defined in a cell-level security XML document. The security configuration determines whether product security is enforced, whether Java 2 security is enforced, the authentication mechanism and user registry configuration, security protocol configurations, JAAS login configurations, and Secure Sockets Layer configurations. Applications can have their own unique security requirements. Each application server process can create a per server security configuration to address its own security requirement. Not all security configurations can be modified at the application server level. Those can be modified at application server level include whether application security should be enforced, whether Java 2 security should be enforced, and security protocol configurations. The administrative subsystem security configuration is always determined by the cell level security document. The Web container and EJB container security configuration are determined by the optional per server level security document, which has precedence over the cell-level security document.

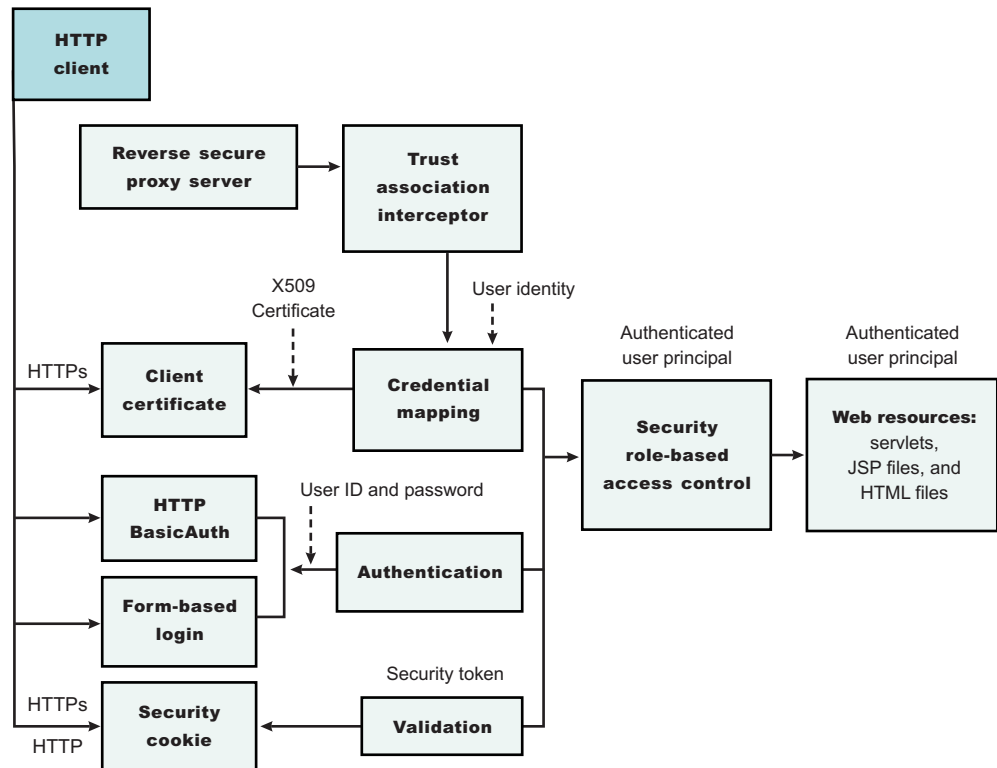
Security configuration, both at the cell level and at the application server level, are managed either by the Web-based administrative console application or by the WSADMIN scripting application.

Web security

When a security policy is specified for a Web resource and IBM WebSphere Application Server security is enforced, 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, ensure the data constraints are met, and determine whether the authenticated user has the required security role. The product supports the following login methods: HTTP basic authentication, Hypertext Transfer Protocol with Secure Sockets Layer (HTTPS) client authentication, and form-based Login. Mapping a client certificate to a product security credential uses the UserRegistry implementation to perform the mapping. The LDAP UserRegistry supports the mapping function while LocalOS UserRegistry does not.

When the LTPA authentication mechanism is configured and single signon (SSO) is enabled, an authenticated client is issued a security cookie, which can represent the user within the specified security domain. It is recommended that you use Secure Sockets Layer (SSL) to protect the security cookie from being intercepted and replayed. When a trust association is configured, the product can map an authenticated user identity to security credentials based on the trust relationship established with the secure reverse proxy server.

Web Security



The Web security collaborator enforces role-based access control by using an access manager implementation. An access manager makes authorization decisions based on the security policy derived from the deployment descriptor. An authenticated user principal can access the requested Servlet or JSP file if it has one of the required security roles. Servlets and JSP files can use the `HttpServletRequest` methods: `isUserInRole` and `getUserPrincipal`. As an example, the administrative console uses the `isUserInRole` method to determine the proper set of administrative functionality to expose to a user principal.

The EJB security collaborator enforces role-based access control by using an access manager implementation. An access manager makes authorization decisions based on the 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`. EJB code also can use the JAAS programming model to perform JAAS login and `WSSubject` `doAs` and `doAsPrivileged` methods. The code in the `doAs` and `doAsPrivileged` `PrivilegedAction` block executes under the `Subject` identity. Otherwise, the EJB method executes under either the `RunAs` identity or the caller identity, depending on the `RunAs` configuration.

EJB security

When security is enabled, the EJB container enforces access control on EJB method invocation. The authentication takes place regardless of whether a method permission is defined for the specific EJB method.

A Java application client can provide the authentication data in several ways. Using the `sas.client.props` file, a Java client can specify whether to use a user ID and password to authenticate or to use an SSL client certificate to authenticate. The client certificate is stored in the key file or in the hardware cryptographic card, as defined in a `sas.client.props` file. The user ID and password can be optionally defined in the `sas.client.props` file. At run time, the Java client can either perform a programmatic login or perform a *lazy authentication*. In lazy authentication when the Java client is accessing a protected enterprise bean for the first time the security run time tries to obtain the required authentication data. Depending on the configuration setting in `sas.client.props` file the security runtime either looks up the authentication data from this file or prompts the user. Alternatively, a Java client can use programmatic login. The product supports the JAAS programming model and the JAAS login (`LoginContext`) is the recommended way of programmatic login. The `login_helper request_login` helper function is deprecated in Version 5. Java clients programmed to the `login_helper` APT can run in this version.

The EJB security collaborator enforces role-based access control by using an access manager implementation.

An access manager makes authorization decisions based on the 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`. EJB code also can use the JAAS programming model to perform JAAS login and `WSSubject doAs` and `doAsPrivileged` methods. The code in the `doAs` and `doAsPrivileged PrivilegedAction` block executes under the Subject identity. Otherwise, the EJB method executes under either the `RunAs` identity or the caller identity, depending on the `RunAs` configuration. The J2EE `RunAs` specification is at the enterprise bean level. When `RunAs` identity is specified, it applies to all bean methods. The method level IBM `RunAs` extension introduced in Version 4.0 is still supported in this version.

Federal Information Processing Standards-approved

Federal Information Processing Standards (FIPS) are standards and guidelines issued by the National Institute of Standards and Technology (NIST) for federal 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.

WebSphere Application Server integrates cryptographic modules including Java Secure Socket Extension (JSSE) and Java Cryptography Extension (JCE), which are undergoing FIPS 140-2 certification. Throughout the documentation and the product, the IBM JSSE and JCE modules undergoing FIPS certification are referred to as `IBMJSSEFIPS` and `IBMJCEFIPS`, which distinguishes the FIPS modules from the IBM JSSE and IBM JCE modules. For more information on the FIPS certification process and to check the status of the IBM submission, see the Cryptographic Module Validation Program FIPS 140-1 and FIPS 140-2 Pre-validation List Web site.

The IBMJSSEFIPS module supports the FIPS-approved TLS cipher suites including:

- SHA
- DES
- TripleDES

The IBMJSSEFIPS module supports the following algorithms:

- RSA public key algorithm
- ANSI X9.31
- IBM Random Number Generator (Patent pending)

The IBMJCEFIPS module supports the following symmetric cipher suites:

- AES (FIPS 197)
- DES and TripleDES (FIPS 46-3)
- SHA1 Message Digest algorithm (FIPS 180-1)

The IBMJCEFIPS module supports the following algorithms:

- Digital Signature DSA and RSA algorithms (FIPS 186-2)
- ANSI X 9.31 (FIPS 186-2)
- IBM Random Number Generator

The IBMJSSEFIPS and IBMJCEFIPS cryptographic modules only contain the algorithms that are approved by FIPS, which form a proper subset of those in the IBM JSSE and IBM JCE modules.

Chapter 2. Securing applications and their environments

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

Consult the J2EE specifications for complete details.

1. Plan to secure your applications and environment. Complete this step before you install the WebSphere Application Server.
2. Consider pre-installation and post-installation requirements. For example, during this step, you learn how to protect security configurations after you install the product.
3. Migrate your existing security systems.
4. Develop secured applications.
5. Assemble secured applications. Development tools, such as the Assembly Toolkit are used to assemble J2EE modules and to set the attributes in the deployment descriptors.

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

Application assemblers combine J2EE modules, resolve references between them, and create from them a single deployment unit, typically an Enterprise Archive (EAR) file. Component providers and application assemblers can be represented by the same person but do not have to be.

6. Deploy secured applications.

Deployer link entities refer to in an enterprise application to the run time environment. The deployer:

- Maps actual users and groups to application roles
- Installs the enterprise application into the environment
- Makes the final adjustments needed to run the application

7. Test secured applications.
8. Manage security configurations.
9. Improve performance by tuning security configurations.
10. Troubleshoot security configurations.

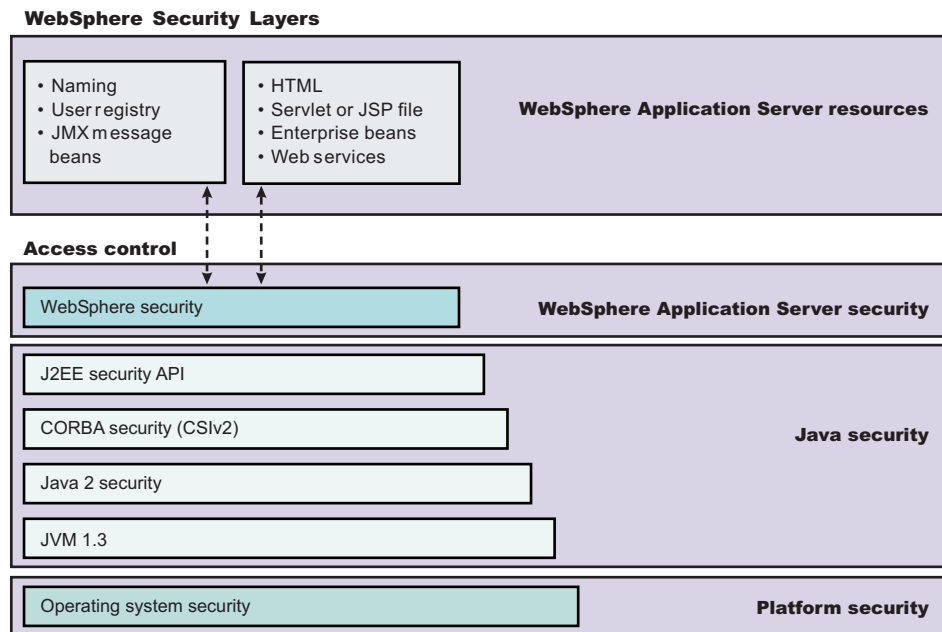
Your applications and production environment are secured.

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

Planning to secure your environment

There are several communication links from a browser on the Internet, through web servers and product servers, to the enterprise data at the back end. This section examines some typical configuration and common security practices. WebSphere Application Server security is built on a layered security architecture as showed below. 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. The following figure illustrates the building blocks that comprise the operating environment of WebSphere Security:



- **Operating System Security** - The security infrastructure of the underlying operating system provides certain security services to the WebSphere Security Application. This includes the file system security support to secure sensitive files in WebSphere product installation. The WebSphere system administrator can configure the product to obtain authentication information directly from the operating system user registry, for example the Windows system Security Access Manager (SAM).
- **Network Security** - The Network Security layers provide transport level authentication and message integrity and encryption. Communication between separate WebSphere Application Servers can be configured to use Secure Socket Layer (SSL) and HTTPS. Additionally IP Security and Virtual Private Network (VPN) might be used for added message protection.
- **JVM 1.3.1** - The JVM security model provides a layer of security above the operating system layer.
- **Java 2 Security** - The Java 2 Security model offers fine grained access control to system resources including file system, system property, socket connection, threading, class loading, and so on. Application code must explicitly grant the required permission to access a protected resource.
- **J2EE Security** - The security collaborator enforces J2EE based security policies and supports J2EE security APIs.
- **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.
- **CORBA Security** - Any calls made among secure ORBs are invoked over the Common Security Interoperability Version 2 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. WebSphere Application Server continues to support the Secure Authentication Service (SAS)

security protocol which was used in prior releases of WebSphere Application Server and other IBM products for backward compatibility.

The following picture shows a typical multiple-tier business computing environment. Shown in the picture is a WebSphere Application Server Network Deployment (ND) installation. Note that there is a Node Agent instance on every computer node which is omitted in the picture. Each product application server consists of a web container and an EJB container shown in the yellow shaded area and the administrative subsystem shown in red shaded area. The WebSphere Application Server Deployment Manager contains only WebSphere administrative code and the administrative console application. The administrative console is a special J2EE Web Application that provides the GUI interface for performing administrative functions. WebSphere Application Server configuration data is stored in XML descriptor files. Those XML configuration files should be protected by operating system security. Passwords and other sensitive configuration data can be modified using the administrative console. Hence, the administrative console Web application has a setup data constraint that requires the administrative console servlets and JSP files to be accessed only through an SSL connection when global security is enabled.

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

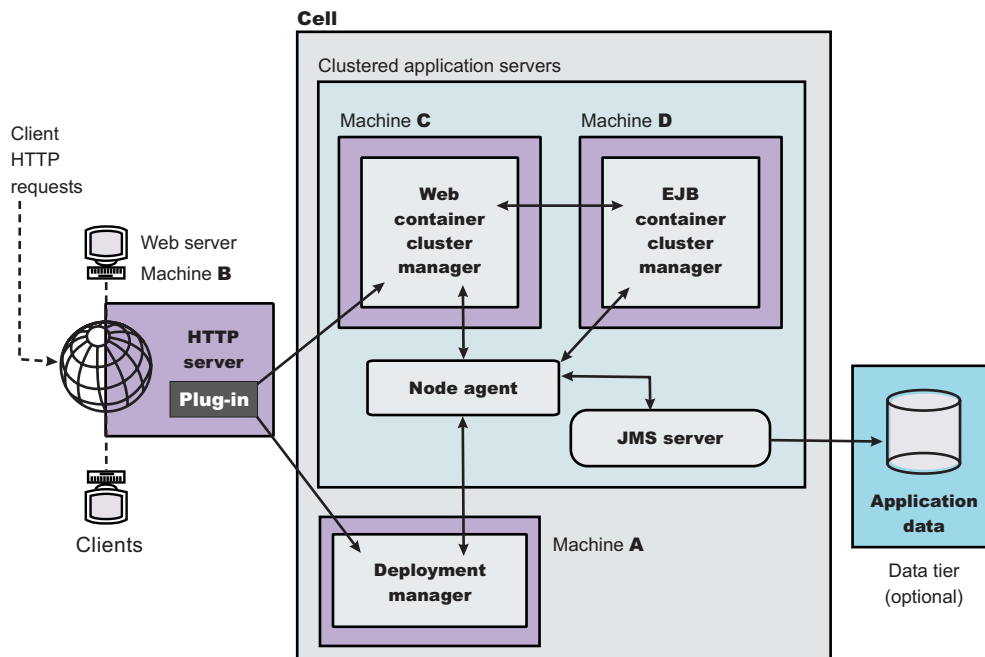


Figure 1. Multiple-tier business computing environment.

WebSphere Application Server servers interact with each other through CSIv2 and SAS security protocols as well and HTTP and or HTTPS protocols. Those protocols can be configured to use SSL when WebSphere Application Server global security

is enabled. The WebSphere Application Server administrative subsystem in every server uses SOAP JMX connectors and or RMI/IIOP JMX connectors to pass administrative commands and configuration data. When global security is disabled, the SOAP JMX connector uses HTTP protocol and the RMI/IIOP connector uses TCP/IP protocol. When global security is enabled, the SOAP JMX connector always uses HTTPS protocol. When global security is enabled, the RMI/IIOP JMX connector may be configured to either use SSL or to use TCP/IP. Again it is recommended to enable global security and enable SSL to protect the sensitive configuration data.

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

While global security is enabled, application security at each individual application server may be disabled by disabling the **per server level security enable** flag. Disabling application server security does not affect the administrative subsystem in that application server which is controlled only by the global security configuration. Both administrative subsystem and application code in an application server share the optional per server security protocol configuration. For more information, see the *Configuring Server Security for Network Deployment* article.

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

In declarative security, an application security structure includes data integrity and confidentiality, authentication requirements, security roles, and access control. Access control is expressed in a form external to the application. In particular the deployment descriptor is the primary vehicle for declarative security in the J2EE platform. The WebSphere Application Server maintains J2EE security policy including information derived from the deployment descriptor and specified by deployers and administrators in a set of XML descriptor files. At run time, the container uses the security policy 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, programmatic security may be used by application code to make access decisions. When global security is enabled and application server security is not disabled at the server level, J2EE applications security will be 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 would challenge the web client for authentication data if none is present according to the specified authentication method, ensure the data constraints are met, and determine 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 decision based on security policy derived from the deployment descriptor. An authenticated user principal is allowed to access the requested Servlet or JSP file if it has one of the required security roles. Servlets and JSP pages may use the `HttpServletRequest` methods `isUserInRole` and `getUserPrincipal`. When global security is enabled and application server security is not disabled, EJB container will enforce access control on EJB method invocation. The authentication would take place regardless of whether method permission was 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 based on security policy derived from the deployment descriptor. An authenticated user

principal is allowed to access the requested EJB method if it has one of the required security roles. EJB code may use the EJBContext methods `isCallerInRole` and `getCallerPrincipal`. The J2EE role based access control should be used to protect valuable business data from being accessed by unauthorized users from both the Internet and the Intranet. Securing Web applications using the Assembly Toolkit and Securing enterprise bean applications using the Assembly Toolkit.

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

- **Monitor role**, which can view configuration information and status but not anything more
- **Operator role**, which is a monitor that can trigger run time state changes, such as start an application server or stop an application, but cannot change configuration
- **Configurator role**, which is a monitor that can modify configuration information but cannot change run-time state
- **Administrator role**, which is an operator as well as a configurator

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

WebSphere Application Server administrative security is enforced when global security is enabled. It is recommended that WebSphere Application Server global security be enabled to protect administrative subsystem integrity. Application server security can be selectively disabled if there is no sensitive information to protect. For securing administrative security, refer to Assigning Users to Admin Roles Task and Assigning Users to Naming Roles WebSphere Application Server uses 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. J2EE Version 1.3 Specification defines typical set of Java 2 Security permissions that Web and EJB components should expect to have, which is shown in the table below.

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 |

Table 2. J2EE Security Permissions set for EJB components (continued)

| Security Permission | Target | Action |
|------------------------------|--------|--------|
| java.util.PropertyPermission | * | read |

WebSphere Application Server Java 2 Security implementation was based on J2EE Version 1.3 Specification. The Specification granted Web components read and write file access permission to any file in the file system, which may be too broad. WebSphere Application Server default policy gives Web components read and write permission to the sub directory and the sub tree where the Web module was installed. The default Java 2 Security policy for all Java virtual machines and WebSphere Application Server server processes are contained in the following policy files:

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

Used as the default policy for the Java Virtual Machine (JVM).

`${user.install.root}/properties/server.policy`

Used as the default policy for all product server processes

To simplify policy management, WebSphere Application Server policy is based on resource type rather than code base (location). Default policy for WebSphere Application Server subsystem that considered as an extension of WebSphere Application Server run time, which is referred to as *SPI*, for library shared by multiple applications, and for J2EE applications, are:

`${was.install.root}/config/cells/<cellname>/nodes/<nodename>/spi.policy`

Used for embedded resources defined in the resources.xml file, such as the Java Messaging Service (JMS), JavaMail and JDBC drivers.

`${was.install.root}/config/cells/<cellname>/nodes/<nodename>/library.policy`

Used by the shared library defined by WebSphere Application Server administrative console.

`${was.install.root}/config/cells/<cellname>/nodes/<nodename>/app.policy`

Used as the default policy for J2EE applications.

In general, applications should not require more permissions to run than those recommended by the J2EE Specification in order to be portable among various application servers. But some applications may require more permissions. WebSphere Application Server allows a per application policy file, `was.policy`, to be packaged together with each application from granting extra permissions to that application. Note that granting extra permissions to an application should be handled with great care because of the potential of compromising system integrity.

WebSphere Application Server uses a permission filtering policy file to alert users when an application requires permissions that are on the filter list during application installation and cause the offended application installation to fail. For example, the `java.lang.RuntimePermission exitVM` permission should not be given to an application so that application code is not allowed to terminate the WebSphere Application Server. The filtering policy is defined by the `filterMask` in `${was.install.root}/config/cells/<cellname>/filter.policy`. Moreover, WebSphere Application Server also performs run time permission filtering based on the run time filtering policy to ensure no application code has been granted any permission that is considered harmful to system integrity. Applying Java 2 Security model to application server is new.

WebSphere Application Server Version 4 supported Java 2 Security but only enforced three permissions checking against `exitVM`, `create` and `set` the Security Manager. Other permission checking is disabled by default. Hence many applications developed for prior releases of WebSphere Application Server might not be Java 2 Security ready. To migrate those applications to WebSphere Application Server Version 5 quickly, you might temporarily give those applications `java.security.AllPermission` in the `was.policy` file. It is recommended to test or make those applications Java 2 Security ready, for example, identify what extra permissions, if any, are required and to just grant those permissions to a particular application. Not granting applications `AllPermission` can certainly reduce the risk of compromising system integrity. For more information on migrating applications to WebSphere Application Server Version 5, refer to *Migrating Java 2 Security*.

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

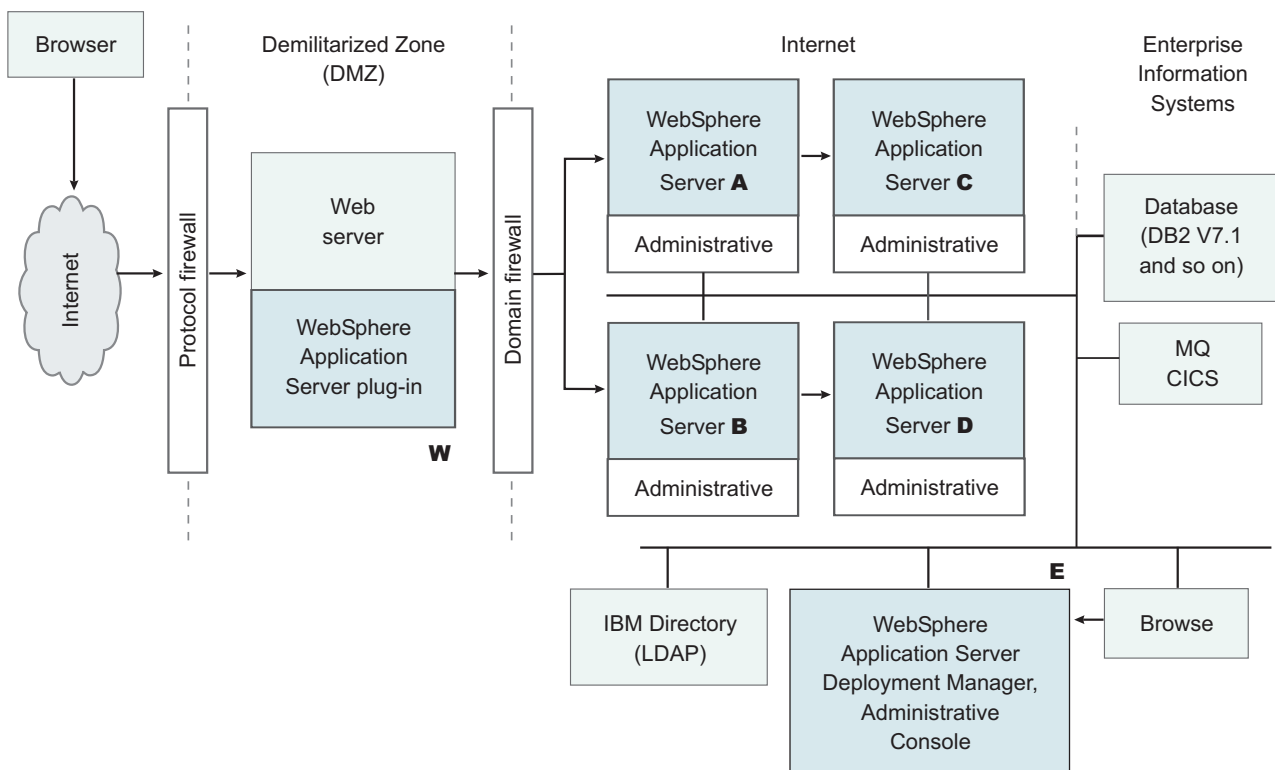
The global security enable flag and Java 2 Security enable flag along with other sensitive configuration data are stored in a set of XML configuration files. Both role based access control and Java 2 Security permission based access control are employed to protect the integrity of the configuration data. We will use configuration data protection as an example to illustrate how system integrity is maintained.

- When Java 2 Security is enforced, application code cannot access the WebSphere Application Server run-time classes that manages the configuration data unless it has been granted the required WebSphere Application Server run-time permissions.
- When Java 2 Security is enforced, application code cannot access the WebSphere Application Server configuration XML files unless it has been granted the required file read and write permissions.
- The JMX administrative subsystem provides SOAP over HTTP or HTTPS and RMI/IIOP remote interface to allow application programs to extract and to modify configuration files and data. When global security is enabled, an application program can modify WebSphere Application Server configuration provided that the application program has presented valid authentication data and that the security identity has the required security roles.
- If a user is allowed to disable Java 2 Security, then that user can modify the WebSphere Application Server configuration including the WebSphere Application Server security identity and authentication data along with other sensitive data. Hence, only users with the administrator security role are allowed to disable Java 2 Security.
- Because WebSphere Application Server security identity is given the administrator role, only users with the administrator role are allowed to disable global security, to change server ID and password, and to map users and groups to administrative roles, and so on.

Other WebSphere Application Server run time resources are protected by similar mechanism as described previously. Hence it is very important to enable WebSphere Application Server global security and to enforce Java 2 Security. J2EE

Specification defines four authentication method for Web components. WebSphere Application Server supports HTTP Basic Authentication, Form Based Authentication, and HTTPS Client Certificate Authentication. When using 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 will automatically redirect it to the HTTPS port. The CSiv2 security protocol also supports client certificate authentication. SSL client authentication can also be used to setup secure communication among selected set of servers based on trust relationship.

If you start from the WebSphere Application Server plug-in at the Web server, SSL mutual authentication can be configured between it and the WebSphere Application Server HTTPS server. When using a self- signed certificate, one can restrict the WebSphere Application Server plug-in to communicate with only the selected two WebSphere Application Server servers as shown in the following picture. Suppose you want to restrict the HTTPS server in WebSphere Application Server **A** and in WebSphere Application Server **B** to accept secure socket connections only from the WebSphere Application Server plug-in **W**. You can generate three self-signed certificate using the IKEYMAN tool and certificate management utility. For example, use certificate **W** and trust certificate **A** and **B**. The HTTPS server of WebSphere Application Server **A** is configured to use certificate **A** and to trust certificate **W**. The HTTPS server of WebSphere Application Server **B** is configured to use certificate **B** and to trust certificate **W**.



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

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

| | | |
|--------------------------------|---|---|
| WebSphere Application Server A | A | W |
| WebSphere Application Server B | B | W |

In a Network Deployment installation, the WebSphere Application Server Deployment Manager is a central point of administration. System management commands are sent from the Deployment Manager to each individual application server. When global security is enabled, all WebSphere Application Server servers can be configured to require SSL and mutual authentication. Suppose you want to further restrict that WebSphere Application Server A so that it can only communicate with WebSphere Application Server C and WebSphere Application Server B can only communicate with WebSphere Application Server D. Note, as mentioned previously, all WebSphere Application Server application servers must be able to communicate with WebSphere Application Server Deployment Manager E. Hence, when using self-signed certificates, You might configure the CSIv2 and SOAP/HTTPS Key and trust relationship as shown in the following table.

| Server | Key | Trust |
|---|-----|------------|
| WebSphere Application Server A | A | C, E |
| WebSphere Application Server B | B | D, E |
| WebSphere Application Server C | C | A, E |
| WebSphere Application Server D | D | B, E |
| WebSphere Application Server Deployment Manager E | E | A, B, C, D |

When WebSphere Application Server is configured to use an LDAP user registry, SSL with mutual authentication also can be configured between every application server and the LDAP server with self-signed certificate so that no password will be passed in clear text from WebSphere Application Server to the LDAP server. In this example, the node agent processes were not discussed. Each node agent needs to communicate with application servers on the same node and with the Deployment Manager. Node agents also need to communicate with LDAP servers when they are configured to use LDAP user registry. It is reasonable to let the Deployment manager and the node agents use the same certificate. Suppose application server A and C are on the same computer node. The Node agent on that node needs to have certificates A and C in its trust file. WebSphere Application Server does not provide a user registry configuration or management utility. In addition, it does not dictate the user registry password policy. It is recommended that you use the password policy recommended by your user registry, including the password length and expiration period.

1. Determine which versions of WebSphere Application Server you are using.
2. Review the WebSphere Application Server security architecture.
3. Review each of the following topics as also defined in Related reference.
 - Authentication Protocol for EJB Security
 - Supported Authentication Protocols: IBM SAS and OMG CSIv2
 - CSIv2 Features

- Identity Assertion
- Authentication Mechanisms
 - LTPA
 - Trust Association
 - Single Sign-On (SSO)
- User Registries
 - LocalOS
 - LDAP
- Custom user registries
- Java 2 Security
 - Dynamic Policy
- Java Authentication and Authorization Service (JAAS)
 - Programmatic login
- Java 2 Connector Security
- Access Control
 - J2EE Role-Based Authorization
 - Administrative Console and Naming Service Authorization
- Secure Socket Layer (SSL) Transport
 - Authenticity
 - Confidentiality
 - Integrity

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

At some point, you might decide to centralize the configuration of your stand-alone base application servers by adding them into a Network Deployment cell. If your base application server is currently configured with security, there are some issues to be considered. The major issue when adding a node to the cell is whether the user registries between the base application server and the Deployment Manager are the same. When adding a node to the cell, you automatically inherit both the user registry and the authentication mechanism of the cell. For distributed security, all servers in the cell must use the same user registry and authentication mechanism. In order to recover from a user registry change, you will need to modify your applications so that the user and group to role mappings are correct for the new user registry. To do this, see the article on Mapping users and groups to J2EE application roles.

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

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

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

-includeapps is optional, but this option attempts to include the server applications into the Deployment Manager. The addNode command might fail if the user registries used by the WebSphere Application Server and the Deployment Manager are not the same. To correct this problem, either make the user registries the same or turn off security. If you change the user registries, remember to verify that the users to roles and groups to roles mappings are correct. See the addNode command for more information on the addNode syntax.

2. Adding a secured remote node through the administrative console is not supported. You can either disable security on the remote node before performing the operation or perform the operation from the command line using the addNode script.
3. Before running the addNode command, you must verify that the truststore files on the nodes can communicate with the keystore files from the Deployment Manager and vice versa. When using the default DummyServerKeyFile and DummyServerTrustFile, you should not see this problem as these are already able to communicate. However, never use these dummy files in a production environment or anytime sensitive data is being transmitted.
4. After running addNode, the application server is in a new SSL domain. It might contain SSL configurations that point to keystore and truststore files that are not prepared to interoperate with other servers in the same domain. Consider which servers will be intercommunicating and ensure that the servers are trusted within your truststore files.

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

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

Security considerations specific to a multi-node or process Network Deployment environment

WebSphere Application Server Network Deployment allows for centralized management of distributed nodes and application servers. This inherently brings complexity, especially when security is included into the mix. Because everything is distributed, security plays an even larger role in ensuring that communications are appropriately secure between applications servers and node agents, and between node agents (a node specific configuration manager) and the Deployment Manager (a domain-wide, centralized configuration manager). The following issues should be considered when operating in this environment, but preferably prior to going to this environment.

Because the processes are distributed, the authentication mechanism that must be used is LTPA. The LTPA tokens are encrypted and signed and therefore, forwardable to remote processes. However, the tokens have expirations. The SOAP connector (the default connector) used for administrative security does not have

retry logic for expired tokens, however, the protocol is stateless so a new token is created for each request (if there is not sufficient time to execute the request with the given time left in the token). An alternative connector is the RMI connector, which is stateful and has some retry logic to correct expired tokens by resubmitting the requests after the error is detected. Also, because tokens have time specific expiration, the synchronization of the system clocks are crucial to the proper operation of token based validation. If the clocks are off by too much (approximately 10-15 minutes), you could easily run into unrecoverable validation failures which would have been avoided by having them in sync. Verify that the clock time, date, and time zones are all the same between systems. It is acceptable for nodes to be across time zones provided that the times are correct within the time zones (for example, 5 PM CST = 6 PM EST, and so on).

The following are issues to consider when using or planning for a Network Deployment environment.

1. When attempting run system management commands such as stopNode, you need to explicitly specify administrative credentials to perform the operation. Most commands accept -username and -password parameters to specify the user ID and password, respectively. The user ID and password that are specified should be an administrative user, for example, a user who is a member of the console users with **Operator** or **Administrator** privileges or the administrative user ID configured in the user registry. An example for stopNode, stopNode -username user -password pass.
2. Verify that the configuration at the node agents are always synchronized with the Deployment Manager prior to starting or restarting a node. To manually get the configuration synchronized, issue the syncNode command from each node that is not synchronized. To synchronize the configuration for node agents that are started, click **System Administration > Nodes** and select all started nodes. Click **Synchronize**.
3. Verify that the clocks on all systems are in sync including the time zone, time and date. If they are out of sync, the tokens will expire immediately when they reach the target server due to the time differences.
4. Verify that the LTPA token expiration period is long enough to complete your longest downstream request. Some credentials are cached and therefore the timeout does not always count in the length of the request. There is a LTPA token expiration *cushion* period for Web requests so that the LTPA token has sufficient time to make a downstream request. The cushion is 20 percent of the LTPA expiration period up to a maximum of ten minutes (configurable by a system property com.ibm.ws.security.cacheCushionMax specified in minutes) and usually not below the ORB request time-out (default 3 minutes, but the minute value is configurable by a system property com.ibm.ws.security.cacheCushionMin specified in minutes). When considering the actual expiration period of the LTPA token, you should consider this cushion. This helps prevent downstream expirations occurring in EJB servers.
5. The administrative connector used by default for system management is SOAP. SOAP is a stateless HTTP protocol. For most situations, this connector is sufficient. When running into a problem using the SOAP connector it might be desirable to change the default connector on all servers from SOAP to RMI. The RMI connector uses CSIv2, a stateful or interoperable protocol, and can be configured to use identity assertion (downstream delegation), message layer authentication (BasicAuth or Token), and or client certificate authentication (for server trust isolation). To change the default connector on a given server, go to the **Administration Services** in Additional Properties for that server.

6. The following error message might occur within the administrative subsystem security. This indicates that the sending process did not supply a credential to the receiving process. Typically the causes for this problem are:
 - The sending process has security disabled while the receiving process has security enabled. This typically indicates one of the two processes are not in sync with the cell.

Note: Having security disabled for a specific application server should not have any effect on administrative security.

- The clocks between the systems are not synchronized; thus invalidating the credential tokens immediately. Verify that the time, date, and time zones are consistent between the two machines. An error similar to the following might occur:

```
[9/18/02 16:48:23:859 CDT] 3b9cef35 RoleBasedAuth A SECJ0305I: Role based authorization check failed for security name <null>, accessId NO_CRED_NO_ACCESS_ID while invoking method propagateNotifications:[Ljavax.management.Notification; on resource NotificationService and module NotificationService.
```

7. When getting the following error message, you should validate that the clocks are synchronized between all servers within the cell, and the configurations are synchronized between all nodes and the Deployment Manager. An error similar to the following might occur:

```
[9/18/02 16:48:22:859 CDT] 3bd06f34 LTPAServerObj E SECJ0372E: Validation of the token failed.
```

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

When you have security problems related to the WebSphere Application Server Network Deployment environment, check the Security Troubleshooting section to find additional information about the problem. When trace is needed to solve a problem, because servers are distributed, quite often it is required to gather trace on all servers simultaneously while recreating the problem. This trace can be enabled dynamically or statically, depending on the type of problem occurring.

Creating login key files

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

```
Realm_name  User_ID  Password
```
3. Make sure that the data conforms to the following rules:
 - One realm name
 - One user ID, and one password defined in each entry
 - One entry per line
 - No blank lines between entries
 - Comments on separate lines only

- Begin any comment with a pound sign (#):

Example:

```
# Sample key file
#
# First target realm
#
TargetRealm serverID serverPassword
#
# Second target realm
#
TargetRealm2 serverID2 serverPassword2
#
# End of key file
```

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

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

Prepare truststore files for the WebSphere Application Server.

Preparing truststore files

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

1. Extract the public key of the z/OS server by using the key management tool from WebSphere Application Server z/OS. For details, see *Configuring the server for request decryption: choosing the decryption method*.
2. With the key management utility (iKeyman) from WebSphere Application Server, add the public key from the WebSphere Application Server for z/OS server as a signer certificate into the requesting WebSphere Application Server truststore file. For details, see the related information about how to *Import a signer certificate*.

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

Configure the WebSphere Application Server for interoperability.

Configuring the application server for interoperability

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

1. Configure the enterprise beans that access WebSphere Application Server for z/OS. Before deploying the enterprise beans, configure the RunAs Identity. Because the Security Authentication Service (SAS) only supports WebSphere

- Application Servers to interoperate with WebSphere Application Server for z/OS, set the RunAs Identity to System Identity.
2. Enable security.
 3. Enable outbound SAS authentication protocol.
 4. Specify the truststore file in an Secure Sockets Layer (SSL) configuration alias and configure the WebSphere Application Server with that alias.
 5. Set the **Request timeout** and **Locate request timeout** values to zero for the Object Request Broker (ORB) service. When the WebSphere Application Server z/OS application server first starts, no server region is available for processing work. It is therefore recommended that you set these two properties to zero to prevent potential timeouts.
 6. Specify a security property named `com.ibm.CORBA.keyFileName` for the absolute path of the login key file created earlier. See related information about how to configure a custom security property.
 7. Restart the WebSphere Application Server.

Implementing security considerations

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

1. Secure your environment before you install the WebSphere Application Server product. This step describes how to install WebSphere Application Server with the proper authority.
2. Install the WebSphere Application Server. This step describes how to install WebSphere Application Server. During installation you are prompted to migrate your security configurations.
3. Secure your environment after you install the WebSphere Application Server product. This step provides information on how to protect password information after you install WebSphere Application Server.

Securing your environment before installation

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

UNIX platforms

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

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

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

Linux platforms and Solaris operating environments

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

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

Windows platforms

On Windows platforms, the logon user must be a member of the administrator group with the rights of **Act as part of the operating system** and **Log on as a service**. If the Windows machine is a member of a Windows NT domain, then the logon user also must be created in the active directory and be a member of the administrator group with the rights of **Act as part of the operating system** and **Log on as a service** in the Windows domain.

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

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

Securing your environment after installation

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

The files in the `<install_root>\config` and `<install_root>\properties`, except for those in the following list, need protection. For example, give permission to the user who logs onto the system for WebSphere Application Server primary administrative tasks. Other users or groups, such as WebSphere Application Server console users and console groups, who perform partial WebSphere Application Server administrative tasks, like configuring, starting servers and stopping servers, need permissions as well. The files in the `install_root\properties` directory that should not be protected are:

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

1. Secure files on a Windows system:
 - a. Open the browser for a view of the files and directories on the machine.
 - b. Locate and right-click the file or the directory to protect.
 - c. Click **Properties**.
 - d. Click the **Security** tab.
 - e. Remove the Everyone entry and any other user or group that should not have access to the file.

- f. Add the users who should be allowed to access the files with the proper permission.
2. Secure files on UNIX systems. This procedure applies only to the ordinary UNIX file system. If your site uses access-control lists, secure the files by using that mechanism. Any site-specific requirements can affect the desired owner, group and corresponding privileges. For example, on AIX,
 - a. Go to the *install_root* directory and change the ownership of the directory configuration and properties to the user who logs onto the system for WebSphere Application Server primary administrative tasks. Execute the following command: `chown -R logon_name directory_name`
Where:
 - *login_name* is a specified user or group.
 - *directory_name* is the name of the directory that contains the files.
 It is recommended that you assign ownership of the files containing password information to the user who runs the application server. If more than one user runs the application server, provide permission to the group in which the users are assigned in the user registry.
 - b. Set up the permission by executing the following command: `chmod -R 770 directory_name`.
 - c. Go to the *install_root*\properties directory and set the following file permission to **everybody** by executing the following command: `chmod 777 file_names`, where *file_names* are the following files:
 - TraceSettings.properties
 - client.policy
 - client_types.xml
 - implfactory.properties
 - sas.client.props
 - sas.stdclient.properties
 - sas.tools.properties
 - soap.client.props
 - wsadmin.properties
 - wsjaas_client.conf
 - d. Create a group for WebSphere Application Server and put the users who perform full or partial WebSphere Application Server administrative tasks in that group.
 - e. Restrict access to the /var/mqm directories and log files needed for WebSphere embedded messaging or WebSphere MQ as the JMS provider. Give write access only to the user ID mqm or members of the mqm user group. For detailed information, see Securing messaging directories and log files.

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

If there are any failures caused by file accessing permissions, check the permission settings.

Protecting plain text passwords

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

| File name | Additional information |
|--|---|
| security.xml | The following fields contain encoded passwords: <ul style="list-style-type: none"> • LTPA password • JAAS Auth Data • User Registry server password • LDAP User Registry bind password • Key file password • Trust file password • Crypto token device password |
| sas.client.props | |
| war/WEB-INF/ibm_web_bnd.xml | Specify passwords for the default basic authentication for the "resource-ref" bindings within all descriptors (except in the Java cryptography architecture) |
| ejb_jar/META-INF/ibm_ejbjar_bnd.xml | Specify passwords for the default basic authentication for the "resource-ref" bindings within all descriptors (except in the Java cryptography architecture) |
| client_jar/META-INF/ibm-appclient_bnd.xml | Specify passwords for the default basic authentication for the "resource-ref" bindings within all descriptors (except in the Java cryptography architecture) |
| ear/META-INF/ibm_application_bnd.xml | Specify passwords for the default basic authentication for the "run as" bindings within all descriptors |
| server.xml | The following fields contain encoded passwords: <ul style="list-style-type: none"> • key file password • trust file password • crypto token device password • auth target password • Session persistence password • DRS Client data replication password (not available in WebSphere Application Server, Version 5) |
| resource.xml (for cells, servers, and nodes) | The following fields contain encoded passwords: <ul style="list-style-type: none"> • WAS40Datasource password • mailTransport password • mailStore password • MQQueue queue mgr password |

| File name | Additional information |
|--------------------------------------|------------------------|
| ws-security.xml | |
| ibm-webservices-bnd.xmi | |
| ibm-webservicesclient-bnd.xmi | |
| /properties/soap.client.props | |
| /properties/sas.tools.properties | |
| /properties/sas.stdclient.properties | |
| wsserver.key | |

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

1. Access the file using a text editor and type over the encoded password in plain text. The new password is shown in plain text and must be encoded.
2. Use the PropFilePasswordEncoder.bat file in the <install_dir>/bin/ directory to re-encode the password.
 - If you are re-encoding SAS properties files, type <file_name> -sas and the PropFilePasswordEncoder.bat file encodes the known SAS properties.
 - If you are encoding files that are not SAS properties files, type <file_name> -sas <password_properties_list>

<file_name> is the name of the SAS properties file. <password_properties_list> is the name of the properties to encode within the file.

If you reopen the affected file or files, the passwords do not display in plain text. Instead, the passwords appear encoded.

Migrating security configurations from previous releases

This article addresses the need to migration your security configurations from a previous release of IBM WebSphere Application Server to WebSphere Application Server, Version 5. Complete the following steps to migrate your security configurations:

- Before migrating your configurations, verify that the administrative server of the previous release is running.
 - If security is enabled in the previous release, obtain the server ID and password of the previous release. This information is needed to log onto the administrative server of the previous release during migration.
 - You can optionally disable security in the previous release before migrating the installation. There is no logon required during the installation.
1. Start the Installation Wizard by running the **install** command.
 2. On the Installation Wizard panel, click **Specify previous version information > Migrate your applications and configuration from the previous version**. Complete the fields for **Install location** and **Configuration file** with corresponding information.
 3. Follow the instructions provided in the Installation Wizard to complete the installation.

The security configuration of previous WebSphere Application Server releases and its applications are migrated to the new installation of WebSphere Application Server Version 5.

This task is for migrating an installation.

After migration is complete, re-enable your security configurations with security disabled in the migrated version. You must re-enable your security configurations even if the previous version had security enabled.

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

If you upgrade from WebSphere Application Server, Version 4.0.x to WebSphere Application Server, Version 5 or later, data associated with Version 4.0.x trust associations is not automatically migrated to Version 5 or later. To migrate trust associations, see “Migrating trust association interceptors” on page 31.

Migrating custom user registries

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

In WebSphere Application Server Version 5, 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.

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

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

If the migration tool was used to migrate the WebSphere Application Server Version 4 configuration to WebSphere Application Server Version 5, be aware that this migration does not involve any changes to your existing code. Since the

WebSphere Application Server Version 4 custom registry works in WebSphere Application Server Version 5 without any changes to the implementation (except when using data sources) you can use the Version 4-based custom registry after the migration without modifying the code. Consider that the migration tool might not copy your implementation files from Version 4 to Version 5. You might have to copy them to the class path in the Version 5 setup (preferably to the `classes` subdirectory, just like in Version 4). If you are using the WebSphere Application Server Network Deployment version, copy the files to the cell and to each of the nodes class paths.

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

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

Before proceeding, look at the new `UserRegistry` interface. The section titled "Developing custom user registries" under **Developing secured applications** describes each of these methods in detail and also indicates the changes from Version 4.

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

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

```
public class FileRegistrySample implements CustomRegistry
to
public class FileRegistrySample implements UserRegistry
```

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

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

Having a certificate chain gives you the flexibility to act on the chain instead of one certificate. If you are only interested in the first certificate just take the first certificate in the chain before processing. In Version 5, the `mapCertificate` method is called to map the user in a certificate to a valid user in the registry, when certificates are used for authentication by the Web or the Java clients (transport layer certificates, Identity Assertion certificates). In Version 4, this was only called by Web clients since the Common Secure Interoperability Version 2 (CSIv2) protocol was not supported.

4. Remove the `getUsers()` method.

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

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

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

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

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

to

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

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

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

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

to

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

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

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

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

```

        EntryNotFoundException {
            return null;
        }

```

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

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

```

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

```

Type the previous lines as one continuous line.

To build the Version 4 custom registry in Version 5 and Version 5.0.1, only the `wssec.jar` file is required.

To build the Version 4 custom registry in Version 5.0.2, 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. If you are using the Network Deployment product, make sure that you copy these files to the cell and all the nodes. Without the files in each of the node class paths the nodes and the application servers in those nodes cannot start when security is on.
13. Use the administrative console to set up the custom registry. Follow the instructions in the [Configuring custom user registries](#) article to set up the custom registry including the `IgnoreCase` flag. Make sure that you add the `WAS_UseDisplayName` properties, if required.

Migrates a Version 4 custom registry to the Version 5 custom registry.

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

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

Migrating trust association interceptors

The following topics are addressed in this document:

- Changes to the product-provided trust association interceptors
- Migrating product-provided trust association interceptors
- Changes to the custom trust association interceptors
- Migrating custom trust association interceptors

Changes to the product-provided trust association interceptors

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

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

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

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

Migrating product-provided trust association interceptors

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

Changes to the custom trust association interceptors

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

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

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

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

Method 1:

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

Method 2:

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

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

The `init(Properties)` method accepts a `java.util.Properties` object which contains the set of properties required to initialize the interceptor. All the properties set for an interceptor (by using the Custom Properties link for that interceptor or using scripting) will be sent to this method. The interceptor can then use these properties

to initialize itself. For example, in the product provided implementation for the WebSEAL server, this method reads the hosts and ports so that a request coming in can be verified to come from trusted hosts and ports. A return value of 0 implies that the interceptor initialization is successful. Any other value implies that the initialization was not successful and the interceptor will not be used.

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

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

Method 1:

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

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

Method 2:

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

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

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

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

Migrating custom trust association interceptors

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

1. Recompile the implementation file, if necessary.

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

To recompile the implementation file, type the following:

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

Note: The previous line of code was broken into two lines due to the width of the page. Type the code as one continuous line.

2. Copy the custom trust association interceptor class files to a location in your product class path. It is suggested that you copy these class files into the %WAS_HOME%/lib/ext directory.
3. Start the WebSphere Application Server
4. Enable security to use the trust association interceptor. *The properties located in your custom trust association properties file and in the trustedserver.properties file are not migrated from previous versions of WebSphere Application Server to version 5. You must migrate the appropriate properties to WebSphere Application Server, version 5 using the trust association panels in the GUI. For more information, see Configuring trust association interceptors.*

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

WebSphere Application Server Version 5 fully supports the Java Authentication and Authorization Service (JAAS) as programmatic login APIs. See Configuring Java Authentication and Authorization Service and Developing with JAAS to log in programmatically, for more details on JAAS support.

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

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

The sampleApp is not included in Version 5.

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

The sampleApp is not included in Version 5.

- **com.ibm.IExtendedSecurity_LoginHelper.**

This API is included with the product, but is deprecated.

- **org.omg.SecurityLevel2.Credentials.** This API is included with the product, but not recommended to use.

The alternative APIs provided in WebSphere Application Server Version 5 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, which is included with the product

(`${was.install.root}/web/docs/jaas/JaasDocs.zip`) and the product Javadoc (`${was.install.root}/web/apidocs/index.html`) for details.

- 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:** A non-prompt CallbackHandler, application pushes basic authentication data (user ID, password, and security realm) or token data to product LoginModules. This API is recommended for server-side login.
 - **com.ibm.websphere.security.auth.callback.WSGUICallbackHandlerImpl:** A GUI login prompt CallbackHandler to gather basic authentication data (user ID, password, and security realm). This API is recommended for client-side login.

Note: If this API is used on the server side, the server is blocked for input.

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

Note: If this API is used on the server side, the server is blocked for input.

 - javax.security.auth.callback.Callback interface:
 - **javax.security.auth.callback.NameCallback:** Provided by JAAS to pass the user name to the LoginModules interface.
 - **javax.security.auth.callback.PasswordCallback:** Provided by JAAS to pass the password to the LoginModules interface.
 - **com.ibm.websphere.security.auth.callback.WSCredTokenCallbackImpl:** Provided by the product to perform a token-based login. With this API, an application can pass a token-byte array to the LoginModules interface. - **javax.security.auth.spi.LoginModule interface:** WebSphere Application Server provides LoginModules implementation for client and server-side login. Refer to Configuring Java Authentication and Authorization Service 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 credentials is created and stored in the Subject.
 - **com.ibm.websphere.security.auth.WSPincipal:** An authenticated principal, that is created and stored in a Subject that is authenticated by the WebSphere LoginModules interface.
1. Use the following as an example of how to perform programmatic login using the CORBA-based programmatic login APIs: The CORBA-based programmatic login APIs are replaced by JAAS login.

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

        // Create a new security context to hold authentication data.
        LoginHelper loginHelper = new LoginHelper();
        try {
            // Provide the ID and password of the user for authentication.
```

```

org.omg.SecurityLevel2.Credentials credentials = loginHelper.login(userid, password);

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

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

```

2. Use the following example to migrate the CORBA-based programmatic login APIs to the JAAS programmatic login APIs. The following example assumes that the application code is granted for the required Java 2 security permissions. See *Configuring Java Authentication and Authorization Service*, *Configuring Java 2 security* and *JAAS* documentation located in the `/${was.install.root}/web/docs/jaas/JaasDocs.zip` file for details.

```

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 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();

// 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 WSPincipal.
java.util.Set ps =
s.getPrincipals(com.ibm.websphere.security.auth.WSPincipal.class);
java.util.Iterator it = ps.iterator();
while (it.hasNext()) {
com.ibm.websphere.security.auth.WSPincipal p =
(com.ibm.websphere.security.auth.WSPincipal) 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 CORBA-based programmatic login application to JAAS-based applications.

Migrating from the CustomLoginServlet class to servlet filters

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

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

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

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

1. Develop a form login page and error page for the application, as described in Developing form login pages.
2. **5.1+** Configure the form login page and the error page for the application as described in “Securing Web applications using the Assembly Toolkit” on page 78.
3. Develop servlet filters if additional processing is required before and after form login authentication. Refer to Developing servlet filters for form login processing for details.
4. Configure the servlet filters developed in the previous step for either the form login page URL or for the /j_security_check URL. Use an assembly tool or development tools like WebSphere Application Development Studio to configure filters. After configuring the servlet filters, the web-xml file contains two stanzas. The first stanza contains the servlet filter configuration, the servlet filter, and its implementation class. The second stanza contains the filter mapping section and a mapping of the servlet filter to the URL. In this case, the servlet filter maps to /j_security_check.

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

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

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

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

Developing secured applications

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

- Development
- Assembly

- Deployment

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

1. Develop secure Web applications.
2. Develop servlet filters for form login processing.
3. Develop form login pages.
4. Develop enterprise bean component applications.
5. Develop with Java Authentication and Authorization Service to log in programmatically.
6. Develop your own Java 2 security mapping module.
7. Develop custom user registries.
8. Develop a custom interceptor for trust associations.

Developing with programmatic security APIs for Web applications

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

getRemoteUser()

Returns the user name the client used for authentication. Returns **null** if no user is authenticated.

isUserInRole

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

getUserPrincipal()

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

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

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

A programmatically secured servlet application.

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

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

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

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

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

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

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

    ....

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

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

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

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

}
```

After developing an application, use the Assembly Toolkit 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 the Assembly Toolkit*.

Example: Web applications code

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

File : HelloServlet.java

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

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

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

        String s = "Hello";

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

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

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

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

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

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

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

Developing servlet filters for form login processing

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

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

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

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

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

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

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

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


```

import javax.servlet.*;

public class LoginFilter implements Filter {

    protected FilterConfig filterConfig;

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

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

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

        // perform pre-login action here

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

        // j_security_check if this filter is
        // mapped to j_security_check.

        // perform post-login action here.

    }
}

```

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

Configuring servlet filters:

WebSphere Application Development Studio or the Assembly Toolkit can configure the servlet filters. There are two steps in configuring a servlet filter.

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

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

```

<filter id="Filter_1">
    <filter-name>LoginFilter</filter-name>
    <filter-class>LoginFilter</filter-class>
    <description>Performs pre-login and post-login
        operation</description>

```

```

    <init-param>// optional
        <param-name>ParameterName</param-name>
        <param-value>ParameterValue</param-value>
    </init-param>
</filter>

```

2. Map the servlet filter to URL or servlet.

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

```

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

```

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

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

Servlet filter source code: LoginFilter.java

```

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

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

public class LoginFilter implements Filter {

    protected FilterConfig filterConfig;

    java.util.List revokeList;

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

```

```

        this.filterConfig = filterConfig;

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

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

    /**
     * doFilter() : doFilter() method called before the servlet to
     * which this 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` was broken into two lines due to the width of the page. The `public void doFilter(ServletRequest request` line and the line after it are one continuous line.

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

```

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

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

```

An example of a revoked user list file:

```

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

```

Developing form login pages

A Web client or browser can authenticate a user to a Web server using one of the following mechanisms:

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

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

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

Form login uses the servlet `sendRedirect()` method, which has several implications for the user. The `sendRedirect()` method is used twice during form login:

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

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

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
2. Create an error page. You can program error pages to retry authentication or display an appropriate error message.

3. Place the login page and error page in the Web archive (WAR) file relative to the top directory. For example, if the login page is configured as /login.html in the deployment descriptor, place it in the top directory of the WAR file. An assembler can also perform this step using the assembly tool.
4. Create a form logout page and insert it to the application only if required.

This step is required when a Web application requires a form-based authentication mechanism.

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

Example: Form login

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

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

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

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

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

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

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

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

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

</body>
</html>
```

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

```
<login-config id="LoginConfig_1">
<auth-method>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 showing login and error pages for the previous login configuration:

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

Form logout

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

Suppose that it is desirable to log out after logging into a Web application and perform some actions. A form logout works in the following manner:

1. The logout-form URI is specified in the Web browser and loads the form.
2. The user clicks **Submit** on the form to log out.
3. The WebSphere security code logs the user out.
4. Upon logout, the user is redirected to a logout exit page.

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

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

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

Developing with programmatic APIs for EJB applications

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

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

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

possible if development tools such as WebSphere Studio Application Developer is used. You also can create the security-role-ref element during the assembly stage using an assembly tool.

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

A programmatically secured EJB application.

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

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

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

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

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

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

```
public class aSessionBean implements SessionBean {

    .....

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

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

```

....

private void aBusinessMethod() {
....

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

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

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

....
}

....
}

```

After developing an application, use the Assembly Toolkit 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 using the Assembly Toolkit.

Example: Enterprise bean application code

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

A remote interface

File : Hello.java

```

package tests;
import java.rmi.RemoteException;
/**
 * Remote interface for Enterprise Bean: Hello
 */
public interface Hello extends javax.ejb.EJBObject {
    public abstract String getMessage()throws RemoteException;
    public abstract void setMessage(String s)throws RemoteException;
}

```

A home interface

File : HelloHome.java

```

package tests;
/**
 * Home interface for Enterprise Bean: Hello
 */
public interface HelloHome extends javax.ejb.EJBHome {
/**
 * Creates a default instance of Session Bean: Hello

```

```

    */
    public tests.Hello create() throws javax.ejb.CreateException,
        java.rmi.RemoteException;
}

```

A bean implementation

File : HelloBean.java

```

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

    public java.lang.String message;

    //business methods

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

```

```

    }

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

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

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

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

```

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

```

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

```

5.1+ For an explanation of how to create a `<security-role-ref>` element, see “Securing enterprise bean applications using the Assembly Toolkit” on page 75. Use the information under Map security-role-ref and role-name to role-link to create the element.

Developing with the Java Authentication and Authorization Service to log in programmatically

Java Authentication and Authorization Service (JAAS) is a new feature in WebSphere Application Server Version 5. Java Authentication and Authorization Service represents the strategic application programming interfaces (API) for authentication and it replaces the CORBA programmatic login APIs. WebSphere Application Server provides some extension to JAAS:

- Refer to the Developing applications that use CosNaming (CORBA Naming interface) article for details on how to set up the environment for thin client applications to access remote resources on a server.
- If the application uses custom JAAS login configuration, verify that it is properly defined. See the Configuring Java Authentication and Authorization Service login configuration article for details.
- Some of the JAAS APIs are protected by Java 2 Security permissions. If these APIs are used by application code, verify that these permissions are added to the application was.policy file. See Adding the was.policy file to Application, Using the Policytool to edit policy file and Configure was.policy articles for details. For more details on which APIs are protected by Java 2 Security permissions, check the IBM Application Developer Kit, Java Technology Edition; JAAS and WebSphere Application Server public APIs javadoc in Resources for

learning. Some of the APIs used in the sample code in this documentation and the Java 2 Security permissions required by these APIs follow:

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

- **ClientContainer JAAS login configuration.** This JAAS login configuration honors the CallbackHandler specified in the client container deployment descriptor. The login module of this login configuration uses the CallbackHandler in the client container deployment descriptor if one is specified, even if the application code specified one CallbackHandler in the LoginContext. This is for a client container application.

A Subject authenticated with the previously mentioned JAAS login configurations contains a `com.ibm.websphere.security.auth.WSPPrincipal` principal and a `com.ibm.websphere.security.auth.WSCredential` credential. If the authenticated Subject is passed in `com.ibm.websphere.security.auth.WSSubject.doAs()` or the other `doAs()` methods, the product security run time can perform authorization checks on J2EE resources based on the Subject `com.ibm.websphere.security.auth.WSCredential`.

- **Customer-defined JAAS login configurations.** You can define other JAAS login configurations to perform programmatic authentication to your authentication mechanism. See the Managing Java Authentication and Authorization Service Login Configuration article for details. For the product security run time to perform authorization checks, the subjects from these customer-defined JAAS login configurations must contain the required principal and credentials.
- **Naming requirements for programmatic login on a pure Java client.** When programmatic login occurs on a pure Java client and the property `com.ibm.CORBA.validateBasicAuth` equals `true`, it is necessary for the security code to know where the SecurityServer resides. Typically, the default InitialContext is sufficient when a `java.naming.provider.url` property is set as a system property or when the property is set in the `jndi.properties` file. In other cases it is not desirable to have the same `java.naming.provider.url` properties set in a system wide scope. In this case, there is a need to specify security specific bootstrap information in the `sas.client.props` file. The following steps present the order of precedence for determining how to find the SecurityServer in a pure Java client:

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

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

If you specify these properties, you are guaranteed that security looks here for the SecurityServer. The host and port specified can represent any valid WebSphere host and bootstrap port. The SecurityServer resides on all server processes and therefore it is not important which host or port you choose. If specified, the security infrastructure within the client process look up the SecurityServer based on the information in the `sas.client.props` file.

2. Place the following code in your client application to get a new InitialContext():

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

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

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

```

Context initialContext = new InitialContext(env);
Object obj = initialContext.lookup("");

// programmatic login code goes here.

```

Complete this step prior to executing any programmatic login. It is in this code that you specify a URL provider for your naming context, but it must point to a valid WebSphere Application Server within the cell that you are authenticating to. This allows thread specific programmatic logins going to different cells to have a single system-wide SecurityServer location.

3. Use the new default InitialContext() method relying on the naming precedence rules. These rules are defined in the article, Example: Getting the default initial context.

See the article, Example: Java Authentication and Authorization Service programmatic login.

Example: JAAS programmatic login

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

```

LoginContext lc = null;

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

```

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

For a pure Java application client, the product provides two other CallbackHandler implementations: WSStdinCallbackHandlerImpl and WSGUICallbackHandlerImpl, which prompt for user ID, password, and realm information on the command line

and pop-up panel, respectively. You can choose either of these product `CallbackHandler` implementations depending on the particular application environment. You can develop a new `CallbackHandler` if neither of these implementations fit your particular application requirement.

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

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

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

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

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

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

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


Developing your own J2C principal mapping module

WebSphere Application Server provides principal mapping when Java 2 Connector (J2C) connection factory is configured to perform container managed sign-on. For example, the application server can map the caller principal to a resource principal to open a new connection to the backend server. With the container-managed signon, WebSphere Application Server creates a Subject instance that contains enterprise information systems (EIS) security domain credentials. A Subject object returned by a principal mapping module contains a Principal object represents the caller identity and a PasswordCredential or a GenericCredential. WebSphere Application Server provides a default principal mapping module that maps any authenticated user credentials to password credentials for the EIS security domain. The default mapping module is defined in the Application Login Configuration panel in the DefaultPrincipalMapping entry. The user ID and password for the EIS security domain is defined under each connection factory by an authDataAlias attribute *container-managed authentication alias* in the administrative console. The authDataAlias attribute does not actually contain the user name and password. An authDataAlias attribute contains an alias that refers to a user name and password pair that is defined in the security configuration document. Since it contains sensitive data, the security configuration document requires the most privileged **administrator** role for both read and write access. This indirection avoids saving sensitive user name and password in configuration documents other than the security document.

The J2C connection factory configuration contains a mapping module, which defines a principal mapping module alias (mappingConfigAlias attribute) and an authentication data alias (authDataAlias attribute). At run time, the J2C-managed connection factory code passes a reference of the ManagedConnectionFactory and an authDataAlias object to the configured principal mapping module through the WSPrincipalMappingCallbackHandler object. WebSphere Application Server supports plugging in a custom principal mapping module for a connection factory if the any-authenticated-to-one mapping provided by the default principal mapping module is insufficient. A custom mapping module is a special purpose Java Authentication and Authorization Service (JAAS) Login Module that performs principal or credential mapping in the login method. The WSSubject.getCallerPrincipal() method can be used to retrieve the application client identity. Plugging in a custom mapping module is very simple. Change the value of the mappingConfigAlias object to the custom mapping module. However, the configuration must be done through the wsadmin tool.

The following steps are needed to perform this task. You can use the administrative console for the first few steps. Use the wsadmin tool for the rest of the configuration. You can use the administrative console for these steps. However, you also can use the wsadmin tool to configure the J2C Connection Factory.

1. Start the administrative console. To add a custom mapping module for an application server, click **Servers > Application Servers**. Click the particular server on the right navigation panel.
2. Click **Security > JAAS Configuration**.
3. Select **JAAS Configuration** and **Application Logins**. Click **New**.
4. Enter a unique alias for the new mapping module, and click **Apply**.
5. Under Additional Properties, click **JAAS Login Modules** to define the custom mapping module class.
6. Click **New** and enter the **Module Classname** and the **Authentication Strategy**.
7. Click **Apply**. Click **Save** to save the new configuration.

8. Configure the J2C Connection Factory to use the new mapping module using either the administrative console or the wsadmin tool.
 - a. Using the administrative console to configure the J2C Connection Factory.
 - 1) Click **Resources > Resource Adapters > resource_adapter**.
 - 2) Under Additional Properties, click **CMP Connection Factories**.
 - 3) Click the name of your connection factory.
 - 4) Enter the resource name, Java Naming and Directory Interface (JNDI) name, a description of the resource, and a category in which to classify the resource.
 - 5) Click **OK**.
 - 6) Click **Save** in the upper-left section of the administrative console to save your configuration changes.
 - b. Using the wsadmin tool to configure the J2C Connection Factory.
 - 1) At the wsadmin prompt, type the following command to show a list of J2CConnectionFactory objects: `wsadmin>$AdminConfig list J2CConnectionFactory`.
 - 2) Select the **J2C Connection Factory** and enter the following command to show all the attributes. For example,


```
wsadmin>$AdminConfig show PetStore_CF(cells/hillsideNetwork/nodes/hillside/servers/server1:resources.xml#CMPConnectorFactory_4)
```

 The previous example was split onto two lines because it displayed beyond the width of the page.
 - 3) Type the following command to examine the current mapping module configuration:


```
wsadmin>$AdminConfig show {mapping (cells/hillsideNetwork/nodes/hillside/servers/server1:resources.xml#MappingModule_7)}
```

 The previous example was split onto two lines because it displayed beyond the width of the page.
 The following shows sample results of the above command:

```
{authDataAlias {}} {mappingConfigAlias DefaultPrincipalMapping}
```

 As shown in the previous example, the J2C Connection factory is configured to use the DefaultPrincipalMapping login configuration.
 - 4) Type the following command to modify the mapping module configuration to use the new mapping module:


```
wsadmin>$AdminConfig modify {mapping (cells/hillsideNetwork/nodes/hillside/servers/server1:resources.xml#MappingModule_7)} {mappingConfigAlias myMappingModule}
```

 The previous example was split onto three lines because it displayed beyond the width of the page.
 You can check the result by typing:


```
wsadmin>$AdminConfig show {mapping (cells/hillsideNetwork/nodes/hillside/servers/server1:resources.xml#MappingModule_7)} {authDataAlias {}} {mappingConfigAlias myMappingModule}
```

 The previous example was split onto three lines because it displayed beyond the width of the page.
 - 5) Type `save` at the wsadmin prompt to save your changes.

Note: The `authDataAlias` is left undefined. In practice, the `authDataAlias` passes at run time to the custom mapping module. But using the `authDataAlias` to look up user ID and password requires the WebSphere Common Configuration Model (WCCM) programming interface, which is not available at this time.

A mapping module is defined and is configured for the specified J2C Connection factory.

Completing this task allows you to use your own mapping module to fit your application environment. The WebSphere Application Server default principal mapping module maps all authenticated user credentials to the same user id and password credentials of the EIS security domain. The user ID and password are stored in the security configuration document and is looked up using the configured alias as a key. Your mapping module may be programmed to perform more sophisticated mapping and store passwords in other persistent storage or from a remote service.

To develop your own principal and credential mapping `LoginModule`, refer to the JAAS documentation for general information. The JAAS documents are shipped with WebSphere Application Server are located in `$(install_root)/web/docs/jaas/JaasDocs.zip` file. Refer to the `login.html` in the `JaasDocs.zip` file for details of how to develop JAAS login module.

In particular, a mapping module needs to obtain the security identity of the caller. The `WSSubject.getCallerPrincipal()` static method returns an `com.ibm.websphere.security.auth.WSPincipal` object, which represents the security identity of an authenticated caller.

Developing custom user registries

WebSphere Application Server security supports the use of custom registries in addition to Local OS and Lightweight Directory Access Protocol (LDAP) registries for authentication and authorization purposes. A custom user registry is a customer implemented user registry which implements the `UserRegistry` Java interface as provided by WebSphere Application Server. A custom implemented user registry can support virtually any type or notion of an accounts repository from a relational database, flat file, and so on. The custom user registry provides considerable flexibility in adapting WebSphere Application Server security to various environments where some notion of a user registry, other than LDAP or LocalOS, already exist in the operational environment.

Implementing a custom user registry is a software development effort. Use the methods defined in the `UserRegistry` interface to make calls to the desired registry to obtain user and group information. The interface defines a very general set of methods, for encapsulating a wide variety of registries. You can configure a custom user registry as the active user registry when configuring WebSphere Application Server global security.

Make sure that your implementation of the custom registry does not depend on any WebSphere Application Server components such as data sources, enterprise beans, and so on. Do not have this dependency because security is initialized and enabled prior to most of the other WebSphere Application Server components during startup. If your previous implementation used these components, make a change that will eliminate the dependency. For example, if your previous

implementation used data sources to connect to a database, use Java database connectivity (JDBC) to connect to the database.

For backward compatibility, the WebSphere Application Server Version 4 custom registry is also supported. Refer to the Migrating custom user registries article 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. If not familiar with the custom user registry concept, refer to the article, Custom user registries. This section explains each of the methods in the interface in detail and the changes for these methods from the version 4 release.
2. Implement all the methods in the interface except for the `CreateCredential` method, which is implemented by WebSphere Application Server. A sample that implements this interface is provided for reference.
3. Build your implementation. You need the `%install_root%/lib/sas.jar` and `%install_root%/lib/wssec.jar` files in your class path. For example:

```
%install_root%\java\bin\javac -classpath  
%install_root%\lib\wssec.jar;%install_root%\lib\sas.jar  
yourImplementationFile.java.
```
4. Copy the class files generated in the previous step to the product class path. The preferred location is the `%install_root%/lib/ext` directory. This should be copied to all the product processes (cell, all NodeAgents) class path.
5. Follow the steps in Configuring custom user registries to configure your implementation using the administrative console.

This step is required to implement custom user registries in Version 5.

If you enabling security, make sure you complete the remaining steps. Once this is done, make sure you save and synchronize the configuration and restart all the servers. Try accessing some J2EE resources to verify that the custom registry implementation is successful.

Example: Custom user registries

A *custom user registry* is a customer-implemented user registry that implements the `UserRegistry` Java interface as provided by WebSphere Application Server. A custom-implemented user registry can support virtually any type or form of an accounts repository from a relational database, flat file, and so on. The custom user registry provides considerable flexibility in adapting WebSphere Application Server security to various environments where some form of a user registry, other than Lightweight Directory Access Protocol (LDAP) or Local OS, already exist in the operational environment.

Implementing a custom user registry is a software development effort. You must use the methods defined in the `UserRegistry` interface to make calls to the desired registry for obtaining user and group information. The interface defines a very general set of methods, so it can encapsulate a wide variety of registries. You can configure a custom user registry as the active user registry when configuring the product global security.

If you are using the WebSphere Application Server Version 4.0 custom registry you can plug in your registry without any changes. However, using the new interface to implement your custom registry is recommended.

To view a sample custom registry, refer to the following files:

-
- FileRegistrySample.java file for WebSphere Application Server Version 5.0.2
- Users.props file
- Groups.props file

UserRegistry interface methods

Implementing this interface enables WebSphere Application Server security to use custom registries. This capability should extend the `java.rmi` file. With a remote registry, you can complete this process remotely.

Implementation of this interface must provide implementations for:

- `initialize(java.util.Properties)`
- `checkPassword(String,String)`
- `mapCertificate(X509Certificate[])`
- `getRealm`
- `getUsers(String,int)`
- `getUserDisplayName(String)`
- `getUniqueUserId(String)`
- `getUserSecurityName(String)`
- `isValidUser(String)`
- `getGroups(String,int)`
- `getGroupDisplayName(String)`
- `getUniqueGroupId(String)`
- `getUniqueGroupIds(String)`
- `getGroupSecurityName(String)`
- `isValidGroup(String)`
- `getGroupsForUser(String)`
- `getUsersForGroup(String,int)`
- `createCredential(String)`

```
public void initialize(java.util.Properties props)
    throws CustomRegistryException,
           RemoteException;
```

This method is called to initialize the UserRegistry method. All the properties defined in the Custom User Registry panel propagate to this method.

For the sample, the initialize method retrieves the names of the registry files containing the user and group information.

This method is called during server bring up to initialize the registry. This method is also called when validation is performed by the administrative console, when security is on. This method remains the same as in version 4.0.

```
public String checkPassword(String userSecurityName, String password)
    throws PasswordCheckFailedException,
           CustomRegistryException,
           RemoteException;
```

The `checkPassword` method is called to authenticate users when they log in using a name (or user ID) and a password. This method returns a string which, in most cases, is the user being authenticated. Then, a credential is created for the user for authorization purposes. This user name is also returned for the enterprise bean call, `getCallerPrincipal()`, and the servlet calls, `getUserPrincipal()` and `getRemoteUser()`. See the `getUserDisplayName` method for more information if you

have display names in your registry. In some situations, if you return a user other than the one who is logged in, verify that the user is valid in the registry.

For the sample, the `mapCertificate` method gets the distinguished name (DN) from the certificate chain and makes sure it is a valid user in the registry before returning the user. For the sample, the `checkPassword` method checks the name and password combination in the registry and (if they match) returns the user being authenticated.

This method is called for various scenarios. It is called by the administrative console to validate the user information once the registry is initialized. It is also called when you access protected resources in the product for authenticating the user and before proceeding with the authorization. This method is the same as in WebSphere Application Server Version 4.

```
public String mapCertificate(X509Certificate[] cert)
    throws CertificateMapNotSupportedException,
           CertificateMapFailedException,
           CustomRegistryException,
           RemoteException;
```

The `mapCertificate` method is called to obtain a user name from an X.509 certificate chain supplied by the browser. The complete certificate chain is passed to this method and the implementation can validate the chain if needed and get the user information. A credential is created for this user for authorization purposes. If browser certificates are not supported in your configuration, you can throw the exception, `CertificateMapNotSupportedException`. The consequence of not supporting certificates is authentication failure if the challenge type is certificates, even if valid certificates are in the browser.

This method is called when certificates are provided for authentication. For Web applications, when the authentication constraints are set to CLIENT-CERT in the `web.xml` file of the application, this method is called to map a certificate to a valid user in the registry. For Java clients, this method is called to map the client certificates in the transport layer, when using the transport layer authentication. Also, when the Identity Assertion Token (when using the CSIV2 authentication protocol) is set to contain certificates, this method is called to map the certificates to a valid user.

In WebSphere Application Server Version 4.0, the input parameter was the `X509Certificate` certificate. In WebSphere Application Server Version 5, this parameter changes to accept an array of `X509Certificate` certificates (such as a certificate chain). In version 4, this parameter was called only for Web applications, but in version 5.0 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 default name of `customRealm` is used.

For the sample, the `getRealm` method returns the string, `customRealm`. One of the calls to this method is when the registry information is validated. This method is the same as in version 4.

```

public Result getUsers(String pattern, int limit)
    throws CustomRegistryException,
           RemoteException;

```

The `getUsers` method returns the list of users from the registry. The names of users depend on the pattern parameter. The number of users are limited by the limit parameter. In a registry that has many users, getting all the users is not practical. So the limit parameter is introduced to limit the number of users retrieved from the registry. A limit of 0 indicates to return all the users that match the pattern and might cause problems for large registries. Use this limit with care.

The custom registry implementations are expected to support at least the wildcard search (*). For example, a pattern of (*) returns all the users and a pattern of (b*) returns the users starting with *b*.

The return parameter is an object of type `com.ibm.websphere.security.Result`. This object contains two attributes, a `java.util.List` and a `java.lang.boolean`. The list contains the users returned and the Boolean flag indicates if there are more users available in the registry for the search pattern. This Boolean flag is used to indicate to the client whether more users are available in the registry.

In the sample, the `getUsers` retrieves the required number of users from the registry and sets them as a list in the result object. To find out if there are more users than requested, the sample gets one more user than requested and if it finds the additional user, it sets the Boolean flag to true. For pattern matching, the `match` method in the `RegExpSample` class is used, which supports wildcard characters such as the asterisk (*) and question mark (?).

This method is called by the administrative console to add users to roles in the various map users to roles panels. The administrative console uses the Boolean set in the result object to indicate that more entries matching the pattern are available in the registry.

In WebSphere Application Server Version 4, this method specifies to take only the pattern parameter. The return is a list. In WebSphere Application Server Version 5, this method is changed to take one additional parameter, the limit. Ideally, your implementation should change to take the limit value and limit the users returned. The return is changed to return a result object, which consists of the list (as in version 4) and a flag indicating if more entries exist. So, when the list returns, use the `Result.setList(List)` to set the List in the result object. If there are more entries than requested in the limit parameter, set the Boolean attribute to true in the result object, using `Result.setHasMore()` method. The default for the Boolean attribute in the result object is false.

```

public String getUserDisplayName(String userSecurityName)
    throws EntryNotFoundException,
           CustomRegistryException,
           RemoteException;

```

The `getUserDisplayName` method returns a display name for a user, if one exists. The display name is an optional string that describes the user that you can set in some registries. This is a descriptive name for the user and does not have to be unique in the registry. For example in Windows systems, you can display the full name of the user. If you do not need display names in your registry, return null or an empty string for this method.

Note: In WebSphere Application Server Version 4, if display names existed for any user these names were useful for the EJB method call `getCallerPrincipal()` and the servlet calls `getUserPrincipal()` and `getRemoteUser()`. If the display names were not the same as the security name for any user, the display names are returned for the previously mentioned enterprise beans and servlet methods. Returning display names for these methods might become problematic in some situations because the display names might not be unique in the registry. Avoid this problem by changing the default behavior to return the user's security name instead of the user's display name in this version of the product. However, if you want to have the same behavior as in version 4, set the property `WAS_UseDisplayName` to `true` in the **Custom Registry Properties** panel in the administrative console. For more information on how to set properties for the custom registry, see the section on *Setting Properties for Custom Registries*.

In the sample, this method returns the display name of the user whose name matches the user name provided. If the display name does not exist this returns an empty string.

This method can be called by the product to present the display names in the administrative console, or using the command line using the `wsadmin` tool. Use this method only for displaying. This method is the same as in Version 4.0.

```
public String getUniqueId(String userSecurityName)
    throws EntryNotFoundException,
           CustomRegistryException,
           RemoteException;
```

This method returns the unique ID of the user given the security name.

In the sample, this method returns the `uniqueId` of the user whose name matches the supplied name. This method is called when forming a credential for a user and also when creating the authorization table for the application.

```
public String getUserSecurityName(String uniqueUserId)
    throws EntryNotFoundException,
           CustomRegistryException,
           RemoteException;
```

This method returns the security name of a user given the unique ID. In the sample, this method returns the security name of the user whose unique ID matches the supplied ID.

This method is called to make sure a valid user exists for a given `uniqueUserId`. This method is called to get the security name of the user when the `uniqueUserId` is obtained from a token. This method is the same as in Version 4.

```
public boolean isValidUser(String userSecurityName)
    throws CustomRegistryException,
           RemoteException;
```

This method indicates whether the given user is a valid user in the registry.

In the Sample, this method returns `true` if the user is found in the registry, otherwise this method returns `false`. This method is primarily called in situations where knowing if the user exists in the directory prevents problems later. For

example, in the `mapCertificate` call, once the name is obtained from the certificate if the user is found to be an invalid user in the registry, you can avoid trying to create the credential for the user. This method is the same as in WebSphere Application Server Version 4.0.

```
public Result getGroups(String pattern, int limit)
    throws CustomRegistryException,
           RemoteException;
```

The `getGroups` method returns the list of groups from the registry. The names of groups depend on the pattern parameter. The number of groups is limited by the limit parameter. In a registry that has many groups, getting all the groups is not practical. So, the limit parameter is introduced to limit the number of groups retrieved from the registry. A limit of 0 implies to return all the groups that match the pattern and can cause problems for large registries. Use this limit with care. The custom registry implementations are expected to support at least the wildcard search (*). For example, a pattern of (*) returns all the users and a pattern of (b*) returns the users starting with *b*.

The return parameter is an object of type `com.ibm.websphere.security.Result`. This object contains two attributes, a `java.util.List` and a `java.lang.Boolean`. The list contains the groups returned and the Boolean flag indicates whether there are more groups available in the registry for the pattern searched. This Boolean flag is used to indicate to the client if more groups are available in the registry.

In the sample, the `getUsers` retrieves the required number of groups from the registry and sets them as a list in the result object. To find out if there are more groups than requested, the sample gets one more user than requested and if it finds the additional user, it sets the Boolean flag to true. For pattern matching, the `match` method in the `RegExpSample` class is used. It supports wildcards like *, ?, ?.

This method is called by the administrative console to add groups to roles in the various map groups to roles panels. The administrative console will use the boolean set in the Result object to indicate that more entries matching the pattern are available in the registry.

In WebSphere Application Server Version 4, this method is used to take the pattern parameter only and returns a list. In WebSphere Application Server Version 5, this method is changed to take one additional parameter, the limit. Change to take the limit value and limit the users returned. The return is changed to return a result object, which consists of the list (as in version 4) and a flag indicating whether more entries exist. Use the `Result.setList(List)` to set the list in the result object. If there are more entries than requested in the limit parameter, set the Boolean attribute to true in the result object using `Result.setHasMore()`. The default for the Boolean attribute in the result object is false.

```
public String getGroupDisplayName(String groupSecurityName)
    throws EntryNotFoundException,
           CustomRegistryException,
           RemoteException;
```

The `getGroupDisplayName` method returns a display name for a group if one exists. The display name is an optional string describing the group that you can set in some registries. This name is a descriptive name for the group and does not have to be unique in the registry. If you do not need to have display names for groups in your registry, return null or an empty string for this method.

In the sample, this method returns the display name of the group whose name matches the group name provided. If the display name does not exist, this method returns an empty string.

The product can call this method to present the display names in the administrative console or through command line using the wsadmin tool. This method is only used for displaying and is the same as in Version 4.0.

```
public String getUniqueGroupId(String groupSecurityName)
    throws EntryNotFoundException,
           CustomRegistryException,
           RemoteException;
```

This method returns the unique ID of the group given the security name.

In the sample, this method returns the unique ID of the group whose name matches the supplied name. This method is called when creating the authorization table for the application and is the same as in Version 4.0.

```
public List getUniqueGroupIds(String uniqueUserId)
    throws EntryNotFoundException,
           CustomRegistryException,
           RemoteException;
```

This method returns the unique IDs of all the groups to which a user belongs.

In the sample, this method returns the unique ID of all the groups that contain this uniqueUserID. This method is called when creating the credential for the user. As part of creating the credential, all the groupUniqueIds in which the user belongs are collected and put in the credential for authorization purposes when groups are given access to a resource. This method is the same as in version 4.

```
public String getGroupSecurityName(String uniqueGroupId)
    throws EntryNotFoundException,
           CustomRegistryException,
           RemoteException;
```

This method returns the security name of a group given its unique ID.

In the sample, this method returns the security name of the group whose unique ID matches the supplied ID. This method verifies that a valid group exists for a given uniqueGroupId. This method is the same as in version 4.

```
public boolean isValidGroup(String groupSecurityName)
    throws CustomRegistryException,
           RemoteException;
```

This method indicates if the given group is a valid group in the registry.

In the sample, this method returns true if the group is found in the registry, otherwise the method returns false. This method can be used in situations where knowing whether the group exists in the directory might prevent problems later. This method is the same as in version 4.

```
public List getGroupsForUser(String userSecurityName)
    throws EntryNotFoundException,
           CustomRegistryException,
           RemoteException;
```

This method returns all the groups to which a user belongs whose name matches the supplied name. This method is similar to the `getUniqueGroupIds` method with the exception that the security names are used instead of the unique IDs.

In the sample, this method returns all the group security names that contain the `userSecurityName`.

This method is called by the administrative console or the scripting tool to verify that the users entered for the RunAs roles are already part of that role in the users and groups to role mapping. This check is required to ensure that a user cannot be added to a RunAs role unless that user is assigned to the role in the users and groups to role mapping either directly or indirectly (through a group that contains this user). Since a group in which the user belongs can be part of the role in the users and groups to role mapping, this method is called to check if any of the groups that this user belongs to mapped to that role. This method is the same as in Version 4.0.

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

This method retrieves users from the specified group. The number of users returned is limited by the `limit` parameter. A limit of `0` indicates to return all the users in that group. This method is not directly called by the WebSphere Application Server security component. However, this can be called by other components. For example, this method issued by the process choreographer when staff assignments are modeled using groups. In rare situations, if you are working with a registry where getting all the users from any of your groups is not practical (for example, if there are a large number of users), you can throw the `NotImplementedException` exception for the particular groups. In this case, verify that if the process choreographer is installed (or if it is installed later) the staff assignments are not modeled using these particular groups. If there is no concern about returning the users from groups in the registry, it is recommended that you do not throw the `NotImplemented` exception when implementing this method.

The return parameter is an object of type `com.ibm.websphere.security.Result`. This object contains two attributes, `java.util.List` and `java.lang.boolean`. The list contains the users returned and the Boolean flag, which indicates whether there are more users available in the registry for the search pattern. This Boolean flag indicates to the client whether users are available in the registry.

In the example, this method gets one user more than the requested number of users for a group if the limit parameter is not set to `0`. If it succeeds in getting one more user, it sets the Boolean flag to `true`.

In WebSphere Application Server Version 4, this method was mandatory for the product. For WebSphere Application Server Version 5, this method can throw the `NotImplementedException` exception in situations where it is not practical to get the requested set of users. However, this exception should be thrown in rare

situations, as other components can be affected. In version 4, this method accepted only the pattern parameter and the returned a list. In version 5, this method accepts one additional parameter, the limit. Change your implementation to take the limit value and limit the users returned. The return changes to return a result object, which consists of the list (as in version 4) and a flag indicating whether more entries exist. As in version 4, when the list is returned, use the `Result.setList(List)` method to set the list in the Result object. If there are more entries than requested in the limit parameter, set the Boolean attribute to true in the result object using `Result.setHasMore()`. The default for the Boolean attribute in the Result object is false.

Attention: The first two lines of the following code sample is one continuous line.

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

In this release of the WebSphere Application Server, this method is not called. You can return *null*. In the example, a *null* is returned.

Developing a custom interceptor for trust associations

If you are using a third party reverse proxy server other than Tivoli WebSEAL, you must provide an implementation class for the product interceptor interface for your proxy server. This article describes the interface you must implement.

1. Define the interceptor class method. WebSphere Application Server provides the interceptor Java interface, `com.ibm.websphere.security.TrustAssociationInterceptor`, which defines the following methods:

- **public boolean isTargetInterceptor(HttpServletRequest req)** throws `WebTrustAssociationException`;

The `isTargetInterceptor` method determines whether the request originated with the proxy server associated with the interceptor. The implementation code must examine the incoming request object and determine if the proxy server forwarding the request is a valid proxy server for this interceptor. The result of this method determines whether the interceptor processes the request or not.

- **public void validateEstablishedTrust (HttpServletRequest req)** throws `WebTrustAssociationException`;

The `validateEstablishedTrust` method determines if the proxy server from which the request originated is trusted or not. This method is called after the `isTargetInterceptor` method. The implementation code must authenticate the proxy server. The authentication mechanism is proxy-server specific. For example, in the product implementation for the WebSEAL server, this method retrieves the basic authentication information from the HTTP header and validates the information against the user registry used by WebSphere Application Server. If the credentials are invalid, the code throws the `WebTrustAssociationException`, indicating that the proxy server is not trusted and the request is to be denied.

- **public String getAuthenticatedUsername(HttpServletRequest req)** throws `WebTrustAssociationException`;

The `getAuthenticatedUsername` method is called after trust is established between the proxy server and WebSphere Application Server. The product has accepted the proxy server authentication of the request and must now authorize the request. To authorize the request, the name of the original requestor must be subjected to an authorization policy to determine if the requestor has the necessary privilege. The implementation code for this method must extract the user name from the HTTP request header and determine if that user is entitled to the requested resource. For example, in the product implementation for the WebSEAL server, the method looks for an `iv-user` attribute in the HTTP request header and extracts the user ID associated with it for authorization.

2. Configuring the interceptor. To make an interceptor configurable, the interceptor must extend `com.ibm.websphere.security.WebSphereBaseTrustAssociationInterceptor`. Implement the following methods:

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

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

Applicability of the following list

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:

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

public void cleanup ();

This method is called when the application server is stopped. It is used to prepare the interceptor for termination.

public void setVersion (String s);

This methods is optional. The method is used to set the version and is for informational purpose only. The default value is `Unspecified`.

You must configure the following methods implemented by the custom interceptor implementation. **This listing only shows the methods and does not include any implementation.**

```

*****
import java.util.*;
import javax.servlet.http.HttpServletRequest;
import com.ibm.websphere.security.*;

public class myTAImpl extends WebSphereBaseTrustAssociationInterceptor
    implements TrustAssociationInterceptor
{

    public myTAImpl ()
    {
    }

    public boolean isTargetInterceptor (HttpServletRequest req)
        throws WebTrustAssociationException
    {

        //return true if this is the target interceptor, else return false.
    }

    public void validateEstablishedTrust (HttpServletRequest req)
        throws WebTrustAssociationFailedException
    {
        //validate if the request is from the trusted proxy server.
        //throw exception if the request is not from the trusted server.
    }

    public String getAuthenticatedUsername (HttpServletRequest req)
        throws WebTrustAssociationUserException
    {
        //Get the user name from the request and if the user is
        //entitled to the requested resource
        //return the user. Otherwise, throw the exception
    }

    public int init (Properties props)
    {
        //initialize the implementation. If successful return 0, else return -1.
    }

    public void cleanup ()
    {
        //Cleanup code.
    }

}
*****

```

Note: If the `init(Properties)` method is implemented as described previously in your custom interceptor, this note does not apply to your

implementation, and you can move on to the next step. Previous versions of `com.ibm.websphere.security.WebSphereBaseTrustAssociationInterceptor` include the public `int init(String propsfile)` method. This method is no longer required since the interceptor properties are not read from a file. The properties are now entered in the administrative console **Custom Properties** link of the interceptor using the administrative console or scripts. These properties then are made available to your implementation in the `init(Properties)` method. However, for backward compatibility, the `init(String)` method still is supported. The `init(String)` method is called by the default implementation of `init(Properties)` as shown in the following example.

```
// Default implementation of init(Properties props) method. A Custom
// implementation should override this.
public int init (java.util.Properties props)
{
    String type =
        props.getProperty("com.ibm.websphere.security.trustassociation.types");
    String classfile=
        props.getProperty("com.ibm.websphere.security.trustassociation."
            +type+".config");
    if (classfile != null && classfile.length() > 0 ) {
        return init(classfile);
    } else {
        return -1;
    }
}
```

Change your implementation to implement the `init(Properties)` method instead of relying on `init(String propsfile)` method. As shown in the previous example, this default implementation reads the properties to load the property file. The `com.ibm.websphere.security.trustassociation.types` property gets the file containing the properties by concatenating `.config` to its value.

Note: The `init(String)` method still works if you want to use it instead of implementing the `init(Properties)` method. The only requirement is that the file name containing the custom trust association properties should now be entered using the **Custom Properties** link of the interceptor in the administrative console or by using scripts. You can enter the property using *either* of the following methods. The first method is used for backward compatibility with previous versions of WebSphere Application Server. You can use the second method with WebSphere Application Server, Version 5.0.2 and later.

Method 1:

The same property names used in the previous release are used to obtain the file name. The file name is obtained by concatenating the `.config` to the `com.ibm.websphere.security.trustassociation.types` property value.

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

Method 2:

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

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

Type the previous code as one continuous line.

The location of the properties file is fully qualified (for example, `C:/WebSphere/AppServer/properties/myTAI.properties`). Since the location can be different in a Network Deployment environment, use variables such as `${USER_INSTALL_ROOT}` to refer to the WebSphere Application Server installation directory. For example, if the file name is called `myTAI.properties`, and it is located in the `C:/WebSphere/AppServer/properties` directory, then set the following properties:

- `com.ibm.websphere.security.trustassociation.types = myTAItype`
 - `com.ibm.websphere.security.trustassociation.myTAItype.config =
c:/WebSphere/AppServer/properties/myTAI.properties`
3. Compile the implementation once you have implemented it. For example,

```
install_root/java/bin/javac -classpath  
install_root/lib/wssec.jar;<install_root>/lib/j2ee.jar myTAIImpl.java
```

 - a. Copy the class file to a location in the class path (preferably the `install_root/lib/ext` directory).
 - b. Restart all the servers.
 4. Delete the default WebSEAL interceptor in the administrative console and click **New** to add your custom interceptor. Verify that the class name is dot separated and appears in the class path.
 5. Click the **Custom Properties** link to add additional properties that are required to initialize the custom interceptor. These properties are passed to the `init(Properties)` method of your implementation when it extends the `com.ibm.websphere.security.WebSphereBaseTrustAssociationInterceptor` as described in the previous step.
 6. Save and synchronize (if applicable) the configuration.
 7. Restart the servers for the custom interceptor to take effect.

Refer to the Security: Resources for Learning article, which references the WebSphere Application Server Version 5 Redbook to view an example of a custom interceptor.

Assembling secured applications

The Assembly Toolkit is a graphical user interface for assembling enterprise (J2EE) applications. You can use these tools to assemble an application and secure EJB and Web modules in that application. An EJB module consists of one or more beans. You can enforce security at the EJB method level. A Web module consists of one or more Web resources (an HTML page, a JSP file or a servlet). You can also enforce security for each Web resource. You can use the assembly tools to secure an EJB module (Java archive (JAR) file) or a Web module (Web archive (WAR) file) or an application (enterprise archive (EAR) file). You can create an

application, an EJB module, or a Web Module and secure them using the Assembly Toolkit or development tools like the IBM WebSphere Studio Application Developer.

1. Secure EJB applications using the Assembly Toolkit.
2. Secure Web applications using the Assembly Toolkit.
3. Add users and groups to roles while assembling secured application using the Assembly Toolkit.
4. Map users to RunAs roles using the Assembly Toolkit.
5. Add the `was.policy` file to applications for Java 2 security.
6. Assemble the application components that you just secured using the Assembly Toolkit..

After securing an application, the resulting `.ear` file contains security information in its deployment descriptor. The EJB module security information is stored in the `ejb-jar.xml` file and the Web module security information is stored in the `web.xml` file. The `application.xml` file of the application EAR file contains all the roles used in the application. The user and group to roles mapping is stored in the `ibm-application-bnd.xmi` file of the application EAR file. The `was.policy` file of the application EAR contains the permissions granted for the application to access system resources.

This task is required to secure EJB modules and Web modules in an application. This task is also required for applications to run properly when Java 2 security is enabled. If the `was.policy` file is not created and it does not contain required permissions, the application might not be able to access system resources.

After securing an application, you can install an application using the administrative console. When you install a secured application, refer to the Deploying secured applications article to complete this task.

Enterprise bean component security

An EJB module consists of one or more beans. You can use development tools such as WebSphere Studio 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 being accessed by anyone mark them **excluded**. You can give everyone access to a set of enterprise beans method by clearing those methods. You can run enterprise beans as a different identity (runAs identity) before invoking other enterprise beans.

Securing enterprise bean applications using the Assembly Toolkit

You can protect enterprise bean methods by assigning security roles to them. Before you assign security roles, you need to know which EJB methods need protecting and how.

1. Open the EJB application file. This file can be an EJB `.jar` file or an application `.ear` file that contains one or more EJB modules. In the Assembly Toolkit, open a deployment descriptor editor on the EJB application file. In a J2EE Hierarchy view, right-click the file and click **Open With > Deployment Descriptor Editor**. If you selected an EJB `.jar` file, an EJB deployment descriptor editor opens. If

you selected an application .ear file, an application deployment descriptor editor opens. To see online information about the editor, press F1 and click the editor name.

2. Create security roles. You can create security roles at the application level or at the EJB module level. If you create a security role at the EJB module level, the role displays in the application level. If a security role is created at the application level, the role does not appear in all the EJB modules. You can copy and paste one or more EJB module security roles that you create at application level:
 - Create a role at an EJB module level. In an EJB deployment descriptor editor, select the **Assembly Descriptor** tab. Under **Security Roles**, click **Add**. In the Add Security Role wizard, name and describe the security role; then click **Finish**.
 - Create a role at the application level. In an application deployment descriptor editor, select the **Security** tab. Under the list of security roles, click **Add**. In the Add Security Role wizard, name and describe the security role; then click **Finish**.
3. 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 2.0 EJB projects, there is an unchecked option that specifies that the selected methods from the selected beans do not require authorization to execute. To add a method permission to an enterprise bean:
 - a. On the **Assembly Descriptor** tab of an EJB deployment descriptor editor, under **Method Permissions**, click **Add**. The Add Method Permission wizard opens.
 - b. Select a security role from the list of roles found and click **Next**.
 - c. Select one or more enterprise beans from the list of beans found. You can click **Select All** or **Deselect All** to select or deselect all of the enterprise beans in the list. Click **Next**.
 - d. Select the methods that you want to bind to your security role. The Method Elements page lists all methods associated with the enterprise bean(s). You can click **Apply to All** or **Deselect All** to quickly select or clear multiple methods. It selects only the * method for each bean. Creating a method permission for the exact method signature overrides the default (*) method permission setting. The * method represents all methods within the bean. There are * for each interface as well. By not selecting all of the individual methods in the tree, you can set other permissions on the remaining methods.
 - e. Click **Finish**.

After the method permission is created, you can see the new method permission in the tree. Expand the tree to see the bean and methods defined in the method permission.
4. Exclude user access to methods. Users cannot access excluded methods. Any method in the enterprise beans that is not assigned to a role or is not excluded, is deselected during the application installation by the deployer.
 - a. On the **Assembly Descriptor** tab of an EJB deployment descriptor editor, under **Excludes List**, click **Add**. The Exclude List wizard opens.
 - b. Select one or more enterprise beans from the list of beans found and click **Next**.

- c. Select one or more of the method elements for the security identity and click **Finish**.
5. Map security-role-ref and role-name to role-link. When developing enterprise beans, you can create the security-role-ref element. The security-role-ref element contains only the role-name field. The role-name field determines if the caller is in a specified role(`isCallerInRole()`) and contains the name of the role that is referenced in the code. Since you create security roles during the assembly stage, the developer uses a *logical rolename* in the **role-name** field and provides enough information in the **description** field for the assembler to map the actual role (role-link). The security-role-ref element is located at the EJB level. Enterprise beans can have zero or more security-role-ref elements.
 - a. On the **References** tab of an EJB deployment descriptor editor, under the list of references, click **Add**. The Add Reference wizard opens.
 - b. Select **Security role reference** and click **Next**.
 - c. Name the security role reference, select a security role to link the reference to, describe the security role reference, and click **Finish**.
 - d. Map every role-name used during development to the role (role-link) using the previous steps.
 6. Specify the RunAs Identity for enterprise beans components. The RunAs Identity of the enterprise bean is used to invoke the next enterprise beans in the chain of EJB invocations. When the next enterprise beans are invoked, the RunAsIdentity passes to the next enterprise beans for performing an authorization check on the next enterprise bean. If the RunAs Identity is not specified, the client identity is propagated to the next enterprise bean. The RunAs Identity can represent each of the enterprise beans or can represent each method in the enterprise beans.
 - a. On the **Access** tab of an EJB deployment descriptor editor, under **Security Identity (Bean Level)**, click **Add**. The Add Security Identity wizard opens.
 - b. Select the run as mode, describe the security identity, and click **Next**. Select the **Use identity of caller** mode to instruct the security service to make no changes to the principal's credential settings. Select the **Use identity assigned to specific role (below)** mode to use a principal that has been assigned to the specified security role for running of the bean's methods. This association is part of the application binding in which the role is associated with a user ID and password of a user who is granted that role. If you selected **Use identity assigned to specific role (below)**, 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**.
 7. 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 article to complete the task of securing the EJB application.

Web component security

A Web module consists of servlets, JSP files, server-side utility classes, static Web content (HTML, images, sound files, Cascading Style Sheets (CSS)), and client-side classes (applets). You can use development tools such as IBM WebSphere Studio Application Developer to develop a Web module and enforce security at the method level of each Web resource.

You can identify a Web resource by its URI pattern. A Web resource method can be any HTTP method (GET, POST, DELETE, PUT, for example). You can group a set of URI patterns and a set of HTTP methods together and assign this grouping a set of roles. When a Web resource method is secured by associating a set of roles, grant a user at least one role in that set to access that method. You can exclude anyone from accessing a set of Web resources by assigning an empty set of roles. A servlet or a JSP file can run as different identities (RunAs identity) before invoking another enterprise bean component. All the secured Web resources require the user to log in by using a configured login mechanism. There are three types of Web login authentication mechanisms: basic authentication, form-based authentication and client certificate-based authentication.

For more detailed information on Web security see the product architectural overview article.

Securing Web applications using the Assembly Toolkit

There are three types of Web login authentication mechanisms that you can configure on a Web application: basic authentication, form-based authentication and client certificate-based authentication. Protect Web resources in a Web application by assigning security roles to those resources.

To secure Web applications, determine the Web resources that need protecting and determine how to protect them.

1. Open the Web application file. This file can be a Web archive (WAR) file or an application archive (EAR) file that contains one or more Web modules. In the Assembly Toolkit, open a deployment descriptor editor on the Web application file. In a J2EE Hierarchy view, right-click the file and click **Open With > Deployment Descriptor Editor**. If you selected Web archive (WAR) file, a Web deployment descriptor editor opens. If you selected an enterprise application (EAR) file, an application deployment descriptor editor opens. To see online information about the editor, press F1 and click the editor name.
2. Create security roles either at the application level or at Web module level. If a security role is created at the Web module level, the role also displays in the application level. If a security role is created at the application level, the role does not display in all the Web modules. You can copy and paste a security role at the application level to one or more Web module security roles.
 - Create a role at a Web-module level. In a Web deployment descriptor editor, select the **Security** tab. Under **Security Roles**, click **Add**. Double-click (**New Security Role**) and type the security role. Under **Details**, describe the security role.
 - 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**.
3. Create security constraints. Security constraints are a mapping of one or more Web resources to a set of roles.

- a. On the **Security** tab of a Web deployment descriptor editor, click **Security Constraints**. On the Security Constraints tab that opens, you can do the following:
 - Add or remove security constraints for specific security roles.
 - Add or remove Web resources and their HTTP methods.
 - Define which security roles are authorized to access the Web resources.
 - Specify *None*, *Integral*, or *Confidential* constraints on user data. *None* means that the application requires no transport guarantees. *Integral* means that data cannot be changes in transit between client and server. And *Confidential* means that data content cannot be observed while it is in transit. *Integral* and *Confidential* usually require the use of SSL.
 - b. Under **Security Constraints**, click **Add**.
 - c. Under **Details**, specify a display name for the security constraint.
 - d. Under Web Resource Collections, click **Add**. The Web Resource Collections wizard opens.
 - e. Type a name and description for the Web resource collection.
 - f. Select one or more HTTP methods. The HTTP method options are: GET, PUT, HEAD, TRACE, POST, DELETE, and OPTIONS.
 - g. Beside **URL Patterns**, click **Add**. Double-click on (**New URL pattern**) and type a URL pattern (for example: - /*, *.jsp, /hello). Consult the Servlet specification Version 2.3 for instructions on mapping URL patterns to servlets. Security run time uses the exact match first to map the incoming URL with URL patterns. If the exact match is not present, the security run time uses the longest match. The wild card (*.*.jsp) URL pattern matching is used last.
 - h. Repeat these steps to create multiple security constraints.
4. Map security-role-ref and role-name elements to the role-link element. During the development of a Web application, you can create the security-role-ref element. The security-role-ref element contains only the role-name field at this stage. The role-name field contains the name of the role that is referenced in the servlet or JSP code to determine if the caller is in a specified role (isUserInRole()). Since security roles are created during the assembly stage, the developer uses a logical role name in the **role-name** field and provides enough description in the **description** field for the assembler to map the role actual (role-link). The Security-role-ref element is at the servlet level. A servlet or JSP file can have zero or more security-role-ref elements.
- a. Go to the **References** tab of a Web deployment descriptor editor. On the **References** tab, you can add or remove the name of an enterprise bean reference to the deployment descriptor. There are 5 types of references you can define on this tab:
 - EJB
 - EJB Local (J2EE 1.3 only)
 - Resource
 - Resource Environment (J2EE 1.3 only)
 - JSP Tag Library

This article describes how to define an EJB reference for a Web application.
 - b. Under the list of EJB references, click **Add**. Double-click on (**New EJB Ref**) and type an EJB reference.
 - c. Under **Details**, click **Browse** beside **Link** and select a link for the EJB reference. Select a link type of ENTITY or SESSION. Select **Home** and **Remote** values, and describe the link.

- d. Map every role-name used during development to the role (role-link) using the previous steps. Every role name used during development maps to the actual role.
5. Specify the RunAs identity for servlets and JSP files. The RunAs identity of a servlet is used to invoke enterprise beans from within the servlet code. When enterprise beans are invoked, the RunAs identity is passed to the enterprise bean for performing an authorization check on the enterprise beans. If the RunAs identity is not specified, the client identity is propagated to the enterprise beans. The RunAs identity is assigned at the servlet level.
 - a. On the **Servlets** tab of a Web deployment descriptor editor, under **Servlets and JSPs**, click **Add**. The Add Servlet or JSP wizard opens.
 - b. Select whether to add a servlet or JavaServer page (JSP), define which servlet or JSP to add, and click **OK**.
 - c. Under **Run As**, select the security role and describe the role.
 - d. Specify a RunAs identity for each servlet and JSP file used by your Web application.
6. Configure the login mechanism for the Web module. This configured login mechanism applies to all the servlets, JavaServer page (JSP) files and HTML resources in the Web module.
 - a. On the **Pages** tab of a Web deployment descriptor editor, under **Login**, select the required authentication method. Available method values include: Unspecified, Basic, Digest, Form, and Client-Cert.
 - b. Specify a realm name.
 - c. If you select the Form authentication method, select a login page and an error page URLs (for example: /login.jsp and /error.jsp). The specified login and error pages are present in the .war file.
 - d. Install the client certificate on the browser or Web client and place the client certificate in the server trust keyring file, if ClientCert is selected.
7. Close the deployment descriptor editor and, when prompted, click **Yes** to save the changes.

After securing a Web application, the resulting WAR file contains security information in its deployment descriptor. The Web module security information is stored in the web.xml file. When you work in the Web deployment descriptor editor, you also can edit other deployment descriptors in the Web project, including information on bindings and IBM extensions in the ibm-web-bnd.xmi and ibm-web-ext.xmi files.

After using the Assembly Toolkit to secure a Web application, you can install the Web application using the administrative console. During the Web application installation, complete the steps in the Deploying secured applications article to finish securing the Web application.

Role-based authorization

Use authorization information to determine whether a caller has the necessary privileges to request a service.

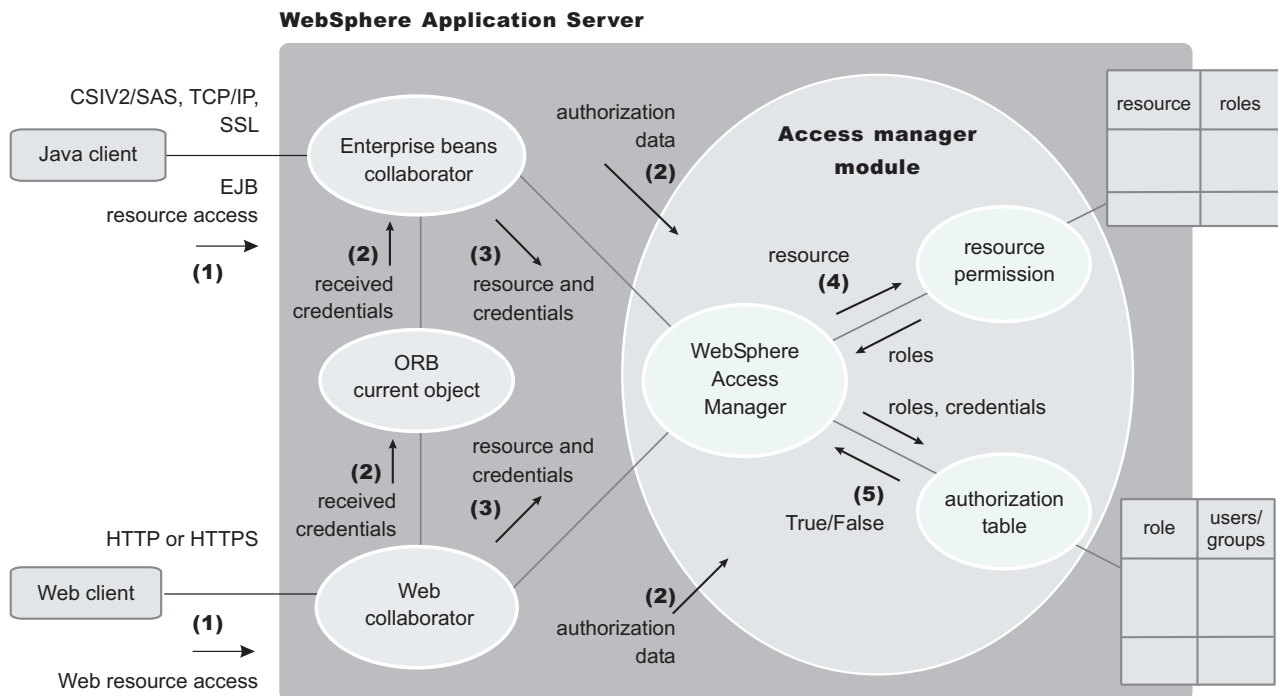
The following figure illustrates the process used during authorization. Web resource access from a Web client is handled by a Web collaborator. The EJB resource access from a Java client (can be enterprise beans or a servlet) is handled by an EJB Collaborator. The EJB collaborator and the Web collaborator extract the client credentials from the object request broker (ORB) current object. The client credentials are set during the authentication process as received credentials in the

ORB Current. The resource and the received credentials are presented to WSAccessManager to check whether access is permitted to the client for accessing the requested resource.

The access manager module contains two main modules:

- Resource permission module helps determine the required roles for a given resource. It uses a resource to roles mapping table that is built by the security run time during application startup. To build the resource-to-role mapping table, the security run time reads the deployment descriptor of the enterprise beans or the Web module (ejb-jar.xml or web.xml)
- Authorization table module consults a role to user or group table to determine whether a client is granted one of the required roles. The role to user or group mapping table, also known as the *authorization table*, is created by the security run time during application startup. To build the authorization table, the security run time reads the application binding file (ibm-application-bnd.xml file).

Authentication



Use authorization information to determine whether a caller has the necessary privilege to request a service. You can store authorization information many ways. For example, with each resource, you can store an *access-control list*, which contains a list of users and user privileges. Another way to store the information is to associate a list of resources and the corresponding privileges with each user. This list is called a *capability list*.

WebSphere Application Server uses the Java 2 Enterprise Edition (J2EE) authorization model. In this model, authorization information is organized as follows:

- During the assembly of an application, permission to invoke methods is granted to one or more roles. A role is a set of permissions; for example, in a banking application, roles can include teller, supervisor, clerk, and other industry-related positions. The teller role is associated with permissions to run methods related

to managing the money in an account, such as the withdraw and deposit methods. The teller role is not granted permission to close accounts; this permission is given to the supervisor role. The application assembler defines a list of method permissions for each role; this list is stored in the deployment descriptor for the application.

There are two *special subjects* that are not defined by J2EE: AllAuthenticatedUsers, Everyone. A special subject is a product-defined entity independent of the user registry. It is used to generically represent a class of users or groups in the registry.

- AllAuthenticatedUsers is a special subject that permits all authenticated users to access protected methods. As long as the user can authenticate successfully, the user is permitted access to the protected resource.
- Everyone is a special subject that permits unrestricted access to a protected resource. Users do not have to authenticate to get access; this special subject provides access to protected methods as if the resources were unprotected.

During the deployment of an application, real users or groups of users are assigned to the roles. 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. When a user is assigned to a role, the user gets all the method permissions that are granted to that role. Users can be assigned to more than one role; the permissions granted to the user are the union of the permissions granted to each role. Additionally, if the authentication mechanism supports the grouping of users, these groups can be assigned to roles. Assigning a group to a role has the same effect as assigning each individual user to the role.

A best practice during deployment is to assign groups, rather than individual users to roles for the following reasons:

- Improves performance during the authorization check. Typically far fewer groups exist than users.
- Provides greater flexibility, by using group membership to control resource access.
- Supports the addition and deletion of users from groups outside of the product environment. This action is preferred to adding and removing them to WebSphere Application Server roles. Stop and restart the enterprise application for these changes to take effect. This action can be very disruptive in a production environment.

At run time, WebSphere Application Server authorizes incoming requests based on the user's identification information and the mapping of the user to roles. If the user belongs to any role that has permission to execute a method, the request is authorized. If the user does not belong to any role that has permission, the request is denied.

The J2EE approach represents a declarative approach to authorization, but it also recognizes that you cannot deal with all situations declaratively. For these situations, methods are provided for determining user and role information programmatically. For Enterprise JavaBeans, the following two methods are supported by WebSphere Application Server:

- **getCallerPrincipal**: This method retrieves the user identification information.
- **isCallerInRole**: This method checks the user identification information against a specific role.

For servlets, the following methods are supported by WebSphere Application Server:

- getRemoteUser
- isUserInRole
- getUserPrincipal

These methods correspond in purpose to the enterprise bean methods.

For more information on the J2EE security authorization model see the following Web site: <http://java.sun.com>

Admin roles

The J2EE role-based authorization concept has been extended to protect the WebSphere Application Server administrative subsystem. A number of administrative roles have been defined to provide degrees of authority needed to perform certain WebSphere administrative functions from either the Web-based administrative console or the system management scripting interface. The authorization policy is only enforced when global security is enabled. The following table describes the admin roles:

Admin roles

| Role | Description |
|---------------|---|
| monitor | Least privileged that basically allows a user to view the WebSphere Application Server configuration and current state. |
| configurator | Monitor privilege plus the ability to change the WebSphere Application Server configuration. |
| operator | Monitor privilege plus the ability to change run-time state, such as starting or stopping services for example. |
| administrator | Operator plus configuration privilege. |

The identity specified when enabling global security is automatically mapped to the administrator role. Users, groups, can be added or removed from the admin roles from the WebSphere Application Server administrative console at anytime. However, a server restart is required for the changes to take effect. A best practice is to map a group or groups, rather than specific users, to admin roles because it is more flexible and easier to administer in the long run. By mapping a group to an admin role, adding or removing users to or from the group occurs outside of WebSphere Application Server and does not require a server restart for the change to take effect.

In addition to mapping user or groups, a special-subject can also be mapped to the admin roles. A special-subject is a generalization of a particular class of users. The AllAuthenticated special subject means that the access check of the admin role ensures that the user making the request has at least been authenticated. The Everyone special subject means that anyone, authenticated or not, can perform the action, as if security was not enabled.

Naming roles

The J2EE role-based authorization concept has been extended to protect the WebSphere CosNaming service.

CosNaming security offers increased granularity of security control over CosNaming functions. CosNaming functions are available on CosNaming servers such as the WebSphere Application Server. They affect the content of the WebSphere Name Space. There are generally two ways in which client programs will result in CosNaming calls. The first is through the JNDI interfaces. The second is CORBA clients invoking CosNaming methods directly.

Four security roles are introduced: **CosNamingRead**, **CosNamingWrite**, **CosNamingCreate**, and **CosNamingDelete**. The name of the four roles are the same with WebSphere Advanced Edition Version 4.0.2. However, the roles now have authority level from low to high as follows:

- **CosNamingRead**. Users who have been assigned the CosNamingRead role will be allowed to do queries of the WebSphere Name Space, such as through the JNDI "lookup" method. The special-subject Everyone is the default policy for this role.
- **CosNamingWrite**. Users who have been assigned the CosNamingWrite role will be allowed to do write operations such as JNDI "bind", "rebind", or "unbind", plus CosNamingRead operations. The special-subject AllAuthenticated is the default policy for this role.
- **CosNamingCreate**. Users who have been assigned the CosNamingCreate role will be allowed to create new objects in the Name Space through such operations as JNDI "createSubcontext", plus CosNamingWrite operations. The special-subject AllAuthenticated is the default policy for this role.
- **CosNamingDelete**. And finally users who have been assigned CosNamingDelete role will be able to destroy objects in the Name Space, for example using the JNDI "destroySubcontext" method, as well as CosNamingCreate operations. The special-subject AllAuthenticated is the default policy for this role.

Users, groups, or the special subjects AllAuthenticated and Everyone can be added or removed to or from the naming roles from the WebSphere Application Server administrative console at anytime. However, you must restart the server for the changes to take effect. A best practice is to map groups or one of the special-subjects, rather than specific users, to Naming roles because it is more flexible and easier to administer in the long run. By mapping a group to an naming role, adding or removing users to or from the group occurs outside of WebSphere Application Server and does not require a server restart for the change to take effect.

If a user is assigned a particular naming role and that user is a member of a group that has been assigned a different naming role, the user will be granted the most permissive access between the role he was assigned and the role his group was assigned. For example, assume that user MyUser has been assigned the CosNamingRead role. Also, assume that group MyGroup has been assigned the CosNamingCreate role. If MyUser is a member of MyGroup, MyUser will be assigned the CosNamingCreate role because he is a member of MyGroup. If MyUser were not a member of MyGroup, he would be assigned the CosNamingRead role.

The CosNaming authorization policy is only enforced when global security is enabled. When global security is enabled, attempts to do CosNaming operations without the proper role assignment will result in a `org.omg.CORBA.NO_PERMISSION` exception from the CosNaming Server.

In WebSphere Application Server Version 4.0.2, each CosNaming function is assigned to only one role. Therefore, users who have been assigned CosNamingCreate role will not be able to query the Name Space unless they have

also been assigned `CosNamingRead`. In most cases a creator would need to be assigned three roles: `CosNamingRead`, `CosNamingWrite`, and `CosNamingCreate`. This has been changed in the release. The `CosNamingRead` and `CosNamingWrite` roles assignment for the creator example in above have been included in `CosNamingCreate` role. In most of the cases, WebSphere Application Server administrators do not have to change the roles assignment for every user or group when they move to this release from previous one.

Although the ability exist to greatly restrict access to the Name space by changing the default policy, doing so may result in unexpected `org.omg.CORBA.NO_PERMISSION` exceptions at run time. Typically, J2EE applications access the Name space and the identity they use is that of the user that authenticated to WebSphere Application Server when they access the J2EE application. Unless the J2EE application provider clearly communicates the expected Naming roles, care should be taken when changing the default naming authorization policy.

Adding users and groups to roles using the Assembly Toolkit

Before you perform this task, you should have already completed the steps in the Securing Web applications and Securing EJB applications articles where you created new roles and assigned those roles to EJB and Web resources. Complete these steps during application installation. This is because the environment (user registry) under which the application is running is not known until deployment.

If you already know the environment in which the application is running and the user registry that is used, then you can use the Assembly Toolkit to assign users and groups to roles. Using the administrative console to assign users and groups to roles is recommended.

1. In the J2EE Hierarchy view of the Assembly Toolkit, right-click an enterprise application project (EAR file) and click **Open With > Deployment Descriptor Editor**. An application deployment descriptor editor opens on the EAR file. To access information about the editor, press F1 and click **Application deployment descriptor editor**.
2. Click the **Security** tab and, under the main pane, click **Add**.
3. In the Add Security Role wizard, name and describe the security role. Then click **Finish**.
4. Under **WebSphere Bindings**, select the user or group extension properties for the security role. Available values include: Everyone, All authenticated users, and Users/Groups.
5. If you selected Users/Groups, click **Add** beside the **Users** or **Groups** panes. In the wizard that opens, specify a user or group name and click **Finish**. Repeat this step until you have added all users and groups to which the security role applies.
6. Close the application deployment descriptor editor and, when prompted, click **Yes** to save the changes.

The `ibm-application-bnd.xmi` file in the application contains the users and groups to roles mapping table (*authorization table*).

After securing an application, install the application using the administrative console.

Mapping users to RunAs roles using the Assembly Toolkit

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. You must define RunAs roles when a servlet or an enterprise bean in an application is configured with RunAs settings. 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.
 - Assign users and groups to roles. 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 the Assembly Toolkit to assign users to RunAs roles.
1. In the J2EE Hierarchy view of the Assembly Toolkit, right-click an enterprise application project (EAR file) and click **Open With > Deployment Descriptor Editor**. An application deployment descriptor editor opens on the EAR file. To access information about the editor, press F1 and click **Application deployment descriptor editor**.
 2. On the **Security** tab, under **Security Role Run As Bindings**, click **Add**.
 3. Click **Add** under **RunAs Bindings**.
 4. In the Security Role wizard, select one or more roles and click **Finish**.
 5. Repeat steps 3 through 5 for all the RunAs roles in the application.
 6. Close the application deployment descriptor editor and, when prompted, click **Yes** to save the changes.

The `ibm-application-bnd.xmi` file in the application contains the user to RunAs role mapping table.

After securing an application, you can install the application using the administrative console. You can change the RunAs role mappings of an installed application.

Deploying secured applications

Before you perform this task, verify that you have already designed, developed and assembled an application with all the relevant security configurations. For more information on these tasks refer to the *Developing secured applications* and *Assembling secured applications* articles. In this context, deploying and installing an application are considered the same task.

Deploying applications that have security constraints (secured applications) is not much different than deploying applications any security constraints. The only difference is that you might need to assign users and groups to roles for a secured application, which requires that you have the correct active registry. To deploy a newly secured application click **Applications > Install New Application** in the navigation panel on the left and follow the prompts. If you are installing a secured application, roles would have been defined in the application. If delegation was required in the application, RunAs roles also are defined.

One of the steps required to deploy secured applications is to assign users and groups to roles defined in the application. This task is completed as part of the step titled *Map security roles to users and groups*. This assignment might have already been done through the Assembly Toolkit. In that case you can confirm the

mapping by going through this step. You can add new users and groups and modify existing information during this step.

If the applications support delegation, then a RunAs role is already defined in the application. If the delegation policy is set to **Specified Identity** (during assembly) the intermediary invokes a method using an identity setup during deployment. Use the RunAs role to specify the identity under which the downstream invocations are made. For example, if the RunAs role is assigned user "bob" and the client "alice" is invoking a servlet, with delegation set, which in turn calls the enterprise beans, then the method on the enterprise beans is invoked with "bob" as the identity. As part of the deployment process one of the steps is to assign or modify users to the RunAs roles. This step is titled "Map RunAs roles to users". Use this step to assign new users or modify existing users to RunAs roles when the delegation policy is set to Specified Identity.

These steps are common for both installing an application and modifying an existing application. If the application contains roles, you see the "Map security roles to users and groups" link during application installation and also during managing applications, as a link in the Additional Properties section.

1. Click **Applications > Install New Application**. Complete the steps (non-security related) required prior to the step titled **Map security roles to users and groups**.
2. Assign users and groups to roles.
3. Map users to RunAs roles if RunAs roles exist in the application.
4. Click **Correct use of System Identity** to specify RunAs roles if needed. Complete this action if the application has delegation set to use System Identity (applicable to enterprise beans only). System Identity uses the WebSphere Application Server security server ID to invoke downstream methods and should be used with caution as this ID has more privileges than other identities in terms of accessing WebSphere Application Server internal methods. This task is provided to make sure that the deployer is aware that the methods listed in the panel have System Identity set up for delegation and to correct them if necessary. If no changes are necessary, skip this task.
5. Complete the remaining (non-security related) steps to finish installing and deploying the application.

Once a secured application is deployed, verify that you can access the resources in the application with the correct credentials. For example, if your application has a protected Web module, make sure only the users that you assigned to the roles are able to use the application.

Assigning users and groups to roles

Before you perform this task:

- Secure the Web applications and EJB applications where new roles were created and assigned to Web and EJB resources.
- Create all the roles in your application.
- Verify that you have properly configured the user registry that contains the users that you want to assign. It is preferable to have security turned on with the user registry of your choice before beginning this process.
- Make sure that if you change anything in the security configuration (for example, enable security or change the user registry) you save the configuration and restart the server before the changes become effective.

Since the default active registry is LocalOS, it is not necessary, although it is recommended, that you enable security if you want to use the LocalOS registry to assign users and groups to roles. You can enable security once the users and groups are assigned in this case. The advantage of enabling security with the appropriate registry before proceeding with this task is that you can validate the security setup (which includes checking the user registry configuration) and avoid any problems using the registry.

These steps are common for both installing an application and modifying an existing application. If the application contains roles, you see the Map security roles to users/groups link during application installation and also during application management, as a link in the Additional Properties section at the bottom.

1. Access the administrative console by typing `http://localhost:9090/admin` in a Web browser.
2. Click **Map security roles to users/groups**. A list of all the roles that belong to this application displays. If the roles already had users or special subjects (All Authenticated, Everyone) assigned, they display here.
3. To assign the special subjects, select either the **Everyone** or the **All Authenticated** check box for the appropriate roles.
4. Click **Apply** to save any changes and then continue working with user or group 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 **Lookup Users** or **Lookup groups**.
7. Get the appropriate users and groups from the registry by completing the **limit** (number of items) and the **Search String** fields and clicking **Search**. The **limit** field limits the number of users that are obtained and displayed from the registry. The pattern is a searchable pattern matching one or more users and groups. For example, `user*` lists users like `user1`, `user2`. A pattern of asterisk (*) indicates all users or groups.

Use the limit and the search strings cautiously so as not to overwhelm the registry. When using large registries (like Lightweight Directory Access Protocol (LDAP)) where information on thousands of users and groups resides, a search for a large number of users or groups can make the system very slow and can make it fail. When there are more entries than requests for entries, a message displays on top of the panel. You can refine your search until you have the required list.

8. Select the users and groups to include as members of these roles from the **Available** box and click **>>** to add them to the roles.
9. To remove existing users and groups, select them from the **Selected** box and click **<<**. When removing existing users and groups from roles use caution if those same roles are used as RunAs roles.

For example, if `user1` is assigned to RunAs role, `role1`, and you try to remove `user1` from `role1`, the GUI validation does not delete the user since a user can only be a part of a RunAs role if the user is already in a role (`User1` should be in `role1` in this case) either directly or indirectly through a group. For more information on the validation checks that are performed between RunAs role mapping and user and group mapping to roles, see the Mapping users to RunAs roles section.

10. Click **OK**. If there are any validation problems between the role assignments and the RunAs role assignments the changes are not committed and an error message indicating the problem displays at the top of the panel. If there is a

problem, make sure that the user in the RunAs role is also a member of the regular role. If the regular role contains a group which contains the user in the RunAs role, make sure that the group is assigned to the role using the administrative console GUI. Follow steps 4 and 5. Avoid using the Assembly Toolkit or any other manual process where the complete name of the group, host name, group name, or distinguished name (DN) is not used.

The user and group information is added to the binding file in the application. This information is used later for authorization purposes.

This task is required to assign users and groups to roles, which enables the correct users and groups to access a secured application.

If you are installing an application, complete your installation. Once the application is installed and running you can access your resources according to the user and group mapping you did in this task. If you are managing applications and have modified the users and groups to role mapping, make sure you save, stop and restart the application so that the changes become effective. Try accessing the J2EE resources in the application to verify that the changes are effective.

Security role to user and group mappings

Use this page to map security roles to users. You can map roles to specific users, to specific groups, or to different categories.

To view this administrative console page, click **Application > Install New Application**. While running the Application Installation Wizard, prompts appear to help you map security roles to users or groups. To change role to user or group mappings for deployed applications, click **Application > Enterprise Application > *deployed_application* > Map security roles to users/groups**.

Users:

Specifies the users for role mapping. Verify that the users are defined in your chosen user registry.

To change the roles to users mapping, click **Manage Application > *application* > Map security roles to users**.

Data type: String

Groups:

Specifies the groups for role mapping. Verify that the groups are defined in your chosen user registry.

To change the roles to users mapping, click **Manage Application > *application* > Map security roles to groups**.

Data type: String

Roles:

Specifies the roles to which you want to map users and groups. Role privileges give users and groups permission to run as specified.

Select the check boxes to choose a role or a set of roles. Click **Look-up Users** to map users to the roles that you have selected. Click **Look-up Groups** to map groups to the selected roles. Use the check boxes to map roles to **EVERYONE** or **ALL AUTHENTICATED** special subject.

Data type: String

Everyone:

Specifies to map roles to everyone. Mapping a role to everyone means that anyone can access resources protected by this role, and essentially, there is no security.

Data type: Boolean

All Authenticated:

Specifies to authenticate all users. Roles are mapped to all authenticated users, and all authenticated users in the selected user registry are granted access to the role.

Data type: Boolean

Security role to user and group selections

Use this page to select users and groups for security roles.

To view this administrative console page, click **Application > Install New Application**.

While using the Install New Application Wizard, prompts appear to help you map security roles to users. You also can configure security roles to user mappings of deployed applications. Different roles can have different security authorizations. Mapping users or groups to a role authorizes those users or groups to access applications defined by the role. Users, groups and roles are defined when an application is installed or configured.

You also can select role to user and group mappings while you are deploying applications. After deployment in **Additional Properties**, click **Map Security roles to users** to change user and group mappings to a role.

Look up users:

Specifies whether the server looks up selected users.

Choose the role by selecting the check box beside the role and clicking **Lookup users**. Complete the **Limit** and the **Pattern** fields. The **Limit** field contains the number of entries that the search function returns. The **Pattern** field contains the search pattern used for searching entries. For example, bob* searches all users or groups starting with bob. A limit of zero returns all the entries that match the pattern. Use this value only when a small number of users or groups match this pattern in the registry. If the registry contains more entries that match the pattern than requested, a message appears in the console to indicate that there are more entries in the registry. You can either increase the limit or refine the search pattern to get all the entries.

Look up groups:

Specifies whether the server looks up selected groups.

Choose the role by selecting the check box beside the role and clicking **Lookup groups**. Complete the **Limit** and the **Pattern** fields. The **Limit** field contains the number of entries that the search function returns. The **Pattern** field contains the search pattern used for searching entries. For example, `bob*` searches all users or groups starting with bob. A limit of zero returns all the entries that match the pattern. Use this value only when a small number of users or groups match this pattern in the registry. If the registry contains more entries that match the pattern than requested, a message appears in the console to indicate that there are more entries in the registry. You can either increase the limit or refine the search pattern to get all the entries.

Role:

Specifies user roles.

A number of administrative roles are defined to provide degrees of authority needed to perform certain WebSphere administrative functions from either the Web-based administrative console or the system management scripting interface. The authorization policy is only enforced when global security is enabled. The following roles are valid:

- **Monitor**--least privileged that basically allows a user to view the server configuration and current state
- **Configurator**--monitor privilege plus the ability to change the server configuration
- **Operator**--monitor privilege plus the ability to change the run time state, such as starting or stopping services
- **Administrator**--operator plus configurator privilege

Range Monitor, Configurator, Operator, Administrator

Everyone:

Specifies to authenticate everyone.

Range Monitor, Configurator, Operator, Administrator

All authenticated:

Range Monitor, Configurator, Operator, Administrator

Mapped users:

Mapped groups:

Look up users and groups settings

Use this page to select users and groups for security roles.

To view this administrative console page, click **Applications > Enterprise Applications > *application_name* > Map security roles to users/groups > Look up users or groups** button.

Different roles can have different security authorizations. Mapping users or groups to a role authorizes those users or groups to access applications defined by the role. Users, groups and roles are defined when an application is installed or configured. Use the Search field to display users in the Available Users list. Click the arrows to add users from the Available Users list to the Selected Users list.

Limit:

Specifies the maximum number of users/groups that can be returned when assigning users/groups to roles.

A value of 0 implies a return of all users/groups that match the pattern. You can either increase the limit or refine the search pattern to get all the entries.

| | |
|------------------|-----------|
| Data type | Integer |
| Units | User name |
| Default | 20 |
| Range | 0 or more |

Pattern:

Indicates the search pattern used to search for the entries.

The pattern field should contain the search pattern that should be used to search for the entries. For example, bob* will search all users or groups starting with bob. A limit of 0 gets all the entries that match the pattern and should be used only when a small number users/groups match that pattern in the registry. If the registry contains more entries that match the pattern than requested for, a message shows in the console to indicate that there are more entries in the registry.

| | |
|------------------|-----------------|
| Data type | String |
| Units | Number of users |
| Default | 20 |
| Range | A-Z with * |

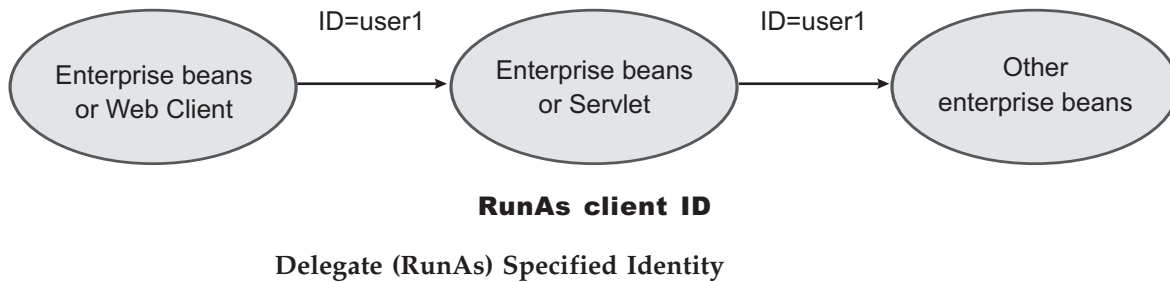
Delegations

Delegation is a process security identity propagation from a caller to a called object. As per the J2EE specification, a servlet and enterprise beans can propagate either the client (remote user) identity when invoking enterprise beans or they can use another specified identity as indicated in the corresponding deployment descriptor. The IBM extension supports Enterprise JavaBeans (EJB) to propagate to the server ID when invoking other entity beans. There are three types of delegations:

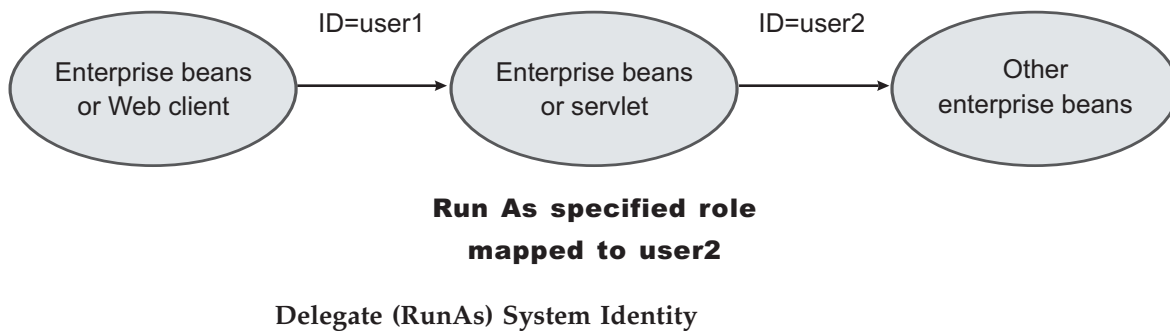
- Delegate (RunAs) Client Identity
- Delegate (RunAs) Specified Identity
- Delegate (RunAs) System Identity

Delegate (RunAs) Client Identity

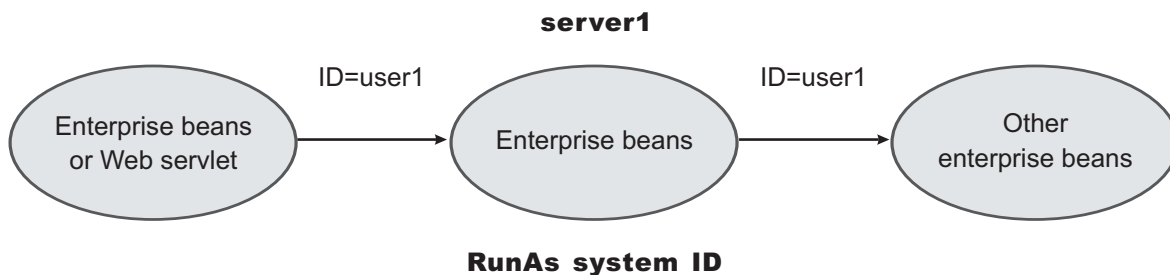
Delegate Client Identity



Delegate Specified Identity



Delegate System Identity



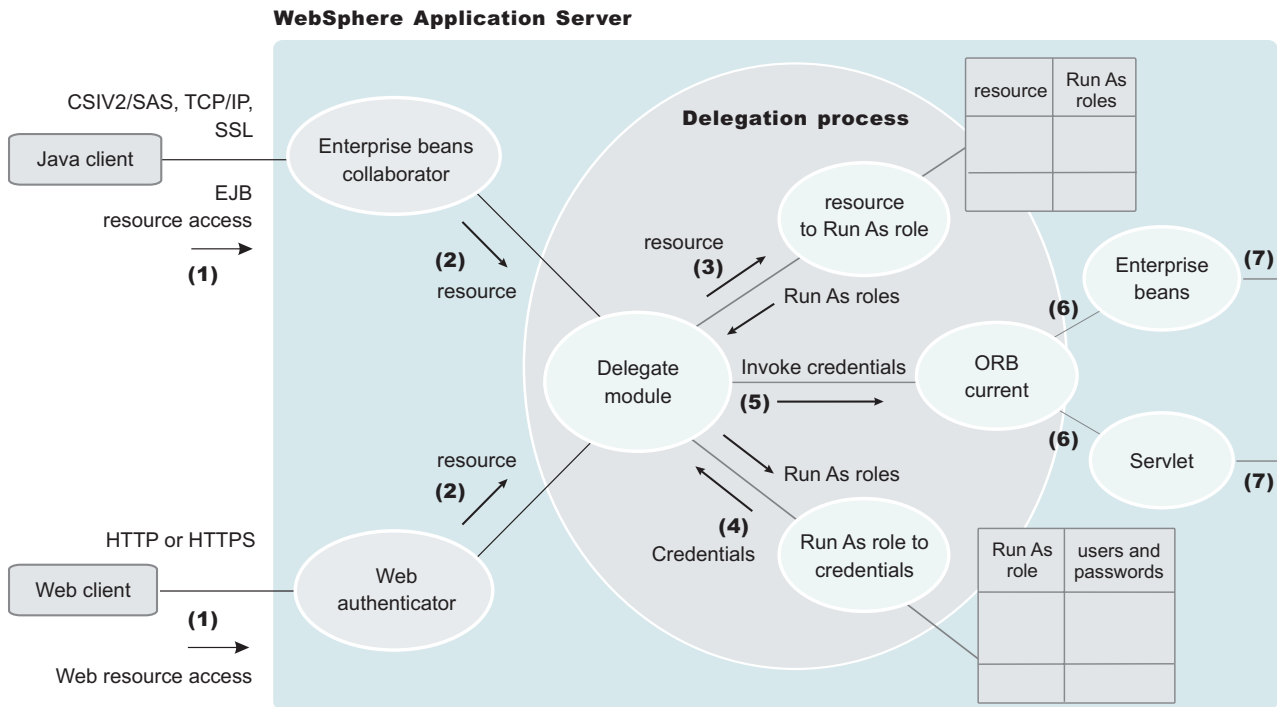
The EJB specification only supports delegation (RunAs) at the EJB level. But an IBM extension allows EJB method level RunAs specification. Method EJB method level runAs specification allows one to specify a different RunAs role for different methods within the same enterprise beans.

The RunAs specification is detailed in the deployment descriptor (the `ejb-jar.xml` file in the EJB module and the `web.xml` file in the Web module). The IBM extension to the RunAs specification is included in the `ibm-ejb-jar-ext.xmi` file.

There is also an IBM specific binding file for each application that contains a mapping from the RunAs role to the user. This file is specified in the `ibm-application-bnd.xmi` file.

These specifications are read by the run time during application startup. The following figure illustrates the delegation mechanism as implemented in the WebSphere Application Server security model.

Delegation



Delegation Process

There are two tables that help in the delegation process:

- Resource to RunAs role mapping table
- RunAs role to user ID and password mapping table

Use the Resource to RunAs role mapping table to get the role that is used by a servlet or by enterprise beans to propagate to the next enterprise beans call.

Use the RunAsRole to User ID and Password mapping table to get the user ID that belongs to the RunAs role and its password.

Delegation is performed after successful authentication and authorization. During this process, the delegation module consults the Resource to RunAs role mapping table to get the RunAs role (3). The delegation module consults the RunAs role to user ID and password mapping table to get the user that belongs to the RunAs role (4). The user ID and password is used to create a new credential using the authentication module, which is not shown in figure. The resulting credential is stored in the ORB Current as an invocation credential (5). Servlet and enterprise beans when invoking other enterprise beans pick up the invocation credential from the ORB Current (6) and call the next enterprise beans (7).

Assigning users to RunAs roles

Before you perform this task,

- Secure the Web applications and EJB applications where new RunAs roles were created and assigned to Web and EJB resources.

- Create all the RunAs roles in your application. The user in the RunAs role can only be entered if that user or a group to which that user belongs is already part of the regular role.
- Assign users and groups to security roles. Refer to Assigning users and groups to security roles for more information.
- Verify that the user registry requirements are met. These requirements are the same as those discussed in the same as in the case of Assigning users and groups to security roles task. For example, if role1 is a role that is also used as a RunAs role, then the user, user1, can be added to the RunAs role. role1, if user1 or a group that user1 belongs to, already is assigned to role1. The administrative console checks this logic when **Apply** or **OK** is clicked. If the check fails, the change is not made and an error message displays at the top of the panel.

If the special subjects "Everyone" or "All Authenticated" are assigned to a role, then no check takes place for that role.

The checking is done every time **Apply** in this panel is clicked or when **OK** is clicked in the **Map security roles to users and groups** panel. The check verifies that all the users in all the RunAs roles do exist directly or indirectly (through a group) in those roles in the **Map security roles to users and groups** panel. If a role is assigned both a user and a group to which that user belongs, then either the user or the group (not both) can be deleted from **Map security roles to users and groups** panel.

If the RunAs role user belongs to a group and if that group is assigned to that role, make sure that the assignment of this group to the role is done through administrative console and not through the Assembly Toolkit or any other method. When using the administrative console, the full name of the group is used (for example, hostname\groupName in windows systems, and distinguished names (DN) in Lightweight Directory Access Protocol (LDAP)). During the check, all the groups to which the RunAs role user belongs are obtained from the registry. Since the list of groups obtained from the registry are the full names of the groups, the check works correctly. If the short name of a group is entered using the Assembly Toolkit (for example, group1 instead of CN=group1, o=myCompany.com) then this check fails.

These steps are common to both installing an application and modifying an existing application. If the application contains RunAs roles, you see the **Map RunAs roles to users** link during application installation and also during managing applications as a link in the **Additional Properties** section at the bottom.

1. Click **Map RunAs roles to users**. A list of all the RunAs roles that belong to this application displays. If the roles already had users assigned, they display here.
2. 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.
3. Enter the user's name and password in the designated fields. The user name entered can be either the short name (preferred) or the full name (as seen when getting users and groups from the registry).
4. 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.
5. To remove a user from a RunAs role, select the roles and click **Remove**.

The RunAs role user is added to the binding file in the application. This file is used for delegation purposes when accessing J2EE resources.

This step is required to assign users to RunAs roles so that during delegation the appropriate user is used to invoke the EJB methods.

If you are installing the application, complete installation. Once the application is installed and running you can access your resources according to the RunAS role mapping. Save the configuration.

If you are managing applications and have modified the RunAs roles to users mapping, make sure you save, stop and restart the application so that the changes become effective. Try accessing your J2EE resources to verify that the new changes are in effect.

Unprotected EJB 2.0 methods protection settings

Use this page to verify that unprotected EJB 2.0 methods have the correct level of protection before you map users to roles.

To view this administrative console page, click **Application > Install New Application**. While running the Install New Application Wizard, prompts appear to help you map security roles to users.

Exclude:

Specifies that the method is completely protected.

Data type: Check box
Default: Cleared

Uncheck:

Specifies that everyone can access the security method.

Data type: Check box
Default: Uncheck

Specify role:

Specifies the EJB level of protection based on the security role.

The roles listed in this menu are obtained from the application scope. If the selected role is not in the module, then it is added to the modules or Java archive (JAR) files.

Data type: String
Units: Role

Module name:

Specifies the name of the module.

If a module name appears in this list, then the module contains unprotected EJB methods.

Data type: String
Units: Module name

Protection:

Specifies the level of protection assigned to a particular module name.

Data type: String
Default: Cleared

EJB 1.0 method protection level settings

Use this page to verify that all unprotected EJB 1.0 methods have the correct level of protection before you map users to roles.

To view this administrative console page, click **Applications > Install New Application**. While running the Install New Application Wizard, prompts appear to help you determine that all unprotected EJB 1.0 methods have the correct level of protection.

EJB Module:

Specifies the enterprise bean module name.

Data Type: String
Units: EJB module name

Module URI:

Specifies the Java archive (JAR) file name.

Data Type: String
Units: JAR file name

Method protection:

Specifies the level of protection assigned to the EJB module.

A selected box means to *Deny All* and that the method is completely protected.

Data Type: Check box
Default: Cleared
Range: Yes or No

RunAs roles to users mapping

Use this page to map RunAs roles to users. You can change the RunAs settings after an application deploys.

To view this administrative console page, click **Applications > Install New Application**. While running the application installation wizard, prompts appear to help you map RunAs roles to users. You can change the RunAs roles to users mappings for deployed applications. Click **Applications > application_name > Map RunAs roles to users** in the Additional Properties section.

The enterprise beans you are installing contain predefined RunAs roles. RunAs roles are used by enterprise beans that need to run as a particular role for recognition while interacting with another enterprise bean.

User name:

Specifies a user name for the RunAs role user.

This user already maps to the role specified in the Mapping users and groups to roles panel. You can map the user to its appropriate role by either mapping the user to that role directly or mapping a group that contains the user to that role.

Data type: String

Password:

Specifies the password for the RunAs user.

Data type: String

Confirm password:

Specifies the confirmed password of the administrative user.

Data type: String

Role:

Specifies administrative user roles.

A number of administrative roles have been defined to provide degrees of authority needed to perform certain WebSphere administrative functions from either the web based administrative console or the system management scripting interface. The authorization policy is only enforced when global security is enabled. The following roles are valid:

- **Monitor**--least privileged that basically allows a user to view the WebSphere configuration and current state
- **Configurator**--monitor privilege plus the ability to change the WebSphere configuration
- **Operator**--monitor privilege plus the ability to change runtime state, such as starting or stopping services for example
- **Administrator**--operator plus configurator privilege

Updating and redeploying secured applications

Before you perform this task, secure Web applications, secure EJB applications, and deploy them in WebSphere Application Server. This section addresses the way to update existing applications.

1. Use the administrative console to modify the existing users and groups mapping to roles. The task titled Assigning users and groups to roles details the required steps.
2. Use the administrative console to modify the users for the RunAs roles. The task entitled, Assigning users to RunAs roles details the required steps.

3. Complete the changes and save them.
4. Stop and restart the application for the changes to become effective.
5. Use the Assembly Toolkit to update any other security related information.
6. Use the Assembly Toolkit to modify roles, method permissions, auth-constraints, data-constraints and so on.
7. Save the Enterprise Archive (EAR) file, uninstall the old application, deploy the modified application and start the application to make the changes effective.

The applications are modified and redeployed.

This step is required to modify existing secured applications.

If information about roles is modified make sure you update the user and group information using the administrative console. Once the secured applications are modified and either restarted or redeployed, make sure that the changes are effective by accessing the resources in the application.

Testing security

After configuring global security and restarting all of your servers in a secure mode, it is best to validate that security is properly enabled. There are a few techniques that you can use to test the various security login types. For example, you can test the Web-based BasicAuth login, Web-based form login, and the Java client BasicAuth login. There are basic tests that show that the fundamental security components are working properly. Complete the following steps to validate your security configuration:

1. Test the Web-based BasicAuth with *Snoop*, by accessing the following URL: `http://hostname.domain:9080/snoop`. A login panel appears. If a login panel does not appear, then a problem exists. If the panel appears, type in any valid user ID and password in your configured user registry.

Note: In a Network Deployment environment, the Snoop servlet is only available in the domain if you included the **DefaultApplication** option when adding the application server to the cell. The **-includeapps** option for the **addNode** command migrates the **DefaultApplication** option to the cell. Otherwise, skip this step.

2. Test the Web-based form login by bringing up the administrative console: `http://hostname.domain:9090/admin`. A form-based login page appears. If a login page does not appear, try accessing the administrative console by typing `https://myhost.domain:9043/admin`. Type in the administrative user ID and password used for configuring your user registry when configuring security. When the authentication mechanism is set as Lightweight Third Party Authentication (LTPA), represent the host name as a fully qualified host name (that is, `myhost.mycompany.com:9090` rather than just `myhost:9090`).
3. Test Java Client BasicAuth with *dumpNameSpace* by executing the `install_root\bin\dumpNameSpace.bat` file. A login panel appears. If a login panel does not appear, there is a problem. Type in any valid user ID and password in your configured user registry.
4. Thoroughly test all of your applications in secure mode.
5. After enabling security, verify that your system comes up in secure mode.
6. If all tests pass, proceed with more rigorous testing of your secured applications. If you have any problems, review the output logs in the WebSphere Application Server `/logs/nodeagent` or WebSphere Application

Server `/logs/server_name` directories, respectively. Then check the security troubleshooting article to see if it references any common problems.

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

Managing security

Administering secure applications requires access to the WebSphere Application Server administrative console. Otherwise, log in with a valid user ID and password that have administrative access. To administer security, complete these steps:

1. Configure global security.
2. Assign users to administrative roles.
3. Assign users to naming roles.
4. Configure authentication mechanisms.
5. Configure Lightweight Third Party Authentication.
6. Configure trust association interceptors.
7. Configure single signon.
8. Configure user registries.
 - a. Configure local operating system user registries.
 - b. Configure Lightweight Directory Access Protocol user registries.
 - c. Configure custom user registries.
9. Configure Java Authentication and Authorization Service login.
10. "Configuring Common Secure Interoperability Version 2 and Security Authentication Service authentication protocols" on page 235
11. Configure Secure Sockets Layer.
12. Configure Java 2 Security Manager

Global security

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

The term *global security* represents the security configuration that is effective for the entire security domain. A *security domain* consists of all servers configured with the same user registry *realm* name. In some cases, the realm can be the machine name of a Local OS user registry. In this case, all application servers must reside on the same physical machine. In other cases, the realm can be the machine name of an Lightweight Directory Access Protocol (LDAP) user registry. Since LDAP is a distributed user registry, a multiple node configuration is supported, such as the case for a Network Deployment environment. The basic requirement for a security domain is that the access ID returned by the registry from one server within the security domain is the same access ID as that returned from the registry on any other server within the same security domain. The *access ID* is the unique identification of a user and is used during authorization to determine if access is permitted to the resource.

Configuration of global security for a security domain consists of configuring the common user registry, the authentication mechanism, and other security information, which defines the behavior of a security domain. The other security information that you can configure includes Java 2 Security Manager, Java Authentication and Authorization Service (JAAS), Java 2 Connector authentication

data entries, Common Secure Interoperability Version 2 (CSIV2)/Security Authentication Service (SAS) authentication protocol (Remote Method Invocation over the Internet Inter-ORB Protocol (RMI/IIOP) security), and other miscellaneous attributes. The global security configuration usually applies to every server within the security domain.

Configuring global security

It is helpful to understand security from an infrastructure standpoint so that you know the advantages of different authentication mechanisms, user registries, authentication protocols, and so on. Picking the right security components to meet your needs is a part of configuring global security. The following sections help you make these decisions. Read the following articles before continuing with the security configuration.

- Global security
- Welcome to security

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

1. Start the WebSphere Application Server administrative console by clicking `http://yourhost.domain:9090/admin` after starting the WebSphere Application Server. If security is currently disabled, log in with any user ID. If security is currently enabled, log in with a predefined administrative ID and password (this is typically the server user ID specified when you configured the user registry).
2. Click **Security** from the left navigation menu. Configure the authentication mechanism, user registry, and so on. The configuration order is not important. However, when you select the **Enabled** flag in the **Global Security** panel, verify that all these tasks are completed. When you click **Apply** or **OK** and the **Enabled** flag is set, a verification occurs to see if the administrative user ID and password can be authenticated to the configured user registry. If you do not configure these, the validation fails.
3. Configure a user registry. For more information, see *Configuring user registries*. Configure (LocalOS, LDAP, or Custom), and then specify details about that registry. One of the details common to all user registries is the **server user ID**. This ID is a member of the chosen user registry, but also has special privileges in WebSphere Application Server. The privileges for this ID and the privileges associated with the administrative role ID are the same. The server user ID can access all protected administrative methods. On Windows systems, the ID must not be the same name as the machine name of your system, since the registry sometimes returns machine-specific information when querying a user of the same name. In LDAP user registries, verify that the server user ID is a member of the registry and not just the LDAP administrative role ID. The entry must be searchable.

The server user ID does **not** run WebSphere Application Server processes. Rather, the process ID runs the WebSphere Application Server processes. The process ID is determined by the way the process starts. For example, if you use a command line to start processes, the user ID that is logged into the system is the process ID. If running as a service, the user ID that is logged into the system is the user ID running the service. If you choose the LocalOS registry, the process ID requires special privileges to call the operating system APIs. Specifically, the process ID must have the **Act as Part of Operating System** privileges on Windows systems or **root** privileges on a UNIX system.

4. Configure the authentication mechanism. To get details about configuring authentication mechanisms, read the *Configuring Authentication Mechanisms* article. There are two authentication mechanisms to choose from in the Global Security panel: Simple WebSphere Authentication Mechanism (SWAM) and Lightweight Third-Party Authentication (LTPA). However, only LTPA requires any additional configuration parameters. Use the SWAM option for single server requirements. Use the LTPA option for multi-server distributed requirements. SWAM credentials are not forwardable to other machines and for that reason do not expire. Credentials for LTPA are forwardable to other machines and for security reasons do expire. This expiration time is configurable. If you choose to go with LTPA, then Configuring single signon support. This support permits browsers to visit different product servers without having to authenticate multiple times.
5. Configure the authentication protocol for special security requirements from Java clients, if needed. This task entails choosing a protocol, either Common Secure Interoperability Version 2 (CSIv2) or Security Authentication Service (SAS). The CSIv2 protocol is new to WebSphere Application Server Version 5 and has many new and improved features. The SAS protocol is still provided as a backwards compatibility to previous product releases, but is being deprecated. For details on configuring CSIv2 or SAS, see the article, *Configuring Common Secure Interoperability Version 2 and Security Authentication Service authentication protocols*.
6. Modify the default Secure Sockets Layer (SSL) keystore and truststore files that are packaged with the product. This action protects the integrity of the messages sent across the Internet. The product provides a single location where you can specify SSL configurations that the various WebSphere Application Server features that use SSL can utilize, including the LDAP user registry, Web container and the authentication protocol (CSIv2 and SAS). Create a new keystore and truststore, by referring to the *Creating keystore files* and *Creating truststore files* articles. You can create different keystore files and truststore files for different uses or you can create just one set for everything that the server uses SSL for. Once you create these new keystore and truststore files, specify them in the SSL configuration repertoire. See the article, *Configuring Secure Sockets Layer* for more information. To get to the SSL Configuration Repertoire, click **Security > SSL**. You can either edit the `DefaultSSLConfig` file or create a new SSL configuration with a new alias name. If you create a new alias name for your new keystore and truststore files, change every location that references the `DefaultSSLConfig` SSL configuration alias. The following list provides these locations:
 - **Security > User Registries > LDAP** (at the bottom of the panel)
 - **Security > Authentication Protocol > CSIv2 Inbound Transport**
 - **Security > Authentication Protocol > CSIv2 Outbound Transport**
 - **Security > Authentication Protocol > SAS Inbound Transport**
 - **Security > Authentication Protocol > SAS Outbound Transport**
 - **Servers > Application Servers > *application_server_name* > Web Container > HTTP transports > *host_link***
7. Click **Security > Global Security** to configure the rest of the security settings and enable security. This panel performs a final validation of the security configuration. When you click **OK** or **Apply** from this panel, the security validation routine is performed and any problems are reported at the top of the page. See the *Global security settings* article for detailed information about these fields. When you complete all of the fields, click **OK** or **Apply** to accept the selected settings. Click **Save** to persist these settings out to a file. If you see any informational messages in red text color, then a problem has occurred with the security validation. Typically, the message indicates the problem. So, review

your configuration to verify that the user registry settings are accurate and the correct registry is selected. In some cases the LTPA configuration might not be fully specified.

8. Store the configuration for the server to use once it restarts. Complete this action if you have clicked **OK** or **Apply** on the **Security > Global Security** panel, and there are no validation problems. To save the configuration, click **Save** in the menu bar at the top. This action writes the settings out to the configuration repository. If you do not click **Apply** or **OK** in the **Global Security** panel before clicking **Save** on the main menu, your changes are not written to the repository.
9. Start the WebSphere Application Server administrative console by typing `http://yourhost.domain:9090/admin` after the WebSphere Application Server deployment manager has been started. If security is currently disabled, log in with any user ID. If security is currently enabled, log in with a predefined administrative ID and password, which is typically the server user ID specified when you configure the user registry.

Enabling and disabling global security

You can decide whether to enable IBM WebSphere Application Server security. You must enable security for all other security settings to function.

1. Enable global security in the WebSphere Application Server. It is important to click **Security > Global Security** and set the **Enabled** flag to on so that security is enabled upon a server restart.
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 `<install_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 `<install_root>/logs/server_name` directory. Check the Troubleshooting Security article for any common problems.

Disabling global security:

1. Click **Security > Global Security** and set the **Enabled** flag to off so that security gets disabled upon a server restart.
2. Before restarting the server, log off of the administrative console. You can log out by clicking **Log off** at the top menu bar.
3. Stop the server by going to the command line, accessing the WebSphere Application Server `/bin` directory, and issuing the following command on one continuous line:

```
stopServer server_name -username <administrative_user_name> -password  
-<administrative_password>
```

You must include the administrative user ID to stop the server when security is enabled. Alternatively, you can edit the `sas.client.props` file in the `<install_root>/properties` directory and edit the `com.ibm.SOAP.loginUserId`

or `com.ibm.SOAP.loginPassword` properties to contain these administrative IDs. If you enter `stopServer server_name`, the administrative user ID is picked up from the `sas.client.props` file.

4. Issue the following command to restart the server in secure mode:
`startServer <server_name>`
5. If you have any problems restarting the server, review the output logs in the `<install_root>/logs/server_name` directory.

This scenario is specifically for a stand-alone setup where you have a single application server and likely utilize your Local OS registry for your repository of users. The authentication mechanism is probably Simple WebSphere Authentication Mechanism (SWAM). The application server cannot communicate securely to other application servers as the SWAM authentication mechanism does not contain a forwardable token to send to downstream servers.

After restarting the server in secure mode, run a couple of simple tests to verify that most facets of security are working properly.

1. Test basic authentication with snoop by accessing the following URL:
`http://hostname.domain:9080/snoop`. A login panel appears. Type in any valid user ID and password in your configured user registry. If the login panel fails to appear, there is a problem.
2. Test the Java client with `dumpNameSpace` by executing the `<install_dir>\bin\dumpNameSpace.bat` file. A login panel appears. Type in any valid user ID and password in your configured user registry. If the login panel fails to appear, there is a problem.
3. Test form login by bringing up the administrative console:
`http://hostname.domain:9090/admin`. A form-based login page appears. Type in the administrative user ID and password that was used for configuring your user registry when configuring security. When the Authentication Mechanism is set as **LTPA**, provide a fully qualified host name (for example, `myhost.mycompany.com:9090`, rather than just `myhost:9090`). If the login panel fails to appear, there is a problem.

If you encountered a problem with any of these tests, check the WebSphere Application Server `/logs/server_name/SystemOut.log` file for hints about the problems that occurred. Also refer to *Troubleshooting Security* for solutions.

Global security settings:

Use this page to configure security. When you enable security, you are enabling security settings on a global level. When security is disabled, WebSphere Application Server performance is increased between 10-20%. Therefore, consider disabling security when it is not needed.

To view this administrative console page, click **Security > Global Security**.

If you are configuring security for the first time, complete the steps in "Configuring global security" in the documentation to avoid problems. When security is configured, validate any changes to the registry or authentication mechanism panels. Click **Apply** to validate the user registry settings. An attempt is made to authenticate the server ID to the configured user registry. Validating the user registry settings after enabling global security can avoid problems when you restart the server for the first time.

Enabled:

Specifies for the server to enable security subsystems.

This flag is commonly referred to as the *global security flag* in WebSphere Application Server information. When enabling security, set the authentication mechanism configuration and specify a valid user ID and password in the selected user registry configuration.

| | |
|-------------------|---------|
| Data type: | Boolean |
| Default: | Disable |

Enforce Java 2 Security:

Specifies whether to enable or disable Java 2 security permission checking. By default, Java 2 security is disabled. However, enabling global security, automatically enables Java 2 security. You can choose to disable Java 2 security, even when global security is enabled.

When Java 2 security is enabled and if an application requires more Java 2 security permissions than are granted in the default policy, then the application might fail to run properly until the required permissions are granted in either the `app.policy` file or the `was.policy` file of the application. `AccessControl` exceptions are generated by applications that do not have all the required permissions. Consult the WebSphere Application Server documentation and review the Java 2 Security and Dynamic Policy sections if you are unfamiliar with Java 2 security.

If your server does not restart after you enable global security, you can disable security. Go to your `$(install_root)\bin` directory and execute the `wsadmin -conntype NONE` command. At the `wsadmin>` prompt, enter `securityoff` and then type `exit` to return to a command prompt. Restart the server with security disabled to check any incorrect settings through the administrative console.

| | |
|-------------------|---------------------|
| Data type: | Boolean |
| Default: | Disabled |
| Range: | Enabled or Disabled |

Use Domain Qualified User Names:

Specifies the user names to qualify with the security domain within which they reside.

| | |
|-------------------|-------------------|
| Data type: | Boolean |
| Default: | Disabled |
| Range: | Enable or Disable |

When you specify **Use Domain Qualified User Names** from the **Security > Global Security** configuration panel, the run-time call to the `getCallerPrincipal()` API from an enterprise bean returns the qualified name with the realm prepended twice. For example, the format return is `realm/realm/user`. You can strip the first realm from the returned value when making API calls. The servlet API `getUserPrincipal()` works correctly.

Cache Timeout:

Specifies the timeout value in seconds for security cache. This value is a relative timeout.

If WebSphere Application Server security is enabled, the security cache timeout can influence performance. The timeout setting specifies how often to refresh the security-related caches. Security information pertaining to beans, permissions, and credentials is cached. When the cache timeout expires, all cached information becomes invalid. Subsequent requests for the information result in a database lookup. Sometimes, acquiring the information requires invoking a Lightweight Directory Access Protocol (LDAP)-bind or native authentication. Both invocations are relatively costly operations for performance. Determine the best trade off for the application, by looking at usage patterns and security needs for the site.

In a 20-minute performance test, setting the cache timeout so that a timeout does not occur yields a 40% performance improvement.

| | |
|-------------------|-------------------------|
| Data type: | Integer |
| Units: | Seconds |
| Default: | 600 |
| Range: | Greater than 30 seconds |

Issue Permission Warning:

Specifies that when the Issue permission warning option is enabled, during application deployment and application start, the security run time emits a warning if applications are granted any custom permissions. Custom permissions are permissions defined by the user applications, not Java API permissions. Java API permissions are permissions in package `java.*` and `javax.*`.

The WebSphere product provides support for policy file management. A number of policy files are available in this product, some of them are static and some of them are dynamic. Dynamic policy is a template of permissions for a particular type of resource. There is no code base defined or relative code base used in the dynamic policy template. The real code base is dynamically created from the configuration and run-time data. The `filter.policy` file contains a list of permissions that an application should not have according to the J2EE 1.3 specification. For more information on permissions, see the Java 2 Security policy management article.

| | |
|-------------------|-------------------|
| Data type: | Boolean |
| Default: | Disabled |
| Range: | Enable or Disable |

Active Protocol:

Specifies the active authentication protocol for Remote Method Invocation over the Internet Inter-ORB Protocol (RMI IIOP) requests when security is enabled. In previous releases the Security Authentication Service (SAS) protocol was the only available protocol.

An Object Management Group (OMG) protocol called Common Secure Interoperability Version 2 (CSIv2) supports increased vendor interoperability and additional features. If all of the servers in your security domain are Version 5 servers, specify CSI as your protocol. If some servers are 3.x or 4.x servers, specify CSI and SAS.

Data type: String
Default: BOTH
Range: CSI and SAS, CSI

Active Authentication Mechanism:

Specifies the active authentication mechanism when security is enabled.

Data type: String
Default: SWAM (WebSphere Application Server)
Range: SWAM, LTPA

Active User Registry:

Specifies the active user registry, when security is enabled.

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

- Local operating system.
- LDAP user registry. The LDAP user registry settings are used when users and groups reside in an external LDAP directory. When security is enabled and any of these properties change, go to the Global Security panel and click **Apply** or **OK** to validate the changes.
- Custom user registry

Data type: String
Default: Local OS
Range: Local OS, LDAP, Custom

Use FIPS:

Enables the use of FIPS (Federal Information Processing Standard)-approved cryptographic algorithms.

When **Use FIPS** is enabled, the Lightweight Third Party Authentication (LTPA) implementation uses IBMJCEFIPS. IBMJCEFIPS supports the Federal Information Processing Standard (FIPS)-approved cryptographic algorithms for DES, Triple DES, and AES. Although the LTPA keys are backwards compatible with prior releases of WebSphere Application Server, the LTPA token is not compatible with prior releases.

Important: The IBMJSSEFIPS and IBMJCEFIPS modules are undergoing certification.

WebSphere Application Server provides a FIPS-approved Java Secure Socket Extension (JSSE) provider called IBMJSSEFIPS. A FIPS-approved JSSE requires the Transport Layer Security (TLS) protocol as it is not compatible with the Secure Sockets Layer (SSL) protocol. If you select the **Use FIPS** checkbox prior to specifying a FIPS-approved JSSE provider and a TLS protocol, the following error message displays at the top of the **Global Security** panel:

The security policy is set to use only FIPS approved cryptographic algorithms. However at least one SSL configuration may not be using a FIPS approved JSSE provider. FIPS approved cryptographic algorithms may not be used in those cases.

To correct this problem, configure your JSSE provider and security protocol on the **SSL Configuration Repertoires** panel by completing one of the following tasks:

- Clicking **Security** > **SSL** and modifying an existing configuration
- Clicking **New** and creating a new configuration

Global security and server security

The term *global security* refers to the security configuration that is effective for the entire security domain. A security domain consists of all the servers configured with the same user registry realm name.

In some cases, the realm can be the machine name of a Local OS user registry. In this case, all application servers must reside on the same physical machine. In other cases, the realm can be the machine name of a Lightweight Directory Access Protocol (LDAP) user registry. Since LDAP is a distributed user registry, this allows for a multiple node configuration in a Network Deployment environment. The basic requirement for a security domain is that the access ID returned by the registry from one server within the security domain is the same access ID that is returned from the registry on any other server within the same security domain. The access ID is the unique identification of a user and is used during authorization to determine if access is permitted to the resource.

Configuration of global security for a security domain consists of configuring the common user registry, the authentication mechanism, and other security information that defines the behavior of a security domain. The other security information that is configured includes Java 2 Security Manager, Java Authentication and Authorization Service (JAAS), Java 2 Connector authentication data entries, Common Secure Interoperability Version 2 (CSIv2)/Security Authentication Service (SAS) authentication protocol (Remote Method Invocation over the Internet Inter-ORB Protocol (RMI/IIOP) security), and other miscellaneous attributes. The global security configuration usually applies to every server within the security domain. You can override some portions of the configuration at the server level.

In a Network Deployment environment, where multiple nodes and multiple servers within a node are possible, you can configure certain attributes at a server level. The attributes that are configurable at a server level include security enablement for the server, Java 2 Security Manager enablement, and CSIv2/SAS authentication protocol (RMI/IIOP security). You can disable security on individual application servers while global security is enabled, however, you cannot enable security on an individual application server while global security is disabled.

While application server security is disabled for user requests, administrative and naming security is still enabled for that application server so that the administrative and naming infrastructure remains secure. If cell security is enabled, but security for individual servers is disabled, J2EE applications are not authenticated or authorized. However, naming and administrative security is still enforced. Consequently, because Naming Services can be called from user applications you need to grant Everyone access to the naming functions that are required so that these functions accept unauthenticated requests. User code does not directly access administrative security except through the supported scripting tools.

Configuring server security

You can customize security to some extent at the application server level. You can disable user security on an application server (administrative security remains

enabled when global security is enabled). You can also modify Java 2 Security Manager, CSIV2 or Secure Authentication Service (SAS), and some of the other security attributes that are found on the global security (also called *cell-level* security) panel. You cannot configure a different authentication mechanism or user registry on an individual server basis. This feature is limited to cell-level configuration only. Also, when global security is disabled, you cannot enable application server security.

By default, server security inherits all of the values that are configured in global security (cell-level security). To override the security configuration at the server level, click **Servers > Application Servers > *server name***. Under Additional Properties, click **Server Security** and click any of the following panels: **Server Level Security, CSI Authentication > Inbound, CSI Authentication > Outbound, CSI Transport > Inbound, CSI Transport > Outbound, SAS Transport > Inbound, and SAS Transport > Outbound**. After modifying the configuration in any of these panels and clicking **OK** or **Apply**, the security configuration for that panel or set of panels now overrides cell-level security. Other panels that are not overridden continue to be inherited at the cell-level. However, you can always revert back to the cell-level configuration at any time. On the Server Security panel, click **Use Cell Security, Use Cell CSI, and Use Cell SAS** to revert back to the global security configuration on these panels.

1. Start the administrative console for the deployment manager. To get to the administrative console, go to `http://host.domain:9090/admin`. If security is disabled, you can enter any ID. If security is enabled, you must enter a valid user ID and password, which is either the administrative ID (configured for the user registry) or a user ID entered as an administrative user. To add a user ID as an administrative user, click **System Administration > Console Users**.
2. Configure global security if you have not already done so. Go to the Configure Global Security article for detailed steps. After global security is configured, configure server-level security.
3. To configure server-level security, click **Servers > Application Servers > *server name***. Under Additional Properties, click **Server Security**. The status of the security level that is in use for this application server is displayed. By default, you can see that global security, CSI, and SAS have not been overridden at the server level. CSI and SAS are authentication protocols for RMI/IIOP requests. The Server Level Security panel lists attributes that are on the Global Security panel and can be overridden at the server level. Not all of the attributes on the Global Security panel can be overridden at the server level, including Active Authentication Mechanism and Active User Registry.
4. To disable security for this application server, go to the Server Level Security panel, clear the **Enabled** flag and click **OK** or **Apply**. Click **Save**. By modifying the Server Level Security panel, you can see that this flag overrides the cell-level security.
5. To configure CSI at the server level, you can change any panel that starts with CSI. By doing so, all panels that start with CSI will override the CSI settings specified at the Cell-level. This includes all authentication and transport panels for CSI. See the Configuring CSIV2 and SAS authentication protocols article for more detailed steps regarding configuring CSI authentication protocol.

Typically server-level security is used to disable user security for a specific application server. However, this can also be used to disable (or enable) the Java 2 Security Manager, and configure the authentication requirements for RMI/IIOP requests both incoming and outgoing from this application server.

Once you have modified the configuration for a particular application server, you must restart the application server for the changes to become effective. To restart the application server, go to **Servers > Application Servers** and click the server name that you recently modified. Then, click the **Stop** button and then the **Start** button.

If you disabled security for the application server, you can typically test a URL which is protected when security is enabled. One URL that usually is installed when the DefaultApplication is installed is the snoop application. If the DefaultApplication is installed on the application server, test that security is disabled by going to the following URL: `http://host.domain:9080/snoop`. If you have disabled security, you should not get prompted. This is just one method of validating the configuration. You should validate that the configuration is appropriate for your applications.

Server security settings

Use this page to configure server security and override the global security settings. If you need to revert to the global security defaults, deselect the appropriate check box in the administrative console.

To view this administrative console page, click **Servers > Application servers > server_name > Server Security**. Note that these settings only apply to Network Deployment.

You can disable security on individual application servers while global security is enabled. However, you cannot enable security on an individual application server while global security is disabled. While application server security is disabled for user requests, administrative and naming security is still enabled for that application server so that the administrative and naming infrastructure remains secure. To avoid problems, verify that the naming security has **Everyone** access to the naming function that you use within your user code. You do not need to configure administrative security, because user code does not directly access administrative functions. User code accesses administrative functions through the supported scripting tools.

Server Level Security:

Specifies whether the server overrides cell defaults for security.

To revert to the cell defaults for Server Level Security, click **Use Cell Security**. Click **Apply** and then select **Save** to validate the changes at the server level.

Default False

CSI:

Specifies whether the server overrides cell defaults for the CSI protocol.

To revert back to the cell defaults for the CSI protocol, click **Use Cell CSI**. Click **Apply** and then select **Save** to validate the changes at the server level.

Default False

SAS:

Specifies whether the server overrides cell defaults for the Secure Authentication Service (SAS) protocol.

To revert back to the cell defaults for the SAS protocol, click **Use Cell SAS**. Click **Apply** and then select **Save** to validate the changes at the server level.

Default False

Server-level security settings

Use this page to enable server level security and specify other server level security configurations.

To view this administrative console page, click **Servers > Application Servers > *server_name* > Server Security > Server Level Security**.

Enabled:

Use this flag to disable or enable security again for this application server while global security is enabled. Server security is enabled by default when global security is enabled. You cannot enable security on an application server while global security is disabled. Administrative (administrative console and wsadmin) and naming security remain enabled while global security is enabled, regardless of the status of this flag.

Data type Boolean
Default Disable

Enforce Java 2 Security:

Specifies that the server enforces Java 2 Security permission checking at the server level. When cleared, the Java 2 server-level security manager is not installed and all of the Java 2 Security permission checking is disabled at the server level.

If your application policy file is not set up correctly, see *Configuring the was.policy* file for information on how to configure an application policy file.

Data type Boolean
Default Disabled
Range Enabled or Disabled

Use Domain Qualified User IDs:

Specifies whether user IDs returned by `getUserPrincipal()`-like calls are qualified with the server level security domain within which they reside.

Data type Boolean
Default Disabled
Range Enable or Disable

Cache Timeout:

Specifies the timeout value for server level security cache in seconds.

Data type Integer
Units Seconds

| | |
|----------------|---|
| Default | 600 |
| Range | Greater than 30 seconds. Avoid setting cache timeout value to 30 seconds or less. |

Issue Permission Warning:

Specifies whether a warning is issued during application installation when an application requires a Java 2 permission that is normally not granted to an application.

WebSphere Application Server provides support for policy file management. A number of policy files are included in WebSphere Application Server. Some of these policy files are static and some of them are dynamic. Dynamic policy is a template of permissions for a particular type of resource. In dynamic policy files, the code bases are evaluated at run time using configuration data. You can add or remove permissions, as needed, for each code base. However, do not add, remove, or modify the existing code bases. The real code base is dynamically created from the configuration and run-time data. The `filter.policy` file contains a list of permissions that an application does not have, according to the J2EE 1.3 Specification. For more information on permissions, see the Java 2 Security permissions article.

| | |
|------------------|-------------------|
| Data type | Boolean |
| Default | Enabled |
| Range | Enable or Disable |

Active Protocol:

Specifies the active server level security authentication protocol when server level security is enabled.

You can use an Object Management Group (OMG) protocol called Common Secure Interoperability Version 2 (CSIv2) for more vendor interoperability and additional features. If all of the servers in your entire security domain are Version 5.0 servers, it is best to specify **CSI** as your protocol. If some servers are Version 3.x or Version 4.x servers, it is best to specify **CSI and SAS**. However, by specifying **CSI and SAS**, you now have two interceptors invoking each request. However, by specifying **CSI and SAS**, you now have two interceptors invoking each request.

| | |
|------------------|------------------|
| Data type | String |
| Default | CSI and SAS |
| Range | CSI, CSI and SAS |

Administrative console and naming service authorization

WebSphere Application Server extends the Java 2 Platform, Enterprise Edition (J2EE) security role-based access control to protect the product administrative and naming subsystems.

Administrative console

Four administrative roles are defined to provide degrees of authority needed to perform certain WebSphere Application Server administrative functions from either the administrative console or the system management scripting interface. The authorization policy is only enforced when global security is enabled. The four

administrative security roles are defined in the following table:

administrative roles

| Role | Description |
|---------------|---|
| monitor | Least privileged where a user can view the WebSphere Application Server configuration and current state. |
| configurator | Monitor privilege plus the ability to change the WebSphere Application Server configuration. |
| operator | Monitor privilege plus the ability to change the run-time state, such as starting or stopping services. |
| administrator | Operator plus configuration privilege and the permission required to access sensitive data including the server password, LTPA password, LTPA, keys, and so on. |

When WebSphere Application Server global security is enabled, the administrative subsystem role-based access control is enforced. The administrative subsystem includes security server, user registry, and all the Java Management Extensions (JMX) MBeans. When security is enabled, both the administrative console and the administrative scripting tool require users to provide the required authentication data. Moreover, the administrative console is designed so the control functions that display on the pages are adjusted according to the security roles that a user has. For example, a user who has only the monitor role can see only the non-sensitive configuration data. A user with the operator role can change the system state.

The server identity specified when enabling global security is automatically mapped to the administrative role. You can add or remove users and groups to or from the administrative roles from the WebSphere Application Server administrative console. However, a server restart is required for the changes to take effect. A best practice is to map a group, rather than specific users, to administrative roles because it is more flexible and easier to administer. By mapping a group to an administrative role, adding or removing users to or from the group occurs outside of WebSphere Application Server and does not require a server restart for the change to take effect.

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 has at least been authenticated. The Everyone special subject means that anyone, authenticated or not, can perform the action, as if security is not enabled.

When global security is enabled, WebSphere Application Servers run under the server identity that is defined under the active user registry configuration. Although it is not shown on the administrative console and in other tools, a special Server subject is mapped to the administrator role. The WebSphere Application Server run-time code, which runs under the server identity, requires authorization to run run-time operations. If no other user is assigned administrative roles, you can log into the administrative console or to the wsadmin scripting tool using the server identity to perform administrative operations and to assign other users or groups to administrative roles. Because the server identity is assigned to the

administrative role by default, the administrative security policy requires the administrative role to perform the following operations:

- Change server ID and server password
- Enable or disable WebSphere Application Server global security
- Enforce or disable Java 2 Security
- Change the LTPA password or generate keys
- Assign users and groups to administrative roles

When enabling security, you can assign one or more users and groups to administrative roles. For more information, see *Assigning users to naming roles*. However, before assigning users to naming roles, configure the active user registry. User and group validation depends on the active user registry. For more information, see *Configuring user registries*.

Naming service authorization

CosNaming security offers increased granularity of security control over CosNaming functions. CosNaming functions are available on CosNaming servers such as the WebSphere Application Server. They affect the content of the WebSphere Application Server name space. There are generally two ways in which client programs result in CosNaming calls. The first is through the JNDI interfaces. The second is with CORBA clients invoking CosNaming methods directly.

Four security roles are introduced :

- CosNamingRead
- CosNamingWrite
- CosNamingCreate
- CosNamingDelete

The names of the four roles are the same with WebSphere Application Server Advanced Edition Version 4.0.2. The roles now have authority levels from low to high:

CosNamingRead

Users can query of the WebSphere Application Server name space, using, for example, the JNDI lookup method. The special-subject Everyone is the default policy for this role.

CosNamingWrite

Users can perform write operations such as JNDI **bind**, **rebind**, or **unbind**, and CosNamingRead operations. The special-subject AllAuthenticated is the default policy for this role.

CosNamingCreate

Users can create new objects in the name space through such operations as JNDI createSubcontext and CosNamingWrite operations. The special subject AllAuthenticated is the default policy for this role.

CosNamingDelete

Users can destroy objects in the name space, for example using the JNDI destroySubcontext method and CosNamingCreate operations. The special-subject AllAuthenticated is the default policy for this role.

Additionally, a Server special-subject is assigned to all the four CosNaming roles by default. The Server special-subject provides a WebSphere Application Server server process, which runs under the server identity, access to all the CosNaming operations. Note that the Server special-subject does not display and cannot be modified through the administrative console or other administrative tools.

Users, groups, or the special subjects AllAuthenticated and Everyone can be added or removed to or from the naming roles from the WebSphere Application Server administrative console at any time. However, a server restart is required for the changes to take effect. A best practice is to map groups or one of the special-subjects, rather than specific users, to naming roles because it is more flexible and easier to administer in the long run. By mapping a group to a naming role, adding or removing users to or from the group occurs outside of WebSphere Application Server and does not require a server restart for the change to take effect.

The CosNaming authorization policy is only enforced when global security is enabled. When global security is enabled, attempts to do CosNaming operations without the proper role assignment result in an `org.omg.CORBA.NO_PERMISSION` exception from the CosNaming Server.

In WebSphere Application Server Version 4.0.2, each CosNaming function is assigned to only one role. Therefore, users who are assigned the CosNamingCreate role cannot query the name space unless they have also been assigned CosNamingRead. And in most cases a creator needs to be assigned three roles: CosNamingRead, CosNamingWrite, and CosNamingCreate. The CosNamingRead and CosNamingWrite roles assignment for the creator example are included in the CosNamingCreate role. In most of the cases, WebSphere Application Server administrators do not have to change the roles assignment for every user or group when they move to this release from a previous one.

Although the ability exists to greatly restrict access to the name space by changing the default policy, unexpected `org.omg.CORBA.NO_PERMISSION` exceptions can occur at run time. Typically, J2EE applications access the name space and the identity they use is that of the user that authenticated to WebSphere Application Server when they access the J2EE application. Unless the J2EE application provider clearly communicates the expected Naming roles, use caution when changing the default naming authorization policy.

Assigning users to administrator roles

The following steps are needed to assign users to administrative roles.

In the administrative console, expand the **System Administration** folder and click **Console Users** or **Console Groups**.

1. To add a user or a group, click **Add** on the **Console users** or **Console groups** panel.
2. To add a new administrative user, enter a user identity in the User field, highlight **Administrator**, and click **OK**. If there is no validation error, the specified user is displayed with the assigned security role.
3. To add a new administrative group, either enter a group name in the **Specify group** field or select **EVERYONE** or **ALL AUTHENTICATED** from the Select from special subject menu, and click **OK**. If no validation error exists, the specified group or special subject displays with the assigned security role.
4. To remove a user or group assignment, click **Remove** on the Console Users or the Console Groups panel. On the Console Users or the Console Groups panel, select the check box of the user or group to remove and click **OK**.
5. To manage the set of users or groups to display, expand the **filter** folder on the right panel and modify the filter. For example, setting the filter to `user*` only displays users with the user prefix.

6. After the modifications are complete, click **Save** to save the mappings.
7. Restart the server for changes to take effect.

The task of assigning users and groups to administrative roles is performed to identify users for performing WebSphere Application Server administrative functions. There are four roles: administrator, configurator, operator and monitor. Users and groups assigned to the administrator role can perform all administrative operations and can set up both J2EE role-based and Java 2 security policy. Users assigned to the configurator role can perform all day-to-day configuration tasks including installing and uninstalling applications, assigning users and groups to role mapping for applications, setting run-as configurations, setting up Java 2 security permissions for applications, and customizing Common Secure Interoperability Version 2 (CSIv2), Security Authentication Service (SAS), and Secure Sockets Layer (SSL) configurations. Users assigned to the operator role can view the WebSphere Application Server configuration and its current state, but also can change the run-time state such as stopping and starting services. Users assigned the monitor state can view the WebSphere Application server configuration and its current state only.

Before you assign users to administrative roles (administrator, configurator, operator, and monitor), you must set up your user registry, which can be LDAP, local OS, or a custom registry. You can set up your user registries without enabling security. Once you assign users to administrative roles, you must restart the server for the new roles to take effect. However, the administrative resources are not protected until you enable security.

Console users settings and CORBA naming service user settings

Use the Console users settings page to give users specific authority to administer WebSphere Application Server using tools such as the administrative console or wsadmin scripting. The authority requirements are only effective when global security is enabled. Use the common object request broker architecture (CORBA) naming service users settings page to manage CORBA naming service users settings.

To view the Console users administrative console page, click **System Administration > Console Users**.

To view the CORBA naming service users administrative console page, click **Environment > Naming > CORBA Naming Service users**.

User (Console users):

Specifies users.

The users entered must exist in the configured active user registry.

Data type: String

User (CORBA naming service users):

Specifies CORBA naming service users.

The users entered must exist in the configured active user registry.

Data type: String

Role (Console users):

Specifies user roles.

The following administrative roles provide different degrees of authority needed to perform certain WebSphere Application Server administrative functions:

Administrator

The administrator role has operator permissions, configurator permissions, and the permission required to access sensitive data including server password, Lightweight Third Party Authentication (LTPA) password and keys, and so on.

Configurator

The configurator role has monitor permissions and can change the WebSphere Application Server configuration.

Operator

The operator role has monitor permissions and can change the run-time state. For example, the operator can start or stop services.

Monitor

The monitor role has the least permissions. This role primarily confines the user to viewing the WebSphere Application Server configuration and current state.

| | |
|-------------------|--|
| Data type: | String |
| Range: | Administrator, Configurator, Operator, and Monitor |

Role (CORBA naming service users):

Specifies naming service user roles.

A number of naming roles are defined to provide degrees of authority needed to perform certain WebSphere naming service functions. The authorization policy is only enforced when global security is enabled. The following roles are valid: CosNamingRead, CosNamingWrite, CosNamingCreate, and CosNamingDelete.

The names of the four roles are the same with WebSphere Application Server, Advanced Edition Version 4.0.2. However, the roles now have authority levels from low to high:

CosNamingRead

Users can query the WebSphere name space using, for example, the Java Naming and Directory Interface (JNDI) lookup method. The special-subject EVERYONE is the default policy for this role.

CosNamingWrite

Users can perform write operations such as JNDI bind, rebind, or unbind, plus CosNamingRead operations. The special-subject ALL AUTHENTICATED is the default policy for this role.

CosNamingCreate

Users can create new objects in the name space through operations such as JNDI createSubcontext and CosNamingWrite operations. The special-subject ALL AUTHENTICATED is the default policy for this role.

CosNamingDelete

Users can destroy objects in the name space, for example using the JNDI destroySubcontext method and CosNamingCreate operations. The special-subject ALL AUTHENTICATED is the default policy for this role.

Data type: String
Range: CosNamingRead, CosNamingWrite, CosNamingCreate and CosNamingDelete

Console groups and CORBA naming service groups

Use the Console Groups page to give groups specific authority to administer the WebSphere Application Server using tools such as the administrative console or wsadmin scripting. The authority requirements are only effective when global security is enabled. Use the CORBA naming service groups page to manage CORBA Naming Service groups settings.

To view the Console Groups administrative console page, click **System Administration > Console Groups**.

To view the CORBA naming service groups administrative console page, click **Environment > Naming > CORBA Naming Service Groups**.

Group (Console groups):

Specifies groups.

The ALL_AUTHENTICATED and the EVERYONE groups can have the following role privileges: Administrator, Configurator, Operator, and Monitor.

Data type: String
Range: ALL_AUTHENTICATED, EVERYONE

Group (CORBA naming service groups):

Identifies CORBA naming service groups.

The ALL_AUTHENTICATED group has the following role privileges: CosNamingRead, CosNamingWrite, CosNamingCreate, and CosNamingDelete. The EVERYONE group indicates that the users in this group have CosNamingRead privileges only.

Data type: String
Range: ALL_AUTHENTICATED, EVERYONE

Role (Console group):

Specifies user roles.

The following administrative roles provide different degrees of authority needed to perform certain WebSphere Application Server administrative functions:

Administrator

The administrator role has operator permissions, configurator permissions, and the permission required to access sensitive data including server password, LTPA password and keys, and so on.

Configurator

The configurator role has monitor permissions and can change the WebSphere Application Server configuration.

Operator

The operator role has monitor permissions and can change the run-time state. For example, the operator can start or stop services.

Monitor

The monitor role has the least permissions. This role primarily confines the user to viewing the WebSphere Application Server configuration and current state.

| | |
|-------------------|--|
| Data type: | String |
| Range: | Administrator, Configurator, Operator, and Monitor |

Role (CORBA naming service groups):

Identifies naming service group roles.

A number of naming roles are defined to provide degrees of authority needed to perform certain WebSphere naming service functions. The authorization policy is only enforced when global security is enabled.

Four name space security roles are available: `CosNamingRead`, `CosNamingWrite`, `CosNamingCreate`, and `CosNamingDelete`. The names of the four roles are the same with WebSphere Advanced Edition, Version 4.0.2. However, the roles now have authority levels from low to high:

CosNamingRead

Users can query the WebSphere name space using, for example, the Java Naming and Directory Interface (JNDI) lookup method. The special-subject `EVERYONE` is the default policy for this role.

CosNamingWrite

Users can perform write operations such as JNDI bind, rebind, or unbind, and `CosNamingRead` operations. The special-subject `ALL_AUTHENTICATED` is the default policy for this role.

CosNamingCreate

Users can create new objects in the name space through operations such as JNDI `createSubcontext` and `CosNamingWrite` operations. The special-subject `ALL_AUTHENTICATED` is the default policy for this role.

CosNamingDelete

Users can destroy objects in the name space, for example using the JNDI `destroySubcontext` method and `CosNamingCreate` operations. The special-subject `ALL_AUTHENTICATED` is the default policy for this role.

| | |
|-------------------|--|
| Data type: | String |
| Range: | <code>CosNamingRead</code> , <code>CosNamingWrite</code> , <code>CosNamingCreate</code> , and <code>CosNamingDelete</code> |

Assigning users to naming roles

The following steps are needed to assign users to naming roles. In the administrative console, expand **Environment > Naming**, and click **CORBA Naming Service Users** or **CORBA Naming Service Groups**.

1. Click **Add** on the **CORBA Naming Service Users** or **CORBA Naming Service Groups** panel.
2. To add a new naming service user, enter a user identity in the **User** field, highlight one or more naming roles, and click **OK**. If no validation errors occur, the specified user is displayed with the assigned security role.

3. To add a new naming service group, either select **Specify group** and enter a group name or select **Select from special subject** and then select either **EVERYONE** or **ALL AUTHENTICATED**. Click **OK**. If no validation errors occur, the specified group or special subject is displayed with the assigned security role.
4. To remove a user or group assignment, go to the **CORBA Naming Service Users** or **CORBA Naming Service Groups** panel. Select the check box next to the user or group that you want to remove and click **Remove**.
5. To manage the set of users or groups to display, expand the **Filter** folder on the right panel, and modify the filter text box. For example, setting the filter to `user*` displays only users with the user prefix.
6. After modifications are complete, click **Save** to save the mappings. Restart the server for the changes to take effect.

The default naming security policy is to grant all users read access to the CosNaming space and to grant any valid user the privilege to modify the contents of the CosNaming space. You can perform the previously mentioned steps to restrict user access to the CosNaming space. However, use caution when changing the naming security policy. Unless a Java 2 Platform, Enterprise Edition (J2EE) application has clearly specified its naming space access requirements, changing the default policy can result in unexpected `org.omg.CORBA.NO_PERMISSION` exceptions at run time.

Authentication mechanisms

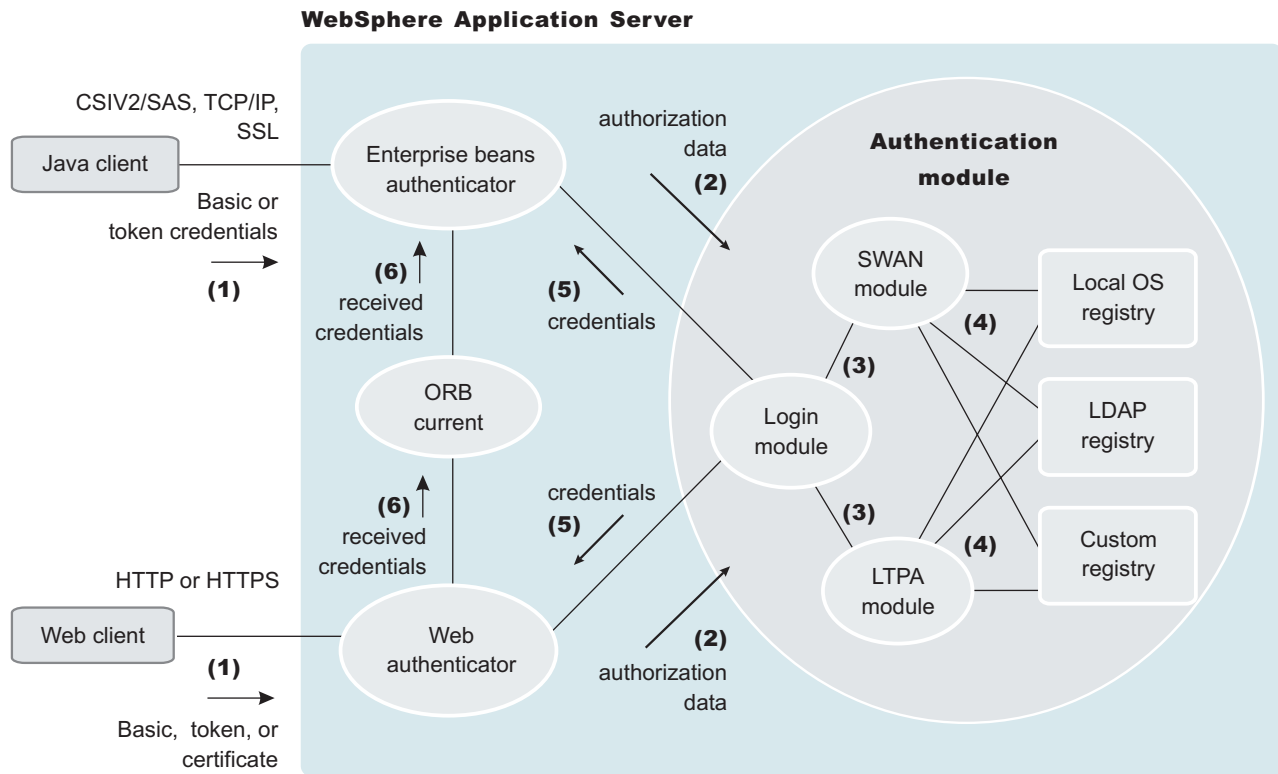
An *authentication mechanism* defines rules about security information (for example, whether a credential is forwardable to another Java process), and the format of how security information is stored in both credentials and tokens.

Authentication is the process of establishing whether a client is valid in a particular context. A client can be either an end user, a machine, or an application.

An authentication mechanism in WebSphere Application Server typically collaborates closely with a *user registry*. The user registry is the user and groups account repository that the authentication mechanism consults with when performing authentication. The authentication mechanism is responsible for creating a *credential*, which is an internal product representation of a successfully authenticated client user. Not all credentials are created equally. The abilities of the credential are determined by the configured authentication mechanism.

Although this product provides several authentication mechanisms, you can only configure a single *active* authentication mechanism at a time. The active authentication mechanism is selected when configuring WebSphere Application Server global security.

Authentication



Authentication Process

The figure demonstrates the authentication process. Basically, authentication is required for enterprise bean clients and Web clients when they access protected resources. Enterprise bean clients (a servlet or other enterprise beans or a pure client) send the authentication information to a Web application server using the Common Secure Interoperability Version 2 (CSIV2) or the Security Authentication Service (SAS) protocol. Web clients use the HTTP or HTTPS protocol to send the authentication information as shown in the previous figure. The authentication information can be BasicAuth (user ID and password), credential token (in case of Lightweight Third Party Authentication (LTPA)), or client certificate. The Web authentication is performed by the Web Authentication module and the enterprise bean authentication is performed by the Enterprise JavaBean (EJB) authentication module, which resides in the CSIV2 and SAS layer.

The authentication module is implemented using the Java Authentication and Authorization Service (JAAS) login module. The Web authenticator and the EJB authenticator pass the authentication data to the login module (2), which can be either Lightweight Third Party Authentication (LTPA) or Simple WebSphere Authentication Mechanism (SWAM).

The authentication module uses the registry that is configured on the system to perform the authentication (4). Three types of registries are supported: LocalOS, Lightweight Directory Access Protocol (LDAP), and custom registry. External registry implementation following the registry interface specified by IBM can replace either the LocalOS or the LDAP registry.

The login module creates a JAAS subject after authentication and stores the Common Object Request Broker Architecture (CORBA) credential derived from the

authentication data in the public credentials list of the subject. The credential is returned to the Web authenticator or EJB authenticator (5).

The Web authenticator and the EJB authenticator store the received credentials in the Object Request Broker (ORB) current for the authorization service to use in performing further access control checks.

WebSphere Application Server provides two authentication mechanisms: SWAM and LTPA. These authentication mechanisms differ primarily in the distributed security features that each supports.

Configuring authentication mechanisms

Configure authentication mechanisms by clicking **Authentication Mechanisms** under **Security** in the administrative console.

- If you are using Simple WebSphere Authentication Mechanism (SWAM), no setup is needed. Follow the instructions in *Configuring Lightweight Third Party Authentication* to set up Lightweight Third Party Authentication (LTPA).
- For LTPA, follow the steps in *Configure single signon (SSO)* for most situations. If trust association is required, follow the steps in *Configuring a trust association interceptor*.
- **5.1+** If you are using Tivoli Access Manager, follow the instructions in *Configuring WebSphere Application Server to use Tivoli Access Manager for authentication*.

Simple WebSphere authentication mechanism

The Simple WebSphere authentication mechanism (SWAM) is intended for simple, non-distributed, single application server run-time environments. The single application server restriction is due to the fact that SWAM does not support *forwardable* credentials. If a servlet or enterprise bean in application server process 1, invokes a remote method on an enterprise bean living in another application server process 2, the identity of the caller identity in process 1 is not transmitted to server process 2. What is transmitted is an unauthenticated credential, which, depending on the security permissions configured on the EJB methods, can cause authorization failures.

Since SWAM is intended for a single application server process, single signon (SSO) is not supported.

The SWAM authentication mechanism is suitable for simple environments, software development environments, or other environments that do not require a distributed security solution.

Lightweight Third Party Authentication

Lightweight Third Party Authentication (LTPA) is intended for distributed, multiple application server and machine environments. It supports forwardable credentials and single signon (SSO). LTPA can support security in a distributed environment through cryptography. This supports permits LTPA to encrypt, digitally sign, and securely transmit authentication-related data, and later decrypt and verify the signature.

The Lightweight Third Party Authentication (LTPA) protocol enables the WebSphere Application Server to provide security in a distributed environment using cryptography. Application servers distributed in multiple nodes and cells can securely communicate using this protocol. It also provides the single signon (SSO)

feature wherein a user is required to authenticate only once in a domain name system (DNS) domain and can access resources in other WebSphere Application Server cells without getting prompted. This protocol uses cryptographic keys (LTPA keys) to encrypt and decrypt user data that passes between the servers. These keys need to be shared between the different cells for the resources in one cell to access resources in other cells (assuming that all the cells involved use the same LDAP or custom registry).

When using LTPA, a token is created with the user information and an expiration time and is signed by the keys. The LTPA token is time sensitive. All product servers participating in a protection domain must have their time, date, and time zone synchronized. If not, LTPA tokens appear prematurely expired and cause authentication or validation failures. This token passes to other servers, in the same cell or in a different cell, either through cookies (for Web resources when SSO is enabled) or through the authentication layer (Security Authentication Service (SAS) or Common Secure Interoperability Version 2 (CSIV2) for enterprise beans). If the receiving servers share the same keys as the originating server, the token can be decrypted to obtain the user information, which then is validated to make sure it has not expired and the user information in the token is valid in its registry. On successful validation, the resources in the receiving servers are accessible after the authorization check.

All the WebSphere Application Server processes in a cell (cell, nodes, application servers) share the same set of keys. If key sharing is required between different cells, export them from one cell and import them to the other. For security purposes, the exported keys are encrypted with a user-defined password. This same password is needed when importing the keys into another cell.

In the base version of WebSphere Application Server, LTPA, ICSF, and the Simple WebSphere Authentication Mechanism (SWAM) protocols are supported. When security is enabled for the first time in WebSphere Application Server with LTPA, configuring LTPA is normally the initial step performed.

LTPA requires that the configured user registry be a centrally shared repository such as LDAP or a Windows domain type registry so that users and groups are the same regardless of the machine.

The following table summarizes the authentication mechanism capabilities and user registries with which LTPA can work.

| | Forwardable Credentials | SSO | LocalOS User Registry | LDAP User Registry | Custom User Registry |
|------|------------------------------------|------------|--------------------------------------|-------------------------------|-------------------------------------|
| SWAM | No | No | Yes | Yes | Yes |
| LTPA | Yes | Yes | Yes | Yes | Yes |
| ICSF | Yes | Yes | Yes | Yes | Yes |

Configuring Lightweight Third Party Authentication

The following steps are needed to configure Lightweight Third Party Authentication (LTPA) when setting up security for the first time:

1. Access the administrative console by typing `http://localhost:9090/admin` in a Web browser.

2. Click **Security > Authentication mechanisms > LTPA** in the Navigation panel on the left.
3. Enter the password and confirm it in the password fields. This password is used to encrypt and decrypt the LTPA keys during export and import of the keys. Remember this password because you enter it again when the keys from this cell are exported to another cell.
4. Enter a positive integer value in the **Timeout** field. This timeout value refers to how long an LTPA token is valid in minutes. The token contains this expiration time so that any server that receives the token can verify that the token is valid before proceeding further. When the token expires, the user is prompted to log in. An optimal value for this field depends on your configuration. The default value is 30 minutes.
5. Click **Apply** or **OK**. The LTPA configuration is now set. Do not generate the LTPA keys in this step because they are automatically generated later. Proceed with the rest of the steps required to enable security, starting with single signon (SSO) (if SSO is required).
6. Complete the information in the Global Security panel and click OK. The LTPA keys are generated automatically the first time. Do not generate the keys manually.

The previous steps configure LTPA by setting passwords that generate LTPA keys.

After configuring LTPA, complete the following steps to work with your key files:

1. Generate key files.
2. Export key files.
3. Import key files.
4. If you are enabling security, make sure that you complete the remaining steps starting with enabling single signon (SSO).
5. If you generated a new set of keys or imported a new set of keys, verify that the keys are saved by clicking **Save** at the top of the panel. Because LTPA authentication uses time sensitive tokens, verify that the time, date, and time zone are synchronized among all product servers that are participating in the protection domain. If the clock skew is too high between servers, the LTPA token appears prematurely expired and causes authentication or validation failures.

Configuring Lightweight Third Party Authentication keys:

Generating keys:

Lightweight Third Party Authentication (LTPA) keys are automatically generated when a password change is detected. The first time that you set the LTPA password, as part of enabling security, the LTPA keys are automatically generated after **OK** or **Apply** is clicked in the LTPA panel. You do not have to click **Generate Keys** in this situation. Complete the following steps in the administrative console to generate a new set of LTPA keys.

1. Access the administrative console by typing `http://localhost:9090/admin` in a Web browser.
2. Verify that all the WebSphere Application Server processes are running (cell, nodes, and all of the application servers). If any of the servers are down at the time of key generation and then brought back up later, these servers might contain old keys. Copy the new set of keys to these servers to bring them back up.

3. Click **Security > Authentication mechanisms > LTPA** in the navigation panel on the left.
4. Click **Generate Keys** if you want to use the existing password. This action generates a new set of keys that are encrypted with the same password as the old set of keys. Regardless of the password change, a new set of keys is generated when you click **Generate Keys**. This new set of keys is not propagated to the run time unless saved; save the files immediately.
5. Enter the new password and confirm it, to use a new password to generate keys. Click **OK** or **Apply**. A new set of keys is generated. A message indicating that a new set of keys is generated displays on the console. Do not click **Generate Keys**. These new keys are propagated to the run time once you save them.
6. Click **Save** to save the keys. After a new set of keys is generated and saved, the key propagation is dynamic. All of the processes running at that time (cells, node agents, application servers) are updated with the new set of keys. The next sections describe the process of exporting and importing the keys.

Exporting keys:

To support single signon (SSO) in WebSphere Application Server across multiple WebSphere Application Server domains or cells, share the LTPA keys and the password among the domains. Make sure that the time on the domains is similar to prevent the tokens from appearing as expired between the cells. You can use **Export Keys** to export the LTPA keys to other domains or cells. Complete the following steps in the administrative console to export key files for LTPA.

1. Access the administrative console by typing `http://localhost:9090/admin` in a Web browser.
2. Click **Security > Authentication mechanisms > LTPA** in the navigation panel on the left.
3. In the **Key File Name** field, enter the full path of a file for key storage. This file needs write permissions.
4. Click **Export Keys**. A file is created with the LTPA keys. Exporting keys fails if a new set of keys is generated or imported and not saved prior to exporting. To avoid failure, make sure that you save the new set of keys (if any) prior to exporting them.
5. Click **Save** to save the configuration.

Importing keys:

To support single signon (SSO) in WebSphere Application Server across multiple WebSphere Application Server domains or cells, share the LTPA keys and the password among the domains. You can use **Import Keys** to import the LTPA keys from other domains. Verify that key files are exported from one of the cells involved, into a file. Complete the following steps in the administrative console to import key files for LTPA.

Importing keys is a dynamic operation. All of the servers that are running at this time are updated with the new set of keys. Any back-level tokens signed with the back-level keys fail validation and the user is prompted to log in again.

1. Access the administrative console by typing `http://localhost:9090/admin` in a Web browser.
2. Click **Security > Authentication mechanisms > LTPA** in the navigation panel on the left.

3. Change the password in the **password** fields to match the password in the cell from which you are importing the keys.
4. Click **Save** to save the new set of keys in the repository. This step is important to complete before importing the keys. If the password and the keys do not match, the servers fail. If the servers fail, turn off security and redo these steps.
5. In the **Key File Name** field, enter the full path of a file for key storage. This file needs read permissions.
6. Click **Import Keys**. The keys are now imported into the system.
7. Click **Save** to save the new set of keys in the repository. It is important to save the new set of keys to match the new password so that no problems are encountered starting the servers later.

Lightweight Third Party Authentication settings:

Use this page to configure Lightweight Third Party Authentication (LTPA) settings.

To view this administrative console page, click **Security > Authentication Mechanisms > LTPA**.

If you are configuring security for the first time, only the password is required. After the password is entered, click **Apply**. Click **Single signon (SSO)** and enter the domain name. Make sure that SSO is enabled. Click **Apply**. To complete the security setup, make sure that the appropriate registry is set up and click **Apply** from the Global Security panel. When security is enabled and any of these properties change, go to the Global Security panel and click **Apply** to validate the changes.

Generate Keys:

Specifies whether the server generates new Lightweight Third Party Authentication (LTPA) keys.

When security is turned on for the first time with LTPA as the authentication mechanism, the LTPA keys are automatically generated with the password entered in the panel. If you need a new set of keys to generate using the previously set password, click **Generate Keys**. If a new password is used, do not click this option. After the new password is entered and **OK** or **Apply** is clicked, a new set of keys is generated. *A new set of generated keys is not used until you save them.*

Import Keys:

Specifies whether the server imports new LTPA keys.

To support single signon (SSO) in the WebSphere product across multiple WebSphere domains (cells), share the LTPA keys and the password among the domains. You can use the **Import Keys** option to import the LTPA keys from other domains. The LTPA keys are exported from one of the cells to a file. To import a new set of LTPA keys, enter the appropriate password, click **OK** and click **Save**. Then, enter the directory location where the LTPA keys are located prior to clicking **Import keys**. Do not click **OK** or **Apply**, but save the settings.

Export Keys:

Specifies whether the server exports LTPA keys.

To support single signon (SSO) in the WebSphere product across multiple WebSphere Application Server domains (cells), share the LTPA keys and the password among the domains. Use the **Export Keys** option to export the LTPA keys to other domains.

To export the LTPA keys, make sure that the system is running with security enabled and is using LTPA. Enter the file name in the **Key File Name** field and click **Export Keys**. The encrypted keys are stored in the specified file.

Password:

Specifies the password to encrypt and decrypt the LTPA keys. Use this password when importing these keys into other WebSphere Application Server administrative domain configurations (if any) and when configuring SSO for a Lotus Domino server.

After the keys are generated or imported, they are used to encrypt and decrypt the LTPA token. Whenever the password is changed, a new set of LTPA keys are automatically generated when you click **OK** or **Apply**. This new set of keys is used only when you save.

Data type String

Confirm Password:

Specifies the confirmed password used to encrypt and decrypt the LTPA keys.

Use this password when importing these keys into other WebSphere Application Server administrative domain configurations (if any) and when configuring SSO for a Lotus Domino server.

Data type String

Timeout:

Specifies the time period in minutes at which an LTPA token expires. Verify that this time period is longer than the cache timeout configured in the Global Security panel.

Data type Integer
Units Minutes
Default 120

Key File Name:

Specifies the name of the file used when importing or exporting keys.

Enter a fully qualified key file name, and click **Import Keys** or **Export Keys**.

Data type String

Trust Associations

Trust Association enables the integration of IBM WebSphere Application Server security and third-party security servers. More specifically, a reverse proxy server

can act as a front-end authentication server while the product applies its own authorization policy onto the resulting credentials passed by the proxy server. Demand for such an integrated configuration has become more compelling, especially when a single product cannot meet all of the customer needs or when migration is not a viable solution. This article provides a conceptual background behind the approach.

The demand is growing to provide customers with a trust association solution between IBM WebSphere Application Server and other Web authentication servers that act as a reverse proxy security server (IBM Tivoli Access Manager for e-business - WebSEAL, Caching Proxy) as an entry point to all service requests (See the first figure). This implementation design intends to have the proxy server as the only exposed entry point. The proxy server authenticates all requests that come in and provides coarse, granularity junction point authorization.

In this setup, the WebSphere Application Server is used as a back-end server to further exploit its fine-grained access control. The reverse proxy server passes the HTTP request to the WebSphere Application Server that includes the credentials of the authenticated user. WebSphere Application Server then uses these credentials to authorize the request.

Trust association model

The idea that WebSphere Application Server can support trust association implies that the product application security recognizes and processes HTTP requests received from a reverse proxy server. WebSphere Application Server and the proxy server engage in a contract in which the product gives its full trust to the proxy server and the proxy server applies its authentication policies on every Web request that is dispatched to WebSphere Application Server. This trust is validated by the interceptors that reside in the product environment for every request received. The method of validation is agreed upon by the proxy server and the interceptor.

Running in trust association mode does not prohibit WebSphere Application Server from accepting requests that did not pass through the proxy server. In this case, no interceptor is needed for validating trust. It is possible, however, to configure WebSphere Application Server to strictly require that all HTTP requests go through a reverse proxy server. In this case, all requests that do not come from a proxy server are immediately denied by WebSphere Application Server.

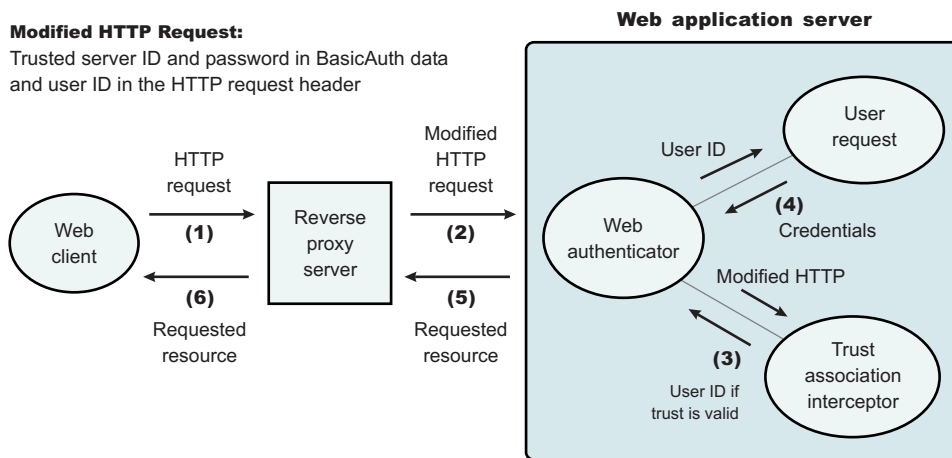
Trust association model

HTTP Request:

User ID and password in BasicAuth data

Modified HTTP Request:

Trusted server ID and password in BasicAuth data and user ID in the HTTP request header



IBM WebSphere Application Server--WebSEAL Integration

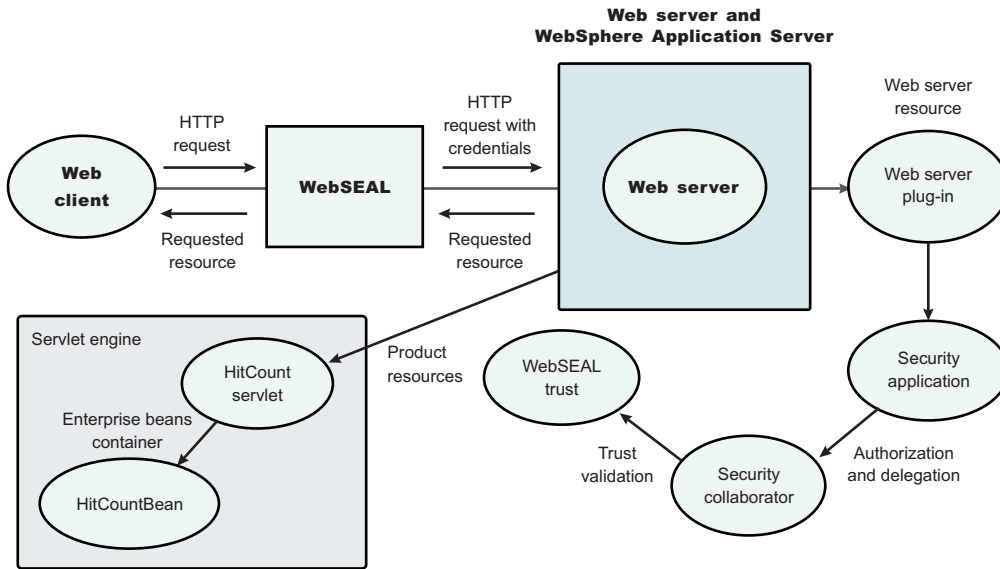
The integration of WebSEAL and WebSphere Application Server security is achieved by placing the WebSEAL server at the front-end as a reverse proxy server. See Figure 2. From a WebSEAL management perspective, a junction is created with WebSEAL on one end, and the product Web server on the other end. A junction is a logical connection created to establish a path from the WebSEAL server to another server.

In this setup, a request for Web resources stored in a protected domain of the product is submitted to the WebSEAL server where it is authenticated against the WebSEAL security realm. If the requesting user has access to the junction, the request is transmitted to the WebSphere Application Server HTTP server through the junction, and then to the application server.

Meanwhile, the WebSphere Application Server validates every request that comes through the junction to ensure that the source is a trusted party. This process is referenced as *validating the trust* and it is performed by a WebSEAL product-designated interceptor. If the validation is successful, the WebSphere Application Server authorizes the request by checking whether the client user has the required permissions to access the Web resource. If so, the Web resource is delivered to the WebSEAL server, through the Web server, which then gives it to the client user.

WebSEAL server

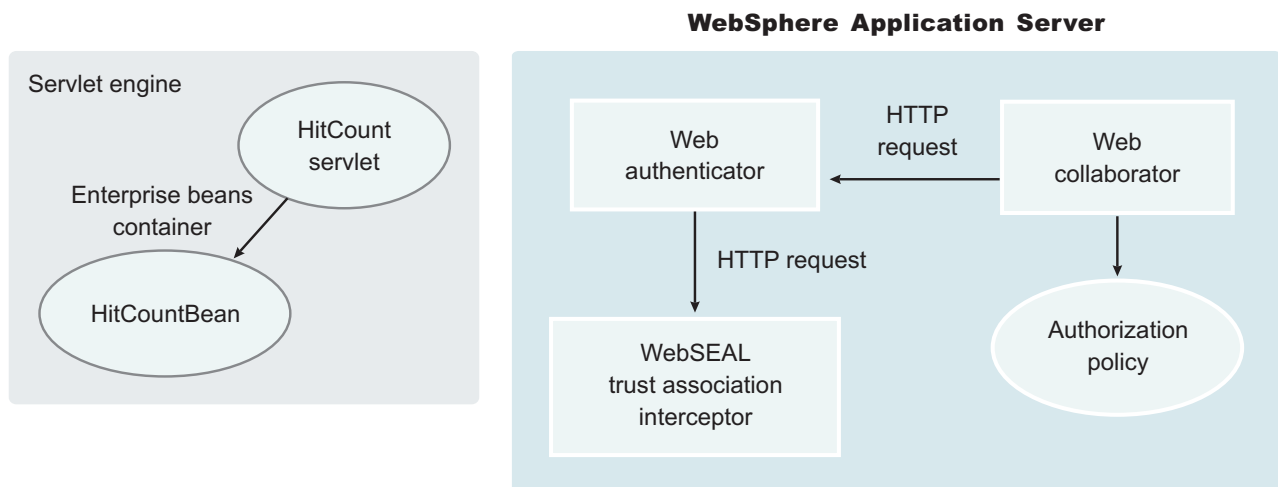
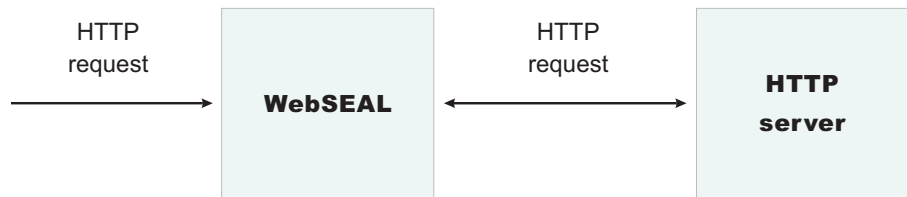
The policy director delegates all of the Web requests to its Web component, the WebSEAL server. One of the major functions of the server is to perform authentication of the requesting user. The WebSEAL server consults a Lightweight Directory Access Protocol (LDAP) directory. It can also map the original user ID to another user ID, such as when global single signon (GSO) is used.



For successful authentication, the server plays the role of a client to WebSphere Application Server when channeling the request. The server needs its own user ID and password to identify itself to WebSphere Application Server. This identity must be valid in the security realm of WebSphere Application Server. The WebSEAL server replaces the basic authentication information in the HTTP request with its own user ID and password. In addition, WebSphere Application Server needs to know the user ID of the requesting client so that it can base its authorization decision from this user ID, and not from a WebSEAL user ID. This information is transmitted through the HTTP request, by creating a header called `iv-user` with the client user ID as its value.

HTTP server

The junction created in the WebSEAL server must get to the HTTP server that serves as the product front end. However, the HTTP server is shielded from knowing that trust association is used. As far as it is concerned, the WebSEAL product is just another HTTP client, and as part of its normal routines, it sends the HTTP request to the product. The only requirement on the HTTP server is a Secure Sockets Layer (SSL) configuration using server authentication only. This requirement protects the requests that flow within the junction.



Web collaborator

When trust association is enabled, the Web collaborator manages the interceptors that are configured in the system. It loads and initializes these interceptors when you restart your servers. When a request is passed to WebSphere Application Server by the Web server, the Web collaborator eventually receives the request for a security check. Two actions must take place:

1. The request must be authenticated.
2. The request must be authorized.

The Web authenticator is called to authenticate the request by passing the HTTP request. If successful, a good credential record is returned by the authenticator, which the Web collaborator uses to base its authorization for the requested resource. If the authorization succeeds, the Web collaborator indicates to WebSphere Application Server that the security check has succeeded and that the requested resource can be served.

Web authenticator

The Web authenticator is asked by the Web collaborator to authenticate a given HTTP request. Knowing that trust association is enabled, the task of the Web authenticator is to find the appropriate trust association interceptor to direct the request for processing. The Web authenticator queries every available interceptor. If no target interceptor is found, the Web authenticator processes the request as though trust association is not enabled.

For an HTTP request sent by the WebSEAL server, the WebSEAL trust association interceptor replies with a positive response to the Web authenticator. Subsequently, the interceptor is asked to validate its trust association with the WebSEAL server and retrieve the user ID of the original user client.

Trust association interceptor feature

The intent of the trust association interceptor feature is to have reverse proxy security servers (RPSS) exist as the exposed entry points to perform authentication and coarse-grained authorization, while the WebSphere Application Server enforces further fine-grained access control. Trust associations improve security by reducing the scope and risk of exposure.

In a typical e-business infrastructure, the distributed environment of a company consists of Web application servers, Web servers, legacy systems, and one or more RPSS, such as the Tivoli WebSEAL product. Such reverse proxy servers, front-end security servers, or security plug-ins registered within Web servers, guard the HTTP access requests to the Web servers and the Web application servers. While protecting access to the Uniform Resource Identifiers (URIs), these RPSS perform authentication, coarse-grained authorization, and request routing to the target application server.

Using the trust association interceptor feature

The following points further describe the benefits of the trust association interceptor (TAI) feature:

- RPSS can authenticate WebSphere Application Server users up front and send credential information about the authenticated user to the product so that the product can trust the RPSS to perform authentication and not prompt the end user for authentication data later. The strength of the trust relationship between RPSS and the product is based on the criteria of trust association that is particular to a RPSS and enforced through the TAI implementation. This level of trust might need relaxing based on the environment. Be aware of the vulnerabilities in cases where the RPSS is not trusted, based on a security technology.
- The end user credentials most likely are sent in a special format as part of the Hypertext Transfer Protocol (HTTP) headers as in the case of RPSS authentication. The credentials can be a special header or a cookie. The data that passes is implementation specific, and the TAI feature considers this fact and accommodates the idea. The TAI implementation works with the credential data and returns a string that represents the end user that WebSphere Application Server uses to enforce security policies.

Configuring trust association interceptors

These steps are required to use either a WebSEAL trust association interceptor or your own trust association interceptor with a reverse proxy security server.

1. Access the administrative console by typing `http://localhost:9090/admin` in a Web browser.
2. Click **Security > Authentication mechanisms > LTPA** in the left navigation panel.
3. Click **Trust Association** under Additional Properties.
4. Select the **Trust Association Enabled** check box.
5. Click **Interceptors** under Additional Properties. The default value appears.
6. Click **com.ibm.ws.security.web.WebSealTrustAssociationInterceptor** if you are using the WebSEAL interceptor. This interceptor is the default value. To use a different interceptor, complete the following steps:
 - a. Click **New**.
 - b. Type the name of the interceptor into the **Interceptor Classname** field.

- c. Click **OK**.
- d. Click the name of the new interceptor.
7. Click **Custom Properties** under Additional Properties.
8. Click **New** to enter the property name and value pairs. The name and value pairs for the WebSEAL server to follow. For a new interceptor, enter the name and value pairs that correspond to your interceptor.
 - com.ibm.websphere.security.trustassociation.types**
WebSEAL
 - com.ibm.websphere.security.webseal.loginId**
This property contains the ID of the WebSEAL server.
 - com.ibm.websphere.security.webseal.id**
iv-user is a special header field that is sent by the WebSEAL server with the request to the WebSphere Application Server.
 - com.ibm.websphere.security.webseal.hostnames**
The host names (case sensitive) that are expected in the request header (the VIA header). This request header also includes the proxy host names (if any) unless the com.ibm.websphere.security.webseal.ignoreProxy interceptor is set to true.
 - com.ibm.websphere.security.webseal.ports**
The corresponding port number of the host names that are expected in the request header (the VIA header). This request header also includes the proxy ports (if any) unless the com.ibm.websphere.security.webseal.ignoreProxy interceptor is set to true.
 - com.ibm.websphere.security.webseal.ignoreProxy**
An optional property that if set to true or yes ignores the proxy host names and ports in the VIA header. By default, this property is set to false.
9. Click **OK**.

Enables trust association.

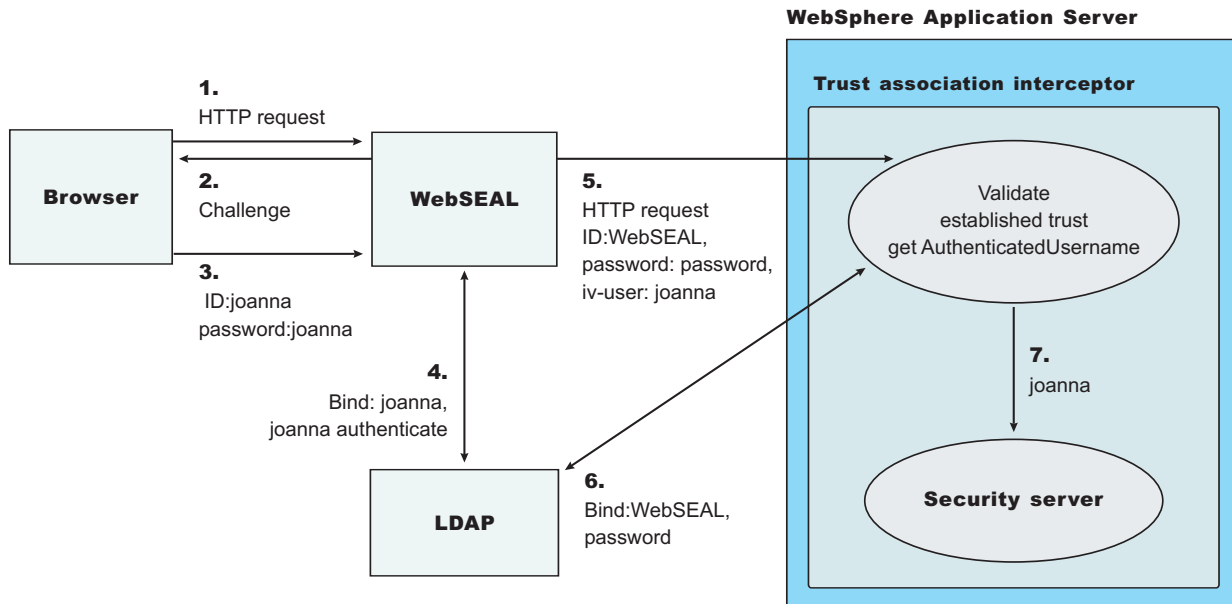
A typical scenario using the trust association interceptor (TAI) includes an environment where IBM Tivoli WebSEAL product is deployed and used with WebSphere Application Server. For the WebSEAL product, an implementation of the TAI is already provided with the product. The following steps outline the typical flow of an HTTP request for a secured WebSphere Application Server resource authenticated by the WebSEAL server through a Web trust association:

1. The browser makes a request for a secured WebSphere resource.
2. The WebSEAL server sends back a challenge, either an HTTP basic authentication or a form-based challenge.
3. A user name and password are supplied.
4. The WebSEAL product authenticates the user to Lightweight Directory Access Protocol (LDAP).
5. The modified request is forwarded by the WebSEAL product to the WebSphere Application Server.
6. The plug-in (TAI) establishes that the WebSphere Application Server trusts the WebSEAL server by using the validateEstablishedTrust method.
7. The plug-in extracts the end-user name from the iv-user header field and passes it to the WebSphere Application Server for authorization.

Note: WebSEAL Version 3.9 and later do not send the user ID and password to the server. Trust is based on a mutual Secure Sockets Layer (SSL) established

between the WebSEAL server and the WebSphere Application Server. Steps 5 and 6 do not apply to WebSEAL Version 3.9 and later.

Web trust association authentication flow



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 (cell, nodes, and all of the application servers) for the changes to take effect.

Trust association settings:

Trust association enables the integration of IBM WebSphere Application Server security and third-party security servers. More specifically, a reverse proxy server can act as a front-end authentication server while the product applies its own authorization policy onto the resulting credentials passed by the proxy server. Use this page to configure trust association settings.

To view this administrative console page, click **Security Center > Authentication Mechanisms > LTPA > Trust Association**.

When security is enabled and any of these properties change, go to the **Global Security** panel and click **Apply** to validate the changes.

Enabled:

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, click **Security > Authentication Mechanisms > LTPA > Trust Association > Interceptors**.

When security is enabled and any of these properties are changed, go to the Global Security panel and click **Apply** to validate the changes.

Interceptor Class Name:

Specifies the trust association interceptor class name.

Data type

String

Default

`com.ibm.ws.security.web.WebSealTrustAssociationInterceptor`

Single Signon

With single signon (SSO) support, Web users can authenticate once when accessing both WebSphere Application Server resources, such as HTML, JavaServer page (JSP) files, servlets, enterprise beans, and Lotus Domino resources, such as documents in a Domino database, or accessing resources in multiple WebSphere domains.

Web users can authenticate once to a WebSphere Application Server or to a Domino server. Without logging in again, Web users can access any other WebSphere Application Servers or Domino servers in the same Domain Name Service (DNS) domain that are enabled for SSO. This authentication is accomplished by configuring the WebSphere Application Servers and the Domino servers to share authentication information.

Enable SSO among WebSphere Application Servers by configuring SSO for WebSphere Application Server. To enable SSO between WebSphere Application Servers and Domino servers, you must configure SSO for both WebSphere Application Server and for Domino.

Prerequisites and conditions

To take advantage of support for single signon between WebSphere Application Servers or between WebSphere Application Server and a Domino server, applications must meet the following prerequisites and conditions:

- Verify that all servers are configured as part of the same DNS domain. For example, if the DNS domain is specified as `mycompany.com`, then SSO is effective with any Domino server or WebSphere Application Server on a host that is part of the `mycompany.com` domain, for example, `a.mycompany.com` and `b.mycompany.com`.
- Verify that all servers share the same user registry. This registry can be either a supported Lightweight Directory Access Protocol (LDAP) directory server or, if SSO is configured between two WebSphere Application Servers, a custom user registry. Domino servers do not support custom registries, but you can use a Domino-supported registry as a custom registry within WebSphere Application Server. For more information on custom registries, see *Introduction to custom registries*.

You can use a Domino directory (configured for LDAP access) or other LDAP directory for the user registry. The LDAP directory product must have WebSphere Application Server support. Supported products include both

Domino and IBM SecureWay LDAP directory servers. Regardless of the choice to use an LDAP or a custom registry, the SSO configuration is the same. The difference is in the configuration of the registry.

- Define all users in a single LDAP directory. Using LDAP referrals to connect more than one directory together is not supported. Using multiple Domino directory assistance documents to access multiple directories also is not supported.
- Enable HTTP cookies in browsers because the authentication information that is generated by the server is transported to the browser in a cookie. The cookie is then used to propagate the user's authentication information to other servers, exempting the user from entering the authentication information for every request to a different server.
- For a Domino server:
 - Domino Release 5.0.6a for iSeries 400 or later and Domino Release 5.0.5 or later for other platforms are supported.
 - A Lotus Notes client Release 5.0.5 or later is required for configuring the Domino server for SSO.
 - You can share authentication information across multiple Domino domains.
- For WebSphere Application Server:
 - WebSphere Application Server Version 3.5 or later for all platforms is supported.
 - You can use any HTTP Web server supported by WebSphere Application Server.
 - You can share authentication information across multiple product administrative domains.
 - Basic authentication (user ID and password) using the basic and form-login mechanisms is supported.
 - By default, WebSphere Application Server does a case-sensitive comparison for authorization. This comparison implies that a user who is authenticated by Domino matches the entry exactly (including the base distinguished name) in the WebSphere Application Server authorization table. If case sensitivity is not considered for the authorization, enable the **Ignore Case** property in the LDAP user registry settings.

Configuring single signon

With single signon (SSO) support, Web users can authenticate once when accessing Web resources across multiple WebSphere Application Servers. Form login mechanisms for Web applications require that SSO is enabled. SSO is supported only when Lightweight Third Party Authentication (LTPA) is the authentication mechanism. SSO uses HTTP cookies to achieve this functionality.

When SSO is enabled, a cookie is created within the LTPA token. 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 LTPA authentication mechanism requires that you enable SSO if any of the Web applications have form login as the authentication method.

The following steps are required to configure SSO for the first time.

1. Access the administrative console by typing `http://localhost:9090/admin` in a Web browser.
2. Click **Security > Authentication mechanisms > LTPA** in the Navigation panel on the left. Click **Single Signon (SSO)** in the Additional Properties section.

3. Click **Enable** if SSO is disabled. After you click **Enable**, make sure you complete the remaining steps to enable security.
4. Enable the **Requires SSL** field if all of the requests are expected to come over HTTPS.
5. Enter the domain name where SSO is effective. The cookie is sent for all of the servers in this domain only. For example, if the domain is `ibm.com`, SSO works between the domains `austin.ibm.com`, `raleigh.ibm.com` and not `austin.otherCompany.com`.

Domain field option: The domain field is optional, and, if left blank, the Web browser defaults to the domain name of the SSO cookie to the WebSphere Application Server that created it. In this case, SSO is only valid for the server that created the cookie. This behavior might be desirable when multiple virtual hosts are defined and need to have a separate domain specified in the SSO cookie.

6. Click **OK**.

For the changes to take effect, save, stop, and restart all the product servers (cell, nodes and all the WebSphere Application Server systems).

Single signon settings:

Use this page to set the configuration values for single signon (SSO).

To view this administrative console page, click **Security > Authentication Mechanisms > LTPA > Single Signon (SSO)**.

Requires SSL:

Specifies that the single signon function is enabled only when requests are made over HTTPS Secure Sockets Layer (SSL) connections.

| | |
|-------------------|-------------------|
| Data type: | Boolean |
| Default: | Disable |
| Range: | Enable or Disable |

Domain Name:

Specifies the domain name (`.ibm.com`, for example) for all single signon hosts.

If no value is specified, the Web browser of the user defaults to the value of the host name where the Web application is running. This default restricts the HTTP cookie (generated for SSO purposes) only to the originating host. Restricting the HTTP cookie can be undesirable if there is more than one host participating in the SSO domain. Leaving the domain name attribute empty is only desirable if multiple virtual hosts with different domain names are running on the same physical host. With this field empty, your Web browser can default the domain name to each different virtual host. If a domain name is explicitly specified in this field, then that value is used for all of the virtual hosts and restricts them to a single domain, which can be undesirable in some situations.

If a domain name is explicitly specified, then all of the Web pages used to access protected Web resources contain the server domain name service (DNS) host name. For example, after global security is configured for LTPA and an explicit SSO domain name is specified, then the administrative console is accessible with the following Web address: `http://yourhost.austin.ibm.com:9090/admin`, where

yourhost.austin.ibm.com is replaced with your server DNS host name.

Data type: String

Enabled:

Specifies that the single signon function is enabled.

Web applications that use J2EE FormLogin style login pages (such as the WebSphere Application Server administrative console) require single signon (SSO) enablement. Only disable SSO for certain advanced configurations where LTPA SSO-type cookies are not required.

Data type: Boolean
Default: Enabled
Range: Enabled or Disabled

Troubleshooting single signon configurations:

This article describes common problems in configuring single signon (SSO) between a WebSphere Application Server and a Domino server and suggests possible solutions.

- Failure to save the Domino Web SSO configuration document

The client must be able to find Domino server documents for the participating SSO Domino servers. The Web SSO configuration document is encrypted for the servers that you specify, so the home server 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 were used to encrypt the document by finding the listed servers, authors, and administrators on the document.

- Failure of the Domino server console to load the Web SSO configuration document at Domino HTTP server startup

During configuration of SSO, the server document is configured for **Multi-Server** in the **Session Authentication** field. The Domino HTTP server tries to find and load a Web SSO configuration document during startup. The Domino server console reports the following information if a valid document is found and decrypted: HTTP: Successfully loaded Web SSO Configuration.

If a server cannot load the Web SSO configuration document, SSO does not work. In this case, a server reports the following message: HTTP: Error Loading Web SSO configuration. Reverting to single-server session authentication.

Verify that only one Web SSO Configuration document is in the Web Configurations view of the Domino directory and in the \$WebSSOConfigs hidden view. You cannot create more than one document, but you can insert additional documents during replication.

If you can verify only one Web SSO Configuration document, consider another condition. When the public key of the Server document does not match the

public key in the ID file, this same error message can display. In this case, attempts to decrypt the Web SSO configuration document fail and the error message is generated.

This situation can occur when the ID file is created multiple times but the Server document is not updated correctly. Usually, an error message is displayed on the Domino server console stating that the public key does not match the server ID. If this situation occurs, then SSO does not work because the document is encrypted with a public key for which the server does not possess the corresponding private key.

To correct a key-mismatch problem:

1. Copy the public key from the server ID file and paste it into the Server document.
 2. Create the Web SSO configuration document again.
- Authentication fails when accessing a protected resource.

If a Web user is repeatedly prompted for a user ID and password, SSO is not working because either the Domino or the WebSphere Application Server security server cannot authenticate the user with the Lightweight Directory Access Protocol (LDAP) server. Check the following possibilities:

- Verify that the LDAP server is accessible from the Domino server machine. Use the **TCP/IP ping** utility to check TCP/IP connectivity and to verify that the host machine is running.
- Verify that the LDAP user is defined in the LDAP directory. Use the **ldapsearch** utility to confirm that the user ID exists and that the password is correct. For example, you can run the following command, entered as a single line, from the OS/400 Qshell, a UNIX shell, or a Windows DOS prompt:

```
% ldapsearch -D "cn=John Doe, ou=Rochester, o=IBM, c=US" -w mypassword  
-h myhost.mycompany.com -p 389  
-b "ou=Rochester, o=IBM, c=US" (objectclass=*)
```

(The percent character (%) indicates the prompt and is not part of the command.) A list of directory entries is expected. Possible error conditions and causes are contained in the following list:

- No such object: This error indicates that the directory entry referenced by either the user's distinguished name (DN) value, which is specified after the -D option, or the base DN value, which is specified after the -b option, does not exist.
- Invalid credentials: This error indicates that the password is invalid.
- Cannot contact the LDAP server: This error indicates that the host name or port specified for the server is invalid or that the LDAP server is not running.
- An empty list means that the base directory specified by the -b option does not contain any directory entries.
- If you are using the user's short name (or user ID) instead of the distinguished name, verify that the directory entry is configured with the short name. For a Domino directory, verify the **Short name/UserID** field of the Person document. For other LDAP directories, verify the userid property of the directory entry.
- If Domino authentication fails when using an LDAP directory other than a Domino directory, verify the configuration settings of the LDAP server in the Directory assistance document in the Directory assistance database. Also verify that the Server document refers to the correct Directory assistance document. The following LDAP values specified in the Directory Assistance document must match the values specified for the user registry in the WebSphere administrative domain:
 - Domain name

- LDAP host name
- LDAP port
- Base DN

Additionally, the rules defined in the Directory assistance document must refer to the base distinguished name (DN) of the directory containing the directory entries of the users.

You can trace Domino server requests to the LDAP server by adding the following line to the server notes.ini file:

```
webauth_verbose_trace=1
```

After restarting the Domino server, trace messages are displayed in the Domino server console as Web users attempt to authenticate to the Domino server.

- Authorization failure when accessing a protected resource.

After authenticating successfully, if an authorization error message is displayed, security is not configured correctly. Check the following possibilities:

- For Domino databases, verify that the user is defined in the access-control settings for the database. Refer to the Domino Administrative documentation for the correct way to specify the user's DN. For example, for the DN cn=John Doe, ou=Rochester, o=IBM, c=US, the value on the access-control list must be set as John Doe/Rochester/IBM/US.
- For resources protected by WebSphere Application Server, verify that the security permissions are set correctly.
 - If granting permissions to selected groups, make sure that the user attempting to access the resource is a member of the group. For example, you can verify the members of the groups by using the following Web site to display the directory contents:
Ldap://myhost.mycompany.com:389/ou=Rochester, o=IBM, c=US??sub
 - If you have changed the LDAP configuration information (host, port, and base DN) in a WebSphere Application Server administrative domain since the permissions were set, the existing permissions are probably invalid and need to be recreated.

- SSO failure when accessing protected resources.

If a Web user is prompted to authenticate with each resource, SSO is not configured correctly. Check the following possibilities:

1. Configure both the WebSphere Application Server and the Domino server to use the same LDAP directory. The HTTP cookie used for SSO stores the full DN of the user, for example, cn=John Doe, ou=Rochester, o=IBM, c=US, and the domain name service (DNS) domain.
2. Define Web users by hierarchical names if the Domino Directory is used. For example, update the **User name** field in the Person document to include names of this format as the first value: John Doe/Rochester/IBM/US.
3. Specify the full DNS server name, not just the host name or TCP/IP address for Web sites issued to Domino servers and WebSphere Application Servers configured for SSO. For browsers to send cookies to a group of servers, the DNS domain must be included in the cookie, and the DNS domain in the cookie must match the Web address. (This requirement is why you cannot use cookies across TCP/IP domains.)
4. Configure both Domino and the WebSphere Application Server to use the same DNS domain. Verify that the DNS domain value is exactly the same, including capitalization. The DNS domain value is found on the Configure Global Security Settings panel of the WebSphere Application Server administrative console and in the Web SSO Configuration document of a

Domino server. If you make a change to the Domino Web SSO Configuration document, replicate the modified document to all of the Domino servers participating in SSO.

5. Verify that the clustered Domino servers have the host name populated with the full DNS server name in the Server document. By using the full DNS server name, Domino Internet Cluster Manager (ICM) can redirect to cluster members using SSO. If this field is not populated, by default, ICM redirects Web addresses to clustered Web servers by using the host name of the server only. It cannot send the SSO cookie because the DNS domain is not included in the Web address. To correct the problem:
 - a. Edit the Server document.
 - b. Click **Internet Protocols > HTTP** tab.
 - c. Enter the full DNS name of the server in the **Host names** field.
6. If a port value for an LDAP server was specified for a WebSphere Application Server administrative domain, edit the Domino Web SSO configuration document and insert a backslash character (\) into the value of the **LDAP Realm** field before the colon character (:). For example, replace `myhost.mycompany.com:389` with `myhost.mycompany.com\ :389`.

Configuring WebSphere Application Server to use Tivoli Access Manager for authentication

Use WebSphere Application Server Version 5.1 to install and pre-configure the Tivoli Access Manager Java run-time component, which uses the WebSphere Application Server version of the Java run time during the installation. This article does not provide support for the HP-UX operating system, and the steps assume that an external Tivoli Access Manager server Version 5.1 already exists. Refer to the IBM Tivoli Access Manager for e-business documentation for more information, including the IBM Tivoli Access Manager for WebSphere Application Server documentation.

To enable and disable Tivoli Access Manager for authentication, complete the following steps:

1. To enable and disable Tivoli Access Manager for authentication, complete the following steps:
 - a. Issue the `PDJrteCfg` command. In a cell, run the following command on the Deployment Manager first and then on the nodes.

Attention: The first two lines of the following code sample are one continuous line. The line was split to fit within the width of the printed page.

```
java -Djava.ext.dirs -Dpd.home="%WAS_HOME%\java\jre\PolicyDirector"  
com.tivoli.pd.jcfg.PDJrteCfg
```

```
-action [ config | unconfig ]  
-was  
-config_type [ full | standalone ]  
-java_home <jre_home>  
-host <Policy_Server_host_name>
```

Detailed information on the `PDJrteCfg` class is located in the `com.tivoli.pd.jcfg.PDJrteCfg` in the Command Reference of the Tivoli Access Manager Version 5.1 product documentation. Here is an example of the configuration script to run.

Attention: The following code example should be written as one line of code.

```
%WAS_HOME%\java\jre\bin\java -Djava.ext.dirs
-Dpd.home="%WAS_HOME%\java\jre\PolicyDirector" \
-cp "%WAS_HOME%\java\jre\lib\ext\PD.jar;
%WAS_HOME%\java\jre\lib\ext\ibmjceprovider.jar;
%WAS_HOME%\java\jre\lib\ext\ibmpkcs.jar;
%CLASSPATH%" \com.tivoli.pd.jcfg.PDJrteCfg -action config -was
-config_type full -host TAM_policy_server_host_name
```

- b. Issue the SvrSslCfg command. In a cell, run the following command on the Deployment Manager first and then on the nodes. Refer to SvrSslCfg usage syntax for more information. Also, see the following example for issuing this command:

Note: The following code example should be written as one line of code.

```
%WAS_HOME%\java\jre\bin\java com.tivoli.pd.jcfg.SvrSslCfg
-action config -admin_id sec_master -admin_pwd password
-appsvr_id appsvr_name -appsvr_pwd security -port 8888
-mode remote -host was_server_host_name
-policysvr TAM_policysvr_host_name:7135:1
-authzsvr TAM_authzsvr_host_name:7136:1
-cfg_file %WAS_HOME%\java\jre\PdPerm.properties
-key_file %WAS_HOME%\java\jre\lib\security\PdPerm.ks
-cfg_action create
```

- c. Start WebSphere Application Server, if not started already.
 - d. Enable Tivoli Access Manager in the WebSphere Application Server administrative console. Check the **Use Tivoli Access Manager for Account Policies** check box on the **Security > User Registries > LDAP** page. If security within the LDAP user registry is not already enabled, then refer to *Configuring Lightweight Access Directory Protocol* for more information.
 - e. Stop and restart WebSphere Application Server for your changes to take effect.
2. To disable Tivoli Access Manager for authentication, complete the following steps:

- a. Deselect the **Use Tivoli Access Manager for Account Policies** check box on the LDAP user registry page in the administrative console.
- b. Stop the WebSphere Application Server.
- c. Run the SvrSslCfg command to unconfigure the WebSphere Application Server to use an existing Tivoli Access Manager server. For more information, see *Configuring WebSphere Application Server to use Tivoli Access Manager server*. For example,

Attention: The following code example should be written as one line of code.

```
%WAS_HOME%\java\jre\bin\java com.tivoli.pd.jcfg.SvrSslCfg
-action unconfig -admin_id sec_master -admin_pwd password
-appsvr_id appsvr_name -policysvr TAM_policysvr_host_name:7135:1
-cfg_file %WAS_HOME%\java\jre\PdPerm.properties
-host was_server_host_name
```

- d. Run the PDJrteCfg command with the unconfig action. Command usage:

Attention: The following code example should be written as one line of code.

```
java -Djava.ext.dirs -Dpd.home="%WAS_HOME%\java\jre\PolicyDirector"  
com.tivoli.pd.jcfg.PDJrteCfg
```

```
-action [ config | unconfig ]  
-was  
-config_type [ full | standalone ]  
-java_home <jre_home>  
-host <Policy_Server_host_name>
```

Detailed information on the PDJrteCfg class is located in the com.tivoli.pd.jcfg.PDJrteCfg in the Command Reference of the Tivoli Access Manager Version 5.1 product documentation. Here is an example of the unconfiguration script.

Attention: The following code example should be written as one line of code.

```
%WAS_HOME%\java\jre\bin\java -Djava.ext.dirs  
-Dpd.home="%WAS_HOME%\java\jre\PolicyDirector" \  
-cp "%WAS_HOME%\java\jre\lib\ext\PD.jar;  
%WAS_HOME%\java\jre\lib\ext\ibmjceprovider.jar;  
%WAS_HOME%\java\jre\lib\ext\ibmpkcs.jar;%CLASSPATH%" \  
com.tivoli.pd.jcfg.PDJrteCfg -action unconfig -was -java_home  
%WAS_HOME%\java\jre
```

Configuring WebSphere Application Server to use Tivoli Access Manager server:

The SvrSslCfg utility is used to configure every WebSphere Developer Kit, Java Technology Edition installation. For example, in a Network Deployment environment, run the SvrSslCfg utility on the Deployment Manager first and then run the utility on all of the application server nodes, even if the nodes are on the same machine.

1. Configure WebSphere Application Server to use Tivoli Access Manager by using the SvrSslCfg utility. This utility is used to configure, to remove, and to modify the configuration information associated with the WebSphere Application Server and the Tivoli Access Manager server that you have configured. After running the SvrSslCfg utility successfully on WebSphere Application Server, a user account and server entries representing the WebSphere Application Server are created in the Tivoli Access Manager user registry. In addition, a configuration file and a Java key store file, which securely stores a client certificate, are created locally on the Application Server. This client certificate permits callers to use Tivoli Access Manager authentication services.
2. Use the SvrSslCfg utility also to unconfigure the communication of the WebSphere Application Server with Tivoli Access Manager. When unconfiguring, you are choosing to remove the user and server entries from the user registry, and clean up the local configuration and key store files.

Configuring an application server in remote mode:

1. After obtaining the necessary information, use the SvrSslCfg class to create the Tivoli Access Manager application name, the configuration file, and the key

store file. Configuring an application server creates user and server information in the user registry as well as local configuration and key store files.

2. Based on the sample information shown in the SvrSslCfg syntax article, the following example script establishes a connection between austin.ibm.com and the Tivoli Access Manager secure domain:

```
java com.tivoli.pd.jcfg.SvrSslCfg -action config \  
-admin_id sec_master -admin_pwd secpw \  
-appsvr_id PDPermissionjapp -appsvr_pwd pw -host austin.ibm.com \  
-mode remote -port 999 -policysvr ampolicy.ibm.com:7135:1 \  
-authzsvr amazn.ibm.com:7136:1 \  
-cfg_file <install_root>/java/jre/PdPerm.properties \  
-key_file <install_root>/java/jre/lib/security/pdperm.ks \  
-cfg_action create
```

Use the `-cfg_action create` option to initially create the configuration and keystore files. Use the `-cfg_action replace` option if these files already exist. If the `-cfg_action create` option is used and the configuration or keystore files already exist, an exception is thrown.

Removing configuration information from an application server:

The `-action unconfig` option removes the user and server information from the user registry, deletes the local keystore file, and removes information for this application from the configuration file. The configuration file is not deleted. This option is illustrated in the following example:

```
java com.tivoli.pd.jcfg.SvrSslCfg -action unconfig \  
-admin_id sec_master -admin_pwd secpw \  
-appsvr_id PDPermissionjapp -host austin.ibm.com \  
-policysvr ampolicy.ibm.com:7135:1
```

This operation fails when the caller is unauthorized. Errors encountered during this process are ignored to ensure that all steps are attempted. This procedure succeeds even if local configuration information or information in the user registry is accidentally deleted.

Enabling WebSphere Application Server to use Tivoli Access Manager for authentication:

The WebSphere Application Server can use the Tivoli Access Manager for authenticating users only when the following LDAP directory servers are used as the user registry:

- IBM Directory Server
- Sun ONE directory server
- Lotus Domino Enterprise Server
- Novell Directory Server eDirectory
- Microsoft Windows Active Directory
- z/OS LDAP Server

When one of these directory servers is used as the registry, you can use the Tivoli Access Manager to authenticate users instead of directly binding to the LDAP registry by accepting the password and the account policies set in the Tivoli Access Manager. The following steps explain how to set a property in the LDAP custom properties section when enabling security.

1. Verify that the PDJrteCfg and SvrSslCfg commands have been run as detailed in Configuring WebSphere Application Server to use Tivoli Access Manager for authentication and Configuring WebSphere Application Server to use Tivoli Access Manager server.
2. Select the LDAP user registry by clicking **Security > User Registries > LDAP** in the administrative console. Confirm that you selected one of the supported LDAP directory servers listed previously.
3. On the LDAP user registry page, select the **Use Tivoli Access Manager for Account Policies** check box in the administrative console. Tivoli Access Manager authentication is valid only when you select **LDAP**. The login policies set in Tivoli Access Manager are honored only when the user logs in with a password. The login policies, however, are not honored when a user logs in without a password (If the user uses X.509 certificates instead of passwords, for example). For more information see the Tivoli Access Manager documentation.
4. If you create groups using Tivoli Access Manager, add the objectClass=accessGroup value to the IBM_Directory_Server group filter. For example the following code is one continuous line:

```
(&(cn=%v)(!(objectclass=groupOfNames)(objectclass=accessGroup)
(objectclass=groupOfUniqueNames)))
```

This addition is required because by default, the Tivoli Access Manager creates the accessGroup object class and uses it for all the groups it creates. The default WebSphere Application Server LDAP filters only look for the groupOfNames or groupOfUniqueNames object classes. If this change is not made, authorization problems result when groups are assigned to roles.

If the groups are first created in the IBM_Directory_Server registry and then imported into the Tivoli Access Manager, you do not need to change the filters because the groups have the object classes that the WebSphere Application Server requires.

5. Enable security. If security is already enabled, save these changes and restart the servers for the Tivoli Access Manager authentication to take effect.

Best practices for establishing Secure Sockets Layer communications: To create the files necessary for establishing Secure Sockets Layer (SSL) communications in the secure domain, the SvrSslCfg class requires information about the secure domain as well as information related to the application. The following information about the Tivoli Access Manager secure domain is required:

- **Administrative user password**
The password associated with the Tivoli Access Manager administrative user. Typically, the user name is sec_master.
- **Policy server name**
The name of the system running the Tivoli Access Manager policy server named ivmgrd.
- **Authorization server name**
The name of the system running the Tivoli Access Manager authorization server named ivacld. This system might be the same system as the policy server.
- **Policy server SSL port number**
The number of the port used for SSL communications with the policy server. The default is 7135.
- **Authorization server SSL port number**

The number of the port used for SSL communications with the authorization server. The default is **7136**.

Note: If either the `pdPerm.properties` file or the SSL keystore file becomes damaged, you must repeat the configuration steps. Creating backups of these files is recommended.

– **Configuration file URL**

The URL to the configuration file that is manipulated by the `SvrSslCfg` class.

– **Keystore file URL**

The URL to the keystore file that is manipulated by the `SvrSslCfg` class.

– **Tivoli Access Manager application name**

The name of the Tivoli Access Manager application name that is created and associated with the SSL connection between this system and the Tivoli Access Manager servers. The configuration and keystore files are sensitive files that need protection. The contents of the configuration file are not externalized and are subject to change without notice in future releases of Tivoli Access Manager. Do not use the information in the configuration file directly.

Use the previously mentioned information about the secure domain to configure WebSphere Application Server to use Tivoli Access Manager.

SvrSslCfg usage syntax:

The following information is a summary of how to use the `com.tivoli.pd.jcfg.SvrSslCfg` class:

```
java com.tivoli.pd.jcfg.SvrSslCfg -action {config |unconfig | addsvr | rmsvr
|chgsvr | setport |setdblisten | replcert }
```

```
-admin_id administrator_user_ID
-admin_pwd administrator_password
-appsvr_id application_server_name
-port port_number
-mode { local | remote }
[Note: local mode is not supported in this release.]
-policysvr policy_server_name:port:rank [...]
-authzsvr authorization_server_name:port:rank [...]
-cfg_file fully_qualified_name_of_configuration_file
-key_file fully_qualified_name_of_keystore_file
-appsvr_pwd application_server_password
-host host_name_of_application_server
-dblisten { true | false }
-dbdir name_of_directory_for_local_policy_database
-dbrefresh refresh_interval_in_seconds
-cfg_action { create | replace }
```

Detailed information on the `SvrSslCfg` class is located in the `com.tivoli.pd.jcfg.SvrSslCfg` class description in the Authorization Java Classes Developer's Reference of the Tivoli Access Manager product documentation.

Information required to run SvrSslCfg command

| Information | Value |
|------------------------|------------|
| Administrator user ID | Sec_master |
| Administrator password | Secpw |

| | |
|---|---|
| Policy server, TCP/IP communications port number, and rank (default port is 7135) | ampolicy.Tivoli.com:7135:1 |
| Authorization server, TCP/IP communications port number, and rank (default port is 7136) | amazn.Tivoli.com:7136:1 |
| Host name of Java application system (used in remote mode examples) | Jsys.Tivoli.com |
| TCP/IP port on which the application server listens for communications from the policy server | 999 |
| Application server password | pw |
| Tivoli Access Manager application ID | PDPermissionjapp The application ID must be unique. Other instances of the application running on this system or other systems must each have a unique ID. You can use a distinguished name if an LDAP-based user registry is used by Tivoli Access Manager. |
| Configuration file | <install_root>/java/jre/PdPerm.properties |
| Key store file | <install_root>/java/jre/lib/security/pdperm.ks |

Best practices for mapping credentials using IBM Tivoli Access Manager

IBM Tivoli Access Manager credential mapping

Currently, IBM Tivoli Access Manager supports a flexible global signon (GSO) solution that features the ability to provide alternate user names and passwords to the back-end Web application server. For detailed information on a global signon solution, refer to Chapter 8 of the "WebSEAL Administrator's Guide" in IBM Tivoli Access Manager Version 5.1 documentation.

Refer to the "IBM WebSphere Application Server Integration Guide" in the IBM Tivoli Access Manager Version 5.1 documentation for more information.

Global signon and IBM Tivoli Access Manager user and group management

For global signon resource management and IBM Tivoli Access Manager user and group management, refer to the IBM Tivoli Access Manager Version 5.1 documentation.

User registries

Information about users and groups reside in a user registry. With WebSphere Application Server, a user registry authenticates a user and retrieves information about users and groups to perform security-related functions, including authentication and authorization.

WebSphere Application Server provides several implementations to support multiple types of operating system base user registries. You can use the custom LDAP feature to support any LDAP server by setting up the correct configuration (user and group filters). However, support is not extended to these custom LDAP servers because there are many configuration possibilities.

In addition to Local operating system (OS) and LDAP registries, WebSphere Application Server also provides a plug-in that supports any registry by using the custom registry feature (also referred to as a custom user registry). The custom registry feature supports any user registry that is not implemented by WebSphere Application Server. You can use any registry used in the product environment by implementing the *UserRegistry interface* interface.

The UserRegistry interface is very helpful in situations where the current user and group information exists in some other format (for example, a database) and cannot move to Local OS or LDAP. In such a case, implement the UserRegistry interface so that WebSphere Application Server can use the existing registry for all of the security-related operations. Using a custom registry is a software implementation effort, it is expected that the implementation does not depend on other WebSphere Application Server resources, for example, data sources, for its operation.

Although WebSphere Application Server supports different types of user registries, only one user registry can be active. This active registry is shared by all of the product server processes. If the product processes in one node or cell need to communicate with other product processes in other nodes or cells using Lightweight Third Party Authentication (LTPA), all of the nodes and cells share the same user registry.

Configuring user registries

Before configuring the user registry, decide which registry to use. Though different types of registries are supported, all of the processes in WebSphere Application Server can use one active registry. Configuring the correct registry is a prerequisite to assigning users and groups to roles for applications. When no registry is configured, the LocalOS registry is used by default. So, if your choice of registry is not Local OS you need to first configure the registry, which is normally done as part of enabling security, restart the servers, and then assign users and groups to roles for all your applications.

After the applications are assigned users and groups, and you need to change the registries (for example from Lightweight Directory Access Protocol (LDAP) to Custom), delete all the users and groups (including any RunAs role) from the applications, and reassign them after changing the registry through the administrative console or by using wsadmin scripting. The following wsadmin command removes all of the users and groups (including the RunAs role) from any application:

```
$AdminApp deleteUserAndGroupEntries yourAppName
```

where *yourAppName* is the name of the application. Backing up the old application is advised before performing this operation. However, if both of the following conditions are true, you might be able to switch the registries without having to delete the users and groups information:

- All of the user and group names (including the password for the RunAs role users) in all of the applications match in both registries.
- The application bindings file does not contain the accessIDs, which are unique for each registry even for the same user or group name.

By default, an application does not contain accessIDs in the bindings file (these IDs are generated when the applications start). However, if you migrated an

existing application from an earlier release, or if you used the wsadmin script to add accessIDs for the applications to improve performance you have to remove the existing user and group information and add the information after configuring the new registry.

For more information on updating accessIDs, see updateAccessIDs in the AdminApp object for scripted administration article.

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

- Configure the local operating system user registry.
- Configure the LDAP user registry.
- Configure the custom user registry.

This step is required as part of enabling security in WebSphere Application Server.

1. If you are enabling security, make sure that you complete the remaining steps. Verify that the Active User Registry field in the **Global Security** panel is set to the appropriate registry. As the final step, validate the user ID and the password by clicking **OK** or **Apply** in the Global Security panel. Save, stop and start all the WebSphere Application Servers.
2. For any changes in user registry panels to be effective, you must validate the changes by clicking **OK** or **Apply** in the Global Security panel. After validation, save the configuration, stop and start all of the WebSphere Application Servers (cells, nodes and all the application servers). To avoid inconsistencies between the WebSphere Application Server processes, make sure that any changes to the registry are done when all of the processes are running. If any of the processes are down, force synchronization to make sure that the process can start later.
3. If the server or servers start without any problems, the setup is correct.

Local operating system user registries

With the local operating system, or Local OS, user registry implementation, the WebSphere authentication mechanism can use the user accounts database of the local operating system.

WebSphere Application Server provides implementations for the Windows local accounts registry and domain registry, as well as implementations for the Linux, Solaris, and AIX user accounts registries. Windows Active Directory is supported through the Lightweight Directory Access Protocol (LDAP) user registry implementation discussed later.

A Local OS user registry is not a centralized user registry like LDAP. Do not use a Local OS user registry in a distributed WebSphere Application Server environment, where application servers are dispersed across several machines because each machine has its own user registry. Exceptions include a Windows domain registry, which is a centralized registry.

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:

```
SECJ0337E: The mapCertificate method is not supported
```

The error is intended for Web client certificates; however, it also displays for Java client certificates. Ignore this error for Java client certificates.

Using Windows operating system registries

When enabling security on Windows operating systems, if the local operating system (LocalOS) is selected as the registry, consider the following points:

Required privileges

The user that is running the WebSphere Application Server process requires enough operating system privilege to call the Windows systems application programming interface (API) for authenticating and obtaining user and group information from the Windows operating system. This user logs into the machine, or if running as a service, is the **Log On As** user. Depending on the machine (whether the machine is a stand-alone machine or a machine that is part of a domain or is the domain controller), the access requirements vary.

- For a stand-alone machine, the user:
 - Is a member of the administrative group.
 - Has the **Act as part of the operating system** privilege.
 - Has the **Log on as a service** privilege, if the server is run as a service.
- For a machine that is a member of a domain, only a domain user can start the server process and:
 - Is a member of the domain administrative groups in the domain controller.
 - Has the **Act as part of the operating system** privilege in the Domain Security Policy on the domain controller.
 - Has the **Act as part of the operating system** privilege in the Local Security Policy on the local machine.
 - Has the **Log on as a service** privilege on the local machine, if the server is run as a service.

The user is a domain user and not a local user, which implies that when a machine is part of a domain, only a domain user can start the server.

- For a domain controller machine, the user:
 - Is a member of the domain administrative groups in the domain controller.
 - Has the **Act as part of the operating system** privilege in the Domain Security Policy on the domain controller.
 - Has the **Log on as a service** privilege on the domain controller, if the server is run as a service.

To give a user the Act as part of the operating system or Log on as a service on Windows 2000 systems:

1. Click **Start > Settings > Control Panel > Administrative Tools > Local Security Policy > Local Policies > User Rights Assignments > Act as part of the operating system (or Log on as a service)** .
2. Add the user name by clicking **Add**.
3. Restart the machine.

Windows 2000 domain controller users: For a Windows 2000 domain controller replace **Local Security Policy** with **Domain Security Policy** in the previous step.

Note: In all of the previous configurations, the server can be run as a service using the LocalSystem for the Log On As entry. LocalSystem has the required privileges and there is no need to give any user special privilege. However, because the LocalSystem has special privileges, make sure that it is appropriate to use it in your environment.

If the user running the server does not have the required privilege, you might see one of the following exception messages in the log files:

- A required privilege is not held by the client.
- Access is denied.

Domain and local registries

When WebSphere Application Server is started, the security run time initialization process dynamically attempts to determine if the local machine is a member of a Windows domain. If the machine is part of a domain then by default both the local registry users or groups and the domain registry users or groups can be used for authentication and authorization purposes with the domain registry taking precedence. The list of users and groups presented during the security role mapping would then include users and groups from both the local user registry and the domain user registry. The users and groups can be distinguished by the host names associated with them.

WebSphere Application Server does not support trusted domains.

If the machine is not a member of a Windows system domain, the user registry local to that machine is used.

Using both the domain registry and the local registry

When the machine hosting the WebSphere Application Server process is a member of a domain, both the local and the domain registries are used by default. The following section describes more on this topic and recommends some best practices to avoid undesirable consequences.

- **Best Practices**

In general, if the local and the domain registries do not contain common users or groups, it is simpler to administer and it eliminates undesirable side effects. So if possible, give users and groups access to unique security roles (including the server ID and administrative roles). In this situation, select the users and groups from either the local registry or the domain registry to map to the roles.

In cases where the same users or groups exist in both the local registry and the domain registry, it is recommended that at least the server ID and the users and groups that are mapped to the administrative roles be unique in the registries (exist only on the domain).

If a common set of users exists, set a different password to make sure that the appropriate user is authenticated.

- **How it works**

When a machine is part of a domain, the domain user registry takes precedence over the local user registry. For example, when a user logs into the system, the domain registry tries to authenticate the user first. If the authentication fails the local registry is used. When a user or a group is mapped to a role, the user and group information is first obtained from the domain registry. In case of failure, the local registry is tried. However, when a fully qualified user or a group name (one that has a domain or host name attached to it) is mapped to a role, then only that registry is used to get the information. Use the administrative console

or scripts to get the fully qualified user and group names and is the recommended way to map users and groups to roles.

Note: A user **Bob** on one machine (the local registry, for example) is not the same as the user **Bob** on another machine (say the domain registry) because the uniqueID of **Bob** (the security identifier [SID], in this case) is different in different registries.

- **Examples**

The machine `MyMachine` is part of the domain `MyDomain`. `MyMachine` contains the following users and groups:

- `MyMachine\user2`
- `MyMachine\user3`
- `MyMachine\group2`

`MyDomain` contains the following users and groups:

- `MyDomain\user1`
- `MyDomain\user2`
- `MyDomain\group1`
- `MyDomain\group2`

Here are some scenarios that assume the previous set of users and groups.

1. When `user2` logs into the system, the domain registry is used for authentication. If the authentication fails (the password is different) the local registry is used.
2. If the user `MyMachine\user2` is mapped to a role, only the `user2` in `MyMachine` has access. So if the `user2` password is the same on both the local and the domain registries, `user2` cannot access the resource, because `user2` is always authenticated using the domain registry. Hence, if both registries have common users, it is recommended that the password be different.
3. If the `group2` is mapped to a role, only the users who are members of the `MyDomain\group2` can access the resource because `group2` information is first obtained from the domain registry.
4. If the group `MyMachine\group2` is mapped to a role, only the users who are members of the `MyMachine\group2` can access the resource. A specific group is mapped to the role (`MyMachine\group2` instead of just `group2`).
5. Use either `user3` or `MyMachine\user3` to map to a role, because `user3` is unique; it exists in one registry only.

Authorizing with the domain user registry first can cause problems if a user exists in both the domain and local user registries with the same password. Role-based authorization can fail in this situation because the user is first authenticated within the domain user registry. This authentication produces a unique domain security ID that is used in WebSphere Application Server during the authorization check. However, the local user registry is used for role assignment. The domain security ID does not match the unique security ID associated with the role. To avoid this problem, map security roles to domain users instead of local users.

Using either the local or the domain registry. If you want to access users and groups from either the local registry or the domain registry, instead of both, set the property `com.ibm.websphere.registry.UseRegistry`. This property can be set to either **local** or **domain**. When this property is set to **local** (case insensitive) only the local registry is used. When this property is set to **domain** (case insensitive) only the domain registry is used. Set this property by clicking **Custom Properties** in the **Security > User Registries > Local OS** panel in the administrative console or by using scripts. When the property is set, the privilege requirement for the user

who is running the product process does not change. For example, if this property is set to **local**, the user running the process requires the same privilege, as if the property was not set.

Using UNIX system registries

When using UNIX system registries, the process ID that runs the WebSphere Application Server process needs the root authority to call the local operating system APIs for authentication and for obtaining user or group information.

Note: In UNIX systems, only the local machine registry is used. Network Information Service (NIS) (Yellow Pages) is not supported.

Using Linux and Solaris system registries

For WebSphere Application Server Local OS security registry to work on the Linux and Solaris platforms, a shadow password file must exist. The shadow password file is named `shadow` and is located in the `/etc` directory. If the shadow password file does not exist, an error occurs after enabling global security and configuring the user registry as Local OS.

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

Remote registries

By default, the registry is local to all of the product processes. The performance is higher, (no need for remote calls) and the registry also increases availability. Any process failing does not effect other processes. When using LocalOS as the registry, every product process must run with privilege access (`root` in UNIX, `Act as part of operating system` in Windows systems). If this process is not practical in some situations, you can use a remote registry from the node (or in very rare situations from the cell). Using a remote registry affects performance and creates a single point of failure. **Use remote registries only in rare situations.**

The node and the cell processes are meant for manipulating configuration information and using them to host the registry for all the application servers creates traffic and can cause problems. Using a node agent (instead of the cell) to host the remote registry is preferable because since the cell process is not designed to be highly available. Also, using a node to host the remote registry indicates that only the application servers in that node are using it. Because the Node Agent does not contain any application code, giving it the access required, privilege is not a concern.

You can set up a remote registry by setting the `WAS_UseRemoteRegistry` property in the Global Security panel using the **Custom Properties** link at the bottom of the administrative console panel. Use either the **Cell** or the **Node** (case insensitive) value. If the value is `Cell`, the cell registry is used by all of the product processes including the node agent and all of the application servers. If the cell process is down for any reason, restart all of the processes after the cell is restarted. If the node agent registry needs is used for the remote registry, set the value, `WAS_UseRemoteRegistry`, to `node`. In this case, all the application server processes use the node agent registry. In this case, if the node agent fails and does not start automatically, then depending on that node agent, you might need to restart all the application servers, after the node agent is started.

Configuring local operating system user registries

For security purposes, the WebSphere Application Server provides and supports the implementation for Windows operating system registries, AIX, Solaris and multiple versions of Linux operating systems. The respective operating system APIs are called by the product processes (servers) for authenticating a user and other security-related tasks (for example, getting user or group information). Access to these APIs are restricted to users who have special privileges. These privileges depend on the operating system and are described below.

Before configuring the LocalOS registry you need to know the user name (ID) and password to use here. This user can be any valid user in the registry. This user is referred to as either a product security server ID, a server ID or a server user ID in the documentation. Having a server ID means that a user has special privileges when calling protected internal methods. Normally, this ID and password are used to log into the administrative console after security is turned on. You can use other users to log in if those users are part of the administrative roles. When security is enabled in the product, this server ID and password are authenticated with the registry during product startup. If authentication fails, the server does not come up. So it is important to choose an ID and password that do not expire or change often. If the product server user ID or password need to change in the registry, ensure that the changes are performed when all the product servers are up and running. After the changes are completed in the registry, use the following steps to change the ID and the password information. Save, stop, and restart all the servers so that the product can use the new ID or password. If any problem arises after starting the product because of authentication problems (that cannot be fixed), disable security before the server can start up. To avoid this step, make sure that the changes are validated in the Global Security panel. After the server is up, change the ID and password information and enable security.

When using the Windows operating system, consider the following issues:

- The server ID needs to be different from the Windows machine name where the product is installed. For example, if the Windows machine name is *vicky* and the security server ID is *vicky*, the Windows system fails when getting the information (group information, for example) for user *vicky*.
- WebSphere Application Server dynamically determines whether the machine is a member of a Windows system domain.
- WebSphere Application Server does not support Windows trusted domains.
- If a machine is a member of a Windows domain, both the domain user registry and the local user registry of the machine participate in authentication and security role mapping.
- The domain user registry takes precedence over the local user registry of the machine and can have undesirable implications if users with the same password exist in both user registries.
- The user that the product processes run under requires the Administrative and Act as part of the operating system privileges to call the Windows operating system APIs that authenticate or collect user and group information. The process needs special authority, which is given by these privileges. The user in this example might not be the same as the security server ID (the requirement for which is a valid user in the registry). This user logs into the machine (if using the command line to start the product process) or the Log On User setting in the services panel if the product processes have started using the services. If the machine is also part of a domain, this user is a part of the Domain Admin group in the domain to call the operating system APIs in the domain in addition to having the Act as part of operating system privilege in the local machine.

When using the UNIX operating systems (AIX and Solaris) and Linux, consider the following points:

- The user that the product processes run under requires the root privilege. This privilege is needed to call the UNIX operating system APIs to authenticate or to collect user and group information. The process needs special authority, which is given by the root privilege. This user may 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.
- When using the Linux operating system, you might need to have the password shadow file in your system.

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

1. Click **Security > User Registries > LocalOS** in the left navigation panel of the administrative console.
2. Enter a valid user name in the Server User ID field.
3. Enter the user password in the Server User Password field.
4. Click **OK**. Validation of the user and password does not happen in this panel. Validation is only done when you click **OK** or **Apply** in the Global Security panel. If you are enabling security for the first time, complete the other steps and go to the Global Security panel. Make sure that LocalOS is the Active User Registry. If security was already enabled and you had changed either the user or the password information in this panel, make sure to go to the Global Security panel and click **OK** or **Apply** to validate your changes. If your changes are not validated, the server might not come up.

The Local OS user registry has been configured.

1. If you are enabling security, complete the remaining steps. As the final step, ensure that you validate the user and password by clicking **OK** or **Apply** in the Global Security panel. Save, stop, and start all the product servers.
2. For any changes in this panel to be effective, you need to save, stop and start all the product servers (cell, nodes and all the application servers).
3. If the server comes up without any problems the setup is correct.

Local operating system user registry settings:

Use this page to configure local operating system user registry settings.

To view this administrative console page, click **Security > User Registries > Local OS**.

Server user ID:

Specifies a valid user ID in the Local OS registry.

This ID is the security server ID, which is only used for WebSphere Application Server security and is not associated with the system process that runs the server. The server calls the Local OS registry to authenticate and obtain privilege information about users by calling the native APIs in that particular registry. Access to native APIs is normally restricted to users having special privileges (for example, **root** in UNIX systems and **Act as part of operating system** in Windows systems). To use security in the application server, the process ID (not the security server ID) on which WebSphere Application Server runs requires enough privileges to call the system APIs. The special privilege means that the process running the WebSphere Application Server needs to be part of the **Administrators** group and

have the **Act as part of operating system** privilege on Windows systems, and be **root**, or have root authority on UNIX systems.

When using a Windows system registry, this ID cannot match the name of the Windows machine. Windows systems treat the machine name bob as having an account similar to user bob.

Data type: String
Units: Alphanumeric characters

Server user password:

Specifies a valid user password that corresponds to a valid user ID in the Local OS registry.

Data type String

Local OS Type:

The local operating system (OS) type is set as the Service Access Facility (SAF).

SAF, which is the local operating system on WebSphere Application Server for z/OS, is automatically specified as the local operating system type. Configure SAF by clicking **Custom Properties** in the Additional Properties section. You can configure the following custom properties:

| Property | Data type | Valid values |
|--|-----------|--|
| com.ibm.security.SAF.authorization | Boolean | true or false |
| com.ibm.security.delegation | Boolean | true or false |
| com.ibm.security.SAF.EJBROLE.Audit.Messages.Suppress | Boolean | true or false |
| com.ibm.security.SAF.unauthenticated | String | A valid MVS user ID. The default value is WSGUEST. |

Ignore Case:

When this option is set to true, a case insensitive authorization check is performed.

SAF user IDs are usually in uppercase letters. Enabling this option is necessary only when your registry is case insensitive and does not provide a consistent case when queried for users and groups.

Lightweight Directory Access Protocol

Lightweight Directory Access Protocol (LDAP) is a user registry in which authentication is performed using an LDAP binding.

WebSphere Application Server security provides and supports implementation of most major LDAP directory servers, which can act as the repository for user and group information. These LDAP servers are called by the product processes (servers) for authenticating a user and other security-related tasks (for example, getting user or group information). This support is provided by using different user and group filters to obtain the user and group information. These filters have default values that you can modify to fit your needs. The custom LDAP feature

enables you to use any other LDAP server (which is not in the product supported list of LDAP servers) for its user registry by using the appropriate filters.

To use LDAP as the user registry, you need to know a valid user name (ID), the user password, the server host and port, the base distinguished name (DN) and if necessary the bind DN and the bind password. You can choose any valid user in the registry that is searchable. In some LDAP servers, the administrative users are not searchable and cannot be used (for example, cn=root in SecureWay). This user is referred to as WebSphere Application Server security server ID, server ID, or server user ID in the documentation. Being a server ID means a user has special privileges when calling some protected internal methods. Normally, this ID and password are used to log into the administrative console after security is turned on. You can use other users to log in if those users are part of the administrative roles.

When security is enabled in the product, this server ID and password are authenticated with the registry during the product startup. If authentication fails, the server does not start. Choosing an ID and password that do not expire or change often is important. If the product server user ID or password need to change in the registry, make sure that the changes are performed when all the product servers are up and running.

When the changes are done in the registry, use the steps described in Configuring LDAP user registries. Change the ID, password, and other configuration information, save, stop, and restart all the servers so that the new ID or password is used by the product. If any problems occur starting the product when security is enabled, disable security before the server can start up (to avoid these problems, make sure that any changes in this panel are validated in the Global Security panel). When the server is up, you can change the ID, password and other configuration information and then enable security.

Supported directory services:

WebSphere Application Server security supports several different LDAP servers. For a list of supported LDAP servers, refer to the **Supported hardware, software and APIs** prerequisite Web site in the "Resources for Learning" article. *The z/OS Security Server LDAP is supported when the DB2 TDBM backend is used. Use the SecureWay Directory Server filters to connect to the z/os LDAP.* It is expected that other LDAP servers that follow the LDAP specification function. Support is limited to these specific directory servers only. You can use any other directory server by using the custom directory type in the list and by filling in the filters required for that directory.

To improve performance for LDAP searches, the default filters for IBM Directory Server, iPlanet Directory Server, and Active Directory are defined such that when you search for a user, the result contains all the relevant information about the user (user ID, groups, and so on). As a result, the product does not call the LDAP server multiple times. This definition is possible only in these directory types, which support searches where the complete user information is obtained.

If you use the IBM Directory Server, enable the Ignore case flag. This flag is required because when the group information is obtained from the user object attributes, the case is not the same as when you get the group information directly. For the authorization to work in this case, perform a case insensitive check and verify the requirement for the Ignore case flag.

Lightweight Directory Access Protocol settings:

Use this page to configure Lightweight Directory Access Protocol (LDAP) settings when users and groups reside in an external LDAP directory.

To view this administrative console page, click **Security > User Registries > LDAP**.

When security is enabled and any of these properties change, go to the Global Security panel and click **Apply** to validate the changes.

Server User ID:

Specifies the user ID under which the server runs, for security purposes.

Although this ID is not the LDAP administrator user ID, specify a valid entry in the LDAP directory located under the Base Distinguished Name.

Server User Password:

Specifies the password corresponding to the security server ID.

Type:

Specifies the type of LDAP server to which you connect.

The type is used to preload default LDAP properties. IBM Directory Server users can choose either `IBM_Directory_Server` or `SecureWay` as the directory type. Use the `IBM_Directory_server` directory type for better performance. Users of the iPlanet Directory Server can choose either `iPlanet Directory Server` or `NetScape` as the directory type. Use the `iPlanet Directory Server` directory type for better performance after configuring iPlanet to use `role (nsRole)` as the grouping method.

5.1+ IBM SecureWay Directory Server is not supported..

For a list of supported LDAP servers, see "Supported directory services." in the documentation.

Host:

Specifies the host ID (IP address or domain name service (DNS) name) of the LDAP server.

Port:

Specifies the host port of the LDAP server.

If multiple WebSphere Application Servers are installed and configured to run in the same single signon domain, or if the WebSphere Application Server interoperates with a previous version of the WebSphere Application Server, then it is important that the port number match all configurations. For example, if the LDAP port is explicitly specified as 389 in a Version 4.0.x configuration, and a WebSphere Application Server at Version 5 is going to interoperate with the Version 4.0.x server, then verify that port 389 is specified explicitly for the Version 5 server.

Default:

389

Base Distinguished Name:

Specifies the base distinguished name of the directory service, indicating the starting point for LDAP searches of the directory service.

For example, for a user with a distinguished name (DN) of cn=John Doe, ou=Rochester, o=IBM, c=US, you can specify the base DN as (assuming a suffix of c=us): ou=Rochester, o=IBM, c=us o=IBM, c=us c=us. For authorization purposes, this field is case sensitive. This specification implies that if a token is received (for example, from another cell or Domino) the base DN in the server must match the base DN from the other cell or Domino server exactly. If case sensitivity is not a consideration for authorization, enable the **Ignore Case** field. This field is required for all Lightweight Directory Access Protocol (LDAP) directories except for the Domino Directory, where this field is optional.

Bind Distinguished Name:

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

If no name is specified, the application server binds anonymously. See the Base Distinguished Name field description for examples of distinguished names.

Bind Password:

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

Search Timeout:

Specifies the timeout value in seconds for an Lightweight Directory Access Protocol (LDAP) server to respond before aborting a request.

Default:

120

Reuse connection:

Specifies whether the server reuses the Lightweight Directory Access Protocol (LDAP) connection. Clear this option only in rare situations where a router is used to spray requests to multiple LDAP servers and when the router does not support affinity.

Default:

Enabled

Range:

Enabled or Disabled

Ignore Case:

Specifies that a case insensitive authorization check is performed.

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

5.1+ This field is required when Sun ONE Directory Server is selected as the LDAP directory server. For more information, see "Using specific directory servers as the LDAP server" in the documentation.

Otherwise, this field is optional and can be enabled when a case-sensitive authorization check is required. For example, use this field when the certificates and the certificate contents do not match the case used for the entry in the LDAP server. You can enable the **Ignore Case** field when using single signon (SSO) between WebSphere Application Server and Lotus Domino.

Default: Disabled
Range: Enabled or Disabled

SSL Enabled:

Specifies whether secure socket communication is enabled to the Lightweight Directory Access Protocol (LDAP) server. When enabled, the LDAP Secure Sockets Layer (SSL) settings are used, if specified.

SSL Configuration:

Specifies the Secure Sockets Layer configuration to use for the Lightweight Directory Access Protocol (LDAP) connection. This configuration is used only when SSL is enabled for LDAP.

Default: DefaultSSLSettings

Use Tivoli Access Manager for Account Policies:

Select this option to indicate that the Tivoli Access Manager is used for authentication to honor password and account policies. This option requires that you have previously installed the Tivoli Access Manager.

Do not select this option unless you have a Tivoli Access Manager Server installed and configured to be used by WebSphere Application Server. The Lightweight Directory Access Protocol (LDAP) directory server used by the Tivoli Access Manager must be the same LDAP directory server that is used by WebSphere Application Server.

Important: When you select this option, IBM SecureWay Directory Server is not supported as an LDAP directory server.

Lightweight Directory Access Protocol advanced settings:

Use this page to configure advanced Lightweight Directory Access Protocol (LDAP) user registry settings when users and groups reside in an external LDAP directory.

To view this administrative page, click **Security > User Registries > LDAP Advanced LDAP settings**.

Default values for all the user and group related filters are already completed in the appropriate fields. You can change these values depending on your requirements. These default values are based on the type of LDAP server selected in the **LDAP settings** panel. If this type changes (for example from Netscape to

Secureway) the default filters automatically change. When the default filter values change, the LDAP server type changes to Custom to indicate that custom filters are used. When security is enabled and any of these properties change, go to the **Global Security** panel and click **Apply** or **OK** to validate the changes.

User Filter:

Specifies the LDAP user filter that searches the registry for users.

This option is typically used for Security Role to User assignments. It specifies the property by which to look up users in the directory service. For example, to look up users based on their user IDs, specify `(ampersand(uid=%v)(objectclass=inetOrgPerson)` where ampersand is the ampersand symbol (&). 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 registry for groups

This option is typically used for Security Role to Group assignments. It specifies the property by which to look up groups in the directory service. For more information about this syntax, see the LDAP directory service documentation.

Data type: String

User ID Map:

Specifies the LDAP filter that maps the short name of a user to an LDAP entry.

Specifies the piece of information that represents users when users appear. For example, to display entries of the type `object class = inetOrgPerson` by their IDs, specify `inetOrgPerson:uid`. This field takes multiple `objectclass:property` pairs delimited by a semicolon (;).

Data type: String

Group ID Map:

Specifies the LDAP filter that maps the short name of a group to an LDAP entry.

Specifies the piece of information that represents groups when groups appear. For example, to display groups by their names, specify `*:cn`. The asterisk (*) is a wildcard character that searches on any object class in this case. This field takes multiple `objectclass:property` pairs delimited by a semicolon (;).

Data type: String

Group Member ID Map:

Specifies the LDAP filter which identifies user to group relationships.

For directory types SecureWay, Netscape, and Domino, this field takes multiple objectclass:property pairs, delimited by a semicolon (;). In an objectclass:property pair, the objectclass value is the same objectclass defined in Group Filter, and the property is the member attribute. If the objectclass value does not match the objectclass in Group Filter, authorization might fail if groups are mapped to security roles. For more information about this syntax, see your LDAP directory service documentation.

For IBM Directory Server, iPlanet Directory Server and Active Directory, this field takes multiple (group attribute:member attribute) pairs delimited by a semicolon (;). They are used to find the group memberships of a user by enumerating all the group attributes possessed by a given user. For example, attribute pair (memberof:member) is used by Active Directory, and (ibm-allGroup:member) is used by IBM Directory Server. This field also specifies which property of an objectclass stores the list of members belonging to the group represented by the objectclass. For supported LDAP directory servers, see "Supported directory services".

Data type: String

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 whether to use the filter certificate mapping property to specify the LDAP filter, which is used to map attributes in the client certificate to entries in the LDAP registry.

To enable this field, click **CERTIFICATE_FILTER** for the certificate mapping. If more than one LDAP entry matches the filter specification at run time, then authentication fails because it results in an ambiguous match. The syntax or structure of this filter is: LDAP attribute=\${Client certificate attribute} (for example, uid=\${SubjectCN}). The left side of the filter specification is an LDAP attribute that depends on the schema that your LDAP server is configured to use. The right side of the filter specification is one of the public attributes in your client certificate. The right side must begin with a dollar sign (\$) and open bracket (()) and end with a close bracket ()). You can use the following certificate attribute values on the right side of the filter specification. The case of the strings is important:

- \${UniqueKey}
- \${PublicKey}
- \${PublicKey}
- \${Issuer}
- \${NotAfter}
- \${NotBefore}
- \${SerialNumber}
- \${SigAlgName}
- \${SigAlgOID}
- \${SigAlgParams}
- \${SubjectCN}

- \${Version}

Data type: String

Using specific directory servers as the LDAP server:

For *Using MS Active Directory server as the LDAP server* below, note that to use Microsoft Active Directory as the LDAP server for authentication with WebSphere Application Server you must take specific steps. By default, Microsoft Active Directory does not permit anonymous LDAP queries. To create LDAP queries or to browse the directory, an LDAP client must bind to the LDAP server using the distinguished name (DN) of an account that belongs to the administrator group of the Windows system. A group membership search in the Active Directory is done by enumerating the memberof attribute possessed by a given user entry, rather than browsing through the member list in each group. If you change this default behavior to browse each group, you can change the **Group Member ID Map** field from memberof:member to group:member.

Using IBM Directory Server as the LDAP server

To use IBM Directory Server, choose **IBM Directory Server** as the directory type. For supported directory servers, refer to the article, Supported directory services. The difference between these two types is group membership lookup. It is recommended that you choose the IBM Directory Server for optimum performance during run time. In the IBM 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, rather than selecting a group and browsing through the members list. 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.

Using a Lotus Domino Server as the LDAP server

If you choose the Lotus Domino LDAP server Version 6 and the attribute short name is not defined in the schema, you can take either of the following actions:

- Change the schema to add the short name attribute.
- Change the user ID map filter to replace the short name with any other defined attribute (preferably to UID). For example, change person:shortname to person:uid.

5.1 + The userID map filter has been changed to use the **uid** attribute instead of the **shortname** attribute as the current version of Lotus Domino does not create the **shortname** attribute by default. If you want to use the **shortname** attribute, define the attribute in the schema and change the userID map filter to the following:

User ID Map : person:shortname

Roles unify entries. Roles are designed to be more efficient and easier to use for applications. For example, an application can locate the role of an entry by enumerating all the roles possessed by a given entry, rather than selecting a group and browsing through the members list. With the iPlanet Directory Server directory, WebSphere Application Server security supports groups defined by *nsRole* only. If you plan to use traditional grouping methods to group entries in the iPlanet Directory Server, select **Netscape** as the directory type.

5.1+ Using Sun ONE Directory Server as the LDAP server

5.1+ You can choose **Sun ONE Directory Server** for your Sun ONE Directory Server system. For supported directory servers, refer to the article, Supported directory services. In Sun ONE Directory Server, the default object class is `groupOfUniqueName` when you create a group. For better performance, WebSphere Application Server uses the user object to locate the user group membership from the `nsRole` attribute. Thus, create the group from the role. If you want to use `groupOfUniqueName` to search groups, specify your own filter setting. Roles unify entries. Roles are designed to be more efficient and easier to use for applications. For example, an application can locate the role of an entry by enumerating all the roles possessed by a given entry, rather than selecting a group and browsing through the members list.

Using Microsoft Active Directory server as the LDAP server

To set up Microsoft Active Directory as your LDAP server, complete the following steps.

1. Determine the full DN and password of an account in the **administrators** group. For example, if the Active Directory administrator creates an account in the Users folder of the Active Directory Users and Computers Windows control panel and the DNS domain is `ibm.com`, the resulting DN has the following structure:

```
cn=<adminUsername>, cn=users, dc=ibm,  
dc=com
```

2. Determine the short name and password of any account in the Microsoft Active Directory. This password does not have to be the same account that is used in the previous step.
3. Use the WebSphere Application Server administrative console to set up the information needed to use Microsoft Active Directory:
 - a. Start the administrative server for the domain, if necessary.
 - b. On the administrative console, click **Security** on the left navigation panel.
 - c. Click the **Authentication mechanisms** tabbed page. Select **Lightweight Third Party Authentication (LTPA)** as the authentication mechanism.
 - d. Enter the following information in the LDAP settings fields:
 - **Security Server ID:** The short name of the account chosen in 2
 - **Security Server Password:** The password of the account chosen in step 2
 - **Directory Type:** Active Directory
 - **Host:** The domain name service (DNS) name of the machine running Microsoft Active Directory
 - **Base Distinguished Name:** The domain components of the DN of the account chosen in step 1. For example: `dc=ibm, dc=com`
 - **Distinguished Name:** The full DN of the account chosen in step 1. For example: `cn=<adminUsername>, cn=users, dc=ibm, dc=com`
 - **Bind Password:** The password of the account chosen in step 1
 - e. Click **OK** to save the changes.
 - f. Stop and restart the administrative server so that the changes take effect.

Configuring Lightweight Directory Access Protocol user registries

Review the article on Lightweight Directory Access Protocol (LDAP) before beginning this task.

1. In the administrative console, click **Security > User Registries > LDAP** in the left navigation panel.
2. Enter a valid user name in the **Server User ID** field. You can either enter the complete distinguished name (DN) of the user or the short name of the user as defined by the User Filter in the Advanced LDAP settings panel. For example, enter the user ID for Netscape.
3. Enter the password of the user in the Server User Password field.
4. Select the type of LDAP server that is used from the **Type** list. The type of LDAP server determines the default filters that are used by the WebSphere Application Server. These default filters change the **Type** field to **Custom**, which indicates that custom filters are used. This action occurs after you click **OK** or **Apply** in the Advanced LDAP settings panel. Choose the **Custom** type from the list and modify the user and group filters to use other LDAP servers, if required. If either the IBM Directory Server or the iPlanet Directory Server is selected, also select the **Ignore Case** field.
5. Enter the fully qualified host name of the LDAP server in the **Host** field.
6. 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.
7. Enter the Base distinguished name (DN) in the **Base Distinguished Name** field. The Base DN indicates the starting point for searches in this LDAP directory server. For example, for a user with a DN of `cn=John Doe, ou=Rochester, o=IBM, c=US`, specify the Base DN as any of the following options (assuming a suffix of `c=us`): `ou=Rochester, o=IBM, c=us` or `o=IBM c=us` or `c=us`. This field can be case sensitive. Match the case in your directory server. This field is required for all LDAP directories except the Domino Directory. The Base DN field is optional for the Domino server.
8. Enter the Bind DN name in the **Bind Distinguished Name** field, if necessary. The Bind DN is required if anonymous binds are not possible on the LDAP server to obtain user and group information. If the LDAP server is set up to use anonymous binds, leave this field blank.
9. Enter the password corresponding to the Bind DN in the **Bind password** field, if necessary.
10. Modify the **Search Time Out** value if required. This timeout value is the maximum amount of time that the LDAP server waits to send a response to the product client before aborting the request. The default is 120 seconds.
11. Disable the **Reuse Connection** field only if you use routers to send requests to multiple LDAP servers, and if the routers do not support affinity. Leave this field enabled for all other situations.
12. Enable the **Ignore Case** flag, if required. When this flag is enabled, the authorization check is case insensitive. Normally, an authorization check involves checking the complete DN of a user, which is unique in the LDAP server and is case sensitive. However, when using either the IBM Directory Server or the iPlanet Directory Server LDAP servers, this flag needs enabling because the group information obtained from the LDAP servers is not consistent in case. This inconsistency only effects the authorization check.

13. Enable Secure Sockets Layer (SSL) if the communication to the LDAP server is through SSL. For more information on setting up LDAP for SSL, refer to Configuring SSL for LDAP clients.
14. If SSL is enabled, select the appropriate SSL alias configuration from the list in the **SSL configuration** field.
15. Click **OK**. The validation of the user, password, and the setup do not take place in this panel. Validation is only done when you click **OK** or **Apply** in the **Global Security** panel. If you are enabling security for the first time, complete the remaining steps and go to the **Global Security** panel. Select **LDAP** as the Active User Registry. If security is already enabled, but information on this panel changes, go to the **Global Security** panel and click **OK** or **Apply** to validate your changes. If your changes are not validated, the server might not come up.

Sets the LDAP registry configuration.

This step is required to set up the LDAP registry. This step is required as part of enabling security in the WebSphere Application Server.

1. If you are enabling security, complete the remaining steps. As the final step, validate this setup by clicking **OK** or **Apply** in the Global Security panel.
2. Save, stop, and restart all the product servers (cell, nodes and all the application servers) for changes in this panel to take effect.
3. If the server comes up without any problems the setup is correct.

Configuring Lightweight Directory Access Protocol search filters:

The 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 **OK** or **Apply** is clicked) the directory type in the **LDAP Registry** panel changes to custom, which indicates that custom filters are used. Also, you can develop filters to support any additional type of LDAP server. The effort to support additional LDAP directories is optional and other LDAP directory types are not supported.

1. In the administrative console, click **Security > User Registries > LDAP** in the left navigation panel. Click **Advanced LDAP Setting** in Additional Properties.
2. Modify the **User** filter, if necessary. The user filter is used for searching the registry for users and is typically used for the security role to user assignment. Also, the filter is used to authenticate a user using the attribute specified in the filter. The filter specifies the property used to look up users in the directory service. In the following example, the property that is assigned to %v, which is the short name of the user, must be a unique key. Two LDAP entries with the same object class cannot have the same short name. To look up users based on their user IDs (uid) and to use the inetOrgPerson object class, specify the following syntax:

```
(&(uid=%v)(objectclass=inetOrgPerson)
```

For more information about this syntax, see the LDAP directory service documentation.

3. Modify the **Group** filter, if necessary. The group filter is used in searching the registry for groups and is typically used for the security role to group assignment. Also, the filter is used to specify the property by which to look up groups in the directory service. In the following example, the property that is assigned to %v, which is the short name of the group, must be a unique key.

Two LDAP entries with the same object class cannot have the same short name. To look up groups based on their common names (CN) and to use either the `groupOfNames` or the `groupOfUniqueNames` object class, specify the following syntax:

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

For more information about this syntax, see the LDAP directory service documentation.

4. Modify the **User ID map** filter, if necessary. This filter maps the short name of a user to an LDAP entry. It specifies the piece of information that represents users when these users are displayed with their short names. For example, to display entries of the type object class = `inetOrgPerson` by their IDs, specify `inetOrgPerson:uid`. This field takes multiple `objectclass:property` pairs delimited by a semicolon (;). To provide a consistent value for methods like `getCallerPrincipal()`, `getUserPrincipal()` the short name obtained by using this filter is used. For example, the user `CN=Bob Smith, ou=austin.ibm.com, o=IBM, c=US` can log in using any attributes that are defined (for example, e-mail address, social security number, and so on) but when these methods are called, the user ID **bob** is returned no matter how the user logs in.
5. Modify the **Group ID Map** filter, if necessary. This filter maps the short name of a group to an LDAP entry. It specifies the piece of information that represents groups when groups display. For example, to display groups by their names, specify `*:cn`. The (*) is a wildcard character that searches on any object class in this case. This field takes multiple `objectclass:property` pairs delimited by a semicolon (;).
6. Modify the **Group Member ID Map** filter, if necessary. This filter identifies user to group memberships. For SecureWay, Netscape, and Domino directory types, this field is used to query all the groups that match the specified object classes to find if the user is contained in the attribute specified. For example, to get all the users belonging to groups with the `groupOfNames` object class and the users contained in the member attributes, specify `groupOfNames:member`. This syntax which property of an objectclass stores the list of members belonging to the group represented by the objectclass. This field takes multiple `objectclass:property` pairs delimited by a semicolon (;). For more information about this syntax, see the LDAP directory service documentation.

For the IBM Directory Server, iPlanet Directory Server, and Active Directory, this field is used to query all users in a group by using the information stored in the user object (instead of querying all the groups individually to find if the user exists in that group). For example, the `memberof:member` filter (for Active Directory) is used to get the `memberof` attribute of the user object to get all the groups to which the user belongs. The `member` attribute is used to get all the users in a group using the group object. Using the user object to obtain the group information is expected to improve performance.
7. Modify the **Certificate Map Mode**, if necessary. You can use the X.590 certificates for user authentication when LDAP is selected as the user registry. This field is used to indicate whether to map the X.509 certificates into an LDAP directory user by **EXACT_DN** or **CERTIFICATE_FILTER**. If **EXACT_DN** is selected, the DN in the certificate must exactly match the user entry in the LDAP server (including case and spaces). Use the **Ignore Case** field in the LDAP settings to make the authorization case insensitive. If you select **CERTIFICATE_FILTER**, fill in the appropriate certificate filter (in the next field) to use for mapping the certificate to a user in LDAP.
8. If you specify the filter certificate mapping in step 7, use this property to 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.

9. Click **Apply**.

When any LDAP user or group filter is modified in the Advanced LDAP Settings panel click **Apply**. Clicking **OK** navigates you to the LDAP User Registry panel, which contains the previous LDAP directory type, rather than the custom LDAP directory type. Clicking **OK** or **Apply** in the LDAP User Registry panel saves the back-level LDAP directory type and the default filters of that directory. This action overwrites any changes to the filters that you made. To avoid overwriting changes, you can take either of the following actions:

- Click **Apply** in the Advanced LDAP Settings panel. To proceed to another panel, use the left navigation. Using the navigation to access the LDAP User Registry panel changes the directory type to Custom.
- Choose **Custom** type from the LDAP User Registry panel. Click **Apply** and then change the filters by clicking the **Advanced LDAP Settings** panel. After you complete your changes, click **Apply** or **OK**.

The validation of the changes (if any) does not take place in this panel. Validation is done when you click **OK** or **Apply** in the Global Security panel. If you are in the process of enabling security for the first time, complete the remaining steps and go to the Global Security panel. Select **LDAP** as the Active User Registry. If security already is enabled and any information on this panel changes, go to the Global Security panel and click **OK** or **Apply** to validate your changes. If your changes are not validated, the server might not come up.

Sets the LDAP search filters.

This step is required to modify existing user and group filters for a particular LDAP directory type. It is also used to set up certificate filters to map certificates to entries in the LDAP server.

1. If you are enabling security, complete the remaining steps. As the final step make sure that you validate this setup by clicking **OK** or **Apply** in the Global Security panel.

2. Save, stop, and start all the product servers (cell, nodes and all the application servers) for any changes in this panel to become effective.
3. After the server comes up, go through all the security-related tasks (getting users, getting groups and so on) to verify that the changes to the filters function.

Custom user registries

A *custom user registry* is a customer-implemented user registry, which implements the `UserRegistry` Java interface as provided by the product. A custom-implemented user registry can support virtually any type of an account repository from a relational database, flat file, and so on. The custom user registry provides considerable flexibility in adapting product security to various environments where some form of a user registry, other than Lightweight Directory Access Protocol (LDAP) or Local Operating System (LocalOS), already exists in the operational environment.

WebSphere Application Server security provides an implementation that uses various local operating system-based registries (Windows, AIX, Solaris, Linux) and various Lightweight Directory Access Protocol (LDAP)-based registries. However, situations can exist where your user and group data resides in other repositories or custom registries (a database, for example) and moving this information to either a LocalOS or an LDAP registry implementation might not be feasible. For these situations WebSphere Application Server security provides a service provider interface (SPI) that you can implement to interact with your current registry. The SPI is the `UserRegistry` interface. This interface has a set of methods to implement for the product security to interact with your registries for all security-related tasks. The LocalOS and LDAP registry implementations that are provided also implement this interface. Custom user registries are sometimes called the *pluggable user registries* or *custom registries* for short. Your custom user registry implementation is expected to be thread-safe.

The *UserRegistry interface* is a collection of methods required to authenticate individual users using either password or certificates and to collect information about the user (privilege attributes) for authorization purposes. This interface also includes methods that obtain user and group information so that they can be given access to resources. When implementing the methods in the interface, you must decide how to map the information manipulated by the `UserRegistry` interface to the information in your registry.

Make sure that your implementation of the custom registry does not depend on any WebSphere Application Server components such as data sources, enterprise beans, and so on. Do not have this dependency because security is initialized and enabled prior to most of the other WebSphere Application Server components during startup. If your previous implementation used these components, make a change that eliminates the dependency. For example, if your previous implementation used data sources to connect to a database, use Java database connectivity (JDBC) to connect to the database.

The methods in the `UserRegistry` interface operate on the following information for users:

User Security Name

The user name, which is similar to the user name in the Windows systems and the UNIX systems Local OS registries. This name is used to log in when prompted by a secured application. By default, the Enterprise JavaBean (EJB) method `getCallerPrincipal` and the servlet methods `getRemoteUser` and `getUserPrincipal` return this name. The user security name is also referred to as *userSecurityName*, *userName* or *user name*.

Unique ID

This ID represents a unique identifier for the user. The UserRegistry interface requires this identifier to be unique. The unique ID similar to the system ID (SID) in Windows systems, Unique ID (UID) in UNIX systems, distinguished name (DN) in Lightweight Directory Authentication Protocol (LDAP). This ID is also referred to as *uniqueUserId*. The unique ID is used to make the authorization decisions for protected resources.

Display name

This name is an optional string that describes a user, and it is similar to the *FullName* attribute in Windows operating systems. The implementation can use display names for informational purposes only; these names are not required to exist or to be unique. The user interface can use the display name to present more information about the user.

Group Security name

This name, which represents the security group, is also referred to as *groupSecurityName*, *groupName* and *group name*.

Unique ID

The unique ID is the identifier for a group. This name is also referred to as *uniqueGroupId*.

Display name

The display name is an optional string that describes a group.

The article on UserRegistry interface describes each of the methods in the UserRegistry interface that need implementing. An explanation of each of the methods and their usage in the Sample and any changes from the Version 4 interface are provided. The Related references section provides links to all other custom user registries documentation, including a file-based registry Sample. The Sample provided is very simple and is intended to familiarize you with this feature. Do not use this sample in an actual production environment.

Configuring custom user registries

Before you begin this task, implement and build the UserRegistry interface. For more information on developing custom user registries refer to the article, Developing custom user registries. The following steps are required to configure custom user registries through the administrative console.

1. In the administrative console, click **Security > User Registries > Custom** in the left navigation panel.
2. Enter a valid user name in the **Server User ID** field.
3. Enter the password of the user in the **Server User Password** field.
4. Enter the full name of the location of the implementation class file in the **Custom Registry Classname** field a dot-separated file name. For the sample, this file name is `com.ibm.websphere.security.FileRegistrySample`. The file exists in the WebSphere Application Server class path (preferably in the `install_root/lib/ext` directory). This file exists in all the product processes. So, if you are operating in a Network Deployment environment, this file exists in the cell class path and in all of the node class paths.
5. Select the **Ignore Case** check box for the authorization to perform a case insensitive check. Enabling this option is necessary only when your registry is case insensitive and does not provide a consistent case when queried for users and groups.
6. Click **Apply** if you have any other additional properties to enter for the registry initialization. Otherwise click **OK** and complete the steps required to turn on security.

- If you need to enter additional properties to initialize your implementation, click **Custom Properties** at the bottom of the panel. Click **New**. Enter the property name and value. Click **OK**. Repeat this step to add other additional properties. For the sample, enter the following two properties: (assuming that the `users.props` and the `groups.props` file are in the `myDir` directory under the product installation directory).

| Property name | Property value |
|---------------|--|
| usersFile | \${USER_INSTALL_ROOT}/myDir/users.props |
| groupsFile | \${USER_INSTALL_ROOT}/myDir/groups.props |

The Description, Required, and Validation Expression fields are not used and you can leave them blank.

Note: In a Network Deployment environment where multiple WebSphere Application Server processes exist (cell and multiple nodes in different machines), these properties are available for each process. Use the relative name `${USER_INSTALL_ROOT}` to locate any files, as this name expands to the product installation directory. If this name is not used, ensure that the files exist in the same location in all the nodes.

This step is required to set up the custom user registry and to enable security in WebSphere Application Server.

- Complete the remaining steps, if you are enabling security.
- After security is turned on, save, stop, and start all the product servers (cell, nodes and all the application servers) for any changes in this panel to take effect.
- If the server comes up without any problems, the setup is correct.
- Validate the user and password by clicking **OK** or **Apply** in the Global Security panel. Save, synchronize (in the cell environment), stop and start all the product servers.

UserRegistry.java files:

```
// 5639-D57, 5630-A36, 5630-A37, 5724-D18
// (C) COPYRIGHT International Business Machines Corp. 1997, 2003
// All Rights Reserved * Licensed Materials - Property of IBM
//
// DESCRIPTION:
//
// This file is the UserRegistry interface that custom registries in WebSphere
// Application Server implement to enable WebSphere security to use the custom
// registry.
//
package com.ibm.websphere.security;

import java.util.*;
import java.rmi.*;
import java.security.cert.X509Certificate;
import com.ibm.websphere.security.cred.WSCredential;/**
 * Implementing this interface enables WebSphere Application Server Security
 * to use custom registries. This interface extends java.rmi.Remote because the
 * registry can be in a remote process.
 *
 * Implementation of this interface must provide implementations for:
 *
 * initialize(java.util.Properties)
 * checkPassword(String,String)
 * mapCertificate(X509Certificate[])
 * getRealm
 * getUsers(String,int)
```

```

* getUserDisplayName(String)
* getUniqueUserId(String)
* getUserSecurityName(String)
* isValidUser(String)
* getGroups(String,int)
* getGroupDisplayName(String)
* getUniqueGroupId(String)
* getUniqueGroupIds(String)
* getGroupSecurityName(String)
* isValidGroup(String)
* getGroupsForUser(String)
* getUsersForGroup(String,int)
* createCredential(String)
**/

public interface UserRegistry extends java.rmi.Remote
{

    /**
     * Initializes the registry. This method is called when creating the
     * registry.
     *
     * @param props the registry-specific properties with which to
     *             initialize the custom registry
     * @exception CustomRegistryException
     *             if there is any registry specific problem
     * @exception RemoteException
     *             as this extends java.rmi.Remote
     **/
    public void initialize(java.util.Properties props)
        throws CustomRegistryException,
               RemoteException; /**
     * Checks the password of the user. This method is called to authenticate a
     * user when the user's name and password are given.
     *
     * @param userSecurityName the name of user
     * @param password the password of the user
     * @return a valid userSecurityName. Normally this is
     *         the name of same user whose password was checked but if the
     *         implementation wants to return any other valid
     *         userSecurityName in the registry it can do so
     * @exception CheckPasswordFailedException if userSecurityName/
     *         password combination does not exist in the registry
     * @exception CustomRegistryException if there is any registry specific
     *         problem
     * @exception RemoteException as this extends java.rmi.Remote
     **/
    public String checkPassword(String userSecurityName, String password)
        throws PasswordCheckFailedException,
               CustomRegistryException,
               RemoteException; /**
     * Maps a certificate (of X509 format) to a valid user in the registry.
     * This is used to map the name in the certificate supplied by a browser
     * to a valid userSecurityName in the registry
     *
     * @param cert the X509 certificate chain
     * @return the mapped name of the user userSecurityName
     * @exception CertificateMapNotSupportedException if the particular
     *         certificate is not supported.
     * @exception CertificateMapFailedException if the mapping of the
     *         certificate fails.
     * @exception CustomRegistryException if there is any registry specific
     *         problem
     * @exception RemoteException as this extends java.rmi.Remote
     **/
    public String mapCertificate(X509Certificate[] cert)
        throws CertificateMapNotSupportedException,

```

```

        CertificateMapFailedException,
        CustomRegistryException,
        RemoteException; /**
* Returns the realm of the registry.
*
* @return the realm. The realm is a registry-specific string indicating
*         the realm or domain for which this registry
*         applies. For example, for OS400 or AIX this would be the
*         host name of the system whose user registry this object
*         represents.
*         If null is returned by this method realm defaults to the
*         value of "customRealm". It is recommended that you use
*         your own value for realm.
* @exception CustomRegistryException if there is any registry specific
*         problem
* @exception RemoteException as this extends java.rmi.Remote
**/
public String getRealm()
    throws CustomRegistryException,
           RemoteException; /**
* Gets a list of users that match a pattern in the registry.
* The maximum number of users returned is defined by the limit
* argument.
* This method is called by administrative console and by scripting (command
* line) to make available the users in the registry for adding them (users)
* to roles.
*
* @parameter pattern the pattern to match. (For example., a* will match all
*         userSecurityNames starting with a)
* @parameter limit the maximum number of users that should be returned.
* This is very useful in situations where there are thousands of
*         users in the registry and getting all of them at once is not
*         practical. A value of 0 implies get all the users and hence
*         must be used with care.
* @return a Result object that contains the list of users
*         requested and a flag to indicate if more users exist.
* @exception CustomRegistryException if there is any registry specific
*         problem
* @exception RemoteException as this extends java.rmi.Remote
**/
public Result getUsers(String pattern, int limit)
    throws CustomRegistryException,
           RemoteException; /**
* Returns the display name for the user specified by userSecurityName.
*
* This method is called only when the user information displays
* (information purposes only, for example, in the administrative console) and not used
* in the actual authentication or authorization purposes. If there are no
* display names in the registry return null or empty string.
*
* In WebSphere Application Server Version 4.0 custom registry, if you had a display
* name for the user and if it was different from the security name, the display name
* was returned for the EJB methods getCallerPrincipal() and the servlet methods
* getUserPrincipal() and getRemoteUser().
* In WebSphere Application Server Version 5.0 for the same methods the security
* name is returned by default. This is the recommended way as the display name
* is not unique and might create security holes.
* However, for backward compatibility if one needs the display name to
* be returned set the property WAS_UseDisplayName to true.
*
* See the documentation for more information.
*
* @parameter userSecurityName the name of the user.
* @return the display name for the user. The display name
*         is a registry-specific string that represents a descriptive, not
*         necessarily unique, name for a user. If a display name does
*         not exist return null or empty string.

```

```

* @exception EntryNotFoundException if userSecurityName does not exist.
* @exception CustomRegistryException if there is any registry specific
*     problem
* @exception RemoteException as this extends java.rmi.Remote
**/
public String getUserDisplayName(String userSecurityName)
    throws EntryNotFoundException,
           CustomRegistryException,
           RemoteException; /**
* Returns the unique ID for a userSecurityName. This method is called when
* creating a credential for a user.
*
* @parameter userSecurityName the name of the user.
* @return the unique ID of the user. The unique ID for an user is
* the stringified form of some unique, registry-specific, data
* that serves to represent the user. For example, for the UNIX
* user registry, the unique ID for a user can be the UID.
* @exception EntryNotFoundException if userSecurityName does not exist.
* @exception CustomRegistryException if there is any registry specific
*     problem
* @exception RemoteException as this extends java.rmi.Remote
**/
public String getUniqueUserId(String userSecurityName)
    throws EntryNotFoundException,
           CustomRegistryException,
           RemoteException; /**
* Returns the name for a user given its unique ID.
*
* @parameter uniqueUserId the unique ID of the user.
* @return the userSecurityName of the user.
* @exception EntryNotFoundException if the uniqueUserID does not exist.
* @exception CustomRegistryException if there is any registry specific
*     problem
* @exception RemoteException as this extends java.rmi.Remote
**/
public String getUserSecurityName(String uniqueUserId)
    throws EntryNotFoundException,
           CustomRegistryException,
           RemoteException;

/**
* Determines if the userSecurityName exists in the registry
*
* @parameter userSecurityName the name of the user
* @return true if the user is valid. false otherwise
* @exception CustomRegistryException if there is any registry specific
*     problem
* @exception RemoteException as this extends java.rmi.Remote
**/
public boolean isValidUser(String userSecurityName)
    throws CustomRegistryException,
           RemoteException;

/**
* Gets a list of groups that match a pattern in the 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 could
* be the GID.
* @exception EntryNotFoundException if groupSecurityName does not exist.
* @exception CustomRegistryException if there is any registry specific
*         problem
* @exception RemoteException as this extends java.rmi.Remote
**/
public String getUniqueGroupId(String groupSecurityName)
    throws EntryNotFoundException,
           CustomRegistryException,
           RemoteException;

/**
* Returns the unique IDs for all the groups that contain the unique ID of
* a user.
* Called during creation of a user's credential.
*
* @parameter uniqueUserId the unique ID of the user.
* @return a list of all the group unique IDs that the unique user ID
* belongs to. The unique ID for an entry is the stringified
* form of some unique, registry-specific, data that serves
* to represent the entry. For example, for the
* UNIX user registry, the unique ID for a group could be the GID
* and the unique ID for the user could be the UID.
* @exception EntryNotFoundException if unique user ID does not exist.

```

```

* @exception CustomRegistryException if there is any registry specific
*     problem
* @exception RemoteException as this extends java.rmi.Remote
**/
public List getUniqueGroupIds(String uniqueUserId)
    throws EntryNotFoundException,
           CustomRegistryException,
           RemoteException;

/**
* Returns the name for a group given its unique ID.
*
* @parameter uniqueGroupId the unique ID of the group.
* @return the name of the group.
* @exception EntryNotFoundException if the uniqueGroupId does not exist.
* @exception CustomRegistryException if there is any registry-specific
*     problem
* @exception RemoteException as this extends java.rmi.Remote
**/
public String getGroupSecurityName(String uniqueGroupId)
    throws EntryNotFoundException,
           CustomRegistryException,
           RemoteException;

/**
* Determines if the groupSecurityName exists in the registry
*
* @parameter groupSecurityName the name of the group
* @return true if the groups exists, false otherwise
* @exception CustomRegistryException if there is any registry specific
*     problem
* @exception RemoteException as this extends java.rmi.Remote
**/
public boolean isValidGroup(String groupSecurityName)
    throws CustomRegistryException,
           RemoteException;

/**
* Returns the securityNames of all the groups that contain the user
*
* This method is called by administrative console and scripting
* (command line) to verify the user entered for RunAsRole mapping belongs
* to that role in the roles to user mapping. Initially, the check is done
* to see if the role contains the user. If the role does not contain the user
* explicitly, this method is called to get the groups that this user
* belongs to so that checks are made on the groups that the role contains.
*
* @parameter userSecurityName the name of the user
* @return a List of all the group securityNames that the user
*     belongs to.
* @exception EntryNotFoundException if user does not exist.
* @exception CustomRegistryException if there is any registry specific
*     problem
* @exception RemoteException as this extends java.rmi.Remote
**/
public List getGroupsForUser(String userSecurityName)
    throws EntryNotFoundException,
           CustomRegistryException,
           RemoteException;

/**
* Gets a list of users in a group.
*
* The maximum number of users returned is defined by the limit
* argument.
*
* This method is used by the WebSphere Business Integration

```

```

* Server Foundation process choreographer when staff assignments
* are modeled using groups.
*
* In rare situations if you are working with a registry where getting all of
* the users from any of your groups is not practical (for example if
* a large number of users exist) you can throw the NotImplementedException
* for that particular groups. Make sure that if the WebSphere Business
* Integration Server Foundation Process Choreographer is installed (or
* if installed later) that are not modeled using these particular groups.
* If no concern exists about the staff assignments returning the users from
* groups in the registry it is recommended that this method be implemented
* without throwing the 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 throw this exception in rare situations
*         if it is not practical to get this information for any of the
*         groups from the registry.
* @exception EntryNotFoundException if the group does not exist in
*         the registry
* @exception CustomRegistryException if any registry-specific
*         problem occurs
* @exception RemoteException as this extends java.rmi.Remote interface
**/
public Result getUsersForGroup(String groupSecurityName, int limit)
    throws NotImplementedException,
        EntryNotFoundException,
        CustomRegistryException,
        RemoteException;

/**
* This method is implemented internally by the WebSphere Application Server
* code in this release. This method is not called for the custom registry
* implementations for this release. Return null in the implementation.
*
* Note that because this method is not called you can also return the
* NotImplementedException as the previous documentation says.
*
**/
public com.ibm.websphere.security.cred.WSCredential
    createCredential(String userSecurityName)
    throws NotImplementedException,
        EntryNotFoundException,
        CustomRegistryException,
        RemoteException;
}

```

FileRegistrySample.java file for WebSphere Application Server Version 5.0.2:

The user and group information required by this sample is contained in the users.props and groups.props files.

The contents of the FileRegistrySample.java file:

```

//
// 5639-D57, 5630-A36, 5630-A37, 5724-D18
// (C) COPYRIGHT International Business Machines Corp. 1997, 2003
// All Rights Reserved * Licensed Materials - Property of IBM
////-----
// This program may be used, executed, copied, modified and distributed
// without royalty for the purpose of developing, using, marketing, or
// distributing.

```

```

//-----
//

// This sample is for the custom user registry feature in WebSphere
// Application Server.

import java.util.*;
import java.io.*;
import java.security.cert.X509Certificate;
import com.ibm.websphere.security.*;
/**
 * The main purpose of this sample is to demonstrate the use of the
 * custom user registry feature available in WebSphere Application Server. This
 * sample is a file-based registry sample where the users and the groups
 * information is listed in files (users.props and groups.props). As such
 * simplicity and not the performance was a major factor behind this. This
 * sample should be used only to get familiarized with this feature. An
 * actual implementation of a realistic registry should consider various
 * factors like performance, scalability, thread safety, and so on.
 */
public class FileRegistrySample implements UserRegistry {

    private static String USERFILENAME = null;
    private static String GROUPFILENAME = null;

    /** Default Constructor */
    public FileRegistrySample() throws java.rmi.RemoteException {
    }

    /**
     * Initializes the registry. This method is called when creating the
     * registry.
     *
     * @param props - The registry-specific properties with which to
     *                initialize the custom registry
     * @exception CustomRegistryException
     *                if there is any registry-specific problem
     */
    public void initialize(java.util.Properties props)
        throws CustomRegistryException {
        try {
            /* try getting the USERFILENAME and the GROUPFILENAME from
             * properties that are passed in (For example, from the
             * administrative console). Set these values in the administrative
             * console. Go to the special custom settings in the custom
             * user registry section of the Authentication panel.
             * For example:
             * usersFile c:/temp/users.props
             * groupsFile c:/temp/groups.props
             */
            if (props != null) {
                USERFILENAME = props.getProperty("usersFile");
                GROUPFILENAME = props.getProperty("groupsFile");
            }

        } catch (Exception ex) {
            throw new CustomRegistryException(ex.getMessage(), ex);
        }

        if (USERFILENAME == null || GROUPFILENAME == null) {
            throw new CustomRegistryException("users/groups information missing");
        }
    }

    /**
     * Checks the password of the user. This method is called to authenticate
     * a user when the user's name and password are given.
     *
     */
}

```



```

* @param userSecurityName the name of user
* @param password the password of the user
* @return a valid userSecurityName. Normally this is
*         the name of same user whose password was checked
*         but if the implementation wants to return any other
*         valid userSecurityName in the registry it can do so
* @exception CheckPasswordFailedException if userSecurityName/
*         password combination does not exist
*         in the registry
* @exception CustomRegistryException if there is any registry-
*         specific problem
**/
public String checkPassword(String userSecurityName, String passwd)
    throws PasswordCheckFailedException,
           CustomRegistryException {
    String s,userName = null;
    BufferedReader in = null;

    try {
        in = fileOpen(USERFILENAME);
        while ((s=in.readLine())!=null)
        {
            if (!(s.startsWith("#") || s.trim().length() <=0 )) {
                int index = s.indexOf(":");
                int index1 = s.indexOf(":",index+1);
                // check if the userSecurityName:passwd combination exists
                if ((s.substring(0,index)).equals(userSecurityName) &&
                    s.substring(index+1,index1).equals(passwd)) {
                    // Authentication successful, return the userId.
                    userName = userSecurityName;
                    break;
                }
            }
        }
    } catch(Exception ex) {
        throw new CustomRegistryException(ex.getMessage(),ex);
    } finally {
        fileClose(in);
    }

    if (userName == null) {
        throw new PasswordCheckFailedException("Password check failed for user: "
            + userSecurityName);
    }

    return userName;
} /**
* Maps a X.509 format certificate to a valid user in the registry.
* This is used to map the name in the certificate supplied by a browser
* to a valid userSecurityName in the registry
*
* @param cert the X509 certificate chain
* @return The mapped name of the user userSecurityName
* @exception CertificateMapNotSupportedException if the
*         particular certificate is not supported.
* @exception CertificateMapFailedException if the mapping of
*         the certificate fails.
* @exception CustomRegistryException if there is any registry
*         -specific problem
**/
public String mapCertificate(X509Certificate[] cert)
    throws CertificateMapNotSupportedException,
           CertificateMapFailedException,
           CustomRegistryException {
    String name=null;
    X509Certificate cert1 = cert[0];

```

```

    try {
        // map the SubjectDN in the certificate to a userID.
        name = cert1.getSubjectDN().getName();
    } catch(Exception ex) {
        throw new CertificateMapNotSupportedException(ex.getMessage(),ex);
    }

    if(!isValidUser(name)) {
        throw new CertificateMapFailedException("user: " + name
        + " is not valid");
    }
    return name;
} /**
 * Returns the realm of the registry.
 *
 * @return the realm. The realm is a registry-specific string
 * indicating the realm or domain for which this registry
 * applies. For example, for OS/400 or AIX this would be
 * the host name of the system whose user registry this
 * object represents. If null is returned by this method,
 * realm defaults to the value of "customRealm". It is
 * recommended that you use your own value for realm.
 *
 * @exception CustomRegistryException if there is any registry-
 * specific problem
 **/
public String getRealm()
    throws CustomRegistryException {
    String name = "customRealm";
    return name;
} /**
 * Gets a list of users that match a pattern in the registry.
 * The maximum number of users returned is defined by the limit
 * argument.
 * This method is called by the administrative console and scripting
 * (command line) to make the users in the registry available for
 * adding them (users) to roles.
 *
 * @param pattern the pattern to match. (For example, a* will
 * match all userSecurityNames starting with a)
 * @param limit the maximum number of users that should be
 * returned. This is very useful in situations where
 * there are thousands of users in the registry and
 * getting all of them at once is not practical. The
 * default is 100. A value of 0 implies get all the
 * users and hence must be used with care.
 * @return a Result object that contains the list of users
 * requested and a flag to indicate if more users
 * exist.
 * @exception CustomRegistryException if there is any registry-
 * specificproblem
 **/
public Result getUsers(String pattern, int limit)
    throws CustomRegistryException {
    String s;
    BufferedReader in = null;
    List allUsers = new ArrayList();
    Result result = new Result();
    int count = 0;
    int newLimit = limit+1;
    try {
        in = fileOpen(USERFILENAME);
        while ((s=in.readLine())!=null)
        {
            if (!(s.startsWith("#") || s.trim().length() <=0 )) {
                int index = s.indexOf(":");
                String user = s.substring(0,index);

```

```

        if (match(user,pattern)) {
            allUsers.add(user);
            if (limit !=0 && ++count == newLimit) {
                allUsers.remove(user);
                result.setHasMore();
                break;
            }
        }
    }
}
} catch (Exception ex) {
    throw new CustomRegistryException(ex.getMessage(),ex);
} finally {
    fileClose(in);
}

result.setList(allUsers);
return result;
} /**
 * Returns the display name for the user specified by
 * userSecurityName.
 *
 * This method may be called only when the user information
 * is displayed (information purposes only, for example, in
 * the administrative console) and hence not used in the actual
 * authentication or authorization purposes. If there are no
 * display names in the registry return null or empty string.
 *
 * In WebSphere Application Server 4 custom registry, if you
 * had a display name for the user and if it was different from the
 * security name, the display name was returned for the EJB
 * methods getCallerPrincipal() and the servlet methods
 * getUserPrincipal() and getRemoteUser().
 * In WebSphere Application Server Version 5, for the same
 * methods, the security name will be returned by default. This
 * is the recommended way as the display name is not unique
 * and might create security holes. However, for backward
 * compatibility if one needs the display name to be returned
 * set the property WAS_UseDisplayName to true.
 *
 * See the InfoCenter documentation for more information.
 *
 * @param    userSecurityName the name of the user.
 * @return   the display name for the user. The display
 *           name is a registry-specific string that
 *           represents a descriptive, not necessarily
 *           unique, name for a user. If a display name
 *           does not exist return null or empty string.
 * @exception EntryNotFoundException if userSecurityName
 *           does not exist.
 * @exception CustomRegistryException if there is any registry-
 *           specific problem
 */
public String getUserDisplayName(String userSecurityName)
    throws CustomRegistryException,
           EntryNotFoundException {

    String s,displayName = null;
    BufferedReader in = null;

    if(!isValidUser(userSecurityName)) {
        EntryNotFoundException nsee = new EntryNotFoundException("user: "
            + userSecurityName + " is not valid");
        throw nsee;
    }

    try {

```

```

        in = fileOpen(USERFILENAME);
        while ((s=in.readLine())!=null)
        {
            if (!(s.startsWith("#") || s.trim().length() <=0 )) {
                int index = s.indexOf(":");
                int index1 = s.lastIndexOf(":");
                if ((s.substring(0,index)).equals(userSecurityName)) {
                    displayName = s.substring(index1+1);
                    break;
                }
            }
        }
    } catch(Exception ex) {
        throw new CustomRegistryException(ex.getMessage(), ex);
    } finally {
        fileClose(in);
    }

    return displayName;
}

/**
 * Returns the unique ID for a userSecurityName. This method is called
 * when creating a credential for a user.
 *
 * @param    userSecurityName - The name of the user.
 * @return   The unique ID of the user. The unique ID for an user
 *           is the stringified form of some unique, registry-specific,
 *           data that serves to represent the user. For example, for
 *           the UNIX user registry, the unique ID for a user can be
 *           the UID.
 * @exception EntryNotFoundException if userSecurityName does not
 *           exist.
 * @exception CustomRegistryException if there is any registry-
 *           specific problem
 */
public String getUniqueUserId(String userSecurityName)
    throws CustomRegistryException,
           EntryNotFoundException {

    String s,uniqueUsrId = null;
    BufferedReader in = null;
    try {
        in = fileOpen(USERFILENAME);
        while ((s=in.readLine())!=null)
        {
            if (!(s.startsWith("#") || s.trim().length() <=0 )) {
                int index = s.indexOf(":");
                int index1 = s.indexOf(":", index+1);
                if ((s.substring(0,index)).equals(userSecurityName)) {
                    int index2 = s.indexOf(":", index1+1);
                    uniqueUsrId = s.substring(index1+1,index2);
                    break;
                }
            }
        }
    }
    } catch(Exception ex) {
        throw new CustomRegistryException(ex.getMessage(),ex);
    } finally {
        fileClose(in);
    }

    if (uniqueUsrId == null) {
        EntryNotFoundException nsee =
            new EntryNotFoundException("Cannot obtain uniqueId for user: "
                + userSecurityName);
        throw nsee;
    }
}

```

```

    }

    return uniqueUsrId;
} /**
 * Returns the name for a user given its uniqueId.
 *
 * @param    uniqueUserId - The unique ID of the user.
 * @return   The userSecurityName of the user.
 * @exception EntryNotFoundException if the unique user ID does not exist.
 * @exception CustomRegistryException if there is any registry-specific
 *         problem
 **/
public String getUserSecurityName(String uniqueUserId)
    throws CustomRegistryException,
           EntryNotFoundException {
    String s,usrSecName = null;
    BufferedReader in = null;
    try {
        in = fileOpen(USERFILENAME);
        while ((s=in.readLine())!=null)
        {
            if (!(s.startsWith("#") || s.trim().length() <=0 )) {
                int index = s.indexOf(":");
                int index1 = s.indexOf(":", index+1);
                int index2 = s.indexOf(":", index1+1);
                if ((s.substring(index1+1,index2)).equals(uniqueUserId)) {
                    usrSecName = s.substring(0,index);
                    break;
                }
            }
        }
    } catch (Exception ex) {
        throw new CustomRegistryException(ex.getMessage(), ex);
    } finally {
        fileClose(in);
    }

    if (usrSecName == null) {
        EntryNotFoundException ex =
            new EntryNotFoundException("Cannot obtain the
            user securityName for " + uniqueUserId);
        throw ex;
    }

    return usrSecName;
} /**
 * Determines if the userSecurityName exists in the registry
 *
 * @param    userSecurityName - The name of the user
 * @return   True if the user is valid; otherwise false
 * @exception CustomRegistryException if there is any registry-
 *         specific problem
 * @exception RemoteException as this extends java.rmi.Remote
 *         interface
 **/
public boolean isValidUser(String userSecurityName)
    throws CustomRegistryException {
    String s;
    boolean isValid = false;
    BufferedReader in = null;
    try {
        in = fileOpen(USERFILENAME);
        while ((s=in.readLine())!=null)
        {
            if (!(s.startsWith("#") || s.trim().length() <=0 )) {
                int index = s.indexOf(":");

```

```

        if ((s.substring(0,index)).equals(userSecurityName)) {
            isValid=true;
            break;
        }
    }
} catch (Exception ex) {
    throw new CustomRegistryException(ex.getMessage(), ex);
} finally {
    fileClose(in);
}

return isValid;
} /**
 * Gets a list of groups that match a pattern in the registry
 * The maximum number of groups returned is defined by the
 * limit argument. This method is called by administrative console
 * and scripting (command line) to make available the groups in
 * the registry for adding them (groups) to roles.
 *
 * @param      pattern the pattern to match. (For example, a* matches
 *              all groupSecurityNames starting with a)
 * @param      Limits the maximum number of groups to return
 *              This is very useful in situations where there
 *              are thousands of groups in the registry and getting all
 *              of them at once is not practical. The default is 100.
 *              A value of 0 implies get all the groups and hence must
 *              be used with care.
 * @return     A Result object that contains the list of groups
 *              requested and a flag to indicate if more groups exist.
 * @exception CustomRegistryException if there is any registry-specific
 *              problem
 */
public Result getGroups(String pattern, int limit)
    throws CustomRegistryException {
    String s;
    BufferedReader in = null;
    List allGroups = new ArrayList();      Result result = new Result();
    int count = 0;
    int newLimit = limit+1;
    try {
        in = fileOpen(GROUPFILENAME);
        while ((s=in.readLine())!=null)
        {
            if (!(s.startsWith("#") || s.trim().length() <=0 )) {
                int index = s.indexOf(":");
                String group = s.substring(0,index);
                if (match(group,pattern)) {
                    allGroups.add(group);
                    if (limit !=0 && ++count == newLimit) {
                        allGroups.remove(group);
                        result.setHasMore();
                        break;
                    }
                }
            }
        }
    }
} catch (Exception ex) {
    throw new CustomRegistryException(ex.getMessage(),ex);
} finally {
    fileClose(in);
}

result.setList(allGroups);
return result;
}

```

```

/**
 * Returns the display name for the group specified by groupSecurityName.
 * For this version of WebSphere Application Server, the only usage of
 * this method is by the clients (administrative console and scripting)
 * to present a descriptive name of the user if it exists.
 *
 * @param groupSecurityName the name of the group.
 * @return the display name for the group. The display name
 *         is a registry-specific string that represents a
 *         descriptive, not necessarily unique, name for a group.
 *         If a display name does not exist return null or empty
 *         string.
 * @exception EntryNotFoundException if groupSecurityName does
 *         not exist.
 * @exception CustomRegistryException if there is any registry-
 *         specific problem
 */
public String getGroupDisplayName(String groupSecurityName)
    throws CustomRegistryException,
           EntryNotFoundException {
    String s,displayName = null;
    BufferedReader in = null;

    if(!isValidGroup(groupSecurityName)) {
        EntryNotFoundException nsee = new EntryNotFoundException("group: "
            + groupSecurityName + " is not valid");
        throw nsee;
    }

    try {
        in = fileOpen(GROUPFILENAME);
        while ((s=in.readLine())!=null)
        {
            if (!(s.startsWith("#") || s.trim().length() <=0 )) {
                int index = s.indexOf(":");
                int index1 = s.lastIndexOf(":");
                if ((s.substring(0,index)).equals(groupSecurityName)) {
                    displayName = s.substring(index1+1);
                    break;
                }
            }
        }
    } catch(Exception ex) {
        throw new CustomRegistryException(ex.getMessage(),ex);
    } finally {
        fileClose(in);
    }

    return displayName;
}

/**
 * Returns the Unique ID for a group.
 *
 * @param groupSecurityName the name of the group.
 * @return The unique ID of the group. The unique ID for
 *         a group is the stringified form of some unique,
 *         registry-specific, data that serves to represent
 *         the group. For example, for the UNIX user registry,
 *         the unique ID might be the GID.
 * @exception EntryNotFoundException if groupSecurityName does
 *         not exist.
 * @exception CustomRegistryException if there is any registry-
 *         specific problem
 * @exception RemoteException as this extends java.rmi.Remote
 */
public String getUniqueGroupId(String groupSecurityName)

```

```

throws CustomRegistryException,
        EntryNotFoundException {
String s,uniqueGrpId = null;
BufferedReader in = null;
try {
    in = fileOpen(GROUPFILENAME);
    while ((s=in.readLine())!=null)
    {
        if (!(s.startsWith("#") || s.trim().length() <=0 )) {
            int index = s.indexOf(":");
            int index1 = s.indexOf(":", index+1);
            if ((s.substring(0,index)).equals(groupSecurityName)) {
                uniqueGrpId = s.substring(index+1,index1);
                break;
            }
        }
    }
} catch(Exception ex) {
    throw new CustomRegistryException(ex.getMessage(),ex);
} finally {
    fileClose(in);
}

if (uniqueGrpId == null) {
    EntryNotFoundException nsee =
        new EntryNotFoundException("Cannot obtain the uniqueId for group: "
        + groupSecurityName);
    throw nsee;
}

return uniqueGrpId;
}

/**
 * Returns the Unique IDs for all the groups that contain the UniqueId
 * of a user. Called during creation of a user's credential.
 *
 * @param    uniqueUserId the unique ID of the user.
 * @return    A list of all the group unique IDs that the unique user
 *            ID belongs to. The unique ID for an entry is the
 *            stringified form of some unique, registry-specific, data
 *            that serves to represent the entry. For example, for the
 *            UNIX user registry, the unique ID for a group might be
 *            the GID and the Unique ID for the user might be the UID.
 * @exception EntryNotFoundException if uniqueUserId does not exist.
 * @exception CustomRegistryException if there is any registry-specific
 *            problem
 */
public List getUniqueGroupIds(String uniqueUserId)
    throws CustomRegistryException,
           EntryNotFoundException {
String s,uniqueGrpId = null;
BufferedReader in = null;
List uniqueGrpIds=new ArrayList();
try {
    in = fileOpen(USERFILENAME);
    while ((s=in.readLine())!=null)
    {
        if (!(s.startsWith("#") || s.trim().length() <=0 )) {
            int index = s.indexOf(":");
            int index1 = s.indexOf(":", index+1);
            int index2 = s.indexOf(":", index1+1);
            if ((s.substring(index1+1,index2)).equals(uniqueUserId)) {
                int lastIndex = s.lastIndexOf(":");
                String subs = s.substring(index2+1,lastIndex);
                StringTokenizer st1 = new StringTokenizer(subs, ",");
                while (st1.hasMoreTokens())

```



```

        uniqueGrpIds.add(st1.nextToken());
        break;
    }
}
}
} catch(Exception ex) {
    throw new CustomRegistryException(ex.getMessage(),ex);
} finally {
    fileClose(in);
}

return uniqueGrpIds;
}

/**
 * Returns the name for a group given its uniqueId.
 *
 * @param    uniqueGroupId the unique ID of the group.
 * @return   The name of the group.
 * @exception EntryNotFoundException if the uniqueGroupId does
 *           not exist.
 * @exception CustomRegistryException if there is any registry-
 *           specific problem
 */
public String getGroupSecurityName(String uniqueGroupId)
    throws CustomRegistryException,
           EntryNotFoundException {
    String s,grpSecName = null;
    BufferedReader in = null;
    try {
        in = fileOpen(GROUPFILENAME);
        while ((s=in.readLine())!=null)
        {
            if (!(s.startsWith("#") || s.trim().length() <=0 )) {
                int index = s.indexOf(":");
                int index1 = s.indexOf(":", index+1);
                if ((s.substring(index+1,index1)).equals(uniqueGroupId)) {
                    grpSecName = s.substring(0,index);
                    break;
                }
            }
        }
    } catch (Exception ex) {
        throw new CustomRegistryException(ex.getMessage(),ex);
    } finally {
        fileClose(in);
    }

    if (grpSecName == null) {
        EntryNotFoundException ex =
            new EntryNotFoundException("Cannot obtain the group
            security name for: " + uniqueGroupId);
        throw ex;
    }

    return grpSecName;
}

/**
 * Determines if the groupSecurityName exists in the registry
 *
 * @param    groupSecurityName the name of the group
 * @return   True if the groups exists; otherwise false
 * @exception CustomRegistryException if there is any registry-
 *           specific problem
 */

```

```

public boolean isValidGroup(String groupSecurityName)
    throws CustomRegistryException {
    String s;
    boolean isValid = false;
    BufferedReader in = null;
    try {
        in = fileOpen(GROUPFILENAME);
        while ((s=in.readLine())!=null)
        {
            if (!(s.startsWith("#") || s.trim().length() <=0 )) {
                int index = s.indexOf(":");
                if ((s.substring(0,index)).equals(groupSecurityName)) {
                    isValid=true;
                    break;
                }
            }
        }
    } catch (Exception ex) {
        throw new CustomRegistryException(ex.getMessage(),ex);
    } finally {
        fileClose(in);
    }

    return isValid;
}

/**
 * Returns the securityNames of all the groups that contain the user
 *
 * This method is called by the administrative console and scripting
 * (command line) to verify the user entered for RunAsRole mapping
 * belongs to that role in the roles to user mapping. Initially, the
 * check is done to see if the role contains the user. If the role does
 * not contain the user explicitly, this method is called to get the groups
 * that this user belongs to so that check can be made on the groups that
 * the role contains.
 *
 * @param    userSecurityName the name of the user
 * @return    A list of all the group securityNames that the user
 *            belongs to.
 * @exception EntryNotFoundException if user does not exist.
 * @exception CustomRegistryException if there is any registry-
 *            specific problem
 * @exception RemoteException as this extends the java.rmi.Remote
 *            interface
 */
public List getGroupsForUser(String userName)
    throws CustomRegistryException,
           EntryNotFoundException {
    String s;
    List grpsForUser = new ArrayList();
    BufferedReader in = null;
    try {
        in = fileOpen(GROUPFILENAME);
        while ((s=in.readLine())!=null)
        {
            if (!(s.startsWith("#") || s.trim().length() <=0 )) {
                StringTokenizer st = new StringTokenizer(s, ":");
                for (int i=0; i<2; i++)
                    st.nextToken();
                String subs = st.nextToken();
                StringTokenizer st1 = new StringTokenizer(subs, ",");
                while (st1.hasMoreTokens()) {
                    if((st1.nextToken()).equals(userName)) {
                        int index = s.indexOf(":");
                        grpsForUser.add(s.substring(0,index));
                    }
                }
            }
        }
    }
}

```

```

        }
    }
} catch (Exception ex) {
    if (!isValidUser(userName)) {
        throw new EntryNotFoundException("user: " + userName
            + " is not valid");
    }
    throw new CustomRegistryException(ex.getMessage(),ex);
} finally {
    fileClose(in);
}

return grpsForUser;
}

/**
 * Gets a list of users in a group.
 *
 * The maximum number of users returned is defined by the
 * limit argument.
 *
 * This method is being used by the process choreographer
 * when staff assignments are modeled using groups.
 *
 * In rare situations, if you are working with a registry where
 * getting all the users from any of your groups is not practical
 * (for example if there are a large number of users) you can throw
 * the NotImplementedException for that particular group. Make sure
 * that if the process choreographer is installed (or if installed later)
 * the staff assignments are not modeled using these particular groups.
 * If there is no concern about returning the users from groups
 * in the registry it is recommended that this method be implemented
 * without throwing the NotImplementedException.
 * @param groupSecurityName the name of the group
 * @param limit Limits the maximum number of users that should be
 * returned. This is very useful in situations where there
 * are lot of users in the registry and getting all of
 * them at once is not practical. A value of 0 implies
 * get all the users and hence must be used with care.
 * @return A result object that contains the list of users
 * requested and a flag to indicate if more users exist.
 * @deprecated This method will be deprecated in future.
 * @exception NotImplementedException throw this exception in rare
 * situations if it is not practical to get this information
 * for any of the group or groups from the registry.
 * @exception EntryNotFoundException if the group does not exist in
 * the registry
 * @exception CustomRegistryException if there is any registry-specific
 * problem
 */
public Result getUsersForGroup(String groupSecurityName, int limit)
    throws NotImplementedException,
        EntryNotFoundException,
        CustomRegistryException {
    String s, user;
    BufferedReader in = null;
    List usrsForGroup = new ArrayList();
    int count = 0;
    int newLimit = limit+1;
    Result result = new Result();

    try {
        in = fileOpen(GROUPFILENAME);
        while ((s=in.readLine())!=null)
        {

```

```

        if (!(s.startsWith("#") || s.trim().length() <=0 )) {
            int index = s.indexOf(":");
            if ((s.substring(0,index)).equals(groupSecurityName))
            {
                StringTokenizer st = new StringTokenizer(s, ":");
                for (int i=0; i<2; i++)
                    st.nextToken();
                String subs = st.nextToken();
                StringTokenizer st1 = new StringTokenizer(subs, ",");
                while (st1.hasMoreTokens()) {
                    user = st1.nextToken();
                    usrsForGroup.add(user);
                    if (limit !=0 && ++count == newLimit) {
                        usrsForGroup.remove(user);
                        result.setHasMore();
                        break;
                    }
                }
            }
        }
    }
} catch (Exception ex) {
    if (!isValidGroup(groupSecurityName)) {
        throw new EntryNotFoundException("group: "
            + groupSecurityName
            + " is not valid");
    }
    throw new CustomRegistryException(ex.getMessage(),ex);
} finally {
    fileClose(in);
}

result.setList(usrsForGroup);
return result;
}

/**
 * This method is implemented internally by the WebSphere Application
 * Server code in this release. This method is not called for the custom
 * registry implementations for this release. Return null in the
 * implementation.
 */
public com.ibm.websphere.security.cred.WSCredential
createCredential(String userSecurityName)
throws CustomRegistryException,
NotImplementedException,
EntryNotFoundException {

    // This method is not called.
    return null;
}

// private methods
private BufferedReader fileOpen(String fileName)
throws FileNotFoundException {
    try {
        return new BufferedReader(new FileReader(fileName));
    } catch(FileNotFoundException e) {
        throw e;
    }
}

private void fileClose(BufferedReader in) {
    try {
        if (in != null) in.close();
    } catch(Exception e) {

```

```

        System.out.println("Error closing file" + e);
    }
}

private boolean match(String name, String pattern) {
    RegExpSample regexp = new RegExpSample(pattern);
    boolean matches = false;
    if(regexp.match(name))
        matches = true;
    return matches;
}
}

//-----
// The program provides the Regular Expression implementation
// used in the sample for the custom user registry (FileRegistrySample).
// The pattern matching in the sample uses this program to search for the
// pattern (for users and groups).
//-----

class RegExpSample
{
    private boolean match(String s, int i, int j, int k)
    {
        for(; k < expr.length; k++)
label0:
        {
            Object obj = expr[k];
            if(obj == STAR)
            {
                if(++k >= expr.length)
                    return true;
                if(expr[k] instanceof String)
                {
                    String s1 = (String)expr[k++];
                    int l = s1.length();
                    for(; (i = s.indexOf(s1, i)) >= 0; i++)
                        if(match(s, i + l, j, k))
                            return true;

                    return false;
                }
                for(; i < j; i++)
                    if(match(s, i, j, k))
                        return true;

                return false;
            }
            if(obj == ANY)
            {
                if(++i > j)
                    return false;
                break label0;
            }
            if(obj instanceof char[][] )
            {
                if(i >= j)
                    return false;
                char c = s.charAt(i++);
                char ac[][] = (char[][] )obj;
                if(ac[0] == NOT)
                {
                    for(int j1 = 1; j1 < ac.length; j1++)
                        if(ac[j1][0] <= c && c <= ac[j1][1])
                            return false;
                }
            }
        }
    }
}

```

```

        break label0;
    }
    for(int k1 = 0; k1 < ac.length; k1++)
        if(ac[k1][0] <= c && c <= ac[k1][1])
            break label0;

    return false;
}
if(obj instanceof String)
{
    String s2 = (String)obj;
    int i1 = s2.length();
    if(!s.regionMatches(i, s2, 0, i1))
        return false;
    i += i1;
}
}

return i == j;
}

public boolean match(String s)
{
    return match(s, 0, s.length(), 0);
}

public boolean match(String s, int i, int j)
{
    return match(s, i, j, 0);
}

public RegExpSample(String s)
{
    Vector vector = new Vector();
    int i = s.length();
    StringBuffer stringbuffer = null;
    Object obj = null;
    for(int j = 0; j < i; j++)
    {
        char c = s.charAt(j);
        switch(c)
        {
            case 63: /* '?' */
                obj = ANY;
                break;

            case 42: /* '*' */
                obj = STAR;
                break;

            case 91: /* '[' */
                int k = ++j;
                Vector vector1 = new Vector();
                for(; j < i; j++)
                {
                    c = s.charAt(j);
                    if(j == k && c == '^')
                    {
                        vector1.addElement(NOT);
                        continue;
                    }
                }
                if(c == '\\')
                {
                    if(j + 1 < i)
                        c = s.charAt(++j);
                }
            }
        }
    }
}

```

```

        else
            if(c == ']')
                break;
            char c1 = c;
            if(j + 2 < i && s.charAt(j + 1) == '-')
                c1 = s.charAt(j += 2);
            char ac1[] = {
                c, c1
            };
            vector1.addElement(ac1);
        }

        char ac[][] = new char[vector1.size()][];
        vector1.copyInto(ac);
        obj = ac;
        break;

    case 92: /* '\\\ ' */
        if(j + 1 < i)
            c = s.charAt(++j);
        break;

    }
    if(obj != null)
    {
        if(stringbuffer != null)
        {
            vector.addElement(stringbuffer.toString());
            stringbuffer = null;
        }
        vector.addElement(obj);
        obj = null;
    }
    else
    {
        if(stringbuffer == null)
            stringbuffer = new StringBuffer();
        stringbuffer.append(c);
    }
}

if(stringbuffer != null)
    vector.addElement(stringbuffer.toString());
expr = new Object[vector.size()];
vector.copyInto(expr);
}

static final char NOT[] = new char[2];
static final Integer ANY = new Integer(0);
static final Integer STAR = new Integer(1);
Object expr[];

}

```

Result.java file: This module is used by user registries in WebSphere Application Server when calling the getUsers and getGroups methods. The user registries use this method to set the list of users and groups and to indicate if there are more users and groups in the registry than requested.

```

// @(#) 1.20 src/en/ae/rsec_result.xml, WEBSJAVA.INFO.DOCSRC,
// ASVINFO1 10/17/02 16:43:01 [10/18/02 07:31:30]
// 5639-D57, 5630-A36, 5630-A37, 5724-D18
// (C) COPYRIGHT International Business Machines Corp. 1997, 2003
// All Rights Reserved * Licensed Materials - Property of IBM
//

```

```

package com.ibm.websphere.security;

import java.util.List;

public class Result implements java.io.Serializable {
    /**
     * Default constructor
     */
    public Result() {
    }

    /**
     * Returns the list of users and groups
     * @return the list of users and groups
     */
    public List getList() {
        return list;
    }

    /**
     * indicates if there are more users and groups in the registry
     */
    public boolean hasMore() {
        return more;
    }

    /**
     * Set the flag to indicate that there are more users and groups
     * in the registry to true
     */
    public void setHasMore() {
        more = true;
    }

    /**
     * Set the list of users and groups
     * @param list list of users/groups
     */
    public void setList(List list) {
        this.list = list;
    }

    private boolean more = false;
    private List list;
}

```

Custom user registry settings:

Use this page to configure the custom user registry.

To view this administrative console page, click **Security > User Registries > Custom**.

After the properties are set in this panel, click **Apply**. Use the Properties panel for additional properties that the custom registry requires. When security is enabled and any of these properties change, go to the Global Security panel and click **Apply** to validate the changes.

Server User ID:

Specifies the user ID under which the server runs, for security purposes.

This server ID represents a valid user in the custom registry.

Data type: String

Server User Password:

Specifies the password corresponding to the security server ID.

Data type: String

Custom Registry Classname:

Specifies a dot-separated class name that implements the `com.ibm.websphere.security.UserRegistry` interface.

Put the custom registry class name in the class path. A suggested location is the `%install_root%/lib/ext` directory. Although the custom registry implements the `com.ibm.websphere.security.UserRegistry` interface, for backward compatibility, a user registry can alternately implement the `com.ibm.websphere.security.CustomRegistry` interface.

Data type: String

Default: `com.ibm.websphere.security.FileRegistrySample`

Ignore Case:

Specifies that a case insensitive authorization check is performed.

Default: Enabled

Range: Enabled or Disabled

Use **Custom Properties** to add any additional properties required to initialize the custom registry. The following property is predefined by the product; set this property when required only:

- `WAS_UseDisplayName`--When set to true, the `getCallerPrincipal()`, `getUserPrincipal()`, and `getRemoteUser()` methods return the display name. By default, the `securityName` of the user is returned. This default is introduced to support backward compatibility with the Version 4.0 custom registry.

users.props file: Following is the format for the `users.props` file:

```
# 5639-D57, 5630-A36, 5630-A37, 5724-D18
# (C) COPYRIGHT International Business Machines Corp. 1997, 2003
# All Rights Reserved * Licensed Materials - Property of IBM
#
# Format:
# name:passwd:uid:gids:display name
# where name = userId/userName of the user
#         passwd = password of the user
#         uid = uniqueId of the user
#         gid = groupIds of the groups that the user belongs to
#         display name = a (optional) display name for the user.
```

```
bob:bob1:123:567:bob
dave:dave1:234:678:
jay:jay1:345:678,789:Jay-Jay
ted:ted1:456:678:Teddy G
jeff:jeff1:222:789:Jeff
vikas:vikas1:333:789:vikas
bobby:bobby1:444:789:
```

groups.props file: The following example illustrates the format for the groups.props file:

```
# 5639-D57, 5630-A36, 5630-A37, 5724-D18
# (C) COPYRIGHT International Business Machines Corp. 1997, 2003
# All Rights Reserved * Licensed Materials - Property of IBM
#
# Format:
# name:gid:users:display name
# where name = groupId of the group
#       gid = uniqueId of the group
#       users = list of all the userIds that the group contains
#       display name = a (optional) display name for the group.
admins:567:bob:Administrative group
operators:678:jay,ted,dave:Operators group
users:789:jay,jeff,vikas,bobby:
```

Java Authentication and Authorization Service

The standard Java 2 security API helps enforce access control, based on the location of the code and who signed it. The current principal of the running thread is not considered in the Java 2 security authorization. Instances where authorization is based on the principal, rather than the code base and the signer 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 signers.

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 users. It implements a Java version of the standard Pluggable Authentication Module (PAM) framework, and extends the access control architecture of the Java 2 platform in a compatible fashion to support user-based authorization. WebSphere Application Server fully supports the JAAS architecture and extends the access control architecture to support role-based authorization for Java 2 Platform, Enterprise Edition (J2EE) resources including servlets, JavaServer Pages (JSP) files, and Enterprise JavaBeans (EJB) components.

The following sections cover the JAAS implementation and programming model:

- Java Authentication and Authorization Service login configuration
- Programmatic Login
- Java Authentication and Authorization Service authorization

The accompanying product Javadoc contains detailed descriptions of the WebSphere Application Server programming APIs and the JAAS Javadoc also ships with the product. Refer to the `${install_root}/web/docs/jaas` directory.

Java Authentication and Authorization Service authorization

Java 2 security architecture uses a security policy to specify which access rights are granted to running code. This architecture is *code-centric*. That is, the permissions are granted based on code characteristics including where the code is coming from, whether it is digitally signed, and by whom. Authorization of the Java Authentication and Authorization Service (JAAS) augments the existing

code-centric access controls with new user-centric access controls. Permissions are granted based on what code is running and who is running it.

When using JAAS authentication to authenticate a user, a *subject* is created to represent the authenticated user. A subject is comprised of a set of principals, where each principal represents an identity for that user. You can grant permissions in the policy to specific principals. After the user is authenticated, the application can associate the subject with the current access control context. For each subsequent security-checked operation, the Java run time automatically determines whether the policy grants the required permission to a specific principal only. If so, the operation is supported if the subject associated with the access control context contains the designated principal only.

Associate a subject with the current access control context by calling the static `doAs` method from the subject class, passing it an authenticated subject and `java.security.PrivilegedAction` or `java.security.PrivilegedExceptionAction`. The `doAs` method associates the provided subject with the current access control context and then invokes the `run` method from the action. The `run` method implementation contains all the code that ran as the specified subject. The action runs as the specified subject.

In the Java 2 Platform, Enterprise Edition (J2EE) programming model, when invoking the EJB method from an enterprise bean or servlet, the method runs under the user identity that is determined by the `run-as` setting. The J2EE Version 1.3 Specification does not indicate which user identity to use when invoking an enterprise bean from a `Subject.doAs` action block within either the EJB code or the servlet code. A logical extension is to use the proper identity specified in the subject when invoking the EJB method within the `Subject.doAs` action block.

This simple rule of letting `Subject.doAs` overwrite the `run-as` identity setting is an ideal way to integrate the JAAS programming model with the J2EE run-time environment. However, a design oversight was introduced into IBM Developer Kit, Java Technology Edition Version 1.3 when integrating the JAAS Version 1.0 implementation with the Java 2 security architecture. A subject, which is associated with the access control context is cut off by a `doPrivileged` call when a `doPrivileged` call occurs within the `Subject.doAs` action block. Until this problem is corrected, no reliable and run-time efficient way is available to guarantee the correct behavior of `Subject.doAs` in a J2EE run-time environment.

The problem can be explained better with the following example:

```
Subject.doAs(subject, new java.security.PrivilegedAction() {
    Public Object run() {
        // Subject is associated with the current thread context
        java.security.AccessController.doPrivileged( new
            java.security.PrivilegedAction() {
                public Object run() {
                    // Subject was cut off from the current
                    // thread context
                }
            }
        );
        return null;
    }
});
// Subject is associated with the current thread context
return null;
}
```

At line three, the subject object is associated with the context of the current thread. As indicated on line 7 within the run method of a doPrivileged action block, the subject object is removed from the thread context. After leaving the doPrivileged block, the subject object is restored to the current thread context. Because doPrivileged blocks can be placed anywhere along the running path and instrumented quite often in a server environment, the run-time behavior of a doAs action block becomes difficult to manage.

To resolve this difficulty, WebSphere Application Server provides a WSSubject helper class to extend the JAAS authorization to a J2EE EJB method invocation as described previously. The WSSubject class provides static doAs and doAsPrivileged methods that have identical signatures to the subject class. The WSSubject.doAs method associates the Subject to the currently running thread. The credential is used by the Security Authentication Service (SAS) run time for EJB invocation. The WSSubject.doAs and WSSubject.doAsPrivileged methods then invoke the corresponding Subject.doAs and Subject.doAsPrivileged methods. The original credential is restored and associated with the running thread upon leaving the WSSubject.doAs and WSSubject.doAsPrivileged methods.

Note that the WSSubject class is not a replacement of the subject object, but rather a helper class to ensure consistent run-time behavior as long as an EJB method invocation is a concern.

The following example illustrates the run-time behavior of the WSSubject.doAs method:

```
WSSubject.doAs(subject, new java.security.PrivilegedAction() {
    Public Object run() {
        // Subject is associated with the current thread context
        java.security.AccessController.doPrivileged( new
            java.security.PrivilegedAction() {
                public Object run() {
                    // Subject was cut off from the current thread
                    // context.

                }
            }
        );
        // Subject is associated with the current thread context

    }
});
```

The Subject.doAs and Subject.doAsPrivileged methods are not integrated with the J2EE run-time environment. EJB methods that are invoked within the Subject.doAs and Subject.doAsPrivileged action blocks run under the identity specified by the run-as setting and not by the subject identity.

- The subject object generated by the WSLoginModuleImpl instance and the WSCientLoginModuleImpl instance contains a principal that implements the WSPrincipal interface. Using the getCredential() method for a WSPrincipal object returns an object that implements the WSCredential interface. You can also find the WSCredential object instance in the PublicCredentials list of the subject instance. Retrieve the WSCredential object from the PublicCredentials list instead of using the getCredential() method.

- The `getCallerPrincipal()` method for the `WSSubject` class returns a string representing the caller security identity. The return type differs from the `getCallerPrincipal` method of the `EJBContext` interface, which is `java.security.Principal`.
- The `Subject` object generated by the `J2C DefaultPrincipalMapping` module contains a resource principal and a `PasswordCredentials` list. The resource principal represents the caller.

Refer to Java 2 Connector security for more information

Configuring Java Authentication and Authorization Service login

Java Authentication and Authorization Service (JAAS) is a new feature in WebSphere Application Server. It is a collection of WebSphere Application Server strategic authentication APIs and replaces the Common Object Request Broker Architecture (CORBA) programmatic login APIs. JAAS also is mandated by the J2EE Version 1.3 specification.

WebSphere Application Server provides some extensions to JAAS:

- **com.ibm.websphere.security.auth.WSSubject.** Due to a design oversight in the JAAS V1.0, `javax.security.auth.Subject.getSubject()` method does not return the subject associated with the running thread inside a `java.security.AccessController.doPrivileged()` code block. This problem presents an inconsistent behavior that is problematic and causes undesirable effort. The `com.ibm.websphere.security.auth.WSSubject` API provides a workaround to associate the subject to a running thread. The `com.ibm.websphere.security.auth.WSSubject` API extends the JAAS authorization model to J2EE resources.
- You can configure JAAS login in the administrative console and store this configuration in WebSphere Common Configuration Model (WCCM). However, WebSphere Application Server still supports the default JAAS login configuration format (plain text file) provided by the JAAS default implementation. If duplicate login configurations are defined in both the WCCM and the plain text file format, the one in the WCCM takes precedence. Advantages to defining the login configuration in the WCCM 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
- **Proxy LoginModule.** The default JAAS implementation does not use the thread context class loader to load classes. The `LoginModule` module cannot load if the `LoginModule` class file is not in the application class loader or the Java extension class loader class path. Due to this class loader visibility problem, WebSphere Application Server provides a proxy `LoginModule` module to load the JAAS `LoginModule` using the thread context class loader. You do not need to place the `LoginModule` implementation on the application class loader or the Java extension class loader class path with this proxy `LoginModule` module.

Two JAAS login configurations are defined in the WCCM security document for applications to use. In the left navigation pane, click **Security > JAAS Configuration > Application Login > WSLogin** and **ClientContainer**. The following three JAAS login configurations are available:

WSLogin

Defines a login configuration and a LoginModule implementation that applications can use in general.

ClientContainer

Defines a login configuration and a LoginModule implementation that is similar to that of the WSLogin configuration, but enforces the requirements of the WebSphere Application Server client container.

DefaultPrincipalMapping

Defines a special LoginModule module that is typically used by Java 2 Connector to map an authenticated WebSphere user identity to a set of user authentication data (user ID and password) for the specified back-end enterprise information system (EIS). For more information about Java 2 Connector and the DefaultMappingModule module, refer to 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 run time.

Attention: Do not remove or delete the predefined JAAS login configurations (ClientContainer, WSLogin and DefaultPrincipalMapping). Deleting or removing them can cause other enterprise applications to fail.

1. Delete a JAAS login configuration.
 - a. Click **Security** in the navigation tree.
 - b. Click **JAAS Configuration > Application Logins**. The Application Login Configuration panel appears.
 - c. Select the check box for the login configurations to delete and click **Delete**.
2. Create a new JAAS login configuration.
 - a. Click **Security** in the navigation tree.
 - b. Click **JAAS Configuration > Application Logins**.
 - c. Click **New**. The Application Login Configuration panel appears.
 - d. Specify the alias name of the new JAAS login configuration and click **Apply**. This value is the name of the login configuration that you pass in the `javax.security.auth.login.LoginContext` implementation for creating a new `LoginContext`.

Click **Apply** to save changes and to add the extra node name that precedes the original alias name. Clicking **OK** does not save the new changes in the `security.xml` file.
 - e. Click **JAAS Login Modules**.
 - f. Click **New**.
 - g. Specify the Module Classname. Specify WebSphere Proxy LoginModule because of the limitation of the class loader visibility problem.
 - h. Specify the LoginModule implementation as the delegate property of the Proxy LoginModule. The WebSphere Proxy LoginModule class name is `com.ibm.ws.security.common.auth.module.proxy.WSLoginModuleProxy`.
 - i. Select **Authentication Strategy** from the list and click **Apply**.
 - j. Click **Custom Properties**. The **Custom Properties** panel is displayed for the selected LoginModule.
 - k. Create a new property with the name `delegate` and the value of the real LoginModule implementation. You can specify other properties like `debug`

with the value true. These properties are passed to the LoginModule class as options to the initialize() method of the LoginModule instance.

l. Click **Save**.

There are several locations within the WebSphere Application Server directory structure where you can place a JAAS login module. The following list provides locations for the JAAS login module in order of recommendation:

- Within an Enterprise Archive (EAR) file for a specific Java 2 Enterprise Edition (J2EE) application.

If you place the login module within the EAR file, it is accessible to the specific application only.

- In the WebSphere Application Server shared library.

If you place the login module in the shared library, you must specify which applications can access the module. For more information on shared libraries, see *Managing shared libraries*.

- In the Java extensions directory (*WAS_HOME\jre\lib\ext*)

If you place the JAAS login module in the Java extensions directory, the login module is available to all applications.

Although the Java extensions directory provides the greatest availability for the login module, it is recommended that you place the login module in an application EAR file. If other applications need to access the same login module, consider using shared libraries.

3. Change the plain text file. WebSphere Application Server supports the default JAAS login configuration format (plain text file) provided by the JAAS default implementation. However, a tool is not provided that edits plain text files in this format. You can define the JAAS login configuration in the plain text file (*install_root/properties/wsjaas.conf*). Any syntax errors can cause the incorrect parsing of the plain JAAS login configuration text file. This problem can cause other applications to fail.

Java client programs that use the Java Authentication and Authorization Service (JAAS) for authentication must invoke with the JAAS configuration file specified. This configuration file is set in the *install_root/bin/launchClient.bat* file as `set JAAS_LOGIN_CONFIG=-Djava.security.auth.login.config=%install_root%\properties\wsjaas_client.conf`. If the *launchClient.bat* file is not used to invoke the Java client program, verify that the appropriate JAAS configuration file is passed to the Java virtual machine with the `-Djava.security.auth.login.config` flag.

A new JAAS login configuration is created or an old JAAS login configuration is removed. An enterprise application can use a newly created JAAS login configuration without restarting the application server process.

However, new JAAS login configurations defined in the *install_root/properties/wsjaas.conf* file, do not refresh automatically. Restart the application servers to validate changes. These JAAS login configurations are specific to a particular node and are not available for other application servers running on other nodes.

Create new JAAS login configurations used by enterprise applications to perform custom authentication.

Use these newly defined JAAS login configurations to perform programmatic login.

Java Authentication and Authorization Service login configuration

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

- `com.ibm.websphere.security.auth.WSSubject`: Due to a design oversight in the JAAS 1.0, `javax.security.auth.Subject.getSubject()` does not return the Subject associated with the thread of execution inside a `java.security.AccessController.doPrivileged()` code block. This can present a inconsistent behavior that is problematic and causes undesirable effort. `com.ibm.websphere.security.auth.WSSubject` provides a work around to associate Subject to thread of execution. `com.ibm.websphere.security.auth.WSSubject` extends the JAAS authorization model to J2EE resources.

Note: You can retrieve the subjects in a `Subject.doAs()` block with the `Subject.getSubject()` call. However, this procedure does not work if there is an `AccessController.doPrivileged()` call within the `Subject.doAs()` block. In the following example, `s1` is equal to `s`, but `s2` is null:

```
* AccessController.doPrivileged() not only truncates the Subject propagation,  
* but also reduces the permissions. It does not include the JAAS security  
* policy defined for the principals in the Subject.
```

```
Subject.doAs(s, new PrivilegedAction() {  
    public Object run() {  
        System.out.println("Within Subject.doAsPrivileged()");  
        Subject s1 = Subject.getSubject(AccessController.getContext());  
        AccessController.doPrivileged(new PrivilegedAction() {  
            public Object run() {  
                Subject s2 = Subject.getSubject(AccessController.getContext());  
                return null;  
            }  
        });  
        return null;  
    }  
});
```

- JAAS Login Configuration can be configured in Admin Console and stored in WCCM (WebSphere Common Configuration Model): Application can define new JAAS login configuration in the Admin Console and the the data is persisted in the configuration respository (stored in the WCCM). However, WebSphere still support the default JAAS login configuration format (plan text file) provided by the JAAS default implementation. But if there are duplication login configurations defined in both the WCCM and the plan text file format, the one in the WCCM takes precedence. There are advantages to define the login configuration in the WCCM:
 - UI support in defining JAAS login configuration.
 - The JAAS configuration login configuration can be managed centrally.
 - The JAAS configuration login configuration is distributed in a Network Deployment installation.
- Proxy LoginModule: The default JAAS implementation does not use the thread context class loader to load classes, the LoginModule could not be loaded if the LoginModule class file is not in the application class loader or the Java extension class loader classpath. Due to this class loader visibility problem, WebSphere provides a proxy LoginModule to load JAAS LoginModule using the thread

context class loader. The LoginModule implementation does not have to be placed on the application class loader or the Java extension class loader classpath with this proxy LoginModule.

Note: Do not remove or delete the pre-defined JAAS Login Configurations (ClientContainer, WSLogin and DefaultPrincipalMapping). Deleting or removing them could cause other enterprise applications to fail.

A system administrator determines the authentication technologies, or LoginModules, to be used for each application and configures them in a login configuration. The source of the configuration information (for example, a file or a database) is up to the current `javax.security.auth.login.Configuration` implementation. The WebSphere Application Server implementation permits the login configuration to be defined in both the WebSphere Common Configuration Model (WCCM) security document and in a JAAS configuration file where the former takes precedence.

Two JAAS login configurations are defined in the WCCM security document for applications to use. They may be found in the left navigation pane at **Security > JAAS Configuration > Application Login Config: WSLogin and ClientContainer**. The **WSLogin** defines a login configuration and LoginModule implementation that may be used by applications in general. The **ClientContainer** defines a login configuration and LoginModule implementation that is similar to that of WSLogin but enforces the requirements of the WebSphere Application Server Client Container. The third entry, **DefaultPrincipalMapping**, defines a special LoginModule that is typically used by Java 2 Connector to map an authenticated WebSphere user identity to a set of user authentication data (user ID and password) for the specified back end Enterprise Information System (EIS). For more information about Java 2 Connector and the DefaultMappingModule please refer to the Java 2 Security section.

New JAAS login configuration may be added and modified using Security Center. The changes are saved in the cell level security document and are available to all managed application servers. An application server restart is required for the changes to take effect at run time.

WebSphere Application Server also reads JAAS Configuration information from the `wsjaas.conf` file under the `properties` sub directory of the root directory under which WebSphere Application Server is installed. Changes made to the `wsjaas.conf` file is used only by the local application server and will take effect after restarting the application server. Note that JAAS configuration in the WCCM security document takes precedence over that defined in the `wsjaas.conf` file. In other words, a configuration entry in `wsjaas.conf` will be overridden by an entry of the same alias name in the WCCM security document.

Note: The Java Authentication and Authorization Service (JAAS) login configuration entries in the Security Center are propagated to the server run time when they are created, not when the configuration is saved. However, the deleted JAAS login configuration entries are not removed from the server run time. To remove the entries, save the new configuration, then stop and restart the server.

Java Authentication and Authorization service configuration entry settings

Use this page to specify a list of Java Authentication and Authorization Service (JAAS) login configurations for the application code to use, including enterprise beans, Java ServerPages (JSP) files, servlets and resource adapters.

To view this administrative console page, click **Security > JAAS Configuration > Application Login Configuration**.

Read the JAAS documentation before you begin defining additional login modules for authenticating to the WebSphere Application Server security run time. You can define additional login configurations for your applications. However, if the WebSphere Application Server LoginModule (`com.ibm.ws.security.common.auth.module.WSLoginModuleImpl`) is not used or the LoginModule does not produce a credential that is recognized by WebSphere Application Server, then the WebSphere Application Server security run time cannot use the authenticated subject from these login configurations for an authorization check for resource access.

Note: You must invoke Java client programs that use Java Authentication and Authorization Service (JAAS) for authentication with a JAAS configuration file specified. The WebSphere product supplies the default JAAS configuration file, `wsjaas_client.conf` under the `install_root/properties` directory. This configuration file is set in the `/install_root/bin/launchClient.bat` file as: `set JAAS_LOGIN_CONFIG=-Djava.security.auth.login.config=%WAS_HOME%\properties\wsjaas_client.conf`

If `launchClient.bat` file is not used to invoke Java client programs, make sure that the appropriate JAAS configuration file is passed to the Java virtual machine with the `-Djava.security.auth.login.config` flag.

ClientContainer:

Specifies the login configuration used by the client container application, which uses the CallbackHandler API defined in the client container deployment descriptor.

ClientContainer is the default login configuration for the WebSphere Application Server. Do not remove this default, as other applications that use it fail.

Default: ClientContainer

DefaultPrincipalMapping:

Specifies the login configuration used by Java 2 Connectors to map users to principals that are defined in the J2C Authentication Data Entries.

ClientContainer is the default login configuration for the WebSphere Application Server. Do not remove this default, as other applications that use it fail.

Default: ClientContainer

WSLogin:

Specifies whether all applications can use the WSLogin configuration to perform authentication for the WebSphere Application Server security run time.

This login configuration does not honor the CallbackHandler defined in the client container deployment descriptor. To use this functionality, use the ClientContainer login configuration.

The WLogin configuration is the default login configuration for the WebSphere Application Server. Do not remove this default, as other applications that use it fail. This login configuration authenticates users for the WebSphere Application Server security run time. Use credentials from the authenticated subject returned from this login configurations as an authorization check for access to WebSphere Application Server resources.

Default: ClientContainer

Java Authentication Authorization Service login module settings

Use this page to define the login module for a Java Authentication Authorization Service (JAAS) login configuration.

To view this administrative page, click **Security > JAAS Configuration > Application Logins > *alias_name* > JAAS Login Modules**.

Module Class Name:

Specifies the class name of the given login module.

The default login modules defined by WebSphere Application Server use a proxy LoginModule class, com.ibm.ws.security.common.auth.module.WLoginModuleProxy. This proxy class loads the WebSphere 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 WebSphere Application Server class loaders.

Data type: String

Authentication Strategy:

Specifies the authentication behavior as authentication proceeds down the list of login modules.

A 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

Specify additional options in **Options Additional Properties**. These name and value pairs are passed to the login modules during initialization. This process is one of the mechanisms used to passed information to login modules.

Application login configuration settings

Use this page to configure application login configurations.

To view this administrative console page, click **Security > JAAS Configuration > Application 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

Java 2 Connector security

Java 2 Connector authentication data entries are used by resource adapters and Java database connectivity (JDBC) data sources. A Java 2 Connector authentication data entry contains authentication data.

The connector architecture defines a standard architecture for connecting the Java 2 Platform, Enterprise Edition (J2EE) to heterogeneous enterprise information systems (EIS). Examples of EIS include Enterprise Resource Planning (ERP), mainframe transaction processing (TP) and database systems.

The connector architecture enables an EIS vendor to provide a standard *resource adapter* for its EIS. A *resource adapter* is a system-level software driver that is used by a Java application to connect to an EIS. The resource adapter plugs into an application server and provides connectivity between the EIS, the application server, and the enterprise application. You must protect information in EIS from unauthorized access. The Java 2 Connector security architecture is designed to extend the end-to-end security model for J2EE-based applications to include integration with EISs. An application server and an EIS collaborate to ensure the proper authentication of a resource principal, which establishes a connection to an underlying EIS. The connector architecture identifies the following mechanisms as the commonly-supported authentication mechanisms:

- BasicPassword: Basic user-password-based authentication mechanism specific to an EIS
- Kerbv5: Kerberos Version 5-based authentication mechanism

WebSphere Application Server implementation of a Java 2 connection supports basic password authentication mechanisms.

The user ID and password for the target EIS is either supplied by applications or by the application server. WebSphere Application Server uses a Java Authentication and Authorization Service (JAAS) pluggable authentication mechanism to perform principal mapping to convert a WebSphere principal to a resource principal. WebSphere Application Server provides a DefaultPrincipalMapping LoginModule, which basically converts any authenticated principal to the pre-configured EIS resource principal and password. Subsequently, you can plug in a principal mapping LoginModule through the JAAS plug-in mechanism.

J2C mapping module configuration

When a Java 2 connection factory is configured for container-managed signon, WebSphere Application Server uses the configured principal mapping module to create a Subject instance that contains a user ID and a password for the target EIS.

Mapping modules are special JAAS login modules that provide principal and credential mapping functionality. You can define and configure custom mapping modules through the administrative console. Associated with the mapping module configuration is a set of user IDs and passwords that you can define in the security configuration with a specified alias name. The WebSphere Application Server run time passes the user ID, password and a reference of the connection factory manager to the configured mapping module to create a subject.

For more information about mapping module requirements, refer to the Javadoc of the `WSDDefaultPrincipalMapping` class. For more detailed information about developing a mapping module, refer to the *Developing your own Java 2 security mapping module* article.

J2C mapping module programming reference

You can develop your own mapping module if your application requires more sophisticated mapping functions. You can use the `WSSubject.getRunAsSubject()` method to retrieve the subject that represents the identity of the current thread of execution. The identity of the current thread of execution is known as the `RunAs` identity. The `RunAs` subject typically contains a `WSPrincipal` in the principal set and a `WSCredential` in the public credential set. The subject instance that is created by your mapping module contains a `Principal` instance in the principals set and a `PasswordCredential` or `GenericCredential` instance in the set of private credentials.

Managing J2EE Connector Architecture authentication data entries

Java 2 Connector authentication data entries are used by resource adapters and Java database connectivity (JDBC) data sources. A Java 2 Connector authentication data entry contains authentication data, which contains the following information:

Alias An identifier used to identify the authenticated data entry. When configuring resource adapters or Java database connectivity (JDBC) data sources, the administrator can specify which authentication data to choose for 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.

This task creates and deletes Java 2 Connector (J2C) authentication data entries.

1. Delete a J2C authentication data entry.
 - a. Click **Security** in the navigation tree, then click **JAAS Configuration > J2C Authentication Data**. The **J2C Authentication Data Entries** panel is displayed.
 - b. Select the check boxes for the entries to delete and click **Delete**. Before deleting or removing an authentication data entry, make sure that it is not used or referenced by any resource adapter or JDBC data source. If the deleted authentication data entry is used or referenced by a resource, the application that uses the resource adapter or JDBC data source fails to connect to the resources.

2. Create a new J2C authentication data entry.
 - a. Click **Security** in the navigation tree, then click **JAAS Configuration > J2C Authentication Data**. The **J2C Authentication Data Entries** panel is displayed.
 - b. Click **New**.
 - c. Enter a unique alias, a value user ID, a valid password, and a short description (optional).
 - d. Click **OK** or **Apply**. No validation for the user ID and password is required.
 - e. Click **Save**. For a Network Deployment installation, make sure that a file synchronized operation is performed to propagate the changes to other nodes.

A new J2C authentication data entry is created or an old entry is removed. The newly created entry is visible without restarting the application server process for use in the data source definition. But the entry is only in effect after the server is restarted. Specifically, the authentication data is loaded by an application server when starting an application and is shared among applications in the same application server.

If you create or update a data source that points to a newly created J2C authentication data alias, Test Connection fails to connect until you restart the deployment manager. After you restart the deployment manager, the J2C authentication data is reflected in the run-time configuration. Any changes to the J2C authentication data fields require a deployment manager restart for the changes to take effect.

This step defines authentication data that you can share among resource adapters and JDBC data sources.

Use the authentication data entry defined in the resource adapters or JDBC data sources.

Java 2 Connector authentication data entry settings:

Use this page as a central place for administrators to define authentication data, which includes user identities and passwords. These values can reference authentication data entries by resource adapters, data sources, and other configurations that require authentication data using an alias.

You can display this page directly from the JAAS configuration page or from other pages for resources that use J2C authentication data entries. For example, to view this administrative page, you can click either **Security > JAAS Configuration > J2C Authentication Data Entries** or **Resources > WebSphere JMS Provider > WebSphere Queue Connection Factories > *connection_factory* > J2C Authentication Data Entries**.

Deleting authentication data entries: Be careful when deleting authentication data entries. If the deleted authentication data is used by other configurations, the initializing resources process fails.

Define a new authentication data entry by clicking **New**.

Alias:

Specifies the name of the authentication data entry.

| | |
|-------------------|--------|
| Data type: | String |
| Units: | String |
| Default: | None |

User ID:

Specifies the user identity.

| | |
|-------------------|--------|
| Data type: | String |
|-------------------|--------|

Description:

Specifies an optional description of the authentication data entry. For example, this authentication data entry is used to connect to DB2.

| | |
|-------------------|--------|
| Data type: | String |
|-------------------|--------|

Programmatic login

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

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

- In a client program
- In a server program

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

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

Applications can present site-specific login forms by using the WebSphere Application Server `form-login` type. The Java 2 Platform, Enterprise Edition (J2EE) specification defines form login as one of the authentication methods for Web applications. However, the Servlet Version 2.2 specification does not define a mechanism for logging out. WebSphere Application Server extends J2EE by also providing a form-logout mechanism.

Java Authentication and Authorization Service programmatic login

Java Authentication and Authorization Service (JAAS) is a new feature in WebSphere Application Server. It is also mandated by the J2EE 1.3 Specification.

JAAS is a collection of WebSphere strategic authentication APIs and replace of the CORBA programmatic login APIs. WebSphere Application Server provides some extensions to JAAS:

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

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

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

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

For more information, see Example: JAAS programmatic login.

- For the pure Java client application or client container application, make sure that the host name and the port number of the target JNDI bootstrap properties are specified properly. See the Developing applications that use CosNaming (CORBA Naming interface) section for details.
- If the application uses custom JAAS login configuration, make sure that the custom JAAS login configuration is properly defined. See the Java Authentication and Authorization Service (JAAS) Login Configuration 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 the application, Using policytool to edit policy file and Configuring was.policy sections for details. For more details of which APIs are protected by Java 2 Security permissions, check the IBM Developer Kit, Java edition; JAAS and the WebSphere public APIs Javadoc for more details. The following list indicates the APIs used in the samples code provided in this documentation.
 - javax.security.auth.login.LoginContext constructors are protected by javax.security.auth.AuthPermission "createLoginContext".
 - javax.security.auth.Subject.doAs() and com.ibm.websphere.security.auth.WSSubject.doAs() are protected by javax.security.auth.AuthPermission "doAs".
 - javax.security.auth.Subject.doAsPrivileged() and com.ibm.websphere.security.auth.WSSubject.doAsPrivileged() are protected by javax.security.auth.AuthPermission "doAsPrivileged".

- `com.ibm.websphere.security.auth.WSSubject`: Due to a design oversight in the JAAS 1.0, `javax.security.auth.Subject.getSubject()` does not return the Subject associated with the thread of execution inside a `java.security.AccessController.doPrivileged()` code block. This can present an inconsistent behavior that is problematic and causes undesirable effort. The `com.ibm.websphere.security.auth.WSSubject` API provides a work around to associate Subject to thread of execution. The `com.ibm.websphere.security.auth.WSSubject` API extends the JAAS model to J2EE resources for authorization checks. The Subject associated with the thread of execution within `com.ibm.websphere.security.auth.WSSubject.doAs()` or `com.ibm.websphere.security.auth.WSSubject.doAsPrivileged()` code block is used for J2EE resources authorization checks.
- UI support for defining new JAAS login configuration: You can configure JAAS login configuration in the administrative console and store it in WCCM (WebSphere Common Configuration Model). Applications can define new JAAS login configuration in the administrative console and the data is persisted in the configuration repository (stored in the WCCM). However, WebSphere Application Server still supports the default JAAS login configuration format (plain text file) provided by the JAAS default implementation. But if there are duplication login configurations defined in both the WCCM and the plain text file format, the one in the WCCM takes precedence. There are advantages to defining the login configuration in the WCCM:
 - UI support in defining JAAS login configuration.
 - You can manage the JAAS configuration login configuration centrally.
 - The JAAS configuration login configuration is distributed in a Network Deployment installation.
- WebSphere Application Server JAAS login configurations: WebSphere Application Server provides JAAS login configurations for application to perform programmatic authentication to the WebSphere Application Server security run time. These WebSphere Application Server JAAS login configurations perform authentication to the WebSphere Application Server configured authentication mechanism (SWAM or LTPA) and user registry (Local OS, LDAP, or Custom) based on the authentication data supplied. The authenticated Subject from these JAAS login configurations contain the required Principal and Credentials that can be used by WebSphere Application Server security run time to perform authorization checks on J2EE role-based protected resources. Here is the JAAS login configurations provided by WebSphere Application Server:
 - *WSLogin JAAS login configuration*: A generic JAAS login configuration that a Java Client, client container application, servlet, JSP file, enterprise bean, and so on, can use to perform authentication based on a user ID and password, or a token to the WebSphere Application Server security run time. However, this does not honor the `CallbackHandler` specified in the Client Container deployment descriptor.
 - *ClientContainer JAAS login configuration*: This JAAS login configuration honors the `CallbackHandler` specified in the client container deployment descriptor. The login module of this login configuration uses the `CallbackHandler` in the client container deployment descriptor if one is specified, even if the application code specified one `CallbackHandler` in the `LoginContext`. This is for client container application.
 - Subject authenticated with the previously mentioned JAAS login configurations contain a `com.ibm.websphere.security.auth.WSPincipal` and a `com.ibm.websphere.security.auth.WSCredential`. If the authenticated Subject is passed the in `com.ibm.websphere.security.auth.WSSubject.doAs()` (or the other `doAs()` methods), the WebSphere Application Server security run time can

perform authorization checks on J2EE resources, based on the Subject `com.ibm.websphere.security.auth.WSCredential`.

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

Finding the root cause login exception from a JAAS login

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

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

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

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

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

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

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

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

    // keep looping until there are no more embedded WSLoginFailedException or
    // WSSecurityException exceptions
```

```

        while (true)
        {
if (e instanceof com.ibm.websphere.security.auth.WSLoginFailedException)
        {
            temp_exception = ((com.ibm.websphere.security.auth.WSLoginFailedException)
e).getCause();
        }
        else if (e instanceof com.ibm.websphere.security.WSSecurityException)
        {
            temp_exception = ((com.ibm.websphere.security.WSSecurityException)
e).getCause();
        }
        else if (e instanceof com.ibm.ws.security.registry.nt.NTException)
            // get NT specific error code, if configured
        {
System.out.println ("Error code from NT exception: "
+ ((com.ibm.ws.security.registry.nt.NTException)e).getErrorCode());
return e;
        }
        else if (e instanceof com.ibm.ws.security.registry.unix.UnixRegistryException)
            // get Unix specific error code, if configured
        {
System.out.println ("Error code from Unix exception: "
+ ((com.ibm.ws.security.registry.unix.UnixRegistryException)e).getErrorCode());
return e;
        }
        else if (e instanceof javax.naming.NamingException)
            // check for Ldap embedded exception
        {
temp_exception = ((javax.naming.NamingException)e).getRootCause();
        }
        else if (e instanceof your_custom_exception_here)
        {
            // your custom processing here, if necessary
        }
        else
        {
            // this exception is not one of the types we are looking for,
            // lets return now, this is the root from the WebSphere
            // Application Server perspective
            return root_exception;
        }

            if (temp_exception != null)
            {
                // we have an exception, let's go back 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 is suitable because it shows the user what happened. The following API can be issued to obtain the root cause exception:

```

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

```

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

Enabling root cause login exception propagation to pure Java clients

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

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

Non-prompt programmatic login

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

```

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

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

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

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

```

```

}

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

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

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

```

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

User interface prompt programmatic login

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

```

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

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

// create a LoginContext and specify a CallbackHandler implementation
// CallbackHandler implementation determine how authentication data is collected
// in this case, the authentication date is collected by GUI login prompt
// and pass to the authentication mechanism implemented by the LoginModule.
} catch (javax.security.auth.login.LoginException e) {

```

```

System.err.println("ERROR: failed to instantiate a LoginContext and the exception: "
+ e.getMessage());
e.printStackTrace();

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

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

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

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

```

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

Stdin prompt programmatic login

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

```

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

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

```

```

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

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

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

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

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

```

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

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

WebSphere Application Server supports plugging in a custom Java Authentication and Authorization Service (JAAS) login module before or after the WebSphere

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

Attention: Information added to the Subject from the custom login module is not propagated downstream in the current implementation.

Only the identity gets propagated. Using a custom login module, the identity can be used to generate similar data downstream.

WebSphere Application Server server-side authentication mechanisms, both Simple WebSphere Authentication Mechanism (SWAM) and Lightweight Third Party Authentication (LTPA), are implemented as the JAAS LoginModule. The SWAM and LTPA login configurations are defined in the WebSphere Common Configuration Module (WCCM) model within the `wsjaas.conf` properties file. There are two sets of JAAS login configuration defined in the WCCM model: the application login configuration and the system login configuration. SWAM and LTPA are defined in the system login configuration.

WebSphere Application Server Version 5.1 enables you to customize the LTPA system configuration. A new `LTPA_WEB` login configuration exists in the WCCM model, which is used exclusively by the Web container. The LTPA login configuration is used by the EJB container. The `LTPA_WEB` login configuration contains the `com.ibm.ws.security.web.AuthenLoginModule` module and the LTPA login configuration contains the `com.ibm.ws.security.server.lm.ltpaLoginModule` module. The new `AuthenLoginModule` is an enhanced `ltpaLoginModule` that can process the `HttpServletRequest` object, the `HttpServletResponse` object, and the Web application name that are passed in using the `CallbackHandler`.

WebSphere Application Server Version 5.1 exposes the server-side login configuration so that you may customize the LTPA authentication process. Both the LTPA login configuration and the `LTPA_WEB` login configuration enable you to add a custom `LoginModule` both before and after the `ltpaLoginModule` and the `AuthenLoginModule`. For example, one might add a `LoginModule` to perform principal or credential mapping function either before or after the authentication. In the Web configuration, a custom `LoginModule` might perform principal mapping using a combination of authenticated user name, application name, and URL patterns. WebSphere Application Server does not allow you to rename nor remove the LTPA and `LTPA_WEB` login configurations.

WebSphere Application Server Version 5.1 and all prior version 5.0.x releases do not expose the system login configuration in the administration console. The WebSphere Application Server version 5.0.x releases do not support the modification of the system login configuration. In WebSphere Application Server Version 5.1, the system login configuration is not exposed in the administrative console. However, you can modify the system login configuration using the `wsadmin` scripting utility. The following sample JACL script adds a custom `LoginModule` into the Lightweight Third-party Authentication (LTPA) Web system login configuration:

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


```

1. #####
2. #
3. # Open security.xml
4. #
5. #####
6.
7.
8. set sec [$AdminConfig getid /Cell:hillside/Security:/]
9.
10.
11. #####
12. #
13. # Locate systemLoginConfig
14. #
15. #####
16.
17.
18. set slc [$AdminConfig showAttribute $sec systemLoginConfig]
19.
20. set entries [lindex [$AdminConfig showAttribute $slc entries] 0]
21.
22.
23. #####
24. #
25. # Append a new LoginModule to LTPA_WEB
26. #
27. #####
28.
29. foreach entry $entries {
30.     set alias [$AdminConfig showAttribute $entry alias]
31.     if {$alias == "LTPA_WEB"} {
32.         set newJAASLoginModuleId [$AdminConfig create JAASLoginModule
33.             $entry {{moduleClassName
34.                 "com.ibm.ws.security.common.auth.module.proxy.WSLoginModuleProxy"}}]
35.         set newPropertyId [$AdminConfig create Property
36.             $newJAASLoginModuleId {{name delegate}{value
37.                 "com.ABC.security.auth.CustomLoginModule"}}]
38.         $AdminConfig modify $newJAASLoginModuleId
39.             {{authenticationStrategy REQUIRED}}
40.         break
41.     }
42. }
43. #
44. # save the change
45. #
46. #####
47. $AdminConfig save

```

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

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

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

```
48. #####
49. #
50. # Open security.xml
51. #
52. #####
53.
54.
55. set sec [$AdminConfig getid /Cell:hillside/Security:/]
56.
57.
58. #####
59. #
60. # Locate systemLoginConfig
61. #
62. #####
63.
64.
65. set slc [$AdminConfig showAttribute $sec systemLoginConfig]
66.
67. set entries [lindex [$AdminConfig showAttribute $slc entries] 0]
68.
69.
70. #####
71. #
72. # Remove the LTPA_WEB login configuration
73. #
74. #####
75.
76. foreach entry $entries {
77.     set alias [$AdminConfig showAttribute $entry alias]
78.     if {$alias == "LTPA_WEB"} {
79.         $AdminConfig remove $entry
80.         break
81.     }
82. }
83.
84.
85. #####
86. #
87. # save the change
88. #
89. #####
90.
91. $AdminConfig save
```

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

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

```
92. #####
93. #
94. # Open security.xml
95. #
96. #####
97.
98.
99. set sec [$AdminConfig getid /Cell:hillside/Security:/]
100.
101.
102. #####
103. #
104. # Locate systemLoginConfig
105. #
106. #####
107.
108.
109. set slc [$AdminConfig showAttribute $sec systemLoginConfig]
110.
111. set entries [lindex [$AdminConfig showAttribute $slc entries] 0]
112.
113.
114.
115. #####
116. #
117. # Recreate the LTPA_WEB login configuration
118. #
119. #####
120.
121.
122. set newJAASConfigurationEntryId [$AdminConfig create JAASConfigurationEntry
    $slc {{alias LTPA_WEB}}]
123.
124. set newJAASLoginModuleId [$AdminConfig create JAASLoginModule
    $newJAASConfigurationEntryId
    {{moduleName
    "com.ibm.ws.security.common.auth.module.proxy.WSLoginModuleProxy"}}]
125.
126. set newPropertyId [$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 5.1 ItpaLoginModule and AuthenLoginModule use the shared state to save state information so that custom LoginModules can modify the information. The ItpaLoginModule initializes the callback array in the login() method using the following code. The callback array is created by ItpaLoginModule only if an array is not defined in the shared state area. In the following code sample, the error handling code was removed to make the sample concise. If you insert a custom LoginModule before the ItpaLoginModule, custom LoginModule might follow the same style to save the callback into the shared state.

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

```

138.         Callback callbacks[] = null;
139.         if (!sharedState.containsKey(
                com.ibm.wsspi.security.auth.callback.Constants.
                CALLBACK_KEY)) {
140.             callbacks = new Callback[3];
141.             callbacks[0] = new NameCallback("Username: ");
142.             callbacks[1] = new PasswordCallback("Password: ", false);
143.             callbacks[2] = new com.ibm.websphere.security.auth.callback.
                WSCredTokenCallbackImpl( "Credential Token: ");
144.             try {
145.                 callbackHandler.handle(callbacks);
146.             } catch (java.io.IOException e) {
147.                 . . .
148.             } catch (UnsupportedCallbackException uce) {
149.                 . . .
150.             }
151.             sharedState.put(
                com.ibm.wsspi.security.auth.callback.Constants.CALLBACK_KEY,
                callbacks);
152.         } else {
153.             callbacks = (Callback [])
                sharedState.get( com.ibm.wsspi.security.auth.callback.
                Constants.CALLBACK_KEY);
154.         }

```

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

AuthenLoginModule uses the following code to initialize the callback array:

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

```
155.         Callback callbacks[] = null;
156.         if (!sharedState.containsKey(
            com.ibm.wsspi.security.auth.callback.Constants.
            CALLBACK_KEY)) {
157.             callbacks = new Callback[6];
158.             callbacks[0] = new NameCallback("Username: ");
159.             callbacks[1] = new PasswordCallback("Password: ", false);
160.             callbacks[2] =
                new com.ibm.websphere.security.auth.callback.WSCredTokenCallbackImpl(
                "Credential Token: ");
161.             callbacks[3] =
                new com.ibm.wsspi.security.auth.callback.WSServletRequestCallback(
                "HttpServletRequest: ");
162.             callbacks[4] =
                new com.ibm.wsspi.security.auth.callback.WSServletResponseCallback(
                "HttpServletResponse: ");
163.             callbacks[5] =
                new com.ibm.wsspi.security.auth.callback.WSApplicationContextCallback(
                "ApplicationContextCallback: ");
164.             try {
165.                 callbackHandler.handle(callbacks);
166.             } catch (java.io.IOException e) {
167.                 . . .
168.             } catch (UnsupportedCallbackException uce {
169.                 . . .
170.             }
171.             sharedState.put( com.ibm.wsspi.security.auth.callback.
                Constants.CALLBACK_KEY, callbacks);
172.         } else {
173.             callbacks = (Callback []) sharedState.get(
                com.ibm.wsspi.security.auth.callback.
                Constants.CALLBACK_KEY);
174.         }
```

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

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

```
175.         HashMap appContext = new HashMap(2);
176.         appContext.put(
```

```

        com.ibm.wsspi.security.auth.callback.Constants.WEB_APP_NAME,
        web_application_name);
177.    appContext.put(
        com.ibm.wsspi.security.auth.callback.Constants.REDIRECT_URL,
        errorPage);

```

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

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

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

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

```

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

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

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

```

```

catch (Exception e)
{
    // log error
}

```

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

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` (documented in the Javadoc information) so that you can put or set data in the hashmap within the credential.

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

```

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

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

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

```

Requirements: You need the following Java 2 Security permissions to run this API: `permission javax.security.auth.AuthPermission "wssecurity.getRunAsSubject;"`.

Overriding the RunAs Subject on the Thread:

To extend the function provided by the Java Authentication and Authorization Service (JAAS) application programming interfaces (APIs), you can set the RunAs subject (or invocation subject) with a different valid entry that is used for outbound requests on this execution thread.

This setting gives flexibility for associating the Subject with all remote calls on this thread without having to do a WSSubject.doAs() 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";
```

User revocation from a cache: In WebSphere Application Server, Version 5.0.2 and later, revocation of a user from the security cache using an MBean interface is supported. The following Java Command Language (JACL) revokes a user when given the realm and user ID, and cycles through all security administration MBean instances returned for the entire cell when run from the Deployment Manager WSADMIN. The command also purges the user from the cache during each process.

Note: This procedure can be called from a JACL script.

Attention: In some of the following lines of code, the lines have been split onto two or more lines.

```
proc revokeUser {realm userid} {
    global AdminControl AdminConfig

    if {[catch {$AdminControl queryNames WebSphere:type=SecurityAdmin,*}
```



```

        result]] {
puts stdout "\$AdminControl queryNames WebSphere:type=SecurityAdmin,*
        caught an exception $result\n"
return
} else {
if {$result != {}} {
    foreach secBean $result {
        if {$secBean != {} || $secBean != "null"} {
            if {[catch {$AdminControl invoke $secBean
purgeUserFromAuthCache "$realm $userid"} result]} {
                puts stdout "\$AdminControl invoke $secBean
purgeUserFromAuthCache $realm $userid caught an
exception $result\n"
                return
            } else {
                puts stdout "\nUser $userid has been purged from the
cache of process $secBean\n"
            }
            } else {
                puts stdout "unable to get securityAdmin Mbean, user
$userid not revoked"
            }
        }
    }
} else {
puts stdout "Security Mbean was not found\n"
return
}
}
return true
}

```

Authentication protocol for EJB security

In WebSphere Application Server Version 5, two authentication protocols are available to choose from: Secure Authentication Service (SAS) and Common Secure Interoperability Version 2 (CSIv2). SAS is the authentication protocol used by all previous releases of WebSphere Application Server and is maintained for backwards compatibility. The Object Management Group (OMG) has defined a new 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.

Invoking EJB methods in a secure WebSphere Application Server environment requires an authentication protocol to determine the level of security and the type of authentication, which occur between any given client and server for each request. It is the job of the authentication protocol during a method invocation to merge the server authentication requirements (determined by the object Interoperable Object Reference (IOR)) with the client authentication requirements (determined by the client configuration) and come up with an authentication policy specific to that client and server pair.

The authentication policy makes the following decisions, among others, which are all based on the client and server configurations:

- What kind of connection can you make to this server--SSL or TCP/IP?
- If Secure Sockets Layer (SSL) is chosen, how strong is the encryption of the data?
- If SSL is chosen, do you authenticate the client using client certificates?

- Do you authenticate the client with a user ID and password? Does an existing credential exist?
- Do you assert the client identity to downstream servers?
- Given the configuration of the client and server, can a secure request proceed?

You can configure both protocols (SAS and CSIV2) to work simultaneously. If a server supports both protocols, it exports an IOR containing tagged components describing the configuration for SAS and CSIV2. If a client supports both protocols, it reads tagged components for both CSIV2 and SAS. If the client supports both and the server supports both, CSIV2 is used. However, if the server supports SAS (for example, it is a previous WebSphere Application Server release) and the client supports both, the client chooses SAS for this request, since the SAS protocol is what both have in common. Choose a protocol by specifying the `com.ibm.CSI.protocol` property on the client side and configuring through the administrative console on the server side. More details are included in the SAS and CSIV2 properties articles.

Common Secure Interoperability Specification, Version 2

The Common Secure Interoperability Specification, Version 2 (CSIV2) defines the Security Attribute Service (SAS) that enables interoperable authentication, delegation and privileges. The CSIV2 SAS and SAS protocols are entirely different. The CSIV2SAS protocol is a subcomponent of CSIV2 that supports SSL and interoperability with the EJB Specification, Version 2.0.

Security Attribute Service

The Common Secure Interoperability Specification, Version 2 Security Attribute Service (CSIV2 SAS) protocol is designed to exchange its protocol elements in the service context of a General Inter-ORB Protocol (GIOP) request and reply messages that are communicated over a connection-based transport. The protocol is intended for use in environments where transport layer security, such as that available through Secure Sockets Layer (SSL) and Transport Layer Security (TLS), is used to provide message protection (that is, integrity and or confidentiality) and server-to-client authentication. The protocol provides client authentication, delegation, and privilege functionality that might be applied to overcome corresponding deficiencies in an underlying transport. The CSIV2 SAS protocol facilitates interoperability by serving as the higher-level protocol under which secure transports can be unified.

Connection and request interceptors

The authentication protocols used by WebSphere Application Server are add-on Interoperable Inter-ORB Protocol (IIOP) services. IIOP is a request-and-reply communications protocol used to send messages between two Object Request Brokers (ORBs). For each request made by a client ORB to a server ORB, an associated reply is made by the server ORB back to the client ORB. Prior to any request flowing, a connection between the client ORB and the server ORB must be established over the TCP/IP transport (SSL is a secure version of TCP/IP). The client ORB invokes the authentication protocol client connection interceptor, which is used to read the tagged components in the IOR of the object located on the server. As mentioned previously, this is where the authentication policy is established for the request. Given the authentication policy (a coalescing of the server configuration with the client configuration), the strength of the connection is returned to the ORB. The ORB makes the appropriate connection, usually over SSL.

After the connection is established, the client ORB invokes the authentication protocol client request interceptor, which is used to send security information other than what is established by the transport. The security information includes the user ID and password token (authenticated by the server), an authentication mechanism-specific token (validated by the server), or an identity assertion token. Identity assertion is a way for one server to trust another server without the need to reauthenticate or revalidate the originating client. However, some work is required for the server to trust the upstream server. This additional security information is sent with the message in a *service context*. A service context has a registered identifier so that the server ORB can identify which protocol is sending the information. The fact that a service context contains a unique identity is another way for WebSphere Application Server to support both SAS and CSiv2 simultaneously because both protocols have different service context IDs. After the client request interceptor finishes adding the service context to the message, the message is sent to the server ORB.

When the message is received by the server ORB, the ORB invokes the authentication protocol server request interceptor. This interceptor looks for the service context ID known by the protocol. When both SAS and CSiv2 are supported by a server, two different server request interceptors are invoked and both interceptors look for different service context IDs. However, only one finds a service context for any given request. When the server request interceptor finds a service context, it reads the information in the service context. A method is invoked to the security server to authenticate or validate client identity. The security server either rejects the information or returns a credential. A credential contains additional information about the client, retrieved from the user registry so that authorization can make the appropriate decision. Authorization is the process of determining if the user can invoke the request based on the roles applied to the method and the roles given to the user. If the request is rejected by the security server, a reply is sent back to the client without ever invoking the business method.

If a service context is not found by the CSiv2 server request interceptor, the interceptor then looks at the transport connection to see if a client certificate chain was sent. This is done when SSL client authentication is configured between the client and server. If a client certificate chain is found, the distinguished name (DN) is extracted from the certificate and is used to map to an identity in the user registry. If the user registry is Lightweight Directory Access Protocol (LDAP), the search filters defined in the LDAP registry configuration determine how the certificate maps to an entry in the registry. If the user registry is local OS, the first attribute of the distinguished name (DN) maps to the user ID of the registry. This attribute is typically the common name. If the certificate does not map, no credential is created and the request is rejected. When invalid security information is presented, the method request is rejected and a `NO_PERMISSION` exception is sent back with the reply. However, when no security information is presented, an unauthenticated credential is created for the request and the authorization engine determines if the method gets invoked or not. For an unauthenticated credential to invoke an Enterprise JavaBean (EJB) method, either no security roles are defined for the method or a special **Everyone** role is defined for the method.

When the method invocation is completed in the EJB container, the server request interceptor is invoked again to complete server authentication and a new reply service context is created to inform the client request interceptor of the outcome. This process is typically for making the request *stateful*. When a stateful request is made, only the first request between a client and server requires that security information is sent. All subsequent method requests need to send a unique context

ID only so that the server can look up the credential stored in a session table. The context ID is unique within the connection between a client and server.

Finally, the method request cycle is completed by the client request interceptor receiving a reply from the server with a reply service context providing information so the client side stateful context ID can be confirmed and reused. Specifying a stateful client is done through the property `com.ibm.CSI.performStateful` (true/false). Specifying a stateful server is done through the administrative console configuration.

Authentication protocol flow

Step 1:

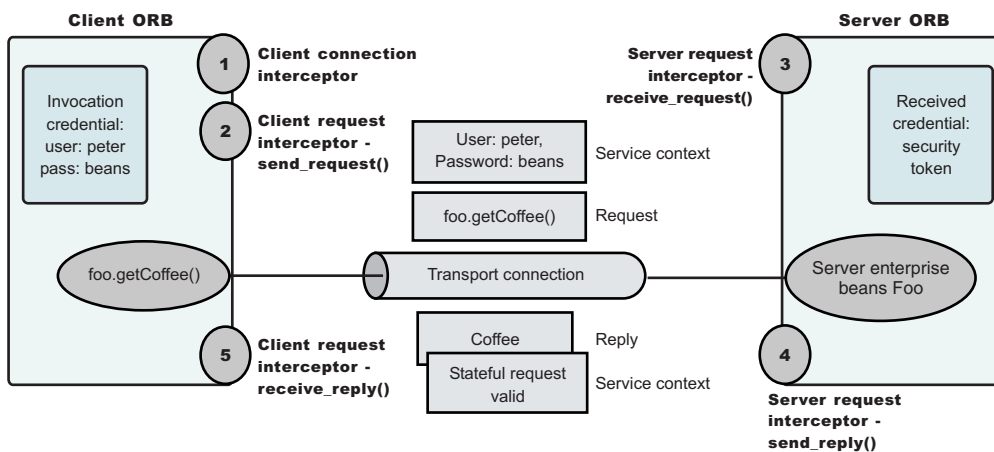
Client ORB calls the connection interceptor to create the connection.

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.



Step 5:

Client ORB calls the request interceptor so that the client can clean up and set the session status as good or bad.

Step 4:

Server ORB calls the request interceptor so that security can send information back to the client with the reply.

. Authentication protocol flow

Authentication policy for each request

The authentication policy of a given request determines the security protection between a client and a server. A client or server authentication protocol configuration can describe required features, supported features and non-supported features. When a client requires a feature, it can only talk to servers that either require or support that feature. When a server requires a feature, it can only talk to clients that either require or support that feature. When a client supports a feature, it can talk to a server that supports or requires that feature, but can also talk to servers that do not support the feature. When a server supports a feature, it can talk to a client that supports or requires the feature, but can also talk to clients that do not support the feature (or chose not to support the feature).

For example, for a client to support client certificate authentication, some setup is required to either generate a self-signed certificate or to get one from a certificate authority (CA). Some clients might not need to complete these actions, therefore, you can configure this feature as not supported. By making this decision, the client cannot communicate with a secure server requiring client certificate authentication. Instead, this client can choose to use the user ID and password as the method of authenticating itself to the server.

Typically, supporting a feature is the most common way of configuring features. It is also the most successful during run time because it is more forgiving than requiring a feature. Knowing how secure servers are configured in your domain, you can choose the right combination for the client to ensure successful method invocations and still get the most security. If you know that all of your servers support both client certificate and user ID and password authentication for the client, you might want to require one and not support the other. If both the user ID and password and the client certificate are supported on the client and server, both are performed but user ID and password take precedence at the server. This action is based on the CSiv2 specification requirements.

Common Secure Interoperability Version 2 features

The following Common Secure Interoperability Version 2 (CSiv2) features are available in IBM WebSphere Application Server: SSL client certificate authentication, message layer authentication and identity assertion.

- SSL Client Certificate authentication.

An additional way to authenticate a client to a server using SSL client authentication.

- Message Layer Authentication.

Authenticates credential information and sends that information across the network so that a receiving server can interpret it.

- Identity Assertion.

Supports a downstream server in accepting the client identity established on an upstream server, without having to reauthenticate. The downstream server trusts the upstream server.

Identity assertion

Identity assertion is the invocation credential that is asserted to the downstream server.

When a client authenticates to a server, the received credential is set. When authorization checks the credential to determine whether access is permitted, it also sets the *invocation* credential so that if the EJB method calls another EJB method located on other servers, the invocation credential can be the identity used to invoke the downstream method. Depending on the RunAs mode for the enterprise beans, the invocation credential is set as the originating client identity, the server identity, or a specified different identity. Regardless of the identity that is set, when identity assertion is enabled, it is the invocation credential that is asserted to the downstream server.

The invocation credential identity is sent to the downstream server in an identity token. In addition, the sending server identity, including the password or token, is sent in the client authentication token when basic authentication is enabled. The sending server identity is sent through a Secure Sockets Layer (SSL) client certification authentication when client certificate authentication is enabled. Basic authentication takes precedence over client certificate authentication. Both tokens are needed by the receiving server to accept the asserted identity. The receiving server completes the following actions to accept the asserted identity:

- The server determines whether the sending server identity, sent with a basic authentication token or with a SSL client certificate, is on the trusted principal list of the receiving server. The server determines whether the sending server can send an identity token to the receiving server.
- After it is determined that the sending server is on the trusted list, the server authenticates the sending server to verify its identity.
- The server is authenticated by comparing the user ID and password from the sending server to the receiving server, or it might require a real authenticated

call. If the credentials of the sending server are authenticated and on the trusted principal list, then the server proceeds to evaluate the identity token.

Evaluation of the identity token consists of the following four identity formats that exist in an identity token:

- Principal name
- Distinguished name
- Certificate chain
- Anonymous identity

The product servers that receive authentication information typically support all four identity types. The sending server decides which one is chosen, based on how the original client authenticated. The existing type depends on how the client originally authenticates to the sending server. For example, if the client uses Secure Sockets Layer (SSL) client authentication to authenticate to the sending server, then the identity token sent to the downstream server contains the certificate chain. This information is important because it permits the receiving server to perform its own certificate chain mapping. It enables more interoperability with other vendors and platforms.

After the identity format is understood and parsed, the identity maps to a credential. For an ITPrincipal identity token, this identity maps one-to-one with the user ID fields. For an ITDistinguishedName identity token, the mapping depends on the user registry. For Lightweight Directory Access Protocol (LDAP), the configured search filter determines how the mapping occurs. For LocalOS, the first attribute of the distinguished name (DN), which is typically the same as the common name, maps to the user ID of the registry. For an ITCertChain identity token, see the section, Map certificates to users for details on how this action is performed for the LDAP user registry. For LocalOS, the first attribute of the DN in the certificate is used to map to the user ID in the registry.

Some user registry methods are called to gather additional credential information used by authorization. In a stateful server, this action completes once for the sending server and receiving server pair where the identity tokens are the same. Subsequent requests are made through a session ID.

Identity assertion is only available using the Common Secure Interoperability Version 2 (CSIv2) protocol.

Message layer authentication

Defines the credential information and sends that information across the network so that a receiving server can interpret it.

When you send authentication information across the network using a token (whether the token is a user ID and password token, that is, Generic Security Services Username Password (GSSUP), or a mechanism-specific format token, Lightweight Third Party Authentication (LTPA), for example), the transmission is considered message layer authentication because the data is sent along with the message inside a service context.

A pure Java client uses basic authentication (GSSUP) as the authentication mechanism to establish client identity. However, a servlet can use either basic authentication (GSSUP) or the authentication mechanism of the server (LTPA) to send security information in the message layer. Use LTPA by authenticating or mapping the basic authentication credentials to the security mechanism of the server.

The security token contained in a token-based credential is authentication mechanism-specific. That is, the way the token is interpreted is only known by the authentication mechanism. Therefore, each authentication mechanism has an object ID (OID) representing it. The OID and the client token are sent to the server, so that the server knows which mechanism to use when reading and validating the token. The following list contains the OIDs for each mechanism:

BasicAuth (GSSUP): oid:2.23.130.1.1.1

LTPA: oid:1.3.18.0.2.30.2 SWAM: No OID because it is not forwardable

On the server, the authentication mechanisms can interpret the token and create a credential, or they can authenticate basic authentication data from the client, and create a credential. Either way, the created credential is the *received* credential that the authorization check uses to determine if the user has access to invoke the method. You can specify the authentication mechanism by using the `com.ibm.CORBA.authenticationTarget` property on the client side. (Basic authentication is currently the only valid value.) You can configure the server through the administrative console.

While this property tells you which authentication mechanism to use, you also need to specify whether you want to perform authentication over the message layer (that is, get a BasicAuth or token-based credential). To complete this task, specify the `com.ibm.CSI.performClientAuthenticationRequired` (**True** or **False**) and `com.ibm.CSI.performClientAuthenticationSupported` (**True** or **False**) properties. Indicating that client authentication is required implies that it must be done for every request. Indicating that the authentication mechanism is supported implies that it might be done but is not required. For some servers, this option is appropriate if no resources are protected. In most cases it is a best practice to indicate that this mechanism is supported so that client authentication is performed if both the client and server support it. Client authentication is not performed when communicating with certain servers that do not want security, yet the method requests still succeed.

Configuring authentication retries

Situations occur where you want a prompt to reappear 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 an invalid user ID and password
- 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. The property used to enable or disable authentication retries is `com.ibm.CORBA.authenticationRetryEnabled` (**True** or **False**). The property used to specify the number of retry attempts is `com.ibm.CORBA.authenticationRetryCount`.

Immediate validating of a basic authentication login

In WebSphere Application Server Version 5, a new behavior is defined during `request_login` for a BasicAuth login. In prior releases, a BasicAuth login takes the

user ID and password entered through the loginSource method and creates a BasicAuth credential. If the user ID or password is invalid, the client program does not find out until the first method request is attempted. When the user ID or password is specified during a prompt or programmatic login, the user ID and password are authenticated by default with the security server, with a True or False being returned as the result. If False, an org.omg.SecurityLevel2.LoginFailed exception is returned to the client indicating the user ID and password are invalid. If True, then the BasicAuth credential is returned to the caller of the request_login. To disable this feature on the pure client, specify com.ibm.CORBA.validateBasicAuth=false. By default this feature is set to True. On the server side, specify this property in the security dynamic properties.

Secure Sockets Layer client certificate authentication

An additional way to authenticate a client to a server is using Secure Sockets Layer (SSL) client authentication.

Using SSL client authentication is another way of authenticating a client to a server. This form of authentication does not occur at the message layer as described previously (using a user ID and password or tokens). This authentication occurs during the connection handshake using SSL certificates. When the client is configured with a personal certificate in the SSL keystore file, which indicates that SSL client authentication is desired and the server supports SSL client authentication, the following actions occur to establish the identity on the client side.

When a method request is invoked in the client code to a remote enterprise bean, the Object Request Broker (ORB) invokes the client connection interceptor to establish a connection with the server. Because the configuration specifies SSL, and SSL client authentication, the connection type is SSL and the SSL handshake sends the client certificate to the server to validate. If the client certificate does not validate, the connection is not established and an exception is sent back to the client code where the method is invoked, which indicates the failure. If the client certificate is validated, then a connection opens between the client and the server.

The ORB proceeds to call the client request interceptor, which might be busy. If basic authentication is also configured, for example, then the user might be prompted for a user ID and password. Because this action is not necessary, disable this option in the configuration if the SSL certificate is the desired identity against which to invoke the method. If no message layer security exists, then no security context is created and associated with the request.

After the server receives the request, the server-side request interceptor checks for a security context. Because the server does not find a service context, it checks the server socket for a client certificate chain that contains the client identity. In this case, the server finds the certificate chain from the client. The identity in the certificate chain is valid because the connection was made. To create a credential, map the identity from the certificate to the user registry. This action is done differently based on the type of authentication mechanism. Mapping a certificate to a credential is done differently based on the user registry type. See the, Map certificates to users article, for details on how this mapping is performed for the Lightweight Directory Access Protocol (LDAP) user registry. For local OS, the first attribute of the distinguished name (DN) in the certificate is used to map to the user ID in the registry.

One benefit of SSL client certificate authentication is that it optimizes authentication performance, because an SSL connection is typically created anyway.

The extra overhead of sending the client certificate is minimal. While the client-side request interceptor performs no activity, the server side request interceptor maps the certificate to a credential. One disadvantage to this type of authentication is the complexity of setting up the keystore file on each client system.

To enable SSL client certificate authentication on the client side, you must enable the properties, such as SSL. This action is completed using the following two properties:

- `com.ibm.CSI.performTransportAssocSSLTLSRequired` (true or false)
- `com.ibm.CSI.performTransportAssocSSLTLSSupported` (true or false)

Indicating SSL is required implies that every request must generate an SSL connection key. If a server does not support SSL, then the request fails. After you enable SSL by either supporting it or requiring it, you can enable some of the SSL features.

To enable SSL client authentication, you can specify the following two properties:

- `com.ibm.CSI.performTLClientAuthenticationRequired` (true or false)
- `com.ibm.CSI.performTLClientAuthenticationSupported` (true or false)

The TL means *transport layer*. If you indicate that SSL client authentication is required, then you only limit the ability to communicate with servers that support SSL client authentication. For a server to support SSL client authentication, that server must have similarly configured properties through the administrative console, and have an SSL listener port that is open to handle mutual authentication handshakes. Configuration of server properties are done through the administrative console.

SSL client certificate authentication from a Java client is only available using the Common Secure Interoperability Version 2 (CSIV2) protocol.

Supported IBM protocols: Secure Authentication Service and Common Secure Interoperability Version 2

There are two authentication protocols supported by IBM. Secure Authentication Service (SAS) is the authentication protocol used by all previous releases of the WebSphere Application Server product. Common Secure Interoperability Version 2 (CSIV2) is implemented in WebSphere Application Server, Version 5 and is considered the strategic protocol.

You can configure both protocols 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 and the server support both protocols, CSIV2 is used. However, if the server supports SAS (for example, it is a previous WebSphere Application Server release) and the client supports both protocols, the client chooses SAS for this request. Choose a protocol using the `com.ibm.CSI.protocol` property on the client side and configure this protocol through the GUI on the server side.

Configuring Common Secure Interoperability Version 2 and Security Authentication Service authentication protocols

1. Determine how to configure security inbound and outbound at each point in your infrastructure.

For example, you might have a Java client communicating with an Enterprise JavaBean (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 only need to configure outbound security). The upstream EJB application server configures inbound security to handle the right type of authentication from the Java client. The upstream EJB application server utilizes the outbound security configuration when going to the downstream EJB application server.

This type of authentication might be different than what you expect from the Java client into the upstream EJB application server. Security might be tighter between the pure client and the first EJB server, depending on your infrastructure. The downstream EJB server utilizes the inbound security configuration to accept requests from the upstream EJB server. These two servers require similar configuration options as well. If the downstream EJB application server communicates to other downstream servers, then the outbound security might require a special configuration.

2. Specify the type of authentication. By default, authentication using a user ID and password is performed. Both Java client certificate authentication and identity assertion are disabled by default. If you want this type of basic authentication performed at every tier, use the CSIV2 authentication protocol configuration as is. However, if you have any special requirements where some servers authenticate differently from other servers, then consider how to configure CSIV2 to take advantage of its features.
3. Configure clients and servers. Configuring a pure Java client is done through the `sas.client.props` file where properties are modified. Configuring servers is always done from the administrative console, either from the Security navigation for cell-level configurations or from the application server Server security for server-level configurations. If you want some servers to authenticate differently from others, modify some of the server level configurations. When you modify the server-level configurations, you are overriding the cell-level configurations.

Common Secure Interoperability Version 2 and Security Authentication Service client configuration

A secure Java client requires configuration properties to determine how to perform security with a server. These configuration properties are typically put into a properties file somewhere on the client machine and referenced by specifying the following system property on the command line of the Java client. The syntax of this property accepts a valid URL with the protocol type, file.

```
-Dcom.ibm.CORBA.ConfigURL=file:/C:/WebSphere/AppServer/properties/sas.client.props
```

When this file is processed by the object request broker (ORB), security can be enabled between the Java client and the target server. If any syntax problems exist with the `ConfigURL` property and the `sas.client.props` file is not found, the Java client proceeds to connect insecurely. Errors display indicating the failure to read the `ConfigURL` property. Typically the problem is related to having two slashes after `file`, which is invalid.

The following properties are used to configure the SAS and CSIV2 authentication protocols:

- Security Authentication Service and Common Secure Interoperability Version 2 authentication protocol common settings for a client configuration
- CSIV2 authentication protocol client settings
- Security Authentication Service Authentication Protocol client settings

Security Authentication Service and Common Secure Interoperability Version 2 authentication protocol common settings for a client configuration: Use the following settings in the *install_dir\properties\sas.client.props* file to configure Security Authentication Service (SAS) and Common Secure Interoperability Version 2 (CSIV2) clients.

com.ibm.CORBA.securityEnabled:

Use to determine if security is enabled for the client process.

| | |
|----------------------|---------------|
| 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 previous version servers.
- 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 previous version servers and WebSphere Application Server Version 5 or later servers. Typically, specifying both provides greater interoperability with other servers.

| | |
|----------------------|------------------|
| Data type: | String |
| Default: | Both |
| Valid values: | ibm, csiv2, both |

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 SSL transport with this type of authentication is recommended because otherwise the password is not encrypted. The target server must support the specified authenticationTarget.

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.

| | |
|----------------------|-----------------|
| Data type: | String |
| Default: | BasicAuth |
| Valid values: | BasicAuth, LTPA |

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 past releases, BasicAuth logins only validated with the initial method request. During the first request, the user ID and password is sent to the server. This 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 it is not desirable 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 as simple because of error handling considerations.

| | |
|----------------------|---------------|
| Data type: | Boolean |
| Default: | True |
| Valid values: | True or 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 property com.ibm.CORBA.authenticationRetryCount.

| | |
|----------------------|---------------|
| Data type: | Boolean |
| Default: | True |
| Valid values: | True or 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.

| | |
|-------------------|---------|
| 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 only valid if message layer authentication occurs. If only transport layer authentication occurs, this property is ignored. When specifying properties, the following two additional properties need to 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. Unless you want the request to fail, do not set a credential as the invocation credential during a method request.

| | |
|----------------------|---|
| 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 only valid when `com.ibm.CORBA.loginSource=properties`. Also, set the `com.ibm.CORBA.loginPassword` property.

| | |
|-------------------|---|
| Data type: | String |
| Range: | Any string 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 only valid when `com.ibm.CORBA.loginSource=properties`. Also, set the `com.ibm.CORBA.loginUserid` property.

| | |
|-------------------|---|
| Data type: | String |
| Range: | Any string 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 used is the one found in the Interoperable Object Reference (IOR) for the current method request. The value of this property is used when `com.ibm.CORBA.loginSource=key file` is used.

| | |
|-------------------|---|
| Data type: | String |
| Default: | C:/WebSphere/AppServer/properties/wsserver.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 in time that the login prompt stays available before it is considered a failed login.

| | |
|-------------------|-------------------------------|
| 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.

| | |
|-------------------|---------------|
| Data type: | Boolean |
| Default: | True |
| Range: | True or False |

CSIV2 authentication protocol client settings:

In addition to the properties that are valid for both Security Authentication Service (SAS) and Common Secure Interoperability Version 2 (CSIV2), this page documents the properties that are valid for the CSIV2 protocol only.

com.ibm.CSI.performStateful:

Used to determine if the CSIV2 protocol maintains stateful sessions between a client and server after the initial secure association (authentication between a particular client and server).

For performance reasons, it is beneficial to enable this property. Considerations for disabling this property include troubleshooting an authentication protocol session-related problem.

| | |
|-------------------|---------------|
| Data type: | Boolean |
| Default: | True |
| Range: | True or False |

com.ibm.CSI.performClientAuthenticationSupported:

Use to determine if message layer client authentication is supported.

When supported, message layer client authentication is performed when communicating with any server that supports or requires the authentication. Message layer client authentication involves transmitting either a user ID and password or a token from an already authenticated credential. If the authenticationTarget property is BasicAuth, the user ID and password are transmitted to the target server. If the authenticationTarget password is a token-based mechanism such as Lightweight Third Party Authentication (LTPA) or Kerberos, then the credential token is transmitted to the server after authenticating the user ID and password directly to the security server.

| | |
|-------------------|---------------|
| Data type: | Boolean |
| Default: | True |
| Range: | True or False |

com.ibm.CSI.performClientAuthenticationRequired:

Use to determine if message layer client authentication is required.

When required, message layer client authentication must occur when communicating with any server. If transport layer client authentication is also enabled, both authentications are performed, but message layer client authentication takes precedence at the server.

Data type: Boolean
Default: True
Range: True or False

com.ibm.CSI.performTransportAssocSSLTLSSupported:

Use to determine if Secure Sockets Layer (SSL) is supported.

When SSL is supported, this client causes either SSL or TCP/IP to communicate with a server. If SSL is not supported, then the client must communicate over TCP/IP to the server. Supporting SSL is recommended so that any sensitive information is encrypted and digitally signed. When the associated `com.ibm.CSI.performTransportAssocSSLTLSRequired` property is enabled (set to true), this property is ignored. In this case, SSL is always required.

Data type: Boolean
Default: True
Range: True or False

com.ibm.CSI.performTransportAssocSSLTLSRequired:

Use to determine if SSL is required.

When SSL is required, this client must use SSL to communicate to a server. If SSL is not supported by a server, this client does not attempt a connection to that server. When this property is enabled, the associated `com.ibm.CSI.performTransportAssocSSLTLSSupported` property is ignored.

Data type: Boolean
Default: True
Range: True or False

com.ibm.CSI.performTLClientAuthenticationSupported:

Use to determine if transport layer client authentication is supported.

When performing client authentication using SSL, the client key file must have a personal certificate configured. Without a personal certificate, the client cannot authenticate to the server over SSL. If the personal certificate is a self-signed certificate, the server must contain the public key of the client in the server trust file. If the personal certificate is a Certificate Authority (CA) granted certificate, the server must contain the root public key of the CA in the server trust file. This property is only valid when SSL is supported or required. If the associated `com.ibm.CSI.performTLClientAuthenticationRequired` property is enabled, this property is ignored.

Data type: Boolean
Default: True
Range: True or False

com.ibm.CSI.performTLClientAuthenticationRequired:

Use to determine if transport layer client authentication is required.

If required, every secure socket opened between a client and server authenticates using SSL mutual authentication. When performing client authentication using SSL, the client key file must have a personal certificate configured. Without a personal certificate, the client cannot authenticate to the server over SSL.

If the personal certificate is a self-signed certificate, the server must contain the public key of the client in the server trust file. If the personal certificate is a certificate authentication (CA) granted certificate, the server must contain the root public key of the CA in the server trust file. When this property is specified, the associated `com.ibm.CSI.performTLClientAuthenticationSupported` property is ignored.

Data type: Boolean
Default: True
Range: True or False

com.ibm.CSI.performMessageConfidentialitySupported:

Use to determine if 128-bit ciphers are supported to make SSL connections.

If a target server does not support 128-bit ciphers, you can make a connection at a lower encryption strength. This property is only valid when SSL is enabled.

Data type: Boolean
Default: True
Range: True or False

com.ibm.CSI.performMessageConfidentialityRequired:

Use to determine if 128-bit ciphers must be used to make SSL connections.

If a target server does not support 128-bit ciphers, a connection to that server fails. This property is only valid when SSL is enabled. When this property is enabled, the associated `com.ibm.CSI.performMessageConfidentialitySupported` property is ignored.

Data type: Boolean
Default: True
Range: True or False

com.ibm.CSI.performMessageIntegritySupported:

Use to determine if 40-bit ciphers are supported to make SSL connections.

If a target server does not support 40-bit ciphers, you can make a connection using only digital signing ciphers. This property is only valid when SSL is enabled. This property is ignored if the associated `com.ibm.CSI.performMessageIntegrityRequired` property is enabled.

| | |
|-------------------|---------------|
| Data type: | Boolean |
| Default: | True |
| Range: | True or False |

com.ibm.CSI.performMessageIntegrityRequired:

Use to determine if 40-bit ciphers must be used to make SSL connections.

If a target server does not support 40-bit ciphers, a connection to that server fails. This property is only valid when SSL is enabled. When this property is enabled, the associated `com.ibm.CSI.performMessageIntegritySupported` property is ignored.

| | |
|-------------------|---------------|
| Data type: | Boolean |
| Default: | True |
| Range: | True or False |

Security Authentication Service Authentication Protocol client settings:

In addition to those properties which are valid for both Security Authentication Service (SAS) and Common Secure Interoperability Version 2 (CSIv2), this article documents properties which are valid only for the SAS authentication protocol.

com.ibm.CORBA.standardPerformQOPModels:

Specifies the strength of the ciphers when making an SSL connection.

| | |
|-------------------|-------------------|
| Data type: | String |
| Default: | High |
| Range: | Low, Medium, High |

Configuring Common Secure Interoperability Version 2 inbound authentication

Inbound authentication refers to the configuration that determines the type of accepted authentication for inbound requests. This authentication is advertised in the Interoperable Object Reference (IOR) that the client retrieves from the name server.

1. Start the administrative console. Click **Security > Authentication Protocol > CSI Inbound Authentication**.
2. Consider the following three layers of security:
 - Identity assertion (attribute layer).

When selected, this server accepts identity tokens from upstream servers. If the server receives an identity token, the identity is taken from an originating client. For example, the identity is in the same form that the originating client presented to the first server. An upstream server sends the identity of the originating client. The format of the identity can be either a principal name, a distinguished name, or a certificate chain. In some cases, the identity is anonymous. It is important to trust the upstream server that sends the

identity token because the identity is authenticating on this server. Trust of the upstream server is established either using Secure Sockets Layer (SSL) client certificate authentication or basic authentication. You must select one of the two layers of authentication in both inbound and outbound authentication when you choose identity assertion. The server ID is sent in the client authentication token with the identity token. The server ID is checked against the trusted server ID list. If the server ID is on the trusted server list, the server ID is authenticated. If the server ID is valid, then the identity token identity is put into a credential and used for authorization of the request.

- User ID and password (message layer).

This type of authentication is the most typical. The user ID and password or authenticated token is sent from a pure client or from an upstream server. However, the upstream server can not be a z/OS server because z/OS does not support a user ID or password from a server acting as a client. 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 user ID and password are received at the server, they are authenticated with the user registry. When a token is received at the server level, the token is validated to determine whether it has been tampered with or has expired.

- Secure Sockets Layer client certificate authentication (transport layer).

This type of authentication typically occurs from pure clients using the certificate identity and from servers trusting the upstream server. Usually, when a server delegates an identity to a downstream server, the identity comes from either the message layer (a client authentication token) or the attribute layer (an identity token), not from the transport layer, through the client certificate authentication.

A client has an SSL client certificate 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 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.

3. Consider the following points when deciding what type of authentication to accept:
 - A server can receive multiple layers simultaneously, so an order of precedence rule decides which identity to use. The identity assertion layer has the highest priority, the message layer follows, and the transport layer has the lowest priority. The SSL client certificate authentication is used when it is the only layer provided. If the message layer and the transport layer are provided, the message layer is used to establish the identity for authorization. The identity assertion layer is used to establish precedence when provided.
 - Does this server usually receive requests from a client, from a server or both? If the server always receives requests from a client, identity assertion is not needed. You can then choose either the message layer, the transport layer, or both. You also can decide when authentication is required or just supported. To select a layer as required, the sending client must supply this layer, or the request is rejected. However, if the layer is only supported, the layer might not be supplied.

- What kind of client identity is supplied? If the client identity is client certificates authentication and you want the certificate chain to flow downstream so that it maps to the downstream server user registries, then identity assertion is the appropriate choice. Identity assertion preserves the format of the originating client. If the originating client authenticated with a user ID and password, then a principal identity is sent. If authentication is done with a certificate, then the certificate chain is sent. In some cases, if the client authenticated with a token and a Lightweight Directory Access Protocol (LDAP) server is the user registry, then a distinguished name (DN) is sent.
4. 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 as 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.
 5. Configure session management. You can choose either *stateful* or *stateless* security. Performance is optimum when choosing stateful sessions. The first method request between a client and server is authenticated. All subsequent requests (or until the credential token expires) reuse the session information, including the credential. A client sends a context ID for subsequent requests. The context ID is scoped to the connection for uniqueness.

When you finish configuring this panel, you have configured most of the information that a client coalesces when determining what to send to this server. A client or server outbound configuration with this server inbound configuration, determines the security that is applied. When you know what clients send, the configuration is simple. However, if you have a diverse set of clients with differing security requirements, your server considers various layers of authentication.

For an enterprise bean server, the authentication choice is usually either identity assertion or message layer because you want the identity of the originating client delegated downstream. You cannot easily delegate a client certificate using an SSL connection. It is acceptable to enable the transport layer because additional server security, as the additional client certificate portion of the SSL handshake, adds some overhead to the overall SSL connection establishment.

After you determine which type of authentication data this server might receive, you can determine what to select for outbound security. Refer to the article, [Configuring Common Secure Interoperability Version 2 outbound authentication](#).

Common Secure Interoperability inbound authentication settings:

Use this page to specify the features that a server supports for a client accessing its resources.

To view this administrative console page, click **Security > Authentication Protocol > CSI Inbound Authentication**.

Use CSI inbound authentication settings for configuring the type of authentication information 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 expirable authenticated token.
- **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 presented during the request. The client picks up the Interoperable Object Reference (IOR) from the name space and reads the values from the tagged component to determine what the server needs for security.

Basic Authentication:

Specifies that basic authentication occurs over the message layer.

In the message layer, basic authentication (user ID and password) takes place. This type of authentication typically involves sending a user ID and a password from the client to the server for authentication. This authentication also involves delegating a credential token from an already authenticated credential, provided the credential type is forwardable (for example, Lightweight Third Party Authentication (LTPA)). If you select **Basic Authentication** for the server, specify both user ID and password authentication as well as token-based authentication.

When you select **Basic Authentication**, decide whether it is Required or Supported. Selecting Required, indicates that only clients configured to authenticate to this server through the message layer can invoke requests on the server. Selecting Supported, indicates that this server accepts basic authentication. However, other methods of authentication can occur if configured and anonymous requests are accepted. Select **Never** to indicate that the server is not configured to accept message layer authentication from any client.

Data type: String

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 takes place. 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 steps. These additional steps involve verifying that the server has the signer certificate of each client to which it is connected. If the client uses a certificate authority (CA) to create its personal certificate, then 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 specified when configuring LDAP. When the certificate is authenticated to a LocalOS user registry, the first attribute of the DN in the certificate (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.

When you select **Client Certificate Authentication**, decide whether it is Required or Supported. When You select **Required**, only clients that are configured to authenticate to this server through SSL client certificates can invoke requests on the server. When you select **Supported**, this server accepts SSL client certificate authentication, however, other methods of authentication can occur (if configured) and anonymous requests are accepted. When you select **Never**, this server is not configured to accept client certificate authentication from any client.

Data type String

Identity Assertion:

Specifies that identity assertion is a way to assert identities from one server to another during a downstream Enterprise JavaBean (EJB) invocation.

Identity assertion is performed in the attribute layer and is only applicable on servers. The principal determined at the server is based on precedence rules. If identity assertion is performed, the identity is always derived from the attribute. If basic authentication is performed without identity assertion, the identity is always derived from the message layer. Finally, if SSL client certificate authentication is performed without either basic authentication, or identity assertion, then the identity is derived from the transport layer.

The identity asserted is the invocation credential that is determined by the RunAs mode for the enterprise bean. If the RunAs mode is Client, the identity is the client identity. If the RunAs mode is System, the identity is the server identity. If the RunAs mode is Specified, the identity is the one specified. The receiving server receives the identity in an identity token and also receives the sending server identity in a client authentication token. The receiving server validates the sending server identity as a trusted identity through the Trusted Server IDs entry box. . Enter a list of pipe-separated (|) principal names, for example, serverid1|serverid2|serverid3.

When authenticating to a LocalOS user registry, all identity token types map to the user ID field of the active user registry. For an ITTPrincipal identity token, this token maps one-to-one with the user ID fields. For an ITTDistinguishedName identity token, the value from the first equal sign is mapped to the user ID field. For an ITTCertChain identity token, the value from the first equal sign of the distinguished name is mapped to the user ID field.

When authenticating to an LDAP user registry, the LDAP filters determine how an identity of type ITTCertChain and ITTDistinguishedName get mapped to the registry. If the token type is ITTPrincipal, then the principal gets mapped to the UID field in the LDAP registry.

Data type: String

Trusted Server User IDs:

Specifies a pipe-separated (|) list of server user IDs, which are trusted to perform identity assertion to this server. For example, serverid1|serverid2|serverid3.

Use this list to quickly decide whether a server is trusted. Even if the server is on the list, the sending server must still authenticate with the receiving server to accept the identity token of the sending server.

Data type String

Stateful Sessions:

Specifies stateful sessions that are used mostly for performance improvements.

The first contact between a client and server must fully authenticate. However, all subsequent contacts with valid sessions reuse the security information. The client passes a context ID to the server, and the ID is used to look up the session. The context ID is scoped to the connection, which guarantees uniqueness. Whenever the security session is invalid and the authentication retry is enabled (it is by default), the client-side security interceptor invalidates the client-side session and resubmits the request 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 re-authenticate.

Data type String

Configuring Common Secure Interoperability Version 2 outbound authentication

Outbound authentication refers to the configuration that determines the type of authentication performed for outbound requests to downstream servers. Several *layers* or *methods* of authentication can occur. The downstream server inbound authentication configuration must support at least one choice made in this server outbound authentication configuration. If nothing is supported, the request might go outbound as unauthenticated. This situation does not create a security problem because the authorization run time is responsible for preventing access to protected resources. However, if you choose to prevent an unauthenticated credential to go outbound, you might want to designate one of the authentication layers as required, rather than supported. If a downstream server does not support authentication, then when authentication is required, the method request fails to go outbound.

The following choices are available in the Common Secure Interoperability Version 2 (CSIv2) Outbound Authentication panel. Remember that you are not required to complete these steps in the displayed order. Rather, these steps are provided to help you understand your choices for configuring outbound authentication.

1. Select **Identity Assertion** (attribute layer). When selected, this server submits an identity token to a downstream server, if the downstream server supports identity assertion. When an originating client authenticates to this server, the authentication information supplied is preserved in the outbound identity token. If the client authenticating to this server uses client certificate authentication, then the identity token format is a certificate chain, containing the exact client certificate chain on the socket. The same scenario is true for other mechanisms of authentication. Read the Identity Assertion article for more information.
2. Select **User ID** and **Password** (message layer). This type of authentication is the most typical. The user ID and password (if BasicAuth credential) or authenticated token (if authenticated credential) are sent outbound to the downstream server if the downstream server supports message layer

authentication in the inbound authentication panel. Refer to the Message Layer Authentication article for more information.

3. Select **SSL Client certificate authentication** (transport layer). The main reason to enable outbound Secure Sockets Layer (SSL) client authentication from one server to a downstream server is to create a trusted environment between those servers. For delegating client credentials, use one of the two layers mentioned previously. However, you might want to create SSL personal certificates for all the servers in your domain, and only trust those servers in your SSL truststore file. No other servers or clients can connect to the servers in your domain, except at the tiers where you want them. This process can protect your enterprise bean servers from access by anything other than your servlet servers. Refer to the SSL Client Certificate Authentication article for more information.

A server can send multiple layers simultaneously, therefore, an order of precedence rule decides which identity to use. The identity assertion layer has the highest priority, the message layer follows, and the transport layer has the lowest priority. SSL client certificates are only used as the identity for invoking method requests, when that is the only layer provided. SSL client certificates are useful for trust purposes, even if the identity is not used for the request. If only the message layer and transport layer are provided, the message layer is used to establish the identity for authorization. If the identity assertion layer is provided (regardless of what is provided), then the identity from the identity token is always used by the authorization engine as the identity for that request.

Configuring session management:

You can choose either *stateful* or *stateless* security. Performance is optimum when choosing stateful sessions. The first method request between this server and the downstream server is authenticated. All subsequent requests reuse the session information, including the credential. A *unique session entry* is defined as the combination of a unique client authentication token and an identity token, scoped to the connection.

When you finish configuring this panel, you configured the information that this server uses to make decisions about the type of authentication to perform with downstream servers. If the downstream server is configured not to support the outbound configuration of the server, the following exception likely occurs:

```
Exception received: org.omg.CORBA.INITIALIZE:
JSAS1477W: 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: JSAS0607E: The client requires SSL Confidentiality but the server
does not support it.
ERROR 2: JSAS0610E: The server requires SSL Integrity but the client does
not support it.
ERROR 3: JSAS0612E: The client requires client (e.g., userid/password or token),
but the server does not support it.
minor code: 0  completed: No
    at com.ibm.ISecurityLocalObjectBaseL13Impl.SecurityConnectionInterceptor.
getConnectionKey(SecurityConnectionInterceptor.java:1770)
    at com.ibm.ws.orbimpl.transport.WSTransport.getConnection(Unknown Source)
    at com.ibm.rmi.iiop.TransportManager.get(TransportManager.java:79)
    at com.ibm.rmi.iiop.GIOPImpl.locate(GIOPImpl.java:167)
    at com.ibm.CORBA.iiop.ClientDelegate._createRequest(ClientDelegate.java:2088)
    at com.ibm.CORBA.iiop.ClientDelegate.createRequest(ClientDelegate.java:1264)
```

```

at com.ibm.CORBA.iiop.ClientDelegate.createRequest(ClientDelegate.java:1177)
at com.ibm.CORBA.iiop.ClientDelegate.request(ClientDelegate.java:1726)
at org.omg.CORBA.portable.ObjectImpl._request(ObjectImpl.java:245)
at com.ibm.WsnOptimizedNaming._NamingContextStub.get_compatibility_level
(Unknown Source)
at com.ibm.websphere.naming.DumpNameSpace.getIdlLevel(DumpNameSpace.java:300)
at com.ibm.websphere.naming.DumpNameSpace.getStartingContext
(DumpNameSpace.java:329)
at com.ibm.websphere.naming.DumpNameSpace.main(DumpNameSpace.java:268)
at java.lang.reflect.Method.invoke(Native Method)
at com.ibm.ws.bootstrap.WSLauncher.main(WSLauncher.java:163)

```

The reasons for the mismatch are explained in the exception. You can make the corrections when you configure the outbound configuration for this server, or when you configure the inbound configuration of the downstream server. If multiple reasons exist for a failure, the reasons are explained as message text in the exception.

Typically, the outbound authentication configuration is for an upstream server to communicate with a downstream server. Most likely, the upstream server is a servlet server and the downstream server is an EJB server. On a servlet server, the client authentication 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 form a credential of the mechanism type that is supported by the server, such as Lightweight Third Party Authentication (LTPA) or LocalOS. When LTPA is the mechanism, a forwardable token exists in the credential. Choose the message layer (BasicAuth) authentication to propagate the client credentials. If the credential was 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 outbound authentication settings:

Use this page to specify the features that a server supports when acting as a client to another downstream server.

To view this administrative console page, click **Security > Authentication Protocol > CSI Outbound Authentication**.

Authentication features include three layers of authentication that you can use simultaneously:

Transport layer

The transport layer, the lowest layer, might contain a Secure Sockets Layer (SSL) client certificate as the identity.

Message layer

The message layer might contain a user ID and password or authenticated token.

Attribute layer

The attribute layer might contain an identity token, which is an identity from an upstream server that is already authenticated. The attribute layer has the highest priority, followed by the message layer and then the transport layer. If this server sends all three, only the attribute layer is used

by the downstream server. The only way to use the SSL client certificate as the identity is if it is the only information presented during the outbound request.

Basic Authentication:

Specifies whether to send a user ID and a password from the client to the server for authentication.

This type of authentication occurs over the message layer. Basic authentication also involves delegating a credential token from an already authenticated credential, provided the credential type is forwardable (for example, Lightweight Third Party Authentication (LTPA)). Basic authentication refers to any authentication over the message layer and indicates user ID and password as well as token-based authentication.

Select **Basic Authentication** and determine whether this authentication method is required or supported. Select **Required** to indicate that when the server goes outbound to downstream servers, the downstream server must support basic authentication for this server to connect. Select **Supported** to indicate that this server might or might not perform basic authentication to a downstream server. Other methods of authentication can occur if configured. Select **Never** to indicate that this server never sends a message layer token outbound to a downstream server. If the downstream server requires basic authentication, then the connection is not attempted.

Data type: String

Client Certificate Authentication:

Specifies whether a client certificate from the configured keystore file is used to authenticate to the server when the SSL connection is made between this server and a downstream server (provided that the downstream server supports client certificate authentication).

Typically, client certificate authentication has a higher performance than message layer authentication, but requires some additional setup steps. 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, decide whether it is required or supported. Select **Required** to indicate that this server can only connect to downstream servers with client certificate authentication also configured. Select **Supported** to indicate that this server performs client certificate authentication with any downstream server, but might not use client certificate authentication depending on whether it is supported by the downstream server. Select **Never** to indicate that this client does not perform client certificate authentication to any downstream server. This limitation prevents access to any downstream server that requires client certificate authentication.

Data type: String

Identity Assertion:

Specifies whether to assert identities from one server to another during a downstream enterprise bean invocation.

The identity asserted is the invocation credential that is determined by the RunAs mode for the enterprise bean. If the RunAs mode is Client, the identity is the client identity. If the RunAs mode is System, the identity is the server identity. If the RunAs mode is Specified, the identity is the identity specified. The receiving server receives the identity in an identity token and also receives the sending server identity in a client authentication token. The receiving server validates the identity of the sending server to ensure a trusted identity.

When specifying identity assertion on the CSiv2 Authentication Outbound panel, you must also select basic authentication as supported or required on the CSiv2 Authentication Outbound panel. The server identity can then be submitted with the identity token, so that the receiving server can *trust* the sending server. Without specifying basic authentication as supported or required, trust is not established and the identity assertion fails.

Data type: String

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. Whenever the security session is invalid and if authentication retry is enabled (it is enabled by 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 re-authenticate.

Data type: String

Configuring inbound transports

Inbound transports refer to the types of listener ports and their attributes that are opened to receive requests for this server. Both Common Secure Interoperability Specification, Version 2 (CSiv2) and Secure Authentication Service (SAS) have the ability to configure the transport. However, the following differences between the two protocols exist:

- CSiv2 is much more flexible than SAS, which requires Secure Sockets Layer (SSL); CSiv2 does not require SSL.
- SAS does not support SSL client certificate authentication, while CSiv2 does.
- CSiv2 can require SSL connections, while SAS only supports SSL connections.
- SAS always has two listener ports open: TCP/IP and SSL.
- CSiv2 can have as few as one listener port and as many as three listener ports. You can open one port for just TCP/IP or when SSL is required. You can open two ports when SSL is supported, and open three ports when SSL and SSL client certificate authentication is supported.

Complete the following steps to configure the Inbound Transport panels in the administrative console:

1. Click **Security > Authentication Protocol > CSIV2 Inbound Transport** to select the type of transport and the SSL settings. By selecting the type of transport, as noted previously, you choose which listener ports you want to open. In addition, you disable the SSL client certificate authentication feature if you choose TCP/IP as the transport.
2. Select the SSL settings that correspond to an SSL transport. These SSL settings are defined in the **Security > SSL** panel and define the SSL configuration including the keyring, security level, ciphers, and so on.
3. Consider fixing the listener ports that you configured.

You complete this action in a different panel, but this is the time to think about it. Most end points are managed at a single location, which is why they do not appear 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 end points. The location for SSL end points is at each server. The following port names are defined in the End Points panel and are used for object request broker (ORB) security:

- CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS - CSIV2 Client Authentication SSL Port
- CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS - CSIV2 SSL Port
- SAS_SSL_SERVERAUTH_LISTENER_ADDRESS - SAS SSL Port
- ORB_LISTENER_PORT - TCP/IP Port

For an application server, click **Servers > Application Servers > server_name**. Under Additional Properties, click **End Points**. The End Points panel is displayed for the specified server.

The Object Request Broker (ORB) on WebSphere Application Server uses a listener port for Remote Method Invocation over the Internet Inter-ORB Protocol (RMI/IIOP) communications, which is generally not specified and selected dynamically during run time. If you are working with a firewall, you must specify a static port for the ORB listener and open that port on the firewall so that communication can pass through the specified port. The endPoint property for setting the ORB listener port is: ORB_LISTENER_ADDRESS.

Complete the following steps using the administrative console to specify the ORB_LISTENER_ADDRESS port or ports.

- a. Click **Servers > Application Servers > server_name**.
 - b. Click **End Points > New** under Additional Properties.
 - c. Select **ORB_LISTENER_ADDRESS** from the End Point Name field in the Configuration panel.
 - d. Enter the IP address, the fully qualified 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.
 - e. 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 in conjunction with the host name. Using the previous example, the port number might be 9000.
4. Click **Security > Authentication Protocol > SAS Inbound** to select the SSL settings used for inbound requests from SAS clients. Remember that the SAS protocol is used to interoperate with previous releases. When configuring the key store and trust store files in the SSL configuration, these files need the right information for interoperating with previous releases of WebSphere Application

Server. For example, a previous release has a different trust store file than the Version 5 release. If you use the Version 5 key store file, add the signer to the trust store 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 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 transport inbound settings:

Use this page to specify which listener ports to open and which Secure Sockets Layer (SSL) settings to use. These specifications determine which transport a client or upstream server uses to communicate with this server for incoming requests.

To view this administrative console page, click **Security > Authentication Protocol > CSI 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.

By default, SSL ports for Common Secure Interoperability Version 2 (CSIV2) and Security Authentication Service (SAS) are dynamically generated. In cases where you need to fix the SSL ports on application servers, click **Servers > Application Servers > server_name > End Points**. Configure the following ports to be fixed. A zero port number indicates that a dynamic assignment is made at run time.

CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS
CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS
SAS_SSL_SERVERAUTH_LISTENER_ADDRESS

- **TCP/IP:** Only a TCP/IP listener port is opened and all requests inbound do not have SSL protection.

- **SSL-Supported:** Both a TCP/IP and SSL listener port are opened and most requests come inbound by SSL.
- **SSL-Required:** 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.

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. These settings are configured at the SSL Repertoire panel.

Data type: String
Default: DefaultSSLSettings
Range: Any SSL settings configured in the SSL Configuration Repertoire

Secure Authentication Service transport inbound settings:

Use this page to specify transport settings for connections that are accepted by this server using the Secure Authentication Service (SAS) authentication protocol. The SAS protocol is used to communicate securely to enterprise beans with previous releases of the WebSphere Application Server.

To view this administrative console page, click **Security > Authentication Protocol > SAS Inbound Transport**.

SSL Settings:

Specifies a list of predefined SSL settings to choose from for inbound connections. These settings are configured at the SSL Repertoire panel.

Data type: String
Default: DefaultSSLSettings

Configuring outbound transports

Outbound transports refers to the transport used to connect to a downstream server. When you configure the outbound transport, consider the transports that the downstream servers support. If you are considering Secure Sockets Layer (SSL), also consider including the signers of the downstream servers in this server truststore file for the handshake to succeed. When you select an SSL configuration, that configuration points to keystore and truststore files that contain the necessary signers. If you configured client certificate authentication for this server in the **Security > Authentication Protocols > CSiv2 Outbound Authentication** panel, then the downstream servers contain the signer certificate belonging to the server personal certificate.

Complete the following steps to configure the Outbound Transport panels.

1. Select the type of transport and the SSL settings by clicking **Security > Authentication Protocol > CSiv2 Outbound Transport** panel. By selecting the type of transport, you are choosing the transport to use when connecting to

downstream servers. The downstream servers support the transport that you choose. If you choose **SSL-Supported**, the transport used is negotiated during the connection. If both the client and server support SSL, always choose **SSL-Supported** unless the request is considered a special request that does not require SSL, such as if an object request broker (ORB) is a request.

2. Pick the SSL settings that correspond to an SSL transport. Click **Security > SSL**. This panel includes the SSL configuration of keystore files, truststore files, file formats, security levels, ciphers, cryptographic token selections, and so on. Verify that the truststore file in the selected SSL configuration contains the signers for any downstream servers. Also, verify that the downstream servers contain the server signer certificates when outbound client certificate authentication is used.
3. Select the SSL settings used for outbound requests to downstream Secure Authentication Service (SAS) servers. Click **Security > Authentication Protocol > SAS Outbound**. Remember that the SAS protocol allows interoperability with previous releases. When configuring the keystore and truststore files in the SSL configuration, these files have the correct information for interoperating with previous releases of WebSphere Application Server. For example, a previous release has a different personal certificate than the Version 5 release. If you use the keystore file from the Version 5.0 release, you must add the signer to the truststore file of the previous release. Also, you must extract the signer for the Version 5.0 release and import that signer into the truststore file of the previous release.

The outbound transport configuration is complete.

With this configuration you can configure a different transport for inbound security versus outbound security. For example, if the application server is the first server used by end users, the security configuration might be more secure. When requests go to back-end enterprise beans servers, you might consider less security for performance reasons when you go outbound. With this flexibility you can design a transport infrastructure that meets your needs.

When you finish configuring security, perform the following steps to save, synchronize, and restart the servers.

- Click **Save** in the administrative console to save any modifications to the configuration.
- Stop and restart all servers, after synchronization.

Common secure interoperability transport outbound settings:

Use this page to specify which transports and Secure Sockets Layer (SSL) settings this server uses when communicating with downstream servers for outbound requests.

To view this administrative console page, click **Security > Authentication Protocol > CSI 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 which a server supports. If you specify **TCP/IP**, the server only supports 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, then 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 key file and trust file to use for outbound connections to downstream servers. For example, consider the following options:

TCP/IP

This server opens TCP/IP connections with downstream servers only.

SSL Supported

This server opens SSL connections with any downstream servers that support them, and TCP/IP connections with any downstream servers that do not support these SSL connections.

SSL Required

This server always opens SSL connections with downstream servers.

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.

Data type: String
Default: DefaultSSLSettings
Range: Any SSL settings configured in the SSL Configuration Repertoires panel

Secure Authentication Service transport outbound settings:

Use this page to specify transport settings for connections that are accepted by this server using the Secure Authentication Service (SAS) authentication protocol. Use the SAS protocol to communicate securely to enterprise beans with previous releases of WebSphere Application Server.

To view this administrative console page, click **Security > Authentication Protocol > SAS Outbound Transport**.

SSL Settings:

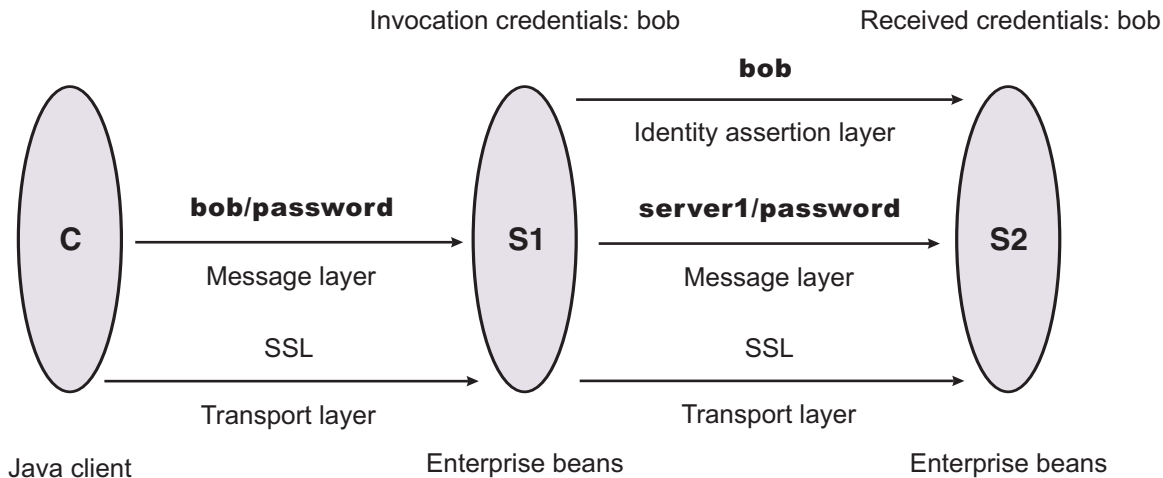
Specifies a list of predefined Secure Sockets Layer (SSL) settings to choose from for outbound connections. These settings are configured at the SSL Repertoire panel.

Data type: String
Default: DefaultSSLSettings

Example: Common Secure Interoperability Version 2 scenarios

The articles included in this section are intended to demonstrate how to configure specific Common Secure Interoperability Version 2 (CSIV2) configuration examples.

Scenario 1: Basic authentication and identity assertion:



This example presents a pure Java client, C, that accesses a secure enterprise bean on server, S1, through user "bob." The enterprise bean code on S1 accesses another enterprise bean on server, S2. This configuration uses identity assertion to propagate the identity of "bob" to the downstream server, S2. S2 trusts that "bob" already is authenticated by S1 because it trusts S1. To gain this trust, the identity of S1 also flows to S2 simultaneously and S2 validates the identity by checking the trustedPrincipalList to verify that it is a valid server principal. S2 also authenticates S1. The following steps take you through the configuration of C, S1, and S2.

Configuring client, C

Client C requires message layer authentication with a Secure Sockets Layer (SSL) transport. To accomplish this task:

1. Point the client to the `sas.client.props` file using the `com.ibm.CORBA.ConfigURL=file:/C:/was/properties/sas.client.props` property.
All further configuration involves setting properties within this file.
2. Enable SSL.
In this case, SSL is supported but not required:
`com.ibm.CSI.performTransportAssocSSLTLSSupported=true,`
`com.ibm.CSI.performTransportAssocSSLTLSRequired=false`
3. Enable client authentication at the message layer.
In this case, client authentication is supported but not required:
`com.ibm.CSI.performClientAuthenticationRequired=false,`
`com.ibm.CSI.performClientAuthenticationSupported=true`
4. Use all of the remaining defaults in the `sas.client.props` file.

Configuring server, S1

In the administrative console, server S1 is configured for incoming requests to support message layer client authentication and incoming connections to support SSL without client certificate authentication. Server S1 is configured for outgoing requests to support identity assertion.

1. Configure S1 for incoming connections.
 - a. Disable identity assertion.
 - b. Enable user ID and password authentication.
 - c. Enable SSL.

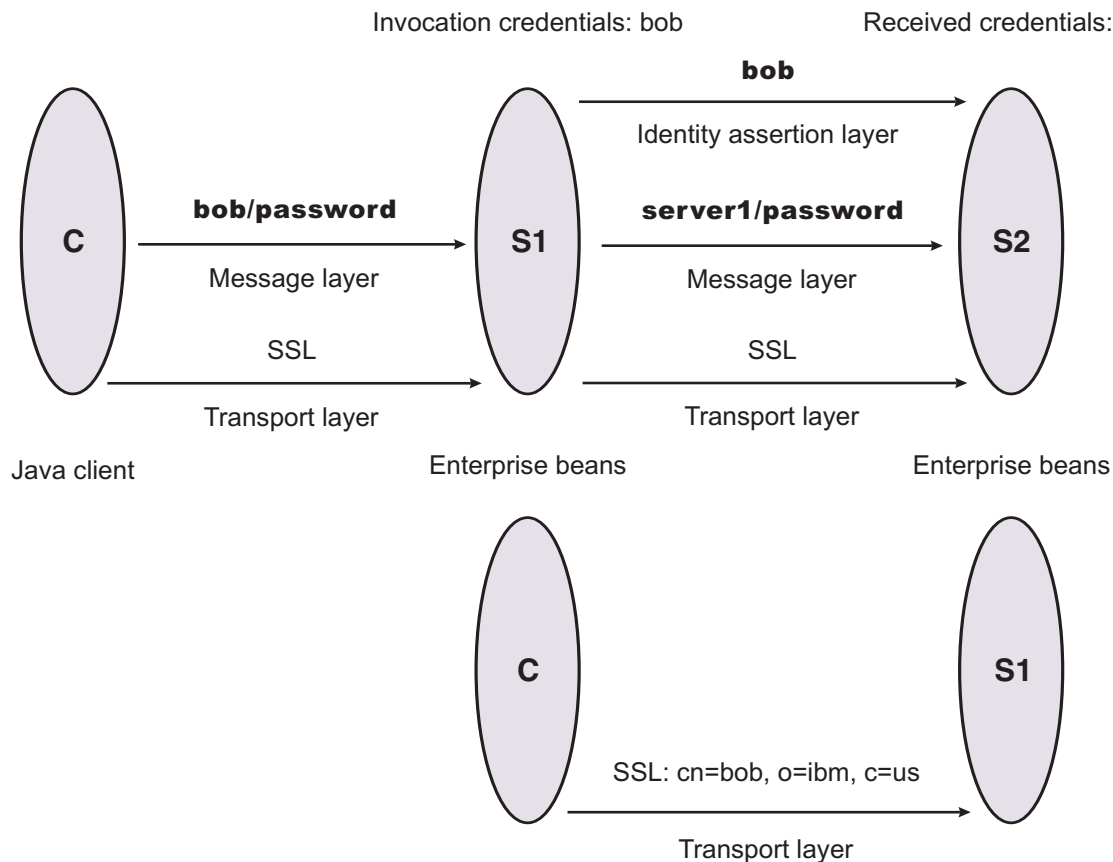
- d. Disable SSL client certificate authentication.
- 2. Configure S1 for outgoing connections.
 - a. Enable identity assertion.
 - b. Disable user ID and password authentication.
 - c. Enable SSL.
 - d. Disable SSL client certificate authentication.

Configuring server, S2

In the administrative console, server S2 is configured for incoming requests to support identity assertion and to accept SSL connections. Complete the following steps to configure incoming connections. Configuration for outgoing requests and connections are not relevant for this scenario.

1. Enable identity assertion.
2. Disable user ID and password authentication.
3. Enable SSL.
4. Disable SSL client authentication.

Scenario 2: Basic authentication, identity assertion and client certificates:



This scenario is the same as Scenario 1, except for the interaction from client C2 to server S2. Therefore, the configuration of Scenario 1 still is valid, but you have to modify server S2 slightly and add a configuration for client C2. The configuration is not modified for C1 or S1.

Configuring client C2

Client C2 requires transport layer authentication (Secure Sockets Layer (SSL) client certificates). To configure transport layer authentication:

1. Point the client to the `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

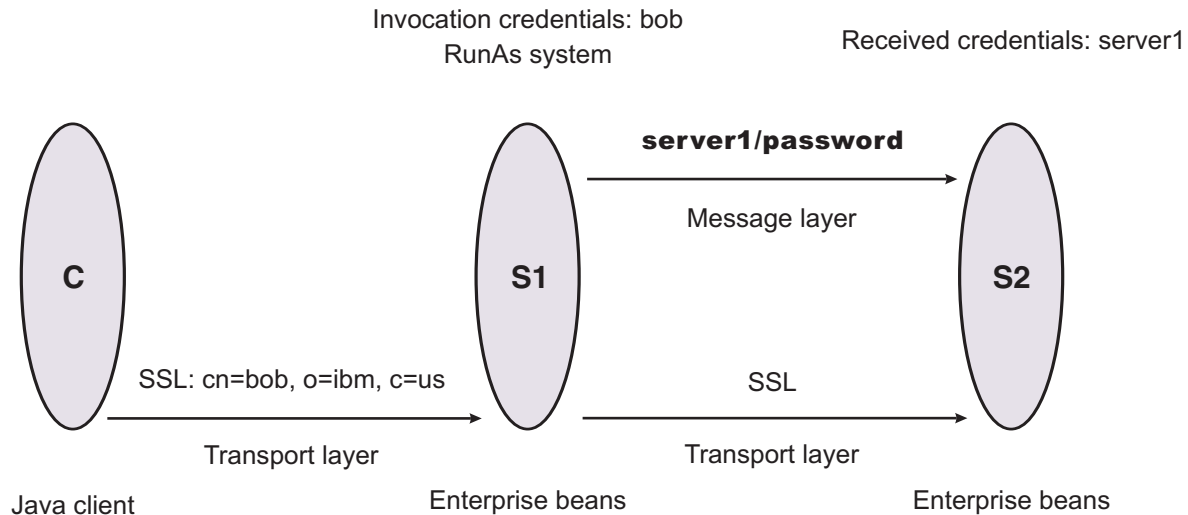
In the administrative console, server S2 is configured for incoming requests to SSL client authentication and identity assertion. Configuration for outgoing requests is not relevant for this scenario.

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, there is a precedence to which authentication features become the identity in the received credential:

1. Identity assertion
2. Message layer client authentication (basic authentication or token)
3. Transport layer client authentication (SSL certificates)

Scenario 3: Client certificate authentication and RunAs system:



This example presents a pure Java client, C, accessing a secure enterprise bean on S1. C authenticates to S1 using Secure Sockets Layer (SSL) client certificates. S1 maps the common name of the distinguished name (DN) in the certificate to a user in the local registry. The user in this case is bob. The enterprise bean code on S1 accesses another enterprise bean on S2. Because the RunAs mode is system, the invocation credential is set as server1 for any outbound requests.

Configuring C

C requires transport layer authentication (SSL client certificates):

1. Point the client to the `sas.client.props` file using the `com.ibm.CORBA.ConfigURL=file:/C:/was/properties/sas.client.props` property.
All further configuration involves setting properties within this file.
2. Enable SSL.
In this case, SSL is supported but not required:
`com.ibm.CSI.performTransportAssocSSLTLSSupported=true,`
`com.ibm.CSI.performTransportAssocSSLTLSRequired=false`
3. Disable client authentication at the message layer:
`com.ibm.CSI.performClientAuthenticationRequired=false,`
`com.ibm.CSI.performClientAuthenticationSupported=false`
4. Enable client authentication at the transport layer. It is supported, but not required: `com.ibm.CSI.performTLClientAuthenticationRequired=false,`
`com.ibm.CSI.performTLClientAuthenticationSupported=true`

Configuring S1

In the administrative console, S1 is configured for incoming connections to support SSL with client certificate authentication. The S1 server is configured for outgoing requests to support message layer client authentication.

1. Configure S1 for incoming connections:
 - a. Disable identity assertion.
 - b. Disable user ID and password authentication.
 - c. Enable SSL.
 - d. Enable SSL client certificate authentication.

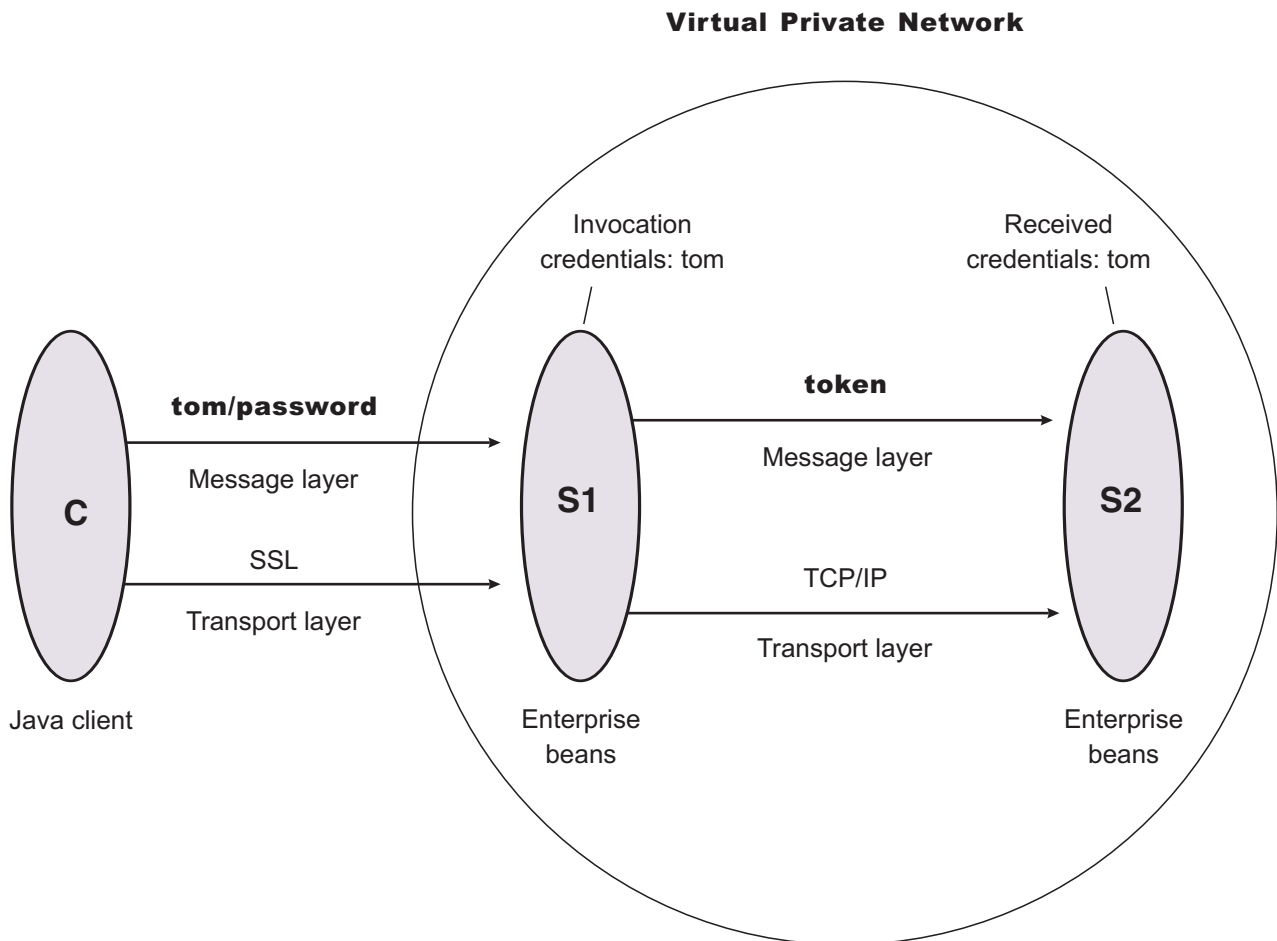
2. Configure S1 for outgoing connections:
 - a. Disable identity assertion.
 - b. Disable user ID and password authentication.
 - c. Enable SSL.
 - d. Enable SSL client certificate authentication.

Configuring S2

In the administrative console, the S2 server is configured for incoming requests to support message layer authentication over SSL. Configuration for outgoing requests is not relevant for this scenario.

1. Disable identity assertion.
2. Enable user ID and password authentication.
3. Enable SSL.
4. Disable SSL client authentication.

Scenario 4: TCP/IP transport using a Virtual Private Network:



This scenario illustrates the ability to choose TCP/IP as the transport when it is appropriate. In some cases, when two servers are on the same Virtual Private Network (VPN), it can be appropriate to select TCP/IP as the transport for performance reasons because the VPN already encrypts the message.

Configuring C

C requires message layer authentication with an SSL transport:

1. Point the client to the `sas.client.props` file using the `com.ibm.CORBA.ConfigURL=file:/C:/was/properties/sas.client.props` property. All further configuration involves setting properties within this file.
2. Enable SSL. In this case, SSL is supported but not required:
`com.ibm.CSI.performTransportAssocSSLTLSSupported=true,`
`com.ibm.CSI.performTransportAssocSSLTLSRequired=false`
3. Enable client authentication at the message layer. In this case, client authentication is supported but not required:
`com.ibm.CSI.performClientAuthenticationRequired=false,`
`com.ibm.CSI.performClientAuthenticationSupported=true`
4. Use the remaining defaults in the `sas.client.props` file.

Configuring the S1 server

In the administrative console, the S1 server is configured for incoming requests to support message layer client authentication and incoming connections to support SSL without client certificate authentication. The S1 server is configured for outgoing requests to support identity assertion.

1. Configure S1 for incoming connections:
 - a. Disable identity assertion.
 - b. Enable user ID and password authentication.
 - c. Enable SSL.
 - d. Disable SSL client certificate authentication.
2. Configure S1 for outgoing connections:
 - a. Disable identity assertion.
 - b. Enable user ID and password authentication.
 - c. Disable SSL.

It is possible to enable SSL for inbound connections and disable SSL for outbound connections. The same is true in reverse.

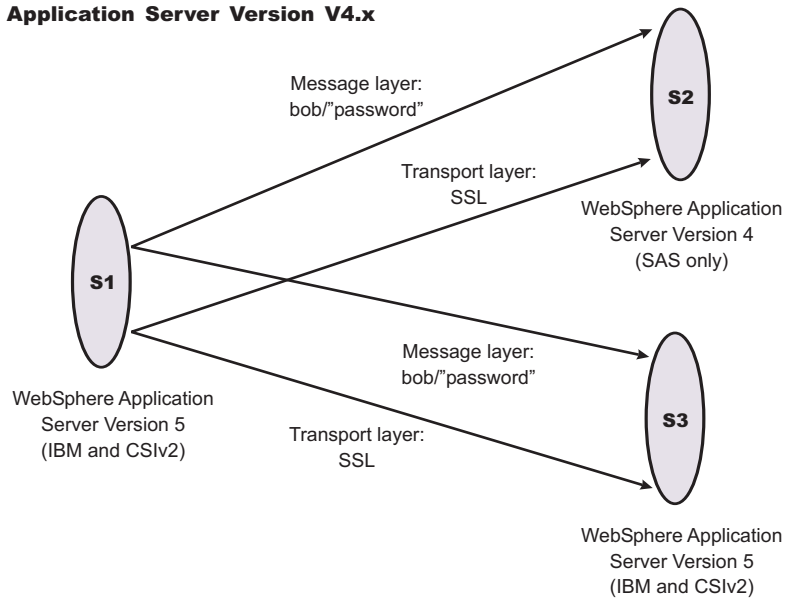
Configuring the S2 server

In the administrative console, the S2 server is configured for incoming requests to support identity assertion and to accept SSL connections. Configuration for outgoing requests and connections are not relevant for this scenario.

1. Disable identity assertion.
2. Enable user ID and password authentication.
3. Disable SSL.

Scenario 5: Interoperability with WebSphere Application Server Version:

Interoperability with WebSphere Application Server Version V4.x



The purpose of this scenario is to show how secure interoperability can occur between different releases simultaneously while using multiple authentication protocols (Security Authentication Service (SAS) and Common Secure Interoperability Version 2 (CSIv2)). For WebSphere Application Server Version 5 to communicate with a WebSphere Application Server Version 4, Version 5 server must support either IBM or BOTH as the protocol choice. By choosing BOTH, the Version 5 server also can communicate with other Version 5 servers that support CSI. If the only servers in your security domain are version 5, it is recommended that you choose CSI as the protocol because this prevents the IBM interceptors from loading. However, a chance exists that any server has to communicate with a previous release of WebSphere Application Server, select the protocol choice of BOTH.

Configuring the S1 server

The S1 server requires message layer authentication with an SSL transport. The protocol for the S1 server must be BOTH. Configuration for incoming requests for the S1 server is not relevant for this scenario. To configure the S1 server for outgoing connections:

1. Disable identity assertion.
2. Enable user ID and password authentication.
3. Enable Secure Sockets Layer (SSL).
4. Disable SSL client certificate authentication.
5. Set authentication protocol to BOTH in the global security settings.

Configuring the S2 server

All previous releases of WebSphere Application Server support the SAS authentication protocol only. No special configuration steps are needed other than enabling global security on the server (S2).

Configuring the S3 server

In the administrative console, the S3 server is configured for incoming requests to message layer authentication and to accept SSL connections. Configuration for outgoing requests and connections are not relevant for this scenario.

1. Enable identity assertion.
2. Disable user ID and password authentication.
3. Enable SSL.
4. Disable SSL client authentication.
5. Set authentication protocol to either CSI or BOTH.

Secure Sockets Layer

The Secure Sockets Layer (SSL) protocol provides transport layer security: authenticity, integrity, and confidentiality, for a secure connection between a client and server in the WebSphere Application Server. The protocol runs above TCP/IP and below application protocols such as Hypertext Transfer Protocol (HTTP), Lightweight Directory Access Protocol (LDAP), and Internet Inter-ORB Protocol (IIOP), and provides trust and privacy for the transport data.

Depending upon the SSL configurations of both the client and server, various levels of trust, data integrity, and privacy can be established. Understanding the basic operation of SSL is very important to proper configuration and to achieve the desired protection level for both client and application data.

Some of the security features provided by SSL are data encryption to prevent the exposure of sensitive information while data flows across the wire. Data signing prevents unauthorized modification of data while data flows across the wire. Client and server authentication ensures that you talk to the appropriate person or machine. SSL can be effective in securing an enterprise environment.

SSL is used by multiple components within WebSphere Application Server to provide trust and privacy. These components are the built-in HTTP transport, the Object Request Broker (ORB), and the secure LDAP client.

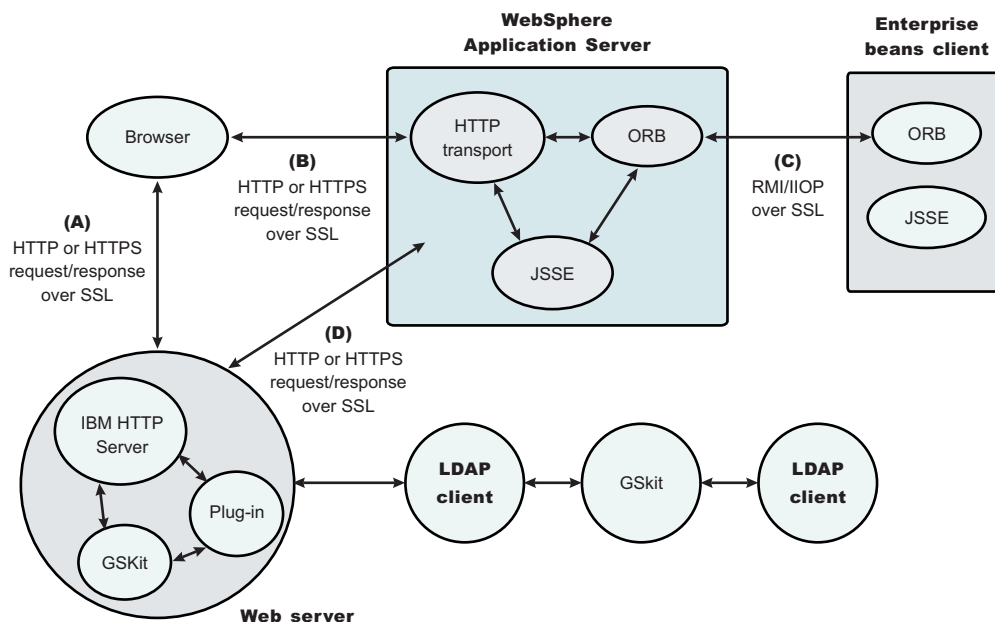


Figure 2. SSL and WebSphere Application Server

In this figure:

- The built-in HTTP transport in a WebSphere Application Server accepts HTTP requests over SSL from a Web client like a browser.
- The Object Request Broker used in WebSphere Application Server can perform Internet Inter-ORB Protocol (IIOP) over SSL to secure the message.
- The secure LDAP client uses LDAP over SSL to securely connect to an LDAP user registry and is present only when LDAP is configured as the user registry.

WebSphere Application Server and the IBMJSSE provider

The SSL implementation used by the WebSphere Application Server is IBM Java Secure Sockets Extension (IBMJSSE). The IBMJSSE provider contains a reference implementation supporting SSL and Transport Layer Security (TLS) protocols and an application programming interface (API) framework. The IBMJSSE provider also comes with a standard provider, which supplies Rivest Shamir Adleman (RSA) support for the signature-related JCA features of the Java 2 platform, common SSL and TLS cipher suites, hardware cryptographic token device, X.509-based key and trust managers, and PKCS12 implementation for a JCA *keystore*. A graphical tool called Key Management Tool (iKeyman) also is provided to manage digital certificates. With this tool, you can create a new key database or a test digital certificate, add certificate authority (CA) roots to the database, copy certificates from one database to another, as well as request and receive a digital certificate from a CA.

Configuring the JSSE provider is very similar to configuring most other SSL implementations (for example, GSKit); however, a couple of differences are worth noting.

- The JSSE provider support both signer and personal certificate storage in an SSL key file, but it also supports a separate file called a *trust file*. A trust file can contain only signer certificates. You can put all of your personal certificates in an SSL key file and your signer certificates in a trust file. This might be desirable, for example, if you have an inexpensive hardware cryptographic device with

only enough memory to hold a personal certificate. In this case, the key file refers to the hardware device and the trust file to a file on disk containing all of the signer certificates.

- The JSSE provider does not recognize the proprietary SSL key file format, which is used by the plug-in (.kdb files). Instead, the JSSE provider recognizes standard file formats such as Java Key Store (JKS). SSL key files might not be shared between the plug-in and application server. Furthermore, a different implementation of the key management utility must be used to manage application server key and trust files.

Certain limitations exist with the Java Secure Socket Extension (JSSE) provider:

- Customer code using JSSE and Java Cryptography Extension (JCE) APIs must reside within a WebSphere Application Server environment. This restriction includes applications deployed in WebSphere Application Server and client applications in the J2EE application client environment.
- Only com.ibm.crypto.provider.IBMJCE, com.ibm.jsse.IBMJSSEProvider, com.ibm.security.cert.IBMCertPath, and com.ibm.crypto.pkcs11.provider.IBMPKCS11 are provided as the cryptography package providers.
- Interoperability of the IBM JSSE implementation with other SSL implementations by vendors is limited to tested implementations. The tested implementations include Microsoft Internet Information Services (IIS), BEA WebLogic Server, IBM AIX, and IBM AS/400.
- Hardware token support is limited to supported cryptographic token devices. .

| Tested for SSL clients | Tested for SSL clients or servers |
|--|-----------------------------------|
| IBM Security Kit Smartcard | IBM 4758-23 |
| GemPlus Smartcards | IBM 4758-23 |
| Rainbow iKey 1000/2000(USB "Smartcard" device) | IBM 4758-23 |

- The SSL protocol of Version 2.0 is not supported. In addition, the JSSE and JCE APIs are not supported with Java applet applications.

WebSphere Application Server and the Federal Information Processing Standards for Java Secure Socket Extension and Java Cryptography Extension providers

The Federal Information Processing Standards (FIPS)-approved Java Secure Socket Extension (JSSE) and Java Cryptography Extension (JCE) providers are optional in WebSphere Application Server. By default, the FIPS-approved JSSE and JCE providers are disabled. When these providers are enabled, WebSphere Application Server uses FIPS-approved cryptographic algorithms in the IBMJSSEFIPS and IBMJCEFIPS provider packages only.

Important: The IBMJSSEFIPS and IBMJCEFIPS modules are undergoing FIPS 140-2 certification. For more information on the FIPS certification process and to check the status of the IBM submission, see the Cryptographic Module Validation Program FIPS 140-1 and FIPS 140-2 Pre-validation List Web site.

Authenticity

Authenticity of client and server identities during a Secure Sockets Layer (SSL) connection is validated by both communicating parties using public key cryptography or asymmetric cryptography, to prove the claimed identity from each other.

Public key cryptography is a cryptographic method that uses public and private keys to encrypt and decrypt messages. The public key is distributed as a public key certificate while the private key is kept private. The public key is also a cryptographic inverse of the private key. Well known public key cryptographic algorithms such as the Rivest Shamir Adleman (RSA) algorithm and Diffie-Hellman (DH) algorithm are supported in the WebSphere Application Server.

Public key certificates are either issued by a trusted organization like a certificate authority (CA) or extracted from a self-signed personal certificate by using the IBM Key Management Tool (iKeyman). A self-signed certificate is less secure and is not recommended for use in a production environment.

The public key certificate includes the following information:

- Issuer of the certificate
- Expiration date
- Subject that the certificate represents
- Public key belonging to the subject
- Signature by the issuer

You can link multiple key certificates into a certificate chain. In a certificate chain, the client is always first, while the certificate for a root CA is last. In between, each certificate belongs to the authority that issued the previous one.

During the Secure Sockets Layer (SSL) connection, a digital signature is also applied to avoid forged keys. The digital signature is an encrypted hash and cannot be reversed. It is very useful for validating the public keys.

SSL supports reciprocal authentication between the client and the server. This process is optional during the handshake. By default, a WebSphere Application Server client always authenticates its server during the SSL connection. For further protection, you can configure a WebSphere Application Server for client authentication.

Refer to the Transport Layer Security (TLS) specification at <http://www.ietf.org/rfc/rfc2246.txt> for further information.

Confidentiality

Secure Sockets Layer (SSL) uses private or secret key cryptography or symmetric cryptography to support message confidentiality or privacy. After an initial handshake (a negotiation process by message exchange), the client and server decide on a secret key and a cipher suite. Between the communicating parties, each message encryption and decryption using the secret key occurs based on the cipher suite.

Private key cryptography requires the two communicating parties to use the same key for encryption and decryption. Both parties must have the key and keep the key private. Well known secret key cryptographic algorithms include the Data Encryption Standard (DES), triple-strength DES (3DES), and Rivest Cipher 4 (RC4), which are all supported in WebSphere Application Server. These algorithms provide excellent security and quick encryption.

A cryptographic algorithm is a *cipher*, while a set of ciphers is a *cipher suite*. A cipher suite is a combination of cryptographic parameters that define the security algorithms and the key sizes used for authentication, key agreement, encryption strength and integrity protection.

The following cipher suites are supported in WebSphere Application Server:

- SSL_RSA_WITH_RC4_128_SHA
- SSL_RSA_WITH_RC4_128_MD5
- SSL_RSA_WITH_DES_CBC_SHA
- SSL_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_DSS_WITH_DES_CBC_SHA
- SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_EXPORT_WITH_RC4_40_MD5
- SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
- SSL_RSA_WITH_NULL_MD5
- SSL_RSA_WITH_NULL_SHA
- SSL_DH_anon_WITH_RC4_128_MD5
- SSL_DH_anon_WITH_DES_CBC_SHA
- SSL_DH_anon_WITH_3DES_EDE_CBC_SHA
- SSL_DH_anon_EXPORT_WITH_RC4_40_MD5
- SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA

All of the previously mentioned cipher suites provide data integrity protection by using hash algorithms like MD5 and SHA-1. The cipher suite names ending with `_SHA` indicate that the SHA-1 algorithm is used. SHA-1 is considered a stronger hash, while MD5 provides better performance.

The `SSL_DH_anon_XXX` cipher suites (for example, those cipher suites that begin with `SSL_DH_anon_`, where, `anon` is *anonymous*) are not enabled on the product client side. Because the Java Secure Socket Extension (JSSE) client trust manager does not support anonymous connections, the JSSE client must always establish trust in the server. However, the `SSL_DH_anon_XXX` cipher suites are enabled on the server side to support another type of client connection. That client might not require trust in the server. These cipher suites are vulnerable to *man-in-the-middle* attacks and are strongly discouraged. In a *man-in-the-middle* attack, an attacker is able to intercept and potentially modify communications between two parties without either party being cognizant of the attack.

Where:

| Name | Description |
|------|--|
| SSL | Secure Sockets Layer |
| RSA | <ul style="list-style-type: none"> • Public key algorithm developed by Rivest, Shamir and Adleman • Requires RSA or DSS key exchange |
| DH | <ul style="list-style-type: none"> • Diffie-Hellman public key algorithm • Server certificate contains the Diffie-Hellman parameters signed by the certificate authority (CA) |
| DHE | <ul style="list-style-type: none"> • Ephemeral Diffie-Hellman public key algorithm • Diffie-Hellman parameters are signed by a DSS or RSA certificate, which is signed by the certificate authority (CA) |

| Name | Description |
|--------|--|
| DSS | Digital Signature Standard, using the Digital Signature Algorithm for digital signatures |
| DES | <ul style="list-style-type: none"> • Data Encryption Standard, an symmetric encryption algorithm • Block cipher • Performance cost is high when using software without the support of a hardware cryptographic device |
| 3DES | <ul style="list-style-type: none"> • Triple DES, increasing the security of DES by encrypting three times with different keys • Strongest of the ciphers • Performance cost is very high when using software without the support of a hardware cryptographic device support |
| RC4 | <ul style="list-style-type: none"> • A stream cipher designed for RSA • Variable key-size stream cipher with key length from 40 bits to 128 bits |
| EDE | Encrypt-decrypt-encrypt for the triple DES algorithm |
| CBC | <ul style="list-style-type: none"> • Cipher block chaining • A mode in which every plain text block encrypted with the block cipher is first exclusive-ORed with the previous ciphertext block |
| 128 | 128-bit key size |
| 40 | 40-bit key size |
| EXPORT | Exportable |
| MD5 | <ul style="list-style-type: none"> • Secure hashing function that converts an arbitrarily long data stream into a digest of fixed size • Produces 128-bit hash |
| SHA | <ul style="list-style-type: none"> • Secure Hash Algorithm, same as SHA-1 • Produces 160-bit hash |
| anon | For anonymous connections |
| NULL | No encryption |
| WITH | The cryptographic algorithm is defined after this key word |

Refer to the Transport Layer Security (TLS) specification at <http://www.ietf.org/rfc/rfc2246.txt> for further information.

Integrity

Secure Sockets Layer (SSL) uses a cryptographic hash function similar to checksum, to ensure data integrity in transit. Use the cryptographic hash function to detect accidental alterations in the data. This function does not require a cryptographic key. After a cryptographic hash is created, the hash is encrypted with a secret key. The private key belonging to the sender encrypts the hash for the digital signature of the message.

When secret key information is included with the cryptographic hash, the resulting hash is known as a *Key-Hashing Message Authentication Code* (HMAC) value. HMAC

is a mechanism for message authentication that uses cryptographic hash functions. Use this mechanism with any iterative cryptographic hash function, in combination with a secret shared key.

In the product, both well known *one-way* hash algorithms, MD5 and SHA-1, are supported. One-way hash is an algorithm that converts processing data into a string of bits known as a *hash value* or a *message digest*. *One-way* means that it is extremely difficult to turn the fixed string back into the original data. The following explanation includes both the MD5 and SHA-1 *one-way* hash algorithms:

- MD5 is a hash algorithm designed for a 32-bit machine. It takes a message of arbitrary length as input and produces a 128-bit hash value as output. Although this process is less secure than SHA-1, MD5 provides better performance.
- SHA-1 is a secure hash algorithm specified in the Secure Hash Standard. It is designed to produce a 160-bit hash. Although it is slightly slower than MD5, the larger message digest makes it more secure against attacks like *brute-force collision*.

Refer to the Transport Layer Security (TLS) specification at <http://www.ietf.org/rfc/rfc2246.txt> for further information.

Configuring Secure Sockets Layer

Secure Sockets Layer (SSL) is used by multiple components within WebSphere Application Server to provide trust and privacy. These components are the built-in HTTP Transport, the Object Request Broker (ORB) (for client and server) and the secure Lightweight Directory Access Protocol (LDAP) client. Configuring SSL is different between client and server with WebSphere Application Server .

1. Configure the client (JSSE). Use the `sas.client.props` file located in the `${install_root}/properties` directory. The `sas.client.props` file is a configuration file that contains lists of property-value pairs, using the syntax `<property> = <value>`. The property names are case sensitive, but the values are not; the values are converted to lowercase when the file is read. By default, the `sas.client.props` file is located in the `properties` directory under the `install_root` of your WebSphere Application Server installation. Specify the following properties for an SSL connection:
 - `com.ibm.ssl.protocol`
 - `com.ibm.ssl.keyStoreType`
 - `com.ibm.ssl.keyStore`
 - `com.ibm.ssl.keyStorePassword`
 - `com.ibm.ssl.trustStoreType`
 - `com.ibm.ssl.trustStore`
 - `com.ibm.ssl.trustStorePassword`
 - `com.ibm.ssl.enabledCipherSuites`
 - `com.ibm.ssl.contextProvider`
 - `com.ibm.ssl.keyStoreServerAlias`
 - `com.ibm.ssl.keyStoreClientAlias`
 - For the Secure Authentication Services (SAS) authentication protocol only:
`com.ibm.CORBA.standardPerformQOPModels`
 - For the cryptographic token device:
 - `com.ibm.ssl.tokenType`
 - `com.ibm.ssl.tokenLibraryFile`
 - `com.ibm.ssl.tokenPassword`

Note: **5.1+** Although WebSphere Application Server supports the IBM Federal Information Processing Standard-approved Java Secure Socket Extension (IBMJSSSEFIPS), IBMJSSEFIPS is not supported on the HP-UX platform.

2. Configure the server. Use the administrative console to configure an application server that makes SSL connections. To start the administrative console, specify the following Web address: `http://server_hostname:9090/admin`.
3. Create an SSL configuration repertoires alias or entry. You can select the alias later when a component is configured for SSL support. An SSL configuration repertoires entry contains the following fields:
 - Typical configuration settings:
 - Alias
 - Key file name
 - Key file password
 - Key file format
 - Trust file name
 - Trust file password
 - Trust file format
 - Client authentication
 - Security level
 - Cipher suites
 - For the cryptographic token device:
 - Cryptographic token (Create the alias first so you can configure these fields).
 - Token type
 - Library file
 - Password
 - For additional Java properties:
 - Custom properties (Create the alias first so you can configure these fields).
 - `com.ibm.ssl.contextProvider`
 - `com.ibm.ssl.protocol`

Note: **5.1+** WebSphere Application Server contains IBM Developer Kit for Java Technology Edition Version 1.4.x , which includes changes from IBM Developer Kit for Java Technology Edition Version 1.3. See Changes to IBM Developer Kit for Java Technology Edition Version 1.4.x for more information.

Configuring Secure Sockets Layer for Web client authentication

To enable client-side certificate-based authentication, you must modify the authentication method defined on the J2EE Web module that you want to manage. The Web module might already be configured to use the basic challenge authentication method. In this case, modify the challenge type to `client certificate`. This functionality is delivered to the WebSphere Application Server administrator in the Assembly Toolkit. However, developers can use the WebSphere Application Server Studio Application Development environment to achieve the same result.

1. Launch the Assembly Toolkit. This step can be done either before an enterprise application archive `.ear` file is deployed into the WebSphere Application Server or after deployment into the product. The latter option is discouraged in a production environment because it involves opening the expanded archive correlating to the enterprise application archive, found in the `installedApps` directory.

2. Locate and expand the Web module package under the application for which you wish to enable the client-side certificate authentication method.
3. Select the appropriate Web application, and switch to the **Advanced** tab. Modify the authentication method to client certificate. The realm name is the scope of the login operation and is the same for all participating resources.
4. Click **OK**, and save the changes you made with Assembly Toolkit.
5. Stop and restart the associated application server containing the resource, so that the security modification is included in the run time. Complete this action if the modification was made to a resource that already is deployed in the WebSphere Application Server.

Now your enterprise application prompts the user for proof of identity with a certificate.

The Web server must also be configured to request a client certificate. If the Web server is external, refer to the appropriate configuration documentation. If the Web server is the Web container transport (for example, 9043) within WebSphere Application Server, verify that the **client authentication** flag is selected in the referenced SSL configuration.

Refer to the Map certificates to users article to determine how a certificate is authenticated within the product.

Configuring Secure Sockets Layer for the Lightweight Directory Access Protocol client

This topic describes how to establish a Secure Sockets Layer (SSL) connection between WebSphere Application Server and a Lightweight Directory Access Protocol (LDAP) server. This page provides an overview. Refer to the linked pages for more details. To understand SSL concepts, refer to Secure Sockets Layer (SSL).

Setting up an SSL connection between WebSphere Application Server and an LDAP server requires the following steps:

1. Set up an LDAP server with users. The server configured in this example is IBM Directory Server. Other servers are configured differently. Refer to the documentation of the directory server you are using for details on SSL enablement. For a product-supported LDAP directory server, see the Supported directory services article.
2. Configure certificates for the LDAP server using the key management utility (iKeyman) that is shipped with the IBM HTTP Server product.
3. Click **Key Database File > New**.
4. Type LDAPkey.kdb as the file name and a proper path.
5. Click **Personal Certificates > New Self-Signed Certificate**. The **Create New Self-Signed Certificate** panel is displayed. Type the following information in the fields and click **OK**:

Key Label

LDAP_Cert

Common Name

droplet.austin.ibm.com

This common name is the host name where the WebSphere Application Server plug-in runs.

Organization

ibm

Country

US

6. Return to the Personal Certificates panel and click **Extract Certificate**.
7. Click the **Base64-encoded ASCII data** data type. Type LDAP_cert.arm as the file name and a proper path. Click **OK**.
8. Enable SSL on the LDAP server:
 - a. Copy the LDAPkey.kdb, LDAPkey.sth, LDAPkey.rdb, and LDAPkey.crl files created previously to the LDAP server system, for example, the \Program Files\IBM\LDAP\ssl\ directory.
 - b. Open the LDAP Web administrator from a browser (http://secnt3.austin.ibm.com/ldap, for example). IBM HTTP Server is running on secnt3.
 - c. Click **SSL properties** to open the SSL Settings window.
 - d. Click **SSL On > Server Authentication** and type an SSL port (636, for example) and a full path to the LDAPkey.kdb file.
 - e. Click **Apply**, and restart the LDAP server.
9. Manage certificates for WebSphere Application Server using the default SSL key files.
 - a. Open the *install_root\etc\DummyServerTrustFile.jks* file using the key management utility that shipped with WebSphere Application Server. The password is WebAS.
 - b. Click **Personal Certificates > Import**. The **Import Key** panel is displayed. Specify LDAP_cert.arm for the file name. Complete this step for all the servers including the deployment manager.
10. Establish a connection between the WebSphere Application Server and the LDAP server.
 - a. In the administrative console, click **User Registry > LDAP User Registry > LDAP Settings**. Fill in the **Server ID**, **Server Password**, **Type**, **Host**, **Port**, and **Base Distinguished Name** fields. Select the **SSL Enabled** check box. The port is the one that the LDAP server is using for SSL (636, for example). Click **Apply**.
 - b. Click **Authentication Mechanisms > LTPA > Single SignOn (SSO)**. Type in a domain name (austin.ibm.com, for example). Click **Apply**.
11. Enable global security.
 - a. Click **Security > Global Security**. Select the **Enabled** check box. Choose **LTPA** as the active authentication mechanism and **LDAP** as the active user registry. Click **Apply** and **Save**.

Note: Verify that the security level for the LDAP server is set to HIGH. The default security level is HIGH (128-bit).
 - b. Check the *LDAP_install_root\etc\slapd32.conf* file; verify that the `ibm-slapdSSLCipherSpecs` parameter has the value, 15360, instead of 12288.
 - c. Restart the servers. Restarting the servers ensures that the security settings are synchronized between the deployment manager and the application servers.

You can test the configuration by accessing https://fully_qualified_host_name:9443/snoop. You are presented with a login challenge.

This test can be beneficial when using LDAP as your user registry. Sensitive information can flow between the WebSphere Application Server and the LDAP server, including passwords. Using SSL to encrypt the data protects this sensitive information.

1. If you are enabling security, make sure that you complete the remaining steps. As the final step, validate this configuration by clicking **OK** or **Apply** in the Global Security panel. Refer to the Configuring global security article for detailed steps on enabling global security.
2. For changes in this panel to become effective, save, stop, and start all WebSphere Application Servers (cells, nodes and all the application servers).
3. After the server starts up, go through all the security-related tasks (getting users, getting groups, and so on) to make sure that the changes to the filters are functioning.

Configuring IBM HTTP Server for secure sockets layer mutual authentication

IBM HTTP Server supports Secure Sockets Layer (SSL) Version 2 and Version 3 and Transport Layer Security (TLS) Version 1. IBM HTTP Server is based on the Apache Web server, but for SSL configuration it requires the IBM-supplied SSL modules, rather than the OpenSSL modules. This document describes configuration of IBM HTTP Server, although it is possible to use another supported Web server.

SSL is disabled by default and it is necessary to modify a configuration file and generate a server-side certificate using the key management utility (iKeyman) provided with IBM HTTP Server to enable SSL.

1. For a single server, enable SSL on IBM HTTP Server (port 443, for example).
2. To set up certificates complete the following: Start the key management utility by clicking **Start > Programs > IBM HTTP Server > Start Key Management Utility**. Refer to Requesting a CA-signed personal certificate, Creating a certificate signing request (CSR), Receiving a CA-signed personal certificate, and Extracting a public certificate for use in a truststore file
3. Create a key database and click **Key Database File > New**.
4. Type a file name, serverkey.kdb, for example, and the location path. Click **OK**.
5. Type a password, select the **Stash the password to a file** check box and click **OK**.
6. Obtain a personal certificate for IBM HTTP Server: Click **Personal Certificate Requests** in the key management utility menu. Click **New**. The **Create New Key and Certificate Request** panel appears. Complete the following information:

Key label
Server_Cert
Common name
droplet.austin.ibm.com
Organization
IBM
Country
US
File name
Server_certreq.arm

The Verisign Test CA Root Certificate is in the set of signer certificates shipped with the IKeyMan for IBM HTTP Server.

7. Go to URL <http://www.verisign.com>, click **Get Free Trial SSL ID**. Complete the profile information, click **Submit**, and click **Continue** twice.
8. Use your favorite text editor to edit the request file `Server_certreq.arm`, and copy the entire contents of the file into the browser request panel. Click **Continue**. VeriSign displays the Personal Certificate in the browser.
9. Copy and paste this certificate into a file, for example `Server_Cert.arm`. Click **Personal Certificate** from the menu in the key management utility. Click **Receive**. Specify the file name, `Server_Cert.arm`, and click **OK**. Close the `serverkey.kdb` file.
10. To allow IBM HTTP Server to support HTTPS, port 443, for example, enable SSL on IBM HTTP Server. Modify the configuration file of IBM HTTP Server, `<IHS_HOME>/conf/httpd.conf`. You also can enable SSL can be enabled through the IBM HTTP Server administrative console also. Open the file `<IHS_HOME>/conf/httpd.conf` and then add the following lines above the line `Alias /IBMWebAS/ "<install_root>/web"`:

```
LoadModule ibm_ssl_module modules/IBMModuleSSL128.dll
<install_root>/bin/mod_ibm_app_server_http.dll
Listen 443
<VirtualHost droplet.austin.ibm.com:443>
ServerName droplet.austin.ibm.com
DocumentRoot <install_root>/htdocs
SSLEnable
#SSLClientAuth required
SSLDisable
Keyfile <IHS_HOME>/serverkey.kdb
```

Note: Change the host name and the path for the key file accordingly. Modify the Web server to support client certificates by uncommenting the `SSLClientAuth` directive shown in the `httpd.conf` file.

```
SSLClientAuth required
```

11. Restart IBM HTTP Server.
12. Test SSL between a browser and IBM HTTP Server. For more information on the default IBM HTTP Server port number, see Port number settings in WebSphere Application Server versions.
13. Follow the prompts to select a personal certificate if the `SSLClientAuth` directive is set to required.
14. To enable the application server to communicate with IBM HTTP Server using port 443, add the host alias on the default_host. Click **Environment > Virtual Hosts > default host > Host Aliases > New**. Enter the following information in the appropriate fields:

| | |
|------------------|-----|
| Host name | * |
| Port type | 443 |
15. Click **Apply** and **Save** to write to the `security.xml` file.
16. Click **Update Web Server Plugin**, and then click **OK**.
17. Restart WebSphere Application Server.
18. Test your connection.

You can connect to the Snoop servlet.

Enable Secure Sockets Layer communication between IBM HTTP Server and the WebSphere Application Server.

Configuring the IBM HTTP Server for distributed platforms and the Web server plug-in for Secure Sockets Layer

This section documents the configuration necessary to instantiate a secure connection between the Web server plug-in and the internal HTTP transport in the WebSphere Application Server Web container on a distributed platform. By default, this connection is not secure, even when global security is enabled. This document discusses the configuration for the IBM HTTP Server; however, the Web server related configuration in this situation is not specific to any distributed platform Web server.

1. Create a self-signed certificate for the Web server plug-in. The Web server plug-in requires a key ring file to store its own private and public key files and to store the public certificate from the Web container key file. The following steps are required to generate a self-signed certificate for the Web server plug-in.
 - a. Create a directory on the Web server host for storing the key ring file referenced by the plug-in and associated files, for example:
<IHS_install_root>\conf\keys.
 - b. Launch the key management utility (iKeyman) packaged with the IBM HTTP Server.
 - c. From the iKeyman menu, click **Key Database File > New**.
 - d. Enter the following settings:
 - Key database file**
CMS Key Database File
 - File name**
WASplugin.kdb
 - Location**
C:\http1324\conf\keys\ (or file of your choice)
 - e. Click **OK**.
 - f. Set the password of your choice at the password prompt. Select the **Stash the Password to a File** check box to save the password to a stash file. This action allows the plug-in to use the password, which provides access to the certificates contained in the key database.
 - g. From the iKeyman menu, click **Create > New Self-Signed Certificate** to create a new self-signed certificate key pair. Specify the following options. Optionally, you can choose to complete all of the remaining fields.
 - Key label**
WASplugin
 - Version**
X509 V3
 - Key size**
1024
 - Common name**
droplet.austin.ibm.com
 - Organization**
IBM
 - Country**
US
 - Validity period**
365
 - h. Click **OK**.

- i. Extract the public self-signed certificate key: this key is used later by the embedded HTTP server peer to authenticate connections originating from the plug-in.
 - j. Click **Personal Certificates** in the menu and select the WASplugin certificate that you just created.
 - k. Click **Extract Certificate**. Extract the certificate to a file:
 - Data type**
Base64-encoded ASCII data
 - Certificate file name**
WASpluginPubCert.arm
 - Location**
C:\http1324\conf\keys (or directory of your choice)
 - l. Click **OK**.
 - m. Close the key database and exit the iKeyman when you finish.
2. Generate a self-signed certificate for the Web container.
 - a. Launch the JKS capable iKeyman version located the product /bin directory.
 - b. Click **Key Database File > New** from the iKeyman menu.
 - c. Enter the following settings:
 - Key database file**
JKS
 - File name**
WASWebContainer.jks
 - Location**
C:\WebSphere\AppServer\etc\ (or directory of your choice)
 - d. Click **OK**.
 - e. Enter the password of your choice at the password prompt window.
 - f. Click **Create > New Self-Signed Certificate** from the iKeyman menu. The following values were used in this example:
 - Key Label**
WASWebContainer
 - Version**
X509 V3
 - Key size**
1024
 - Common name**
droplet.austin.ibm.com
 - Organization**
IBM
 - Country**
US
 - Validity Period**
365
 - g. Click **OK**.
 - h. Extract the public self-signed certificate key: this key is used later by the Web server plug-in peer to authenticate connections originating from the embedded HTTP server in the product.
 - i. Click **Personal Certificates** from the list. Select the **WASWebContainer** certificate that you just created. Click **Extract Certificate**. Extract the certificate to a file:
 - Data type**
Base64-encoded ASCII data

Certificate file name

WASWebContainerPubCert.arm

Location

C:\WebSphere\AppServer\etc\

- j. Click **OK**.
- k. Close the database and exit the key management utility.
3. Exchange the public certificates.
 - a. Copy the WASpluginPubCert.arm file from the Web server machine to the WebSphere Application Server machine. The source directory in this case is C:\http1324\conf\keys, while the destination is C:\WebSphere\Appserver\etc.
 - b. Copy the WASWebContainerPubCert.arm file from the product machine to the Web server machine. The source directory in this case is C:\WebSphere\Appserver\etc, while the destination is C:\http1324\conf\keys.
4. Import the certificate into the Web server plug-in key file.
 - a. On the Web server machine, launch the key management utility that supports the CMS key database format.
 - b. From the iKeyman menu, click **Key Database File > Open** and select the previously created key database file: WASplugin.kdb.
 - c. In the password prompt window, enter the password. Click **OK**.
 - d. Click **Signer Certificates** from the list and click **Add**. This action imports the public certificate previously extracted from the embedded HTTP server (Web container) keystore file.

Data type

Base64-encoded ASCII data

Certificate file name

WASWebContainerPubCert.arm

Location

C:\WebSphere\Appserver\etc\

- e. Click **OK**. You are prompted for a label name that represents the trusted signer public certificate.
- f. Enter a label for the certificate: WASWebContainer.
- g. Close the key database and exit IKeyman when you finish.
5. Import the certificate into the Web container keystore file.
 - a. On the WebSphere Application Server machine, launch the JKS capable iKeyman version, located in the product /bin directory.
 - b. From the iKeyman menu, select **Key Database File > Open**. Select the previously created WASWebContainer.jks file.
 - c. In the password prompt window, enter the password. Click **OK**.
 - d. Click **Signer Certificates** from the list. Click **Add**. This action imports the public certificate previously extracted from the embedded HTTP server (Web container) keystore file.

Data type

Base64-encoded ASCII data

Certificate file name

WASpluginPubCert.arm

Location

C:\WebSphere\Appserver\etc\

- e. Click **OK**. You are prompted for a label name that represents the trusted signer public certificate.

- f. Enter a label for the certificate: WASplugin.
 - g. Close the key database and exit iKeyman when you finish.
6. Modify the Web server plug-in file. In a production environment, add the secure transport definition, port 9443, to the plugin-cfg.xml file. For example, your modified plugin-key.kdb file contains the following lines:

```
<Transport Hostname="hpws07" Port="9080" Protocol="http"/>
<Transport Hostname="hpws07" Port="9443" Protocol="https"/>
```

After you verify that the proper plugin-key.kdb and plugin-key.sth files exist on the Web server, modify the plugin-cfg.xml file that resides on the Web server. You must specify the local path to both the plugin-key.kdb and plugin-key.sth files in the plugin-cfg.xml file. For more information, see plugin-cfg.xml file and Situations requiring manual editing of the plug-in configuration.

Important: If you manually edit the plugin-cfg.xml file and an automatic regeneration of the file occurs, you must replace your manual edits.

7. Modify the Web container to support SSL. To complete the configuration between Web server plug-in and Web container, modify the WebSphere Application Server Web container to use the previously created self-signed certificates.
 - a. Start the WebSphere Application Server administrative console.
 - b. Click **Security > SSL Configuration Repertoires**.
 - c. Click **New** to create a new entry in the repertoire. Provide the following values to complete the form:

Alias WebContainerSSLSettings

Key file name
C:\WebSphere\Appserver\etc\WASWebContainer.jks

Key file password
<key_file_password>

Key file format
JKS

Trust file name
C:\WebSphere\Appserver\etc\WASWebContainer.jks

Trust file password
<trust_file_password>

Trust file format
JKS

Client authentication

Security level
HIGH
 - d. Click **OK**.
 - e. If you want mutual SSL between the two parties, select the **Client Authentication** check box.
 - f. Save the configuration in the administrative console.
 - g. Click **Servers > Application Servers**, *server_name*, in this example, server1.
 - h. Click the Web container located in the server navigation tree.
 - i. Click **HTTP Transport** located in the Web container navigation tree.
 - j. Select the entry for the transfer you want to secure. Click the item under the **Host** column. Select the asterisk (*), in this case, in the line of port **9443**.

- k. On the configuration panel, select the **Enable SSL** check box. Click the desired SSL entry from the SSL repertoire list. In this example, the WebContainerSSLSettings.
 - l. Click **OK**.
8. Test the secure connection. Test the secure connection by accessing a Web application on the WebSphere Application Server using port 9443. For example, <https://droplet.austin.ibm.com:9443/snoop>.
 9. Import the correct certificate with public and private keys into the browser to test the secured connection, when client-side certification is required.
 - a. Launch the iKeyman utility that supports the CMS key database file, on the Web server machine.
 - b. Open the key file for the plug-in, C:\http1324\conf\keys\WASplugin.kdb. Provide the password when prompted.
 - c. Click **WASplugin certificate**, located under the Personal Certificates. Click **Export**.
 - d. Save the certificate in PKCS12 format to a file, for example C:\http1324\conf\keys\WASplugin.p12 . Provide a password to secure the PKCS12 certificate file.
 - e. Close the key file and exit iKeyman.
 - f. Copy the saved WASplugin.p12 file to the client machine from where you access the product server.
 - g. Import the PKCS12 file into your browser. Then, access https://your_server_address:9443/snoop.
 - h. The browser asks which personal certificate to use for the connection. Select the certificate, and continue connecting.
 - i. Once the browser test with direct product access is successful, test the connection through the Web server using port 9443. For example, https://your_server_address:9443/snoop.

The IBM HTTP Server plug-in and the internal Web server are configured for SSL.

Enabling Secure Sockets Layer (SSL) communication between the IBM HTTP Server plug-in and the embedded HTTP server (Web container) in the WebSphere Application Server.

Configuring Secure Sockets Layer for Java client authentication

WebSphere Application Server supports Java client authentication using a digital certificate when the client attempts to make a Secure Sockets Layer (SSL) connection. The authentication occurs during an SSL handshake. The SSL handshake is a series of messages exchanged over the SSL protocol to negotiate for connection-specific protection. During the handshake, the secure server requests the client to send back a certificate or certificate chain for the authentication.

To configure SSL for Java client authentication, consider the following questions:

- Have you enabled security with your WebSphere Application Server? Refer to *Configuring global security* for more details.
- Have you configured Common Secure Interoperability (CSI) authentication protocol for your target application server? Refer to *Configuring global security* for more details.

Note: The Security Authentication Service (SAS) authentication protocol does not support Java client authentication with SSL transport.

- Have you configured your server to support secure transport for the inbound CSI authentication protocol?
- Have you configured your server to support client authentication at the transport layer for the inbound CSI authentication protocol?
- If you are using a self-signed personal certificate, have you exported the public certificate from your client application Java keystore file or cryptographic token device?
- If you are using a certificate authority (CA)-signed personal certificate, have you received the root certificate of the CA?
- If you are using a self-signed personal certificate, have you imported the public certificate into your target Java truststore file as a signer certificate?
- If you are using a CA-signed (certificate authority) personal certificate, have you imported the CA root certificate into your target Java trust store file as a signer certificate?
- Does the common name (CN) specified in your personal certificate name exist in your configured user registry?

If you answer yes to all of these questions, you can configure SSL for Java client authentication.

Note: Java client authentication using digital certificates is supported only by the Common Secure Interoperability Version 2 (CSIv2) authentication protocol.

1. Configure CSIv2 for SSL client authentication.
2. Add keystore files.
3. Add truststore files.
4. Save changes.
5. Restart the server if you have configured the server.

A secure client connects to a secure Internet InterORB Protocol (IIOP) server that requires client authentication at the transport layer.

If a connection problem occurs, you can set a Java property, `javax.net.debug=true`, before you run your client or your server to generate debugging information. See *Troubleshooting security configurations* for further information about how to debug an IBM JSSE problem.

Configuring Common Secure Interoperability Version 2 for Secure Sockets Layer client authentication:

Configure the Secure Sockets Layer (SSL) client authentication using the `sas.client.props` configuration file or the administrative console. To configure a Java client application, use the `sas.client.props` configuration file. By default, the `sas.client.props` file is located in the `properties` directory under the `<install_root>` of your WebSphere Application Server installation.

To configure a WebSphere Application Server, use the administrative console. To start the administrative console, specify URL: `http://<server_host_name>:9090/admin`.

To configure a Java client application, complete the following steps, which explain how to edit the `sas.client.props` file.

1. To require SSL client authentication, set property `com.ibm.CSI.performTLCClientAuthenticationRequired=true`. Do not set this property unless you know your target server also supports SSL client authentication for the inbound CSI authentication protocol.

2. To support SSL client authentication, set the property `com.ibm.CSI.performTLCClientAuthenticationSupported=true`.
3. To specify the CSI protocol, set the property `com.ibm.CSI.protocol=csiv2`.
4. To match the SSL protocol configured with your server, set the property, `com.ibm.ssl.protocol`, accordingly.
5. Specify the `com.ibm.CORBA.ConfigURL` property with the fully qualified path of your Java property file when you run your application. For example, `-Dcom.ibm.CORBA.ConfigURL=file:/c:/WebSphere/AppServer/properties/sas.client.props`

To configure a WebSphere Application Server, complete the following steps

1. Start the administrative console.
2. Expand **Security > Authentication Protocol**.
3. Click **CSiv2 Inbound Authentication**.
4. Select **Supported** or **Required** for Client Certificate Authentication.
5. Click **OK**.
6. If you selected **Required** in step 4, configure the CSiv2 outbound authentication as well to support the client certificate authentication. Otherwise, you can skip this step. Click **CSiv2 Outbound Authentication** and select either **Supported** or **Required** for Client Certificate Authentication.
7. Click **CSiv2 Outbound Transport**. Select an SSL setting from the SSLSettings list for keystore, truststore, cryptographic token, SSL protocol, and ciphers use. Create an alias from the SSL Configuration Repertoires panel for an SSL setting. Update the SSL setting selected in CSiv2 Inbound Transport accordingly.
8. Save your configuration.
9. Restart the server for the changes to become effective.

Client authentication using digital certificates is performed during SSL connection.

A secure client connects using SSL to a secure Internet InterORB Protocol (IIOP) server with client authentication at the transport layer.

Specify the keystore and truststore files in your configuration.

Adding keystore files:

A keystore file contains both public keys and private keys. Public keys are stored as signer certificates while private keys are stored in the personal certificates. In WebSphere Application Server, adding keystore files to the configuration is different between client and server. For the client, a keystore file is added to a property file like `sas.client.props`. For the server, a keystore file is added through the WebSphere Application Server administrative console.

Before you add the keystore file to your configuration, consider the following questions:

- Is a self-signed or a certificate authority (CA)-signed personal certificate created in the keystore file?
 - If you configure client authentication using digital certificates, is the public key of the signed personal certificate imported as a signer certificate into the server truststore file?
1. Add a keystore file into a client configuration by editing the `sas.client.props` file and setting the following properties:
 - **com.ibm.ssl.keyStoreType** for the keystore format. Range: JKS (default), PKCS12KS, JCEK.

- **com.ibm.ssl.keyStore** for a fully qualified path to the keystore file. The keystore file contains private keys and sometimes public keys.
 - **com.ibm.ssl.keyStorePassword** for the password to access the keystore file.
2. Add a keystore file into a server configuration:
 - a. Start the WebSphere administrative console by specifying:
`http://server_hostname:9090/admin`.
 - b. Click **Security > SSL Configuration Repertoires**.
 - c. Create a new Secure Sockets Layer (SSL) setting alias if one does not exist.
 - d. Select the alias that you want to add into the keystore file.
 - e. Type in the Key File Name for the path of the keystore file.
 - f. Type in the Key File Password for the password to access the keystore file.
 - g. Select the **Key File Format** for the keystore type. Range: JKS (default), PKCS12KS, or JCEK.
 - h. Click **OK** and **Save** to save the configuration.

The SSL configuration alias now has a valid keystore file for an SSL connection.

Note: If the Cryptographic Token field is selected and you only want to use cryptographic tokens for your keystore file, leave the Key File Name field and the Key File Password field blank.

- SSL connection for Internet InterORB Protocol (IIOP)
- SSL connection for Lightweight Directory Access Protocol (LDAP)
- SSL connection for Hypertext Transfer Protocol (HTTP)

Adding truststore files:

A *truststore file* is a key database file that contains public keys. The public key is stored as a signer certificate. The keys are used for a variety of purposes, including authentication and data integrity. In WebSphere Application Server, adding truststore files to the configuration is different between client and server. For the client, a truststore file is added to a property file, like `sas.client.props`. For the server, a truststore file is added through the WebSphere Application Server administrative console.

Before you add the truststore file to your configuration, ask the following questions:

- If you configure for client authentication using digital certificate, has the public key of the client personal certificate been imported as a signer certificate into the server truststore file?
 - Does the truststore file contain all the required signer certificates with respect to the keystore files of the target servers?
1. Add a truststore file into a client configuration, by editing the `sas.client.props` file and setting the following properties:
 - **com.ibm.ssl.trustStoreType** for the truststore format. Range: JKS (default), PKCS12KS, JCEK, JCERACFKS. Use JCERACFKS if you are using a RACF key ring as the truststore.
 - **com.ibm.ssl.trustStore** for a fully qualified path to the truststore file. The truststore file contains the public keys.
 - **com.ibm.ssl.trustStorePassword** for the password to access the truststore file. The `com.ibm.ssl.trustStorePassword` property should be set to password if you are using a RACF key ring as a trust store.
 2. Add a truststore file into a server configuration:

- a. Start the WebSphere administrative console by specifying :
`http://server_host_name:9090/admin`.
- b. Click **Security > SSL**.
- c. Create a new Secure Sockets Layer (SSL) setting alias if one does not exist.
- d. Select the alias that you want to add into the truststore file.
- e. Type the Trust File Password for the password to access the truststore file.
Type password if you are using a RACF key ring for the trust store.
- f. Select the Trust File Format for the truststore type. JKS (Default), PKCS12KS, JCEK.
- g. Click **OK** and **Save** to save the configuration.

The SSL configuration alias now contains a valid truststore file for an SSL connection.

- SSL connection for Internet InterORB Protocol (IIOP)
- SSL connection for Lightweight Directory Access Protocol (LDAP)
- SSL connection for Hypertext Transfer Protocol (HTTP)

Creating a Secure Sockets Layer repertoire configuration entry

The first step in configuring Secure Sockets Layer (SSL) is to define an SSL configuration repertoire. A *repertoire* contains the details necessary for building an SSL connection, such as the location of the key files, their type and the available ciphers. WebSphere Application Server provides a default repertoire called `DefaultSSLSettings`. To view this page in the administrative console, click **Security > SSL** to see the list of SSL repertoire settings.

The appropriate repertoire is referenced during the configuration of a service that sends and receives requests encrypted using SSL, such as the Web and enterprise beans containers. If an SSL configuration alias is referenced elsewhere, but the alias is deleted from the SSL Configuration Repertoires panel, the SSL connection fails if the deleted alias is accessed.

With the SSL configuration repertoire, administrators can define SSL settings to use for making Hypertext Transfer Protocol with SSL (HTTPS), Internet InterORB Protocol with SSL (IIOPS) or Lightweight Directory Access Protocol with SSL (LDAPS) connections. You can pick one of the SSL settings defined here from any location within the administrative console, which supports SSL connections. This selection simplifies the SSL configuration process because you can reuse many of these SSL configurations by specifying the alias in multiple places.

1. From the SSL Configuration Repertoire window, click **New**.
2. Enter the information needed to access the key file.
 - a. Type the name of the key file, which must include the fully qualified path to the key file, in the Key File Name field. Type `safkeyring:///` if you are using a RACF key ring for the key file.
 - b. Type the password needed to access the key file in the Key File Password field. Type password if you are using a RACF key ring for the key store.
 - c. Select the format of the key file from the Key File Format menu.
3. Enter the information needed to access the trust file.
 - a. Type the name of the trust file, which must include the fully qualified path to the trust file, in the Trust File Name field. Type `safkeyring:///` if you are using a RACF key ring as the trust store.

- b. Type the password needed to access the trust file in the Trust File Password field. Type password if you are using a RACF key ring as the trust store.
- c. Select the format of the trust file from the Trust File Format menu.
4. Select the **Client Authentication** check box if this configuration supports client authentication. This selection only affects HTTP and LDAP requests.
5. Select the appropriate security level from the Security Level menu. Valid values are low, medium, and high. Low specifies digital signing ciphers only (no encryption), medium specifies 40-bit ciphers only (including digital signing), high specifies 128-bit ciphers only (including digital signing). If you are using a Federal Information Processing Standards (FIPS)-supported Java Secure Socket Extension (JSSE), you must select **High** from the Security Level menu.
6. Select a cipher suite from the Cipher Suites menu. Manually add the cipher suite if the preset security level does not define the required cipher. Select the **Cryptographic Token** check box if the RACF key ring contains keys or certificates that were created using the RACDCERT command with the ICSF keyword specified.
7. Select the **Cryptographic Token** check box if hardware or software cryptographic support is available. See Configuring to use cryptographic tokens for details regarding cryptographic support.
8. Indicate which JSSE provider you are using by either selecting **IBMJSSE** or **IBMJSSEFIPS** from the menu, or by typing the name of the provider. WebSphere Application Server includes the IBMJSSE JSSE provider and the IBMJSSEFIPS JSSE provider.

Important: The IBMJSSEFIPS JSSE provider is undergoing FIPS 140-2 certification.

Use IBMJSSEFIPS only if you are using the Transport Layer Security (TLS) protocol and not the Secure Sockets Layer (SSL) protocol. See Configuring Federal Information Processing Standard Java Secure Socket Extension files for more information. If you are not using the predefined providers, configure the custom provider by clicking **Apply**, then **Custom Properties > New** in the Additional Properties section. After the custom provider is configured, return to the SSL Configuration Repertoires window and continue with these instructions.

9. Select an SSL or TLS protocol version. If you are using a FIPS-approved JSSE, you must select a TLS protocol version.
10. Click **Apply** to apply the changes.
11. If no errors occur, save the changes to the master configuration and restart the WebSphere Application Server. For more information on the FIPS certification process and to check the status of the IBM submission, see the Cryptographic Module Validation Program FIPS 140-1 and FIPS 140-2 Pre-validation List Web site.

You included additional SSL configuration repertoires with the default DefaultSSLSettings repertoire.

The appropriate repertoire is referenced during the configuration of a service that sends and receives requests encrypted using SSL, such as the Web and enterprise bean containers, and Lightweight Directory Access Protocol (LDAP) servers.

For the changes to take effect, restart the server after saving the configuration.

Secure Sockets Layer configuration repertoire settings:

Use this page to define a new Secure Sockets Layer (SSL) alias. Through the SSL configuration repertoire, administrators can define any number of SSL settings to use in configuring the Hypertext Transfer Protocol with SSL (HTTPS), Internet InterORB Protocol with SSL (IIOPS) or Lightweight Directory Access Protocol with SSL (LDAPS) connections. You can pick one of the SSL settings defined here from any location within the administrative console that supports SSL connections. This flexibility simplifies the SSL configuration process because you can reuse many of these SSL configurations by specifying the alias in multiple places.

To view this administrative console page, click **Security** > **SSL**.

Click **New** to create a new SSL Configuration Repertoire alias.

Click **Delete** to remove an SSL Configuration Repertoire alias. If an SSL configuration alias is referenced in the configuration and is deleted here, then an SSL connection fails when the deleted alias is accessed.

Alias:

Specifies the name of the specific SSL setting.

Secure Sockets Layer settings:

Use this page to configure Secure Sockets Layer (SSL) settings for the server.

To view this administrative console page, click **Security** > **SSL** > *alias_name*.

Alias:

Specifies the name of the specific SSL setting

Data type: String

Key File Name:

Specifies the fully qualified path to the SSL key file that contains public keys and private keys.

You can create an SSL key file with the key management utility, or this file can correspond to a hardware device if one is available. In either case, this option indicates the source for personal certificates and for signer certificates unless a trust file is specified. The default SSL key files, `DummyClientKeyFile.jks` and `DummyServerKeyFile.jks`, contain a self-signed personal test certificate expiring on March 17, 2005. The test certificate is only intended for use in a test environment. The default SSL key files should never be used in a production environment because the private keys are the same on all the WebSphere Application Server installations. Refer to the Managing certificates article for information about creating and managing digital certificates for your WebSphere Application Server domain.

Data type: String

Key File Password:

Specifies the password for accessing the SSL key file.

Data type: String

Key File Format:

Specifies the format of the SSL key file.

Data type: String
Default: JKS
Range: JKS, JCEK, PKCS12

Trust File Name:

Specifies the fully qualified path to a trust file containing the public keys.

You can create a trust file with the key management utility included in the WebSphere *bin* directory. Using the key management utility from Global Security Kit (GSKit) (another SSL implementation) does not work with the Java Secure Socket Extension (JSSE) implementation.

Unlike the SSL key file, no personal certificates are referenced; only signer certificates are retrieved. The default SSL trust files, `DummyClientTrustFile.jks` and `DummyServerTrustFile.jks`, contain multiple test public keys as signer certificates that can expire. The public key for the WebSphere Application Server Version 4.0 test certificates expires on January 15, 2004, and the public key for the WebSphere Application Server Version 5 test certificates and WebSphere Application Server CORBA C++ client expires on March 17, 2005. The test certificate is only intended for use in a test environment.

If a trust file is not specified but the SSL key file is specified, then the SSL key file is used for retrieval of signer certificates as well as personal certificates.

Data type: String

Trust File Password:

Specifies the password for accessing the SSL trust file.

Data type: String

Trust File Format:

Specifies the format of the SSL trust file.

Data type: String
Default: JKS
Range: JKS, JCEK, PKCS12

Client Authentication:

Specifies whether to request a certificate from the client for authentication purposes when making a connection.

This attribute is only valid when used by the Web container HTTP transport. When performing client authentication with the Internet InterORB Protocol (IIOP) for EJB requests, click **Security > Authentication Protocol > CSIV2 Inbound or Outbound Authentication** from the left navigation pane of the administrative console. Click **SSL Client Certificate Authentication** to enable it for these requests.

| | |
|-------------------|---------------------|
| Data type: | Boolean |
| Default: | Disabled |
| Range: | Enabled or Disabled |

Security Level:

Specifies whether the server selects from a preconfigured set of security levels.

| | |
|-------------------|--|
| Data type: | Valid values include Low, Medium or High. <ul style="list-style-type: none">• Low specifies only digital signing ciphers (no encryption)• Medium specifies only 40-bit ciphers (including digital signing)• High specifies only 128-bit ciphers (including digital signing). To specify all ciphers or any particular range, you can set the <code>com.ibm.ssl.enabledCipherSuites</code> property. See the SSL documentation for more information. |
| Default: | High |
| Range: | Low, Medium, or High |

Cipher Suites:

Specifies a list of supported cipher suites that can be selected during the SSL handshake. If you select cipher suites individually here, you override the cipher suites set in the Security Level field.

| | |
|-------------------|--|
| Data type: | |
| Default: | |
| Range: | |

Cryptographic Token:

Specifies whether the server enables or disables cryptographic hardware and software support. The SOAP connector does not use hardware cryptography.

| | |
|-------------------|---------------------|
| Data type: | Boolean |
| Default: | Disabled |
| Range: | Enabled or Disabled |

V3 Timeout:

Specifies the length of time that a browser can reuse a System SSL Version 3 session ID without renegotiating encryption keys with the server.

The repertoires that you define for a server require the same V3 timeout value.

| | |
|------------------|------------|
| Data type | integer |
| Default | 100 |
| Range | 1 to 86400 |

Provider:

Refers to a package that supplies a concrete implementation of a subset of the cryptography aspects of the Java Security API.

If you select the first button, select a provider from the menu. WebSphere Application Server has the IBMJSSE predefined provider and the IBMJSSEFIPS predefined provider. IBMJSSEFIPS is a version of the IBMJSSE provider that is undergoing Federal Information Processing Standard (FIPS) certification. If you select the second option, enter a custom provider. For a custom provider, you first must enter the cipher suites through Custom Properties under Additional Properties, Cipher suites and protocol values depend on the Provider.

| | |
|------------------|------------|
| Data type | integer |
| Default | 100 |
| Range | 1 to 86400 |

Protocol:

Specifies the SSL protocol that is used.

If you are using a FIPS-approved JSSE such as IBMJSSEFIPS, you must select a TLS protocol. Because the FIPS-approved JSSE providers are not backwards-compatible, a server that uses the TLS protocol cannot communicate with a client that uses an SSL protocol.

Secure Sockets Layer settings for custom properties:

Use this page to configure additional Secure Sockets Layer (SSL) settings for a defined alias.

To view this administrative console page, click **Security** > **SSL** > *alias_name* > **Custom properties**.

Custom Properties:

Specifies the name-value pairs that you can use to configure additional SSL settings beyond those available in the `com.ibm.ssl.protocol` administrative interface.

This value is the SSL protocol used (including its version). The possible values are SSL, SSLv2, SSLv3, TLS, or TLSv1. The default value, SSL, is backward-compatible with the other SSL protocols.

com.ibm.ssl.keyStoreProvider

The name of the key store provider to use. Specify one of the security providers listed in your `java.security` file, which has a keystore implementation. The default value is IBMJCE.

com.ibm.ssl.keyManager

The name of the key management algorithm to use. Specify any key

management algorithm that is implemented by one of the security providers listed in your `java.security` file. The default value is `IbmX509`.

com.ibm.ssl.trustStoreProvider

The name of the trust store provider to use. Specify one of the security providers listed in your `java.security` file, which has a truststore implementation. The default value is `IBMJCE`.

com.ibm.ssl.trustManager

The name of the trust management algorithm to use. Specify any trust management algorithm that is implemented by one of the security providers listed in your `java.security` file. The default value is `IbmX509`.

com.ibm.ssl.trustStoreType

The type or format of the truststore file. The possible values are `JKS`, `PKCS12`, `JCEK`. The default value is `JKS`.

com.ibm.ssl.enabledCipherSuites

The list of cipher suites to enable. By default, this is not set and the set of cipher suites used is determined by the value of the security level (high, medium, or low). A cipher suite is a combination of cryptographic algorithms used for an SSL connection. Enter a space-separated list of any of the following cipher suites:

- `SSL_RSA_WITH_RC4_128_MD5`
- `SSL_RSA_WITH_RC4_128_SHA`
- `SSL_RSA_WITH_DES_CBC_SHA`
- `SSL_RSA_WITH_3DES_EDE_CBC_SHA`
- `SSL_DHE_RSA_WITH_DES_CBC_SHA`
- `SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA`
- `SSL_DHE_DSS_WITH_DES_CBC_SHA`
- `SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA`
- `SSL_RSA_EXPORT_WITH_RC4_40_MD5`
- `SSL_RSA_EXPORT_WITH_DES40_CBC_SHA`
- `SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5`
- `SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA`
- `SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA`
- `SSL_RSA_WITH_NULL_MD5`
- `SSL_RSA_WITH_NULL_SHA`
- `SSL_DH_anon_WITH_RC4_128_MD5`
- `SSL_DH_anon_WITH_DES_CBC_SHA`
- `SSL_DH_anon_WITH_3DES_EDE_CBC_SHA`
- `SSL_DH_anon_EXPORT_WITH_RC4_40_MD5`
- `SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA`

Data type: String

Cryptographic token:

Specifies information about the cryptographic tokens related to SSL support.

A cryptographic token is a hardware or software device that has a built-in keystore implementation. Document the exact values for the following fields in the found in the literature of your supported cryptographic device.

Configuring Federal Information Processing Standard Java Secure Socket Extension files

The Federal Information Processing Standard (FIPS)-approved Java Secure Socket Extension (JSSE) provider has increased data encryption capabilities.

FIPS-approved JSEE providers support Data Encryption Standard (DES) or Triple

DES with at least 56-bits of encryption. Although this additional encryption capability is available, you must use Transport Layer Security (TLS) and not Secure Sockets Layer (SSL) as FIPS-approved JSSE files are not backwards-compatible and SSL is not FIPS-approved. If the server uses TLS, a client using SSL cannot communicate with the server. Thus, use FIPS-approved JSSE providers if your servers and clients are using WebSphere Application Server, Version 5.0.2 or later as this version supports FIPS.

Attention: The IBMJSSEFIPS and IBMJCEFIPS are undergoing FIPS 140-2 certification. For more information on the FIPS certification process and to check the status of the IBM submission, see the Cryptographic Module Validation Program FIPS 140-1 and FIPS 140-2 Pre-validation List Web site.

If you create your own encryption configurations and enable FIPS, you must add a FIPS-approved JSSE to all of your server and client configurations.

Important: **5.1+** Although WebSphere Application Server supports the IBM Federal Information Processing Standard-approved Java Secure Socket Extension (IBMJSSEFIPS), IBMJSSEFIPS is not supported on the HP-UX platform.

To configure the WebSphere Application Server to use IBMJSSEFIPS and IBMJCEFIPS providers, complete the following steps using the administrative console:

1. Click **Security > Global Security**.
2. Select the **Use FIPS** check box and click **OK**. IBMJCEFIPS is enabled. However, IBMJSSEFIPS is not configured until you complete the remaining steps.
3. Click **Security > SSL**.
4. Click the name of your SSL configuration or click **New** to create a new configuration. For more information on SSL configurations, see “Creating a Secure Sockets Layer repertoire configuration entry” on page 285.
5. Select **High** from the Security Level menu. This action sets the encryption strength to 56-bits and higher.
6. Indicate which JSSE FIPS provider to use. Do one of the following actions:
 - Select **IBMJSSEFIPS** from the Provider menu and select Predefined JSSE provider. For a list of providers that were previously configured, click **Custom Properties** under Additional Properties.
 - Type the name of your custom JSSE FIPS provider and select Custom JSSE provider. To create a custom JSSE FIPS provider, click **Custom Properties > New** under Additional Properties. After configuring your custom FIPS-approved provider, return to the SSL Configuration Repertoires panel for your SSL configuration and enter the name in the Provider field.
7. Select the **TLS** or **TLSV1** option from the Protocol menu. To use a FIPS-approved JSSE, you must choose either the **TLS** or **TLSV1** option. SSL protocol is not FIPS-approved. After you select the protocol, the corresponding custom property value is updated for `com.ibm.ssl.protocol`. You can view this updated property value under Custom Properties after you click **Apply** or **OK**.
8. Click **OK**.
9. If you have a Java client that must access enterprise beans, modify the `install_dir>/properties/sas.client.props` file to comment out the SSL protocol and add the Transport Layer Security (TLS) protocol. To change the

protocol to TLS, make the following changes to the *install_dir>/properties/sas.client.props* file:

```
#com.ibm.ssl.protocol=SSL  
com.ibm.ssl.protocol=TLS
```

10. If the server uses a FIPS-approved provider for the CSiv2/SAS protocol, add IBMJSSEFIPS as the contextProvider and TLS as the protocol to the *install_dir/properties/sas.client.props* file on the application client. In the *install_dir/properties/sas.client.props* file, add the following information:

```
com.ibm.ssl.contextProvider=IBMJSSEFIPS  
com.ibm.ssl.protocol=TLS
```

11. If the server-side SOAP connector configuration uses a FIPS-approved IBMJSSEFIPS provider, add com.ibm.fips.jsse.JSSESocketFactory as the provider and IBMJSSEFIPS as the contextProvider in the *install_dir/properties/soap.client.props* file on the administrative client. In the *install_dir/properties/soap.client.props* file, add the following information:

```
ssl.SocketFactory.provider=com.ibm.fips.jsse.JSSESocketFactory  
com.ibm.ssl.contextProvider=IBMJSSEFIPS
```

12. Verify that a FIPS-approved configuration is specified correctly throughout the administrative console. Verify the configuration settings in the following panels:
 - Click **Servers > Application Servers > *server_name***. Under Additional properties, click **Administration Services > JMX Connectors > SOAPConnector > Custom Properties > sslConfig**.
 - Click **Servers > Application Servers > *server_name***. Under Additional properties, click **Web Container > HTTP Transport**.
 - Click **Environment > Virtual Hosts > *host_name***. Under Additional properties, click **Host Aliases > <*alias_name*>**.
 - Click **Applications > Enterprise Applications > *application_name***. Under Additional properties, click **Map virtual hosts for web modules**.
 - Click **Security > User Registries > LDAP**.
 - Click **Enterprise Applications > *application_name***. Under Related Items, click **Web Module > *URI_file_name* > Web Services: Client Security Bindings**. Verify the configuration settings listed under **HTTP Basic Authentication** and **HTTP SSL Authentication**.

After completing these steps, a FIPS-approved JSSE provides increased encryption capabilities. However, when you use FIPS-approved providers, consider the following points:

- By default, Microsoft Internet Explorer Version 5.5 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** checkbox.
- Netscape Version 4.7.x and earlier versions might not support TLS.
- IBM Directory Server Version 4.1 and earlier versions do not support TLS.
- If you select **IBMJSSEFIPS** from the Provider menu before changing the Security Level to High and the Protocol menu to TLS or TLSV1, WebSphere Application Server changes the Security Level and Protocol menu options automatically. However, if you change the Provider menu option from IBMJSSEFIPS to IBMJSSE, you must manually change the Protocol option to the correct setting. The setting does not change automatically because IBMJSSE supports both SSL and TLS.

- If you have an administrative client that uses a SOAP connector and you enable FIPS, add the following lines to the `install_dir/properties/soap.client.props` file:

```
ssl.SocketFactory.provider=com.ibm.fips.jsse.JSSESocketFactory
com.ibm.ssl.contextProvider=IBMJSSEFIPS
```

- When you select the **Use FIPS** check box on the **Security > Global Security** panel, the LTPA token format is not backwards-compatible with prior releases of WebSphere Application Server. However, you can continue to use the LTPA keys configured using a previous version of WebSphere Application Server.

Attention: If you select **USE FIPS** on the Global Security panel and select an SSL configuration on the SSL Configuration Repertoires panel, the following error message is displayed at the top of the Global Security panel:

The security policy is set to use only FIPS-approved cryptographic algorithms. However at least one SSL configuration may not be using a FIPS-approved JSSE provider. FIPS-approved cryptographic algorithms may not be used in those cases.

Attention: If you use the FIPS-approved JSSE provided with WebSphere Application Server, you must choose **IBMJSSEFIPS** from the Provider menu on the SSL Configuration Repertoires panel. Otherwise, the following message is displayed at the top of the panel:

"Use FIPS" is enabled, but the SSL provider is not IBMJSSEFIPS.
FIPS approved cryptographic algorithms may not be used.

Digital certificates

Certificates provide a way of authenticating users. Instead of requiring each participant in an application to authenticate every user, third-party authentication relies on the use of digital certificates.

A digital certificate is equivalent to an electronic ID card. It serves two purposes:

- Establishes the identity of the owner of the certificate
- Distributes the owner's public key

Certificates are issued by trusted parties, called *certificate authorities (CAs)*. These authorities can be commercial ventures or they can be local entities, depending on the requirements of your application. Regardless, the CA is trusted to adequately authenticate users before issuing certificates. A CA issues certificates with digital signatures. When a user presents a certificate, the recipient of the certificate validates it by using the digital signature. If the digital signature validates the certificate, the certificate is recognized as intact and authentic. Participants in an application only need to validate certificates; they do not need to authenticate users. The fact that a user can present a valid certificate proves that the CA has authenticated the user. The descriptor, *trusted third-party*, indicates that the system relies on the trustworthiness of the CAs.

Contents of a digital certificate

A certificate contains several pieces of information, including information about the owner of the certificate and the issuing CA. Specifically, a certificate includes:

- The distinguished name (DN) of the owner. A DN is a unique identifier, a fully qualified name including not only the common name (CN) of the owner but the owner's organization and other distinguishing information.
- The public key of the owner.

- The date on which the certificate was issued.
- The date on which the certificate expires.
- The distinguished name of the issuing CA.
- The digital signature of the issuing CA. (The message-digest function is run over all the preceding fields.)

The core idea of a certificate is that a CA takes the owner's public key, signs the public key with its own private key, and returns the information to the owner as a certificate. When the owner distributes the certificate to another party, it signs the certificate with its private key. The receiver can extract the certificate (containing the CA signature) with the owner's public key. By using the CA public key and the CA signature on the extracted certificate, the receiver can validate the CA signature. If it is valid, the public key used to extract the certificate is recognized as good. The owner signature is then validated, and if the validation succeeds, the owner is successfully authenticated to the receiver.

The additional information in a certificate helps an application decide whether to honor the certificate. With the expiration date, the application can determine if the certificate is still valid. With the name of the issuing CA, the application can check that the CA is considered trustworthy by the site.

A process that uses certificates must provide its personal certificate, the one containing its public key, and the certificate of the CA that signed its certificate, called a *signer certificate*. In cases where chains of trust are established, several signer certificates can be involved.

Requesting certificates

To get a certificate, send a certificate request to the CA. The certificate request includes:

- The distinguished name of the owner (the user for whom the certificate is requested).
- The public key of the owner.
- The digital signature of the owner.

The message-digest function is run over all these fields.

The CA verifies the signature with the public key in the request to ensure that the request is intact and authentic. The CA then authenticates the owner. Exactly what the authentication consists of depends on a prior agreement between the CA and the requesting organization. If the owner in the request is successfully authenticated, the CA issues a certificate for that owner.

Using certificates: Chain of trust and self-signed certificate

To verify the digital signature on a certificate, you must have the public key of the issuing CA. Because public keys are distributed in certificates, you must have a certificate for the issuing CA that is signed by the issuer. One CA can certify other CAs, so a chain of CAs can issue certificates for other CAs, all of whose public keys you need. Eventually, you reach a root CA that issues itself a self-signed certificate. To validate a user's certificate, you need certificates for all intervening participants, back to the root CA. Then you have the public keys you need to validate each certificate, including the user's.

A self-signed certificate contains the public key of the issuer and is signed with the private key. The digital signature is validated like any other, and if the certificate is

valid, the public key it contains is used to check the validity of other certificates issued by the CA. However, anyone can generate a self-signed certificate. In fact, you can probably generate self-signed certificates for testing purposes before installing production certificates. The fact that a self-signed certificate contains a valid public key does not mean that the issuer is really a trusted certificate authority. To ensure that self-signed certificates are generated by trusted CAs, such certificates must be distributed by secure means (hand-delivered on floppy disks, downloaded from secure sites, and so on).

Applications that use certificates store these certificates in a *keystore* file. This file typically contains the necessary personal certificates, its signing certificates, and its private key. The private key is used by the application to create digital signatures. Servers always have personal certificates in their keystore files. A client requires a personal certificate only if the client must authenticate to the server when mutual authentication is enabled.

To allow a client to authenticate to a server, a server keystore file contains the private key and the certificate of the server and the certificates of its CA. A client truststore file must contain the signer certificates of the CAs of each server to which the client must authenticate.

If mutual authentication is needed, the client keystore file must contain the client private key and certificate. The server truststore file requires a copy of the certificate of the client CA.

Digital signatures:

A *digital signature* is a number attached to a document. For example, in an authentication system that uses public-key encryption, digital signatures are used to sign certificates.

This signature establishes the following information:

- The integrity of the message: Is the message intact? That is, has the message been modified between the time it was digitally signed and now?
- The identity of the signer of the message: Is the message authentic? That is, was the message actually signed by the user who claims to have signed it?

A digital signature is created in two steps. The first step distills the document into a large number. This number is the *digest code* or *fingerprint*. The digest code is then encrypted, resulting in the digital signature. The digital signature is appended to the document from which the digest code was generated.

Several options are available for generating the digest code. WebSphere Application Server supports the MD5 message digest function and the SHA1 secure hash algorithm, but these procedures reduce a message to a number. This process is not encryption, but a sophisticated checksum. The message cannot regenerate from the resulting digest code. The crucial aspect of distilling the document to a number is that if the message changes, even in a trivial way, a different digest code results. When the recipient gets a message and verifies the digest code by recomputing it, any changes in the document result in a mismatch between the stated and the computed digest codes.

To stop someone from intercepting a message, changing it, recomputing the digest code, and retransmitting the modified message and code, you need a way to verify the digest code as well. To verify the digest code, reverse the use of the public and private keys. For private communication, it makes no sense to encrypt messages

with your private key; these keys can be decrypted by anyone with your public key. This technique can be useful for proving that a message came from you. No one can create it because no one else has your private key. If some meaningful message results from decrypting a document by using someone's public key, the decryption process verifies that the holder of the corresponding private key did encrypt the message.

The second step in creating a digital signature takes advantage of this reverse application of public and private keys. After a digest code is computed for a document, the digest code is encrypted with the sender's private key. The result is the digital signature, which is attached to the end of the message.

When the message is received, the recipient follows these steps to verify the signature:

1. Recomputes the digest code for the message.
2. Decrypts the signature by using the sender's public key. This decryption yields the original digest code for the message.
3. Compares the original and recomputed digest codes. If these codes match, the message is both intact and authentic. If not, something has changed and the message is not to be trusted.

Public key cryptography:

All encryption systems rely on the concept of a key. A key is the basis for a transformation, usually mathematical, of an ordinary message into an unreadable message. For centuries, most encryption systems have relied on what is called private-key encryption. Only within the last 30 years has a challenge to private-key encryption appeared: public-key encryption.

Private key encryption

Private-key encryption systems use a single key that is shared between the sender and the receiver. Both must have the key; the sender encrypts the message by using the key, and the receiver decrypts the message with the same key. Both must keep the key private to keep their communication private. This kind of encryption has characteristics that make it unsuitable for widespread, general use:

- Private key encryption requires a key for every pair of individuals who need to communicate privately. The necessary number of keys rises dramatically as the number of participants increases.
- The fact that keys must be shared between pairs of communicators means the keys must somehow be distributed to the participants. The need to transmit secret keys makes them vulnerable to theft.
- Participants can communicate only by prior arrangement. There is no way to send a usable encrypted message to someone spontaneously. You and the other participant must make arrangements to communicate by sharing keys.

Private-key encryption is also called *symmetric encryption*, because the same key is used to encrypt and decrypt the message.

Public key encryption

Public-key encryption uses a pair of mathematically related keys. A message encrypted with the first key must be decrypted with the second key, and a message encrypted with the second key must be decrypted with the first key.

Each participant in a public-key system has a pair of keys. The private key is kept secret. The other key is distributed to anyone who wants it; this key is the public key.

To send an encrypted message to you, the sender encrypts the message by using your public key. When you receive the message, you decrypt it by using your private key. To send a message to someone, you encrypt the message by using the recipient's public key. The message can be decrypted with the recipient's private key only. This kind of encryption has characteristics that make it very suitable for general use:

- Public-key encryption requires only two keys per participant. The increase in the total number of keys is less dramatic as the number of participants increases, compared to private-key encryption.
- The need for secrecy is more easily met. Only the private key needs to be kept private and because it does not need to be shared, the private key is less vulnerable to theft in transmission than the shared key in a private-key system.
- Public keys can be published, which eliminates the need for prior sharing of a secret key before communication. Anyone who knows your public key can use it to send you a message that only you can read.

Public-key encryption is also called *asymmetric encryption*, because the same key cannot be used to encrypt and decrypt the message. Instead, one key of a pair is used to undo the work of the other. WebSphere Application Server uses the Rivest Shamir Adleman (RSA) public and private key-encryption algorithm.

With private-key encryption, you have to be careful of stolen or intercepted keys. In public-key encryption, where anyone can create a key pair and publish the public key, the challenge is in verifying that the owner of the public key is really the person you think it is. Nothing prevents a user from creating a key pair and publishing the public key under a false name. The listed owner of the public key cannot read messages encrypted with that key because the owner does not have the private key. If the creator of the false public key can intercept these messages, that person can decrypt and read messages intended for someone else. To counteract the potential for forged keys, public-key systems provide mechanisms for validating public keys and other information with digital signatures and digital certificates.

Managing digital certificates

Secure Sockets Layer (SSL) connections rely on the existence of *digital certificates*. A digital certificate reveals information about its owner, including their identity. During the initialization of an SSL connection, the server must present its certificate to the client for the client to determine the server identity. The client can also present the server with its own certificate for the server to determine the client identity. SSL is therefore, a means of propagating identity between components. Refer to *Configuring Secure Socket Layer* and *Configuring a Secure Sockets Layer repertoire configuration entry*.

A client can trust the contents of a certificate if that certificate is digitally signed by a trusted third party. A Certificate Authority (CA) acts as a trusted third party and signs certificates on the basis of its knowledge of the certificate requestor. Different approaches for generating certificates include:

1. Use the supplied key management utility. Refer to *Starting the key management utility (iKeyman)*. There are two options for creating a new certificate.

- a. Request that a CA generates the certificates on your behalf. The CA creates a new certificate, digitally signs it, and delivers it to the requester. Popular Web browsers are preconfigured to trust certificates that are signed by certain CAs. No further client configuration is necessary for a client to connect to the server through an SSL connection. Therefore, CA signed certificates are useful where configuration for each and every client that accesses the server is impractical. Refer to Requesting a CA-signed personal certificate, Creating a certificate signing request (CSR), Receiving a CA-signed personal certificate, and Extracting a public certificate for use in a truststore.
 - b. Generate a self-signed certificate. . This option might be the quickest and require the fewest details to create the certificate. However, the certificate is not signed by a CA. Any client that connects to this server over an SSL connection needs configuration to trust the signer of this certificate. Therefore, self-signed certificates are only useful when you can configure each of the clients to trust the certificate. It is possible in some cases to present a self-signed certificate to an untrusting client. In some Web browsers, when the certificate is received and does not match any of those listed in the client trust file, a prompt appears asking if the certificate should be trusted for the connection and added to the trust file. Refer to Create a keystore file, Create a truststore file, Adding keystore files, Adding truststore files, Create a self-signed personal Certificate, and Importing a signer certificate
 - c. Configure server-side options. The WebSphere Application Server stores the keystore information in the repository and the keystore files are referred to in the `security.xml` file. Therefore, complete all server-side configuration through the administration console. For Java clients, refer to Configuring SSL for Java Client Authentication.
2. Use the command line Java utility called *keytool*. With *keytool*, you can create a private and public self-signed certificate key pair. For this example, the first user is `cn=rocaj`.
 - a. Specify **RSA** for the private key to ensure that the *MD5 with RSA* signature algorithm is used. Not all Web browsers support the *DSA* cryptograph algorithm, which is the default when RSA is not specified. Set a password of at least six characters to protect the private key. Finally, specify the keystore file and keystore password (the option is `storepass`):


```
#keytool -genkey -keyalg RSA -dname "cn=rocaj, ou=users, ou=uk, dc=internetchaos,dc=com" -alias rocaj -keypass websphere -keystore testkeyring.jks -storepass websphere
```

The previous three lines of code belong on one line, but were split onto three lines due to the width of the page.
 - b. Create the second private and public self-signed certificate key pair in the same manner for the user `cn=amorv`.


```
#keytool -genkey -keyalg RSA -dname "cn=amorv, ou=users, ou=uk, dc=internetchaos, dc=com" -alias amorv -keypass websphere -keystore testkeyring.jks -storepass websphere
```

The previous three lines of code belong on one line, but were split onto three lines due to the width of the page.

Now the keystore `testkeyring.jks` contains two self-signed certificates with the owner being the same as the issuer for each certificate.
 - c. Verify the integrity and authenticity of the certificates by getting each certificate signed by the certificate authority.

- Generate the Certificate Signing Request, CSR-1 (for the first user `cn=rocaj`).

```
#keytool -v certreq -alias rocaj -file rocajReq.csr -keypass websphere
-keystore testkeyring.jks -storepass websphere
```

The previous two lines of code belong on one line, but were split onto two lines due to the width of the page.

- Generate the CSR-2 (for the second user `cn=amorv`).

```
#keytool -v -certreq -alias amorv -file amorvReq.csr -keypass websphere
-keystore testkeyring.jks -storepass websphere
```

The previous two lines of code belong on one line, but were split onto two lines due to the width of the page.

- d. Use the free Test SSL certificate program offered by Thawte Consulting to sign the Certificate Signing Requests (CSRs) for this example. In each case, select the **Custom Cert** option and set the certificate format to use the default for your kind of certificate. The example also selects the **Generate an X.509v3 Certificate** option and saves the two resulting files as *rocajRes.arm* and *amorvRes.arm*, respectively.
- e. Import the CA trusted root certificate into the keystore. Copy and paste the Thawte test root certificate in BASE64-encoded ASCII data format to a file called *ThawteTestCA.arm*. Add the test root CA certificate into the keystore file with the following command:

```
#keytool -import -alias "Thawte Test CA Root" -file ThawteTestCA.arm
-keystore testkeyring.jks -storepass websphere
```

The previous two lines of code belong on one line, but were split onto two lines due to the width of the page.

- f. Import the two certificate responses from the CA into the keystore file using the same alias name that was first given to the self-signed certificates. In this example, these alias names are *rocaj* and *amorv* respectively. Using an alternative alias name generates a new signer certificate and not a personal certificate chain.
 - Import the certificate response -1 (for the first user `cn=rocaj`).

```
#keytool -import -trustcacerts -alias rocaj -file rocajRec.arm
-keystore testkeyring.jks -storepass websphere.
Certificate reply was installed in keystore
```

The previous three lines of code belong on one line, but were split onto three lines due to the width of the page.

- Import the certificate response -2 (for the second user `cn=amorv`).

```
#keytool -import -trustcacerts -alias amorv -file amorvRec.arm
-keystore testkeyring.jks -storepass websphere.
Certificate reply was installed in keystore
```

The previous three lines of code belong on one line, but were split onto three lines due to the width of the page.

- g. Launch the JSSE `ikeyman` utility, which supports the PKCS12 format and the private key exporting associated with any certificate (the public key is also exported).
- h. Open the *testkeyring.jks* keystore file and select the first certificate from the **Personal Certificates** menu.

- i. Click **Export** and name the file, *rocajprivate.p12*. Export the second personal certificate and name it *amoroprivate.p12*.
 - j. Verify that the same root certificate of the authenticating CA is installed as a trusted authority in the browser.
 - k. To install either of the personal certificates into Netscape Communicator, click **Communicator > Tools > Security Info > Certificates > Yours**. Use the **Import a Certificate** option.
 - l. Enter a password or PIN for the communicator certificate database, when you attempt to import the certificate. Enter the password used when first initializing your certificate database. Enter the password protecting the PKCS#12 certificate file, as set when you exported the personal private and public certificate key pair in iKeyman.
 - m. Click **Verify** to check integrity and validity of the certificate. If you did not install the root CA certificate, your certificate fails the verification.
 - n. Verify that you modified your Web server to support client side certificate requests.
 - o. Go to the following URL: `https://<server_name>/snoop`; the Web browser prompts you to select a personal certificate when accessing a resource protected by the *SSLClientAuth* directive.
 - p. Select the HTTPS information displayed by the snoop servlet; you see the certificate SubjectDN matching the following: **Subject: CN=amorv, OU=users, OU=uk, DC=internetchaos, DC=com**.
3. Refer to Create a Secure Sockets Layer repertoire configuration entry to create a new SSL definition entry for WebSphere Application Server using the administrative console. Once a keystore file is configured, either by creating a self-signed certificate or by creating a certificate request and importing the reply, you can configure WebSphere Application Server to use the certificates. The product uses the certificates to establish a secure connection with a client through SSL.
 4. Set up the appropriate components to use the newly-defined SSL configuration. To ensure a secure connection, configure some non-WebSphere components, such as a Web server. A digital certificate is created for each component. The WebSphere Application Server owns a certificate and the Web server owns another certificate. Refer to Configuring an IBM HTTP Server for Secure Sockets Layer mutual authentication.

Setting up SSL communication between the Web browser and WebSphere Application Server. Using digital signatures, you can communicate securely from the Web browser through the Web server to WebSphere Application Server.

Once you finish configuring security, perform the following steps to save, synchronize, and restart the servers:

1. Click **Save** in the administrative console to save any modifications to the configuration.
2. Synchronize the configuration with all node agents (Network Deployment only).
3. Once synchronized, stop all servers and restart them.

Starting the key management utility (iKeyman):

It is recommended to read the documentation located at *install_root/web/docs/ikeman/ikmuserguide.pdf* for further information.

WebSphere Application Server provides a graphical tool, the key management utility (iKeyman), for managing keys and certificates. With the key management utility, you can:

- Create a new key database
- Create a self-signed digital certificate
- Add certificate authority (CA) roots to the key database as a signer certificate
- Request and receive a digital certificate from a CA

To start the key management utility, complete the following steps:

1. Move to the *install_root/bin* directory.
2. Issue one of the following commands:
 - On Windows systems, *ikeyman.bat*
 - On UNIX systems, *ikeyman.sh*

A graphical user interface of the key management utility appears.

Manage keys and digital certificates.

Creating a keystore file:

The keystore file is a key database file that contains both public keys and private keys. Public keys are stored as signer certificates while private keys are stored in the personal certificates. The keys are used for a variety of purposes, including authentication and data integrity. You can use both the key management utility (iKeyman) and the keytool utility to create keystore files.

Read the documentation located at *install_root/web/docs/ikeyman/ikmuserguide.pdf* for further information.

1. Start the iKeyman utility, if it is not already running.
2. Open a new key database file by clicking **Key Database File > New** from the menu bar.
3. Select the Key Database Type: JKS (default), PKCS12, and JCEK. This is the *key file format* (or the value of *com.ibm.ssl.keyStoreType* property in the *sas.client.props* file) when you configure the SSL setting for your application.
4. Type in the file name and location. The full path of this key database file is used as the *key file name* (or the value of the *com.ibm.ssl.keyStore* property in the *sas.client.props* file) when you configure the SSL setting for your application.
5. Click **OK** to continue.
6. Then, type in password to restrict access to the file. This password is used as the *key file password* (or the value of *com.ibm.ssl.keyStorePassword* property in the *sas.client.props* file) when you configure the SSL setting for your application. Do not set an expiration date on the password or save the password to a file; you must then reset the password when it expires or protect the password file. This password is used only to release the information stored by the key management utility during run time.
7. Click **OK** to continue. The tool displays all of the available default signer certificates. These certificates are the public keys of the most common certificate authorities (CAs). You can add, view or delete signer certificates from this panel.

A new SSL keystore file is created.

Prepare keystore files for an SSL connection.

Specify the keystore file in the configuration of WebSphere Application Server. Create a truststore if one does not yet exist.

Creating self-signed personal certificates:

A self-signed personal certificate is a temporary digital certificate you issue to yourself, acting as the certificate authority (CA). Creating a self-signed certificate creates a private key and a public key within the key database file. The self-signed certificate is created in a keystore file and it is useful when you develop and test your application. You can also create a self-signed personal certificate from your cryptographic token device.

If you want to create a self-signed certificate for a keystore, you must have already created the keystore file. You can later extract the public key and add the key as a signer certificate to other truststore files.

Read the file *install_root/web/docs/ikeman/ikmuserguide.pdf* for further information about how to create a self-signed personal certificate within a key database file.

1. Start the key management utility, if it is not already running.
2. Click **New Self-Signed** from the tool bar or click **Create > New Self-Signed Certificate**.
3. Select the **X509** version and the key size that suits your application.
4. Enter the appropriate information for your self-signed certificate:

Key Label

Give the certificate a key label, which is used to uniquely identify the certificate within the keystore file. If you have only one certificate in each keystore file, you can assign any value to the label. However, it is good practice to use a unique label related to the server name.

Common Name

Enter the common name. This name is the primary, universal identity for the certificate; it should uniquely identify the principal that it represents. In a WebSphere environment, certificates frequently represent server principals, and the common convention is to use common names of the form *host_name* and *server_name*. The common name must be valid in the configured user registry for the secured WebSphere environment.

Organization

Enter the name of your organization.

Optional fields

Enter the organization unit (a department or division), location (city), state and province (if applicable), zip code (if applicable), and select the two-letter identifier of the country in which the server belongs. For a self-signed certificate, these fields are optional. However, commercial CAs might require them.

Validity period

Specify the lifetime of the certificate in days, or accept the default.

5. Click **OK**.

Your key database file now contains a self-signed personal certificate.

Create a self-signed test certificate for testing purposes.

If you need a test certificate signed by a certificate authority, follow the procedure in *Creating a certification request*.

Requesting certificate authority-signed personal certificates:

In a production environment, use a personal certificate signed by a certificate authority (CA). The principal or the owner of the CA-signed personal certificate is authenticated by a CA when the CA signs the principal certificate. Since the certificate authorities (CAs) keep their private keys secure, the signed certificate is more trustworthy than a self-signed certificate. Certificate authorities are entities that issue valid certificates for other entities. Well-known CAs include VeriSign, Entrust, and GTE CyberTrust. You can request a test certificate or a production certificate from some of the CAs like VeriSign.

The authentication process by a CA can take time. Commercial CAs often require up to a week to complete their authentication process. Even on-site CAs can take several minutes, if not hours, or even days, to complete their authentication process. Therefore, you must plan for the certificates that you need.

Considering the following points when you plan for the CA-signed certificate:

- On the certificate signing request that you send to the CA, specify the common name for the certificate. The common name is the primary, universal identity for the certificate. It should uniquely identify the principal that it represents. Verify that the common name is valid in the configured user registry for the WebSphere domain.
 - Check the formatting of the address fields that your CA requires when planning the address for a certificate request.
1. Create and send a certificate signing request (CSR) to the CA.
 2. Visit the CA Web site and follow the instructions to request a test or production certificate.

Once the request is accepted, the certificate authority verifies your identity and finally issues a signed certificate to you. The certificate is usually sent through e-mail.

Request a production certificate from a trusted CA for the production WebSphere Application Server environment.

Once you receive the e-mail from the CA, follow the instructions to store your signed certificate as a file. Receive or store the certificate into the keystore file as a personal certificate.

Creating certificate signing requests:

To obtain a certificate from a certificate authority, submit a certificate signing request (CSR) using the key management utility (iKeyman). You can request either production or test certificates from a CA with a CSR. With the key management utility, generating a certificate signing request also generates a private key for the application for which the certificate is requested. The private key remains in the application keystore file, so it stays private. The public key is included in the certificate requested.

Read the file *install_root/web/docs/ikeyman/ikmuserguide.pdf* for further information about how to create a certificate signing request from a key database file.

1. Start the key management utility, if it is not already running.
2. Open the key database file from which you want to generate the request.
3. Type the password and click **OK**.

4. Click **Create > New Certificate Request**. The Create New Key and Certificate Request window displays.
5. Type a **Key Label**, a **Common Name**, and **Organization**; and select a Country. For the remaining fields, accept the default value, type a value, or select new values. The common name must be valid in the configured user registry for the secured WebSphere environment.
6. Type in a name for the file, such as certreq.arm.
7. Click **OK** to complete.
8. Send the certreq.arm file to the certificate authority (CA) following the instructions from the CA Web site for requesting a new certificate.

The Personal Certificate Requests list shows the key label of the new digital certificate request you just created. Send the file to a CA to request a new digital certificate, or cut and paste the request into the request forms of the CA Web site.

You need to request a certificate authority-signed digital certificate for your secure WebSphere domain.

Once you submit the certificate signing request, wait for the CA to accept the request. After the CA has verified your identity, it sends back the signed certificate usually through e-mail. Receive the signed certificate back to the keystore file from which you generated the CSR.

Receiving certificate authority-signed personal certificates:

Once the certificate signing request (CSR) is accepted, a certificate authority (CA) processes the request and verifies your identity. Once approved, the CA sends the signed certificate back through e-mail. Store the signed certificate in a keystore database file. This procedure describes how to receive the CA-signed certificate into a keystore file using the key management utility (iKeyman). You use this utility the same way for both test certificates and production certificates. The primary difference between the two certificate types is the amount of time it takes for the CA to authenticate the principal your certificate represents. Test certificates are authenticated automatically based on some simple edit checks and returned to you within a few hours. Production certificates may take several days or a week to authenticate and return to you. If the CSR request is made for the cryptographic token, the certificate must be received into that token. If the request is made for the secondary key database of the token, the certificate must be received into that database.

Receive the signed certificate from the CA through e-mail. Follow the instructions from the CA to store the certificate into a file. Read the file *install_root/web/docs/ikeyman/ikmuserguide.pdf* for further information about how to receive a personal certificate into a key database file from the CA.

1. Start IKeyman, if it is not already running.
2. Open the key database file from which you generated the request.
3. Type the password and click **OK**.
4. Select **Personal Certificates** from the pull-down list.
5. Click **Receive**.
6. Click **Data type** and select the data type of the new digital certificate, such as Base64-encoded ASCII data. Select the data type that matches the CA-signed certificate. If the CA sends the certificate as part of an E-mail message, you may first need to cut and paste the certificate into a separate file.

7. Type the certificate file name and location for the new digital certificate, or click **Browse** to locate the CA-signed certificate.
8. Click **OK**.
9. Type a label for the new digital certificate and click **OK**.

The personal certificate list now displays the label you just gave for the new CA-signed certificate.

Needs digital certificate to support SSL for security over the WebSphere domain.

Once the CA-signed certificate is successfully received, you can extract or export the public key of the certificate to a file for distribution to the network.

Extracting public certificates for truststore files:

Use this procedure to extract a public certificate, which includes its public key, from a keystore file. If a target truststore file already contains the signer certificate of the certificate authority (CA) that signed the certificate, you do not need to extract and add the certificate to the target truststore file. However, in general, you need to complete this procedure for a self-signed certificate.

Extracting a certificate from one keystore file and adding it to a truststore file is not the same as exporting the certificate and then importing it. Exporting a certificate copies all the certificate information, including its private key, and is normally only used if you want to copy a personal certificate into another keystore file as a personal certificate.

If a certificate is self-signed, extract the certificate and its public key from the keystore file and add it to the target truststore file.

If a certificate is CA-signed, verify that the CA certificate used to sign the certificate is listed as a signer certificate in the target truststore file. The keystore file must already exist and contain the certificate to be extracted.

Read the file *install_root/web/docs/ikeyman/ikmuserguide.pdf* for further information about how to extract a public certificate from a key database file.

1. Start the key management utility (iKeyman), if it is not already running.
2. Open the keystore file from which the public certificate will be extracted.
3. Select **Personal Certificates**.
4. Click **Extract Certificate**.
5. Click **Base64-encoded ASCII data** under Data type.
6. Enter the **Certificate File Name** and **Location**.
7. Click **OK** to export the public certificate into the specified file.

A certificate file that contains the public key of the signed personal certificate is now available for the target truststore file.

Prepare truststore files for distributing the public keys to support the secure WebSphere domain using Secure Sockets Layer (SSL).

Once the keystore and truststore files are ready, make them accessible by specifying them in your client and server configurations.

Creating truststore files:

A truststore file is a key database file that contains the public keys for target servers. The public key is stored as a signer certificate. If the target uses a self-signed certificate, extract the public certificate from the server keystore file. Add the extracted certificate into the truststore file as a signer certificate. For a commercial certificate authority (CA), the CA root certificate is added. The truststore file can be a more publicly accessible key database file that contains all the trusted certificates.

Read the documentation located at *install_root/web/docs/ikeman/ikmuserguide.pdf* for further information.

1. Start the key management utility (iKeyman), if it is not already running.
2. Open a new key database file by clicking **Key Database File > New** from the menu bar.
3. Click the **Key Database Type**: JKS(Default), PKCS12, and JCEK. The key database type is the *trust file format* (or the value of the `com.ibm.ssl.trustStoreType` property in the `sas.client.props` file) when you configure the SSL setting for your application.
4. Type in the file name and location. The full path of this key database file is used as the *trust file name* (or the value of `com.ibm.ssl.trustStore` property in the `sas.client.props`) when you configure the SSL setting for your application.
5. Click **OK** to continue.
6. Type in a password to restrict access to the file. This password is used as the *trust file password* (or the value of the `com.ibm.ssl.trustStorePassword` property in the `sas.client.props` file) when you configure the SSL setting for your application. Do not set an expiration date on the password or save the password to a file. You must reset the password when it expires or protect the password file. This password is used only to release the information stored by the key management utility during run time.
7. Click **OK** to continue. The tool now displays all of the available default signer certificates. These are the public keys of the most common CAs. You can add, view or delete signer certificates from this screen.

A new SSL truststore file is created.

Prepare truststore files for an SSL connection.

Specify the truststore file in the configuration of WebSphere Application Server. Create a keystore file if one does not exist.

Importing signer certificates:

A *signer certificate* is the trusted certificate entry that is usually in a truststore file. You can import a certificate authority (CA) root certificate from the CA, or a public certificate from the self-signed personal certificate of the target into your truststore file, as a signer certificate.

Read the documentation located at *install_root/web/docs/ikeman/ikmuserguide.pdf* for further information.

1. Start the key management utility (iKeyman), if it is not already running.
2. Open the truststore file. The Password Prompt window displays.
3. Type the password and click **OK**.
4. Select **Signer Certificates** from the menu.
5. Click **Add**.

6. Click **Data type** and select a data type, such as Base64-encoded ASCII data. This data type must match the data type of the importing certificate.
7. Type a certificate file name and location for the CA root digital certificate or click **Browse** to select the name and location.
8. Click **OK**.
9. Type a label for the importing certificate.
10. Click **OK**.

The **Signer Certificates** field now displays the label of the signer certificate you just added.

Receive a CA root certificate or the public key from your secure target.

Map certificates to users:

Client-side certificates support access to secured resources from Web or Java clients. A client presents an X.509-compliant digital certificate to perform mutual authentication with a single sockets layer-enabled server. The product security run time attempts to map the certificate to a known user in the associated Lightweight Directory Access Protocol (LDAP) directory. If the certificate successfully maps to a user, then the holder of the certificate is regarded as the user in the registry and is authorized as this user.

After the single sockets layer-enabled server gets the client certificate, the server needs to map the certificate to a user. WebSphere Application Server supports two techniques for mapping certificates to entries in LDAP registries:

- By exact distinguished name
 - By matching attributes in the certificate to attributes of LDAP entries
1. Map by exact distinguished name (DN).

This approach attempts to map the distinguished name (DN) associated with the **Subject** field in the certificate to an entry in the LDAP directory. If the mapping is successful, the user is authenticated and is authorized according to the privileges granted to the identity in the LDAP directory.

The mapping is case insensitive. For example, the following two DNs match on a case-insensitive comparison:

```
"cn=Smith, ou=NewUnit, o=NewCompany, c=us"
"cn=smith, ou=newunit, o=NewCompany, c=US"
```

If a match is found, authentication succeeds; if no match is found, authentication fails.

2. Map by filtering certificate attributes.

This approach maps certificate attributes to attributes of entries in an LDAP directory. For example, you can specify that the common name (CN) attribute of the **Subject** field in the certificate must match the uid attribute of your LDAP entry. If the mapping is successful, the user is authenticated and is authorized according to the privileges granted to the identity in the LDAP directory.

If you are matching the Subject CN field in the certificate to the uid attribute of the LDAP entry, a certificate with the Subject DN "cn=Smith, ou=NewUnit, o=NewCompany, c=us" matches an LDAP user entry with uid=Smith.

To use this mapping technique, you must request certificate mapping and set up the certificate filter in the administrative console.

This specification extracts the CN field from the Subject attribute in the certificate (Smith) and creates a filter (user ID = Smith) from it. The LDAP directory is searched for a user entry that matches the filter. If an entry matches the filter, authentication succeeds.

Note: The search and match of the LDAP directory are based in part on how your LDAP directory is configured.

Troubleshooting secure sockets layer interoperability

The Secure Sockets Layer (SSL) protocol provides transport layer security: authenticity, integrity, and confidentiality, for a secure connection between a client and server in the WebSphere Application Server.

Secure Sockets Layer Interoperability issues might occur between different releases of WebSphere Application Server

| | |
|-----------------------------|---|
| Symptom | <p>In WebSphere Application Server Version 5.1, the secure sockets later protocol fails when you use the default key files and trust files between version 5.1 and previous releases of the server. For example, when you add a version 5.0.x node to the deployment manager on WebSphere Application Server Network Deployment Version 5.1.x that has security enabled, errors similar to the following might occur:</p> <pre>com.ibm.websphere.management.exception.ConnectorException: ADMC0016E: Can or javax.net.ssl.SSLHandshakeException: unknown certificate;</pre> |
| Explanation | <p>WebSphere Application Server Version 5.1 creates a dummy certificate with a later expiration date. In previous versions of WebSphere Application Server, the signer of the certificate does not exist in the trust files. Because the signer is missing, certificate errors occur when attempting to establish secure sockets layer (SSL) connections between WebSphere Application Server Version 5.1 and previous releases of the server.</p> |
| Recommended response | <p>Copy the dummy key files and trust files provided with WebSphere Application Server Version 5.1 into the previous releases for test interoperability. However, do not use these dummy certificates, key stores, or trust stores in a production environment. These dummy certificates are widely used and thus considered insecure for a production environment.</p> <p>When you apply an interim fix, the installer overwrites these files.</p> |

Changes to IBM Developer Kit for Java Technology Edition Version 1.4.x

WebSphere Application Server, Version 5.1 includes the IBM Developer Kit, Java Technology Edition Version 1.4.x, which contains changes to the IBM Developer Kit, Java Technology Edition Version 1.3.x. This document is intended to assist application developers and system administrators in understanding the changes.

Security packaging changes in IBM Developer Kit, Java Technology Edition Version 1.4.x

In IBM Developer Kit, Java Technology Edition Version 1.4.x, many of the security technologies have been included in the core of the IBM Developer Kit, Java

Technology Edition Version 1.4.x. Because of the packaging changes, we are supporting specific `java.security` configurations for each platform. This document discusses the impact these `java.security` configuration changes have on each platform.

Security providers for the Windows, Linux, and AIX platforms

The Windows, Linux, and AIX platforms use all of the IBM security provider implementations, which is similar to how IBM Developer Kit, Java Technology Edition Version 1.3.x shipped. Because the security technologies in IBM Developer Kit, Java Technology Edition Version 1.3.x, were not part of the core, these technologies were shipped in the `java/jre/lib/ext` directory and provided more flexibility in implementing the technologies. In IBM Developer Kit, Java Technology Edition Version 1.4.x, the security frameworks are located in the `java/jre/lib/security.jar` file, which is loaded by the boot classloader. The providers might not be flexible.

The following list shows the providers and sequence of how these providers are supported on the Windows, Linux, and AIX platforms. Add any additional providers at the end of this list of providers. The IBMJSSE and IBMJSSEFIPS providers are the only providers supported on these platforms.

```
security.provider.1=com.ibm.crypto.provider.IBMJCE
security.provider.2=com.ibm.jsse.IBMJSSEProvider
security.provider.3=com.ibm.security.jgss.IBMJGSSProvider
security.provider.4=com.ibm.security.cert.IBMCertPath
security.provider.5=com.ibm.crypto.pkcs11.provider.IBMPKCS11
```

Security providers for the Sun Solaris environment

In the Sun Solaris environment, by default, we are using the IBM JSSE framework classes. These classes enable you to plug-in the IBMJSSE and IBMJSSEFIPS providers. The following list shows the default provider lists for the Sun Solaris environment. Add any additional providers to the end of this list.

```
security.provider.1=com.ibm.security.cert.IBMCertPath
security.provider.2=com.ibm.security.jgss.IBMJGSSProvider
security.provider.3=sun.security.provider.Sun
security.provider.4=com.ibm.crypto.provider.IBMJCE
security.provider.5=com.ibm.jsse.IBMJSSEProvider
security.provider.6=com.ibm.crypto.pkcs11.provider.IBMPKCS11
```

Security providers for the HP-UX platform

On the HP-UX platform, you must use the SunJSSE and the SunJCE providers due to license restrictions. The IBMJSSE and IBMJSSEFIPS providers are not supported on the HP-UX platform because of a lack of flexibility in plugging into the Sun JSSE framework. Any code that specifically uses the IBMJSSE provider within Java Secure Socket Extension (JSSE) `getInstance` methods does not work on the HP-UX platform. To get the default provider, you can call `getInstance`

methods without explicitly specifying the provider. The following provider list must be used in the `java.security` file on the HP-UX platform. Add additional providers after those listed in the following list.

Attention: Some functions that worked using the IBMJSSE provider might not work using the SunJSSE provider including hardware cryptographic token configurations and certificate alias selection.

```
security.provider.1=com.ibm.security.jgss.IBMJGSSProvider
security.provider.2=com.ibm.security.cert.IBMCertPath
security.provider.3=sun.security.provider.Sun
security.provider.4=com.sun.crypto.provider.SunJCE
security.provider.5=com.sun.net.ssl.internal.ssl.Provider
security.provider.6=com.sun.rsa.jca.Provider
security.provider.7=com.ibm.crypto.provider.IBMJCE
security.provider.8=com.ibm.crypto.pkcs11.provider.IBMPKCS11
```

Changes to the CertPath API package name

In IBM Developer Kit, Java Technology Edition Version 1.3.x, the package for CertPath APIs was `javax.security.cert.*`. However, in IBM Developer Kit, Java Technology Edition Version 1.4.x, the package has changed to `java.security.cert.*`. While your applications might still work using `javax.security.cert.*` due to the `oldcertpath.jar` packaged in `${WAS_INSTALL_ROOT}/java/jre/lib/ext/oldcertpath.jar` file, change your applications to use the new package name for CertPath from this point forward. In this release, either package name should work, but it is recommended that you use the correct package, which is `java.security.cert.*`.

Known problems with IBM Developer Kit, Java Technology Edition Version 1.4.x

For a list of known problems with the various platforms related to the IBM Developer Kit, Java Technology Edition Version 1.4.x changes, please review the release notes for WebSphere Application Server, Version 5.1.

Cryptographic token support

A *cryptographic token* is a hardware or software device with a built-in key store implementation. The cryptographic device is used to manage certificates stored on the cryptographic tokens (also known as *smartcards*).

Both cryptographic accelerators, where the cryptographic hardware device has no persistent key storage, and secure cryptographic hardware, where a cryptographic token generates and securely stores the private key used for Secure Sockets Layer (SSL) key exchange, are supported in the product.

The following token types are supported:

- PKCS#7
- PKCS#11
- PKCS#12
- MSCAPI (only on Windows platforms)

Cryptographic token support is limited to tested devices. These devices include support tested for SSL clients:

- IBM 4758-23
- nCipher nForce
- Rainbow Cryptoswift
- IBM Security Kit Smartcard
- GemPlus Smartcards
- Rainbow iKey 1000/2000(USB "Smartcard" device)
- Eracom CSA8000

Cryptographic token support has also been tested for the following SSL clients and servers:

- IBM 4758-23
- nCipher nForce
- Rainbow Cryptoswift

WebSphere Application Server uses IBMJSSE to support cryptographic token devices. Refer to the document *install_root*\web\docs\jsse\readme.jsse.ibm.html for further information

Opening a cryptographic token using the key management utility (iKeyman)

Verify that your cryptographic token device is installed and functions properly. Create a cryptographic token, following the instructions provided by the manual of the cryptographic device.

From your cryptographic token device documentation, identify the token library. For example, the IBM 4758 PCI Cryptographic Card uses CRYPTOKI.DLL as the PKCS#11-type token library (see <http://www.ibm.com/security/cryptocards/html/library.shtml> for details).

Read the documentation located at *<install_root>/web/docs/ikeman/ikmuserguide.pdf* for further information about using the key management utility (iKeyman).

You can use the key management utility to open a cryptographic token. Once opened, you can manage your keys and certificates just like you do with keystore and truststore files:

- Create a self-signed digital certificate
 - Add certificate authority (CA) roots as a signer certificate
 - Request and receive a digital certificate from a CA
1. Start the key management utility, if it is not already running.
 2. Click **Key DataBase File > Open**.
 3. Click **Cryptographic Token** from the list of key database types.
 4. Fill in the information for **File Name** and **Location**, or browse for the cryptographic device library.
 5. Click **OK** to open the library.
 6. Type in the slot number in the next panel. This is the number of the slot in which you previously created the cryptographic token.
 7. Enter the password. This is the password configured for the cryptographic token that you created.

All of the personal and signer certificates are stored on the cryptographic token card. With the token open, you can create or request digital certificates and receive CA-signed certificates.

Using a cryptographic token device as a key database to manage keys and certificates for an SSL connection.

Once the cryptographic token is open, you can add or delete keys and certificates. Configure the cryptographic token settings in WebSphere Application Server.

Configuring to use cryptographic tokens

You can configure cryptographic token support in both client and server configuration. To configure a Java client application, use the `sas.client.props` configuration file. By default, the `sas.client.props` is located in the `properties` directory under the `<install_root>` of your WebSphere Application Server installation. To configure a WebSphere Application Server, use the administrative console. To start the administrative console, specify URL:
`http://<server_hostname>:9090/admin`.

To understand how to make WebSphere Application Server (both the run time and the key management utility) work correctly with any cryptographic token device, become familiar with the Java Secure Socket Extension (JSSE) documentation available from the application server product installation
`install_root/web/docs/jsse/readme.jsse.ibm.html` and
`install_root/web/docs/ikeyman/readme.jsse_ikeyman.ibm.htm`.

Unzip the `install_root/web/docs/jsse/native-support.zip` file and copy the correct libraries, with respect to target operating system, to the appropriate location. Otherwise, link errors might occur at run time, or the key management tool might not work properly with the cryptographic device library.

Follow the documentation that accompanies your device to install your cryptographic device. Installation instructions for IBM cryptographic hardware devices can be found in the Administration section of Resources for learning.

1. To configure a client to use a cryptographic token, edit the `sas.client.props` file and set the following properties. Leave the **KeyStore File Name** and the **KeyStore File Password** field in a Secure Sockets Layer (SSL) configuration blank, if you want to use only cryptographic tokens as your keystore file.

com.ibm.ssl.tokenType

Specifies the type of built-in keystore file that is implemented in the cryptographic token. (For example, `com.ibm.ssl.tokenType=PKCS\#11`).
The valid values are: **PKCS\#7**, **PKCS\#11**, **PKCS\#12**, and **MSCAPI**.

com.ibm.ssl.tokenLibraryFile

Specifies the token file name for **PKCS#7** tokens, **PKCS#12** tokens, and the library name for **PKCS#11**, **MSCAPI** tokens. Make sure the cryptographic token device is installed and functions properly with a cryptographic token created. Unzip the `native-support.zip` file from `install_root/web/docs/jsse` directory to copy the required libraries with respect to the target operating system.

com.ibm.ssl.tokenPassword

Specifies the password to unlock the cryptographic token.

2. Configure your server to use cryptographic token. Leave the **KeyStore File Name** and the **KeyStore File Password** field in an SSL configuration blank, if you want to use only cryptographic tokens as your keystore file.
 - a. Specify `http://server_hostname:9090/admin` to start the administrative console.
 - b. Click **Security** > **SSL** to open the SSL Configuration Repertoires panel. Create a new SSL setting alias if one does not exist. Otherwise, click the alias that you want to configure for the cryptographic token.
 - c. Click **Cryptographic Token** in the Additional Properties section to open the Cryptographic Token Settings panel. Click **SSL Configuration Repertoires** > **(SSL alias)** > **Cryptographic Token**.

- d. Complete the information for **Token Type** to specify the type of built-in keystore file that is implemented in the cryptographic token. The valid values are: **PKCS#7, PKCS#11, PKCS#12, MSCAPI**.
- e. Complete the information for **Library File** to specify the token and file name for PKCS#7 tokens, PKCS#12 tokens and the library name for PKCS#11, MSCAPI tokens. Make sure the cryptographic token device is installed and functions properly with a new cryptographic token. Unzip the `native-support.zip` file from `install_root/web/docs/jsse` directory to copy the required libraries with respect to the target operating system.
- f. Complete the information for **Password** to specify the password for unlocking the cryptographic token.
- g. Click **Apply** and **OK** to go back to the **SSL Alias** panel.
- h. Select the box to enable **Cryptographic Token**.
- i. Click **Apply, OK, and Save** to save the configuration.

The configuration is enabled to support the specified cryptographic token for and SSL connection.

WebSphere Application Server uses the cryptographic token as a keystore file for and SSL connection.

If the server configuration has changed, restart the configured server.

Cryptographic token settings

Use this page to configure cryptographic token settings.

To view this administrative console page, click **Security > SSL > *alias_name* > Cryptographic Token**.

Token Type:

Specifies the type of built-in keystore file that is implemented in the cryptographic token, such as PKCS#11.

The WebSphere Application Server uses an implementation of Java Secure Socket Extension (JSSE) to support cryptographic token with Secure Sockets Layer (SSL). Different cryptographic devices are supported. For an SSL server, the following devices are supported:

- IBM 4758-23
- nCipher nForce
- Rainbow Cryptoswift

For an SSL client, the following devices are supported:

- IBM 4758-23
- nCipher nForce
- Rainbow Cryptoswift
- IBM Security Kit Smartcard
- GemPlus Smartcards
- Rainbow iKey 1000/2000 (USB "Smartcard" device)
- Eracom CSA8000

Follow the documentation that accompanies your device to install your cryptographic token.

Data type: String

Library File:

Specifies the dynamic link library (DLL) or shared object that implements the interface to the cryptographic token device.

Data type: String

Password:

Specifies the password for the cryptographic token device.

Data type: String

Using Java Secure Socket Extension and Java Cryptography Extension with Servlets and enterprise bean files

Java Secure Socket Extension

Java Secure Socket Extension (JSSE) provides the transport security for WebSphere Application Server. It provides application programming interface (API) framework and the implementation of the APIs, for Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols, including functionality for data encryption, message integrity and authentication. With the JSSE APIs, other SSL or TLS protocols, and Public Key Infrastructure (PKI), implementations can plug in.

IBM Java Secure Socket Extension

The WebSphere Application Server uses the IBMJSSE provider, which is pre-installed and pre-registered with the Java Cryptography Architecture (JCA) of the Java 2 platform. IBMJSSE supports the following cryptographic services:

- Rivest Shamir Adleman (RSA) public key cryptography support
- SSL and TLS security protocols and common cipher suites
- X.509-based key and trust managers
- PKCS12 as JCA keystore type

The IBMJSSE provider is pre-registered in the `java.security` properties file located at `install_root/java/jre/lib/security` directory. It also supports cryptographic token types PKCS#7, PKCS#11, PKCS#12 and MSCAPI (only on Windows platforms) for cryptographic token support.

Note: The IBM Java Secure Socket Extension (JSSE) is currently not supported within applets.

Customizing Java Secure Socket Extension

Note: Make sure you understand the implication to your application before you begin customizing.

You can customize a number of aspects of JSSE by plugging in different implementations of Cryptography Package Provider, X509Certificate and HTTPS protocols, or specifying different default keystore files, key manager factories and trust manager factories. A provided table summarizes which aspects can be customized, what the defaults are, and which mechanisms are used to provide customization. Some of the key customizable aspects follow:

| Customizable item | Default | How to customize |
|-------------------------------|---|---|
| X509Certificate | X509Certificate implementation from IBM | cert.provider.x509v1 security property |
| HTTPS protocol | Implementation from IBM | java.protocol.handler.pkgs system property |
| Cryptography Package Provider | IBMJSSE | A security.provider.n= line in security properties file. See description. |
| Default keystore | None | * javax.net.ssl.keyStore system property |
| Default truststore | jssecacerts, if it exists. Otherwise, cacerts | * javax.net.ssl.trustStore system property |
| Default key manager factory | IbmX509 | ssl.KeyManagerFactory.algorithm security property |
| Default trust manager factory | IbmX509 | ssl.TrustManagerFactory.algorithm security property |

For aspects that you can customize by setting a system property, statically set the system property by using the `-D` option of the Java command (you can set the system property using the administrative console), or set the system property dynamically by calling the `java.lang.System.setProperty` method in your code: `System.setProperty(propertyName, "propertyValue")`.

For aspects that you can customize by setting a Java security property, statically specify a security property value in the `java.security` properties file located in the `install_root/java/jre/lib/security` directory. The security property is `propertyName=propertyValue`. Dynamically set the Java security property by calling the `java.security.Security.setProperty` method in your code.

Application Programming Interface

The JSSE provides a standard application programming interface (API) available in packages of the `javax.net` file, `javax.net.ssl` file, and the `javax.security.cert` file. The APIs cover:

- Sockets and SSL sockets
- Factories to create the sockets and SSL sockets
- Secure socket context that acts as a factory for secure socket factories
- Key and trust manager interfaces
- Secure HTTP UTL connection classes
- Public key certificate API

There is more information documented for the JSSE APIs in the `jssedocs.jar` file located at `install_root/web/docs/jsse` directory. Unzip the Java archive (JAR) file and open `index.html` file with your browser.

Samples using Java Secure Socket Extension

The Java Secure Socket Extension (JSSE) also provides samples to demonstrate its functionality. The samples are included in `install_root/web/docs/jsse/samplejsse.jar`. Unzip the file. The following files display:

| Files | Description |
|--|--|
| ClientJsse.java | Demonstrates a simple client and server interaction using JSSE. All enabled cipher suites are used. |
| ClientJsseProvider.java | Demonstrates a simple client and server interaction using JSSE. All enabled cipher suites are used. |
| ServerJsse.java ServerJsseProvider.java OldClientJsse.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. |
| OldClientPKCS12Jsse.java | Back-level samples |
| OldServerPKCS12Jsse.java | Back-level samples |
| 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 returned from this site displays. |
| HTTPTest.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 returned from this site is displayed. |
| HTTPSPanel.java OldHTTPTest.java | Back-level sample |

See more instructions in the source code. Follow these instructions before you run the samples.

Permissions for Java 2 security

You might need the following permissions to run an application with JSSE: (This is a reference list only.)

- `java.util.PropertyPermission "java.protocol.handler.pkgs", "write"`
- `java.lang.RuntimePermission "writeFileDescriptor"`
- `java.lang.RuntimePermission "readFileDescriptor"`
- `java.lang.RuntimePermission "accessClassInPackage.sun.security.x509"`
- `java.io.FilePermission "${user.install.root}${}/etc${}/.keystore", "read"`
- `java.io.FilePermission "${user.install.root}${}/etc${}/.truststore", "read"`

For the IBMJSSE provider:

- `java.security.SecurityPermission "putProviderProperty.IBMJSSE"`
- `java.security.SecurityPermission "insertProvider.IBMJSSE"`

For the SUNJSSE provider:

- `java.security.SecurityPermission "putProviderProperty.SunJSSE"`
- `java.security.SecurityPermission "insertProvider.SunJSSE"`

Debugging

By configuring through the `javax.net.debug` system property, JSSE provides the following dynamic debug tracing: `-Djavax.net.debug=true`.

A value of **true** turns on the trace facility, provided that the debug version of JSSE is installed. Use the administrative console to set the system property for debugging the application server.

The debug version of JSSE, `ibmjsse-debug.jar`, is located in the `install_root/web/docs/jsse` directory. To collect a trace complete the following steps:

1. Stop your application.
2. Save the default version of JSSE (`ibmjsse.jar`) into a separate directory.
3. Locate the `ibmjsse.jar` file in the `install_root/java/jre/lib/ext` directory.
4. Replace the `ibmjsse.jar` file with the `ibmjsse-debug.jar` file in the `install_root/java/jre/lib/ext` directory.
5. Specify the `javax.net.debug` system property.
6. Restart your application.

The trace is logged in the `trace.log` file for the application server or in the file specified by a system property, `DtraceFileName` for a Java client application.

Documentation

See the Security: Resources for learning article for documentation references to JSSE.

JCE

Java Cryptography Extension (JCE) provides cryptographic, key and hash algorithms for WebSphere Application Server. It provides a framework and implementations for encryption, key generation, key agreement, and Message Authentication Code (MAC) algorithms. Support for encryption includes symmetric, asymmetric, block and stream ciphers.

IBMJCE

The IBM Java Cryptography Extension (JCE) is an implementation of the JCE cryptographic service provider used in WebSphere Application Server. The IBMJCE is similar to SunJCE, except that the IBMJCE offers more algorithms:

- Cipher algorithm
- Signature algorithm
- Message digest algorithm
- Message authentication code
- Key agreement algorithm
- Random number generation algorithm
- Key store

The IBMJCE is also moving `com.sun.crypto.provider.*` packages to `com.ibm.crypto.provider.*` packages. Read `install_root/web/docs/jce/readme.jce.ibm.html` for further details.

Implementing a Java Cryptography Extension cryptographic service provider

A Cryptographic Service Provider, or *provider*, refers to a package (or a set of packages) that supply a concrete implementation of a subset of the cryptography aspects of the Java Security API. A provider can contain an implementation of one or more digital signature algorithms and one or more cipher algorithms. Complete the following steps to implement and integrate a JCE provider:

1. Write your service implementation code.
2. Give your provider a name.
3. Write your *Master Class*, a subclass of your provider.
4. Compile your code.
5. Prepare for testing.
6. Write and compile test programs.
7. Run your test programs.
8. Document your provider and its supported services.
9. Prepare for production.
10. Run your test programs again.
11. Make your provider software and documentation available to clients.

Refer to *install_root/web/docs/jce/HowToImplAProvider.html* for further information.

Application Programming Interface

Java Cryptography Extension (JCE) has a provider-based architecture. Providers can be plugged into the JCE framework by implementing the APIs defined by the JCE. The JCE APIs covers:

- Symmetric bulk encryption, such as DES, RC2, and IDEA
- Symmetric stream encryption, such as RC4
- Asymmetric encryption, such as RSA
- Password-based encryption (PBE)
- Key Agreement
- Message Authentication Codes

Refer to *install_root/web/docs/jce/api_users_guide.html* and *install_root/web/docs/jce/CryptoSpec.html* for more information about Java Cryptography Extension technology. Locate the Javadoc for the JCE APIs by unzipping *install_root/web/docs/jce/jcedocs.jar*.

Samples using Java Cryptography Extension

There are samples provided in *SampleJCE.jar* file located in the *install_root/web/docs/jce* directory. Unzip the file. The following source code displays:

| File | Description |
|--------------------------------|---|
| <i>SampleDSASignature.java</i> | Demonstrates how to generate a pair of DSA keys (a public key and a private key) and use the key to digitally sign a message using the SHA1with DSA algorithm |
| <i>SampleMarsCrypto.java</i> | Demonstrates how to generate a Mars secret key, and how to do Mars encryption and decryption |

| File | Description |
|-----------------------------|--|
| 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 article for documentation on JCE.

Java 2 security

Java 2 security provides a policy-based, fine-grain access control mechanism that increases overall system integrity by checking for permissions before allowing access to certain protected system resources. Java 2 security guards access to system resources such as file I/O, sockets, and properties. J2EE security guards access to Web resources such as servlets, JavaServer pages (JSPs) and EJB methods. WebSphere global security includes J2EE role-based authorization, the Common Secure Interoperability Version 2 (CSIV2) authentication protocol, and Secure Sockets Layer (SSL) configuration.

Since Java 2 security is relatively new, many existing or even new applications might not be prepared for the very fine-grain access control programming model that Java 2 security is capable of enforcing. Administrators should understand the possible consequences of enabling Java 2 security if applications are not prepared for Java 2 security. Java 2 security places some new requirements on application developers and administrators.

Java 2 security for deployers and administrators

Although Java 2 security is supported in WebSphere Application Server Version 5, it is disabled by default. However, it is enabled automatically if you also enable global security when configuring security. Although it becomes enabled automatically when you enable WebSphere global security, you can choose to disable it. You can configure Java 2 security and global security independently of one another. Disabling global security does not disable Java 2 security automatically. You need to explicitly disable it.

If your applications, or third-party libraries are not ready, having Java 2 security enabled causes problems. You can identify these problems as Java 2 security `AccessControlExceptions` in the `SystemOut.log` file, `SystemError.log` file, or the trace log files. If you are unsure about the Java 2 security readiness of your applications, disable Java 2 security initially to get your application installed and verify that it is working properly.

There are implications if Java 2 Security is enabled; deployers or administrators are required to make sure that all the applications are granted the required

permissions, otherwise, applications might fail to run. By default, applications are granted the permissions recommended in the J2EE 1.3 Specification. For details of default permissions granted to applications in the product, refer to the following policy files:

- *install_root/java/jre/lib/security/java.policy*
- *install_root/properties/server.policy*
- *install_root/config/cells/<cell_name>/nodes/<node_name>/app.policy*

Note: This policy embodied by these policy files cannot be made more restrictive because the product might not have the necessary Java 2 security doPrivileged APIs in place. The restrictive policy is the default policy. You can grant additional permissions, but you cannot make the default more restrictive because AccessControlExceptions is generated from within WebSphere Application Server. The product does not support a more restrictive policy than the default defined in the policy files previously mentioned.

There are several policy files used to define the security policy for the Java process. These policy files are static (code base is defined in the policy file) and they are in the default policy format provided by the IBM Developer Kit, Java Technology Edition. For enterprise application resources and utility libraries, WebSphere Application Server provides dynamic policy support. The code base is dynamically calculated based on deployment information and permissions are granted based on template policy files during run time. Refer to the section of Java 2 security policy management.

Note: Syntax errors in the policy files cause the application server process to fail. Edit these policy files carefully using the Policy Tool provided by the IBM Developer Kit, Java Technology Edition for editing the policy files (*install_root/java/jre/bin/policytool*).

If an application is not prepared for Java 2 security, if the application provider does not provide a *was.policy* file as part of the application, or if the application provider does not communicate the expected permissions the application is likely to cause Java 2 security access control exceptions at run time. It might not be obvious that an application is not prepared for Java 2 security. Several run-time debugging aids help troubleshoot applications that might have access control exceptions. See the Java 2 security debugging aids for more details. See Handling applications that are not Java 2 security ready for information and strategies for dealing with such applications.

It is important to note that when Java 2 Security is enabled in the Global Security settings, the installed SecurityManager does not currently check *modifyThread* and *modifyThreadGroup* permissions for non-system threads. Allowing Web and EJB application code to create or modify a thread can have a negative impact on other components of the container and can affect the capability of the container to manage enterprise bean life cycles and transactions.

Java 2 security for application developers

Application developers must understand the permissions granted in the default WebSphere policy and the permission requirements of the SDK APIs that their application calls to know whether additional permissions are required. The "Permissions in the Java 2 SDK" reference in the resources section describes which APIs require which permission.

Application providers can assume that applications have the permissions granted in the default policy previously mentioned. Applications that access resources not covered by the default WebSphere policy are required to grant the additional Java 2 security permissions to the application.

While it is possible to grant the application additional permissions in one of the other dynamic WebSphere policy files or in one of the more traditional static policy files, such as `java.policy`, the `was.policy` (which is embedded in the EAR file) ensures the additional permissions are scoped to the exact application that requires them. Scoping the permission beyond the application code that requires it can permit code that normally does not have permission to access particular resources.

If an application component is being developed, like a library that might actually be included in more than one `.ear` file, then the library developer should document the required Java 2 permissions needed by the application assembler. There is no `was.policy` file for library type components. The developer must communicate the required permissions through Javadoc or some other external documentation.

If the component library is shared by multiple enterprise applications, the permissions can be granted to all enterprise applications on the node in the `app.policy` file.

If the permission is only used internally by the component library and the application should never be granted access to resources protected by the permission, then it might be necessary to mark the code as **privileged** (inserting `doPrivileged`). Refer to the article, `AccessControlException`, for more details. However, improperly inserting a `doPrivileged` might open up security holes. Understand the implication of `doPrivileged` to make a correct judgement whether a `doPrivileged` should be inserted or not.

The section on Dynamic Policy describes how the permissions in the `was.policy` files are granted at run time.

Developing an application with Java 2 security in mind might be a new skill and impose a security awareness not previously required of application developers. Describing the Java 2 security model and the implications on application development is beyond the scope of this section. The following URL can help you get started: <http://java.sun.com/j2se/1.3/docs/guide/security/index.html>.

Debugging Aids

There are two primary aids, the WebSphere `SystemOut.log` file and the `com.ibm.websphere.java2secman.norethrow` property.

The WebSphere SystemOut.log File

The `AccessControl` exception logged in the `SystemOut.log` file contains the permission violation that causes the exception, the exception call stack, and the permissions granted to each stack frame. This information is usually enough to determine the missing permission and the code requiring the permission.

The `com.ibm.websphere.java2secman.norethrow` Property

When Java 2 security is enabled in WebSphere Application Server, the security manager component throws a `java.security.AccessControl` exception when a

permission violation occurs. This exception, if not handled, often causes a run-time failure. This exception is also logged in the `SystemOut.log` file.

However, when the JVM `com.ibm.websphere.java2secman.norethrow` property is set and has a value of **true**, the security manager does not throw the `AccessControl` exception. This information is logged.

To set the `com.ibm.websphere.java2secman.norethrow` property for the server, go to the WebSphere Application Server administrative console and click **Servers > Application Servers**. Under Additional Properties, click **Process Definition > Java Virtual Machine > Custom Properties > New**. In the Name field, type `com.ibm.websphere.java2secman.norethrow`. In the Value field, type **true**.

To set the `com.ibm.websphere.java2secman.norethrow` property for the node agent, go to the WebSphere Application Server administrative console and click **System Administration > Node Agents**. Under Additional Properties, click **Process Definition > Java Virtual Machine > Custom Properties > New**. In the Name field, type `com.ibm.websphere.java2secman.norethrow`. In the Value field, type **true**.

Note: This property is intended for a sandbox or debug environment because it instructs the security manager not to throw the `AccessControl` exception. Java 2 security is not enforced. This property should not be used in a production environment where a relaxed Java 2 security environment weakens the integrity that Java 2 security is intended to produce.

This property is valuable in a sandbox or test environment where the application can be thoroughly tested and the where the `SystemOut.log` file can be inspected for `AccessControl` exceptions. Since this property does not throw the `AccessControl` exception, it does not propagate the call stack and does not cause a failure. Without this property, you have to find and fix `AccessControl` exceptions one at a time.

Handling applications that are not Java 2 security ready

If the increased system integrity that Java 2 security provides is important, then contact the application provider to have the application support Java 2 security or at least communicate the required additional permissions beyond the default WebSphere policy that must be granted.

The easiest way to deal with such applications is to disable Java 2 security in WebSphere Application Server. The downside is that this solution applies to the entire system and the integrity of the system is not as strong as it might be. Disabling Java 2 security might not be acceptable depending on the organization security policies or risk tolerances.

Another approach is to leave Java 2 security enabled, but to grant either just enough additional permissions or grant all permissions to just the problematic application. Granting permissions however, might not be a trivial thing to do. If the application provider has not communicated the required permissions in some way, there is no easy way to determine what the required permissions are and granting all permissions might be the only choice. You minimize this risk by locating this application on a different node, which might help isolate it from certain resources. Grant the `java.security.AllPermission` permission in the `was.policy` file embedded in the application's `.ear` file, for example:

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

install_root/properties/server.policy

This policy defines the policy for the WebSphere classes. At present, all the server processes on the same installation share the same `server.policy` file. However, you can configure this file so that each server process can have a separate `server.policy` file. Define the desired policy file as the value of the Java system properties `java.security.policy`. For details of how to define Java system properties, Refer to the Process definition section of the Manage application servers file.

The `server.policy` file is not a configuration file managed by the repository and the file replication service. Changes to this file are local and do not get replicated to other machines. Use the `server.policy` file to define Java 2 security policy for server resources. Use the `app.policy` file (per node) or `was.policy` file (per enterprise application) to define Java 2 security policy for enterprise application resources.

WAS_HOME/java/jre/lib/security/java.policy

The file represents the default permissions granted to all classes. The policy of this file applies to all the processes launched by the WebSphere Application Server JVM.

Troubleshooting

Symptom:

Error message SECJ0314E: Current Java 2 security policy reported a potential violation of Java 2 security permission. Refer to Problem Determination Guide for further information.
{0}Permission\:{1}Code\:{2}{3}Stack Trace\:{4}Code Base Location\:{5} Current Java 2 security policy reported a potential violation of Java 2 Security Permission. Refer to Problem Determination Guide for further information.
{0}Permission\:{1}Code\:{2}{3}Stack Trace\:{4}Code Base Location\:{5}

Problem:

The Java security manager `checkPermission()` reported a `SecurityException` on the subject permission with debugging information. The reported information can be different with respect to the system configuration. This report is enabled by either configuring RAS trace into debug mode or specifying a Java property. See Enabling trace for information on how to configure RAS trace in debug mode. Specify the following property in the JVM Settings panel from the administrative console:

java.security.debug. Valid values include:

access Print all debug information including: required permission, code, stack, and code base location.

stack Print debug information including: required permission, code, and stack.

failure

Print debug information including: required permission and code.

Recommended response:

The reported exception might be critical to the secure system. Turn on security trace to determine the potential code that might have violated the security policy. Once the violating code is determined, verify if the attempted operation is permitted with respect to Java 2 security, by examining all applicable Java 2 security policy files and the application code.

Note: If the application is running with Java Mail, this message might be benign. User can update the `was.policy` file to grant the following permissions to the application.

```
permission java.io.FilePermission "${user.home}${/}.mailcap", "read";
permission java.io.FilePermission "${user.home}${/}.mime.types", "read";
permission java.io.FilePermission "${java.home}${/}lib${/}mailcap", "read";
permission java.io.FilePermission "${java.home}${/}lib${/}mime.types", "read";
```

Messages

| | |
|-----------------------|---|
| Message: | SECJ0313E: Java 2 security manager debug message flags are initialized\ : TrDebug: {0}, Access: {1}, Stack: {2}, Failure: {3} |
| Problem: | Configured values of the valid debug message flags for security manager. |
| Recommended response: | None. |

| | |
|-----------------------|---|
| Message: | SECJ0307E: Unexpected exception is caught when trying to determine the code base location. Exception: {0} |
| Problem: | An unexpected exception is caught when the code base location is determined. |
| Recommended response: | Contact an IBM representative. |

AccessControlException

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()` is performed or the request is denied (a `java.security.AccessControlException` is thrown). However, if the caller is marked as *privileged* and the class (caller) is granted the

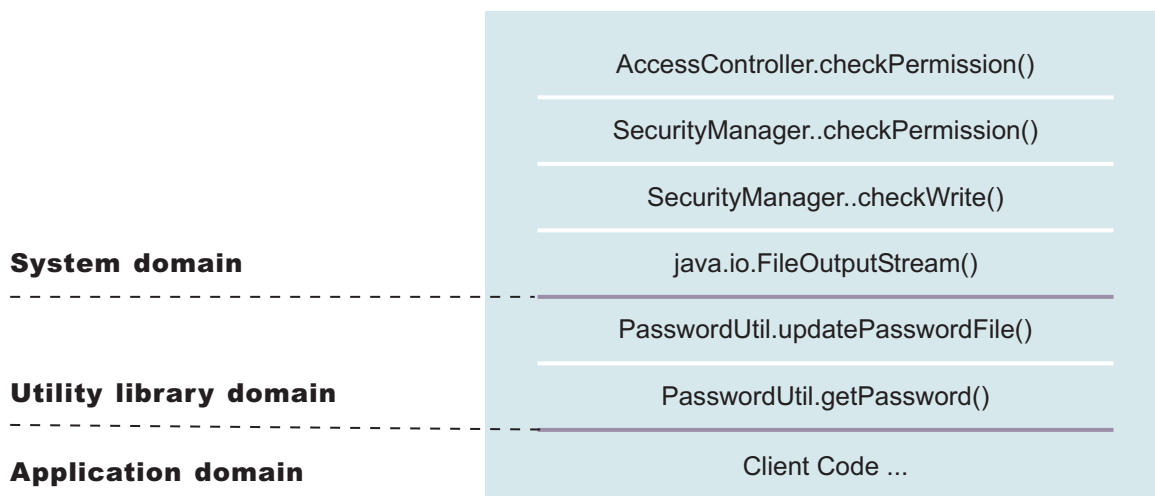
said permissions, the algorithm returns and does not walk the entire call stack. Subsequent classes (callers) do not need the required permission granted.

A `java.security.AccessControlException` exception is thrown as a result of certain classes on the call stack missing the required permissions during a `java.security.AccessController.checkPermission()` method. Two possible resolutions to the `java.security.AccessControlException` exception:

- If the application is calling a Java 2 security-protected API, then grant the required permission to the application Java 2 Security policy. If the application is not calling a Java 2 security-protected API directly and the required permission is because of 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 should. In this case, it is likely that the third party code that accesses the Java 2 Security protected resource is not properly mark 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 is only an example to illustrate the point. The decision as to 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. There are a number of well written publications and books 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 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 should not have permission to write to the password file directly and 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. Then 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 `java.security.AccessControlException`

As described previously, there are two possibilities 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* (considering the concerns and risks).

Configuring Java 2 security

Java 2 security is a new feature in WebSphere Application Server Version 5. It is a new programming model that is very pervasive and has a huge impact on application development. It is disabled by default, but is enabled automatically when global security is enabled. However, Java 2 security is orthogonal to J2EE role-based security; you can disable or enable it independently of Global Security.

However, it does provide an extra level of access control protection on top of the J2EE role-based authorization. It particularly addresses the protection of system resources and APIs. Administrators should need to consider the benefits against the risks of disabling Java 2 Security.

The following recommendations are provided to help enable Java 2 security in a test or production environment:

1. Make sure the application is developed with the Java 2 security programming model in mind. Developers have to know whether or not the APIs used in the applications are protected by Java 2 security. It is very important that the required permissions for the APIs used are declared in the policy file (*was.policy*), or the application fails to run when Java 2 security is enabled. Developers can reference the Web site for Development Kit APIs that are protected by Java 2 security. See the Programming model and decisions section of the Security: Resources for Learning article to visit this Web site.
2. Make sure that migrated applications from previous releases are given the required permissions. Since Java 2 security is not supported or partially supported in previous WebSphere Application Server releases, applications developed prior to Version 5 most likely are not using the Java 2 security programming model. There is no easy way to find out all the required permissions for the application. Following are activities you can perform to determine the extra permissions required by an application:
 - Code review and code inspection
 - Application documentation review
 - Sandbox testing of migrated enterprise applications with Java 2 security enabled in a pre-production environment. Enable tracing in WebSphere Java 2 security manager to help determine the missing permissions in the application policy file. The trace specification is `com.ibm.ws.security.core.SecurityManager=all=enabled`.

- Use the `com.ibm.websphere.java2secman.norethrow` system property to aid debugging. This property should not be used in a production environment. Refer to Java 2 security.

Note: The default permission set for applications is the recommended permission set defined in the J2EE 1.3 Specification. The default is declared in the `config/cells/<cell_name>/nodes/<node_name>/app.policy` policy file with permissions defined in the Development Kit (`${JAVA_HOME}/lib/security/java.policy`) policy file that grant permissions to everyone. However, applications are denied permissions declared in the `config/cells/cell_name/filter.policy` filter policy file. Permissions declared in the *filter.policy* file are filtered for applications during the permission check.

Note: Define the required permissions for an application in a `was.policy` file and embed the `was.policy` file in the application enterprise archive (EAR) file as `YOURAPP.ear/META-INF/was.policy` (see Configuring dynamic policy for details).

Java 2 security is enabled and enforced for the servers. Java 2 security permission is selected when a Java 2 security protected API is called.

When to use Java 2 security.

1. To enable protection on system resources. For example, when opening or listening to a socket connection, reading or writing to operating system file systems, reading or writing Java Virtual Machine system properties, and so on.
2. To prevent application code calling destructive APIs. For example, calling the `System.exit()` method brings down the application server.
3. To prevent application code from obtaining privileged information (passwords) or gaining extra privileges (obtaining server credentials).

The WebSphere Java 2 security manager is enhanced to dump the Java 2 security permissions granted to all classes on the call stack when an application is denied access to a resource (the `java.security.AccessControlException` exception is thrown). However, this tracing capability is disabled by default. You can enable it by specifying the server trace service with the `com.ibm.ws.security.core.SecurityManager=all=enabled` trace specification. When the exception is thrown, the trace dump provides hints to determine whether the application is missing permissions or the product run time code or third party libraries used are not properly marked as *privileged* when accessing Java 2 protected resources. See the Security Problem Determination Guide for details.

Enable or disable Java 2 Security for the cell

1. Click **Security > Global Security** in the navigation tree. The Global Security page appears.
2. Enable Java 2 Security by selecting the check box labeled **Enforce Java 2 Security** (clear the check box to disable Java 2 Security). This enables Java 2 Security for the cell.
3. Click **OK** or **Apply** on the Global Security page.
4. Save the changes and make sure a file sync is performed before restarting the servers.
5. For the changes to take effect, restart all the servers, which include the Network Deployment Manager, all Node Agents, and all application servers.

Enable or disable Java 2 Security for an application server

1. Click **Server > Application Servers** in the navigation tree. The Application Servers page appears.
2. Click the **application server name** in the **Name** column of the Application Server collection table. The configuration panel of the application server selected appears.
3. Click **Server Security** in the Additional Properties section. The Server Security panel of the application server appears.
4. Click **Server Level Security** in the Additional Properties section. The Server Level Security panel of the application server appears.
5. Enable Java 2 Security by selecting the check box labeled **Enforce Java 2 Security** (clear the check box to disable Java 2 Security). This enables Java 2 Security for the selected application server.
6. Click **OK** or **Apply** on the Server Level Security page.
7. Save the changes and make sure a file sync is performed before restarting the application server.
8. Restart the application server for the changes to take effect.

Java 2 Security is enabled and enforced for the servers. Java 2 Security permission is checked when a Java 2 Security protected API is called.

When to use Java 2 Security.

1. To enable protection on system resources. For example, when opening or listening to a socket connection, reading or writing to operating system file systems, reading or writing Java Virtual Machine system properties, and so on.
2. To prevent application code calling destructive APIs. For example, calling *System.exit()* brings down the application server.
3. To prevent application code obtaining privileged information (passwords) or gaining extra privileges (obtaining Server Credentials).

The WebSphere Java 2 Security Manager is enhanced to dump the Java 2 Security permissions granted to all classes on the call stack when an application is denied access to a resource (the `java.security.AccessControlException` exception is thrown). The trace information is dumped to the configured server log files. Check the server log files to access debugging information when an `AccessControlException` is thrown. In addition, the product Java 2 Security Manager trace can be enabled with the trace string, `com.ibm.ws.security.core.SecurityManager=all=enabled`. When the exception is thrown, the trace dump provides hints to determine whether the application is missing permissions or the product run time code or third party libraries used are not properly marked as *privileged* when accessing Java 2 protected resources. See the Security Problem Determination Guide for details.

Using PolicyTool to edit policy files

Java 2 security uses several policy files to determine the granted permission for each Java program. See Dynamic policy for the list of available policy files. The Java Development Kit provides *policytool* to edit these policy files. This tool is recommended for editing any policy file to verify the syntax of its contents. Syntax errors in the policy file cause an `AccessControlException` during application execution, including the server start. Identifying the cause of this exception is not easy because the user might not be familiar with the resource that has an access violation. Be careful when you edit these policy files.

1. Start policytool. Enter `%{was.install.root}/java/jre/bin/policytool` from a command prompt.
The policytool window opens. The policytool looks for the `.java.policy` file in your home directory. If it does not exist, an Error message displays. Click **OK**.
2. Click **File > Open**.
3. Navigate the directory tree in the **Open** window to pick up the policy file that you need to update. After selecting the policy file, click **Open**. The code base entries are listed in the window.
4. Create or modify the code base entry.
 - a. Modify the existing code base entry by double-clicking the code base, or click the code base and click **Edit Policy Entry**. The Policy Entry window opens with the permission list defined for the selected code base.
 - b. Create a new code base entry by clicking **Add Policy Entry**. The Policy Entry window opens. At the code base column, enter the code base information as a URL format, for example, `/WebSphere/AppServer/InstalledApps/testcase.ear`.
5. Modify or add the permission specification
 - a. Modify the permission specification by double-clicking the entry you want to modify, or by selecting the permission and clicking **Edit Permission**. The Permissions window opens with the selected permission information.
 - b. Add a new permission by clicking **Add Permission**. The Permissions window opens. In the Permissions, window there are four rows for **Permission**, **Target Name**, **Actions**, and **Signed By**.
6. Select the permission from the Permission list. The selected permission displays. After a permission is selected, the **Target Name**, **Actions**, and **Signed By** fields automatically show the valid choices or they enable text input in the right text input area.
 - a. Select **Target Name** from the list, or enter the target name in the right text input area.
 - b. Select **Actions** from the list.
 - c. Input **Signed By** if it is needed.

Important: The Signed By keyword is not supported in the following policy files: `app.policy`, `spi.policy`, `library.policy`, `was.policy`, and `filter.policy` files. However, the Signed By keyword is supported in the following policy files: `java.policy`, `server.policy`, and `client.policy` files. The Java Authentication and Authorization Service (JAAS) is not supported in the `app.policy`, `spi.policy`, `library.policy`, `was.policy`, and `filter.policy` files. However, the JAAS principal keyword is supported in a JAAS policy file when it is specified by the Java Virtual Machine (JVM) system property, `java.security.auth.policy`.
7. Click **OK** to close the Permissions window. Modified permission entries of the specified code base display.
8. Click **Done** to close the window. Modified code base entries are listed. Repeat steps 4 through 8 until you complete editing.
9. Click **File > Save** after you finish editing the file.

A policy file is updated. If any policy files need editing, use the policytool. Do not edit the policy file manually. Syntax errors in the policy files can potentially cause

application servers or enterprise applications to not start or function incorrectly. For the changes in the updated policy file to take effect, restart the Java processes.

Java 2 security policy files:

The J2EE 1.3 specification has a well-defined programming model of responsibilities between the container providers and the application code. Using Java 2 security manager to help enforce this programming model is recommended. There are certain operations that are not allowed 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. There are a number of policy files in the product, which are either static or dynamic. *Dynamic policy* is a template of permissions for a particular type of resource. There is no relative codebase defined in the dynamic policy template. The codebase is dynamically calculated from the deployment and run-time data.

Static policy files

| Policy file | Location |
|---------------|--|
| java.policy | <i>install_root</i> /java/jre/lib/security/java.policy. Default permissions granted to all classes. The policy of this file applies to all the processes launched by the WebSphere Application Server. |
| server.policy | <i>install_root</i> /properties/server.policy. Default permissions granted to all the product servers. |
| client.policy | <i>install_root</i> /properties/client.policy. Default permissions for all of the product client containers and applets on a node. |

The static policy files are not managed by configuration and file replication services. Changes made in these files are local and are not replicated to other nodes in the Network Deployment cell.

Dynamic policy files

| Policy file | Location |
|----------------|---|
| spi.policy | <i>install_root</i> /config/cells/ <i>cell_name</i> / <i>nodes/node_name</i> /spi.policy This template is for the Service Provider Interface (SPI) or third-party resources embedded in the product. Examples of SPI are Java Messaging Service (JMS) (MQSeries) and JDBC drivers. The codebase for the embedded resources are dynamically worked out from the configuration (resources.xml file) and run-time data, and permissions defined in the spi.policy files are automatically applied to these resources. The default permission of spi.policy file is java.security.AllPermissions. |
| library.policy | <i>install_root</i> /config/cells/ <i>cell_name</i> / <i>nodes/node_name</i> /library.policy This template is for the library (Java library classes). You can define a shared library to use in multiple product applications. The default permission of the library.policy is empty. |

| Policy file | Location |
|-------------|---|
| app.policy | <i>install_root/config/cells/cell_name</i> <i>/nodes/node_name/app.policy</i> The app.policy file defines the default permissions granted to all enterprise applications running on <i>node_name</i> in <i>cell_name</i> . |
| was.policy | <i>install_root/config/cells/cell_name</i> <i>/applications/ear_file_name/deployments/</i> <i>application_name/META-INF/was.policy</i> Type the previous location on one continuous line. This template is for application-specific permissions. The was.policy 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 defined in the ra.xml file. The ra.xml file is embedded in the RAR file. |

Note: Grant entry specified in the app.policy and was.policy files must have a code base defined. If there are grant entries 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, then 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"
```

Please 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 running on `node_name`. If you define the permissions in the `META-INF/was.policy` file, it only applies to the specific enterprise application. Valid symbols for codebase are listed in the following table:

| 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 EJB 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 |

Other than these entries specified by the codebase symbols, you can specify the module name for a granular setting. For example:

```
grant codeBase "file:DefaultWebApplication.war" {
    permission java.security.SecurityPermission "printIdentity";
};

grant codeBase "file:IncCMP11.jar" {
    permission java.io.FilePermission
    "${user.install.root}${/}bin${/}DefaultDB${/}-",
    "read,write,delete";
};
```

The 6th and 7th lines in the previous code sample are one continuous line.

You can use a relative codebase 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 |
|------------------------------------|--|
| <code>file:\${application}</code> | Permissions apply to all resources within the application |
| <code>file:\${jars}</code> | Permissions apply to all utility JAR files within the application |
| <code>file:\${ejbComponent}</code> | Permissions apply to enterprise beans resources within the application |
| <code>file:\${webComponent}</code> | Permissions apply to Web resources within the application |

| Symbol | Meaning |
|-----------------------------|---|
| file:\${connectorComponent} | Permissions apply to connector resources both within the application and stand-alone connector resources. |

There are five embedded symbols provided to specify the path and name for `java.io.FilePermission`. These symbols enable flexible permission specification. The absolute file path is fixed after the installation of the application.

| Symbol | Meaning |
|--------------------------------------|---|
| <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 |

Note: You must 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. Since dynamic policy resolves the codebase at run time, determining which policy file has a problem is difficult. Add a permission only to the necessary resources. For example, use `${ejbcomponent}`, and etc instead of `${application}`, and update the `was.policy` file instead of the `app.policy` file, if possible.

Static policy filtering

Limited static policy filtering support exists. If the `app.policy` file and the `was.policy` file have permissions defined in the `filter.policy` file with the keyword, `filterMask`, the run time removes the permissions from the applications and an audit message is logged. However, if the permissions defined in the `app.policy` and `was.policy` are compound permissions, for example, `java.security.AllPermission`, the permission is not removed, rather 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 `was.policy` file have permissions defined in the `filter.policy` file with the keyword, `runtimeFilterMask`, the run time removes the permissions from the applications no matter what permissions are granted to the application. For example, even if a `was.policy` file has `java.security.AllPermission` granted to one of its modules, specified permissions such as `runtimeFilterMask` are removed from the granted permission during run time.

If the **Issue Permission Warning** flag in the Global Security panel is enabled and if the `app.policy` file and the `was.policy` file contain custom permissions (non-Developer Kit permissions, where the permissions package name begins with `java` or `javax`), a warning message logs. The permission is not removed. If the permission, `AllPermission`, is listed in the `app.policy` file and the `was.policy` file, a warning message logs.

Policy file editing

Using the policy tool provided by the Developer Kit (*install_root/java/jre/bin/policytool*), to edit the previous policy files is recommended. For Network Deployment, extract the policy files from the repository before editing. After the policy file is extracted, use the policy tool to edit the file. Check the modified policy files into the repository and synchronized them with other nodes.

Note: If there are syntax errors in the policy files, the enterprise application or server process might fail to start. Be very cautious when editing these policy files.

Troubleshooting

To debug the dynamic policy, there are three ways to generate the detail report of the exception, `AccessControlException`.

- **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 property `java.security.debug`. 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.
- **ffdc**. Enable `ffdc`, modify the `ffdcRun.properties` file by changing `Level=4` and `LAE=true`. Look for a keyword `Access Violation` in the log file.

Configuring Java 2 security policy files:

Java 2 security uses several policy files to determine the granted permissions for each Java programs. See the Dynamic policy article for the list of available policy files supported by WebSphere Application Server Version.

There are two types of policy files supported by WebSphere Application Server: dynamic policy files and static policy files. Static policy files provide the default permissions. Dynamic policy files provide application permissions. There are six dynamic policy files:

| Policy file name | Description |
|-------------------------|--|
| <code>app.policy</code> | Contains default permissions for all of the enterprise applications in the cell. |
| <code>was.policy</code> | Contains application-specific permissions for an WebSphere Application Server enterprise application. This file is packaged in an enterprise archive (EAR) file. |
| <code>ra.xml</code> | Contains connector application specific permissions for a WebSphere Application Server enterprise application. This file is packaged in a resource adapter archive (RAR) file. |

| Policy file name | Description |
|------------------|---|
| spi.policy | Contains permissions for Service Provider Interface (SPI) or third-party resources embedded in WebSphere Application Server. The default contents grant everything. Update this file carefully when the cell requires more protection against SPI in the cell. This file is applied to all of the SPIs defined in the resources.xml file. |
| library.policy | Contains permissions for the shared library of enterprise applications. |
| filter.policy | Contains the list of permissions that require filtering from the was.policy file and the app.policy file in the cell. This filtering mechanism only applies to the was.policy and app.policy files. |

Important: The Signed By keyword is not supported in the following policy files: app.policy, spi.policy, library.policy, was.policy, and filter.policy files. However, the Signed By keyword is supported in the following policy files: java.policy, server.policy, and client.policy files. The Java Authentication and Authorization Service (JAAS) is not supported in the app.policy, spi.policy, library.policy, was.policy, and filter.policy files. However, the JAAS principal keyword is supported in a JAAS policy file when it is specified by the Java Virtual Machine (JVM) system property, java.security.auth.policy. You can statically set the authorization policy files in java.security.auth.policy with auth.policy.url.n=URL where URL is the location of the authorization policy.

1. Identify the policy file to update.
 - If the permission is required by an application, update the static policy file. Refer to Configuring static policy files.
 - If the permission is required by all of the WebSphere Application Server enterprise applications in the node, refer to Configuring spi.policy files.
 - If the permission is required only by specific WebSphere Application Server enterprise applications and the permission is required only by connector, update the ra.xml file. Refer to Assembling resource adapter (connector) modules. Otherwise, update the was.policy file. Refer to Configuring was.policy files and Adding the was.policy file to applications.
 - If the permission is required by shared libraries, refer to Configuring library.policy files.
 - If the permission is required by SPI libraries, refer to Configuring spi.policy files.

Note: It is recommended to pick up the policy file with the smallest scope. You can avoid giving an extra permission to the Java programs and protect the resources. You can update the ra.xml file or the was.policy file rather than the app.policy file. Use specific component symbols (`$(ejbcomponent)`, `$(webComponent)`, `$(connectorComponent)` and `$(jars)`) than `$(application)` symbols. Update dynamic policy files than static policy files.

Add any permission that should never be granted to the WebSphere Application Server enterprise application in the cell to the filter.policy file. Refer to Configuring filter.policy files.

2. Restart the WebSphere Application Server enterprise application.

The required permission is granted for the specified WebSphere Application Server enterprise application.

If an WebSphere Application Server enterprise application in a cell requires permissions, some of the dynamic policy files need updating. The symptom of the missing permission is the exception, `java.security.AccessControlException`. The missing permission is listed in the exception data, for example,

```
java.security.AccessControlException: access denied (java.io.FilePermission
C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar read)
```

The previous two lines were split onto two lines because of the width of the page. However, the permission should be on one line.

When a Java program receives this exception and adding this permission is justified, add a permission to an adequate dynamic policy file, for example,

```
grant codeBase "file:<user client installed location>" {
permission java.io.FilePermission
"C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar", "read";
};
```

The previous two lines were split onto two lines because of the width of the page. However, the permission should be on one line.

To decide whether to add a permission, refer to the article `AccessControlException`.

Configuring app.policy files:

Java 2 security uses several policy files to determine the granted permissions for each Java program. See the `Dynamic policy` article for the list of available policy files supported by WebSphere Application Server. The `app.policy` file is a default policy file shared by all of the WebSphere Application Server enterprise applications. The union of the permissions contained in the `app.policy` file, the `server.policy` file, the `app.policy` file, the `application was.policy` file and the permission specification of the `ra.xml` file are applied to the WebSphere Application Server enterprise application. The `app.policy` files are managed by configuration and file replication services.

Important: The `Signed By` and the `Java Authentication and Authorization Service (JAAS)` principal keywords are not supported in the `app.policy` file. However, the `Signed By` keyword is supported in the following files: `java.policy`, `server.policy`, and the `client.policy` files. The `JAAS` principal keyword is supported in a `JAAS` policy file when it is specified by the `Java Virtual Machine (JVM)` system property, `java.security.auth.policy`. You can statically set the authorization policy files in `java.security.auth.policy` with `auth.policy.url.n=URL` where `URL` is the location of the authorization policy.

If the default permissions for enterprise applications (the union of the permissions defined in the `app.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.

The default Java 2 security policies have been changed for the enterprise application.

Several product-reserved symbols are defined to associate the permission lists to a specific type of resource.

| Symbol | Meaning |
|-----------------------------|--|
| file:\${application} | Permissions apply to all resources within the application |
| file:\${jars} | Permissions apply to all utility Java archive (JAR) files within the application |
| file:\${ejbComponent} | Permissions apply to enterprise bean resources within the application |
| file:\${webComponent} | Permissions apply to Web resources within the application |
| file:\${connectorComponent} | Permissions apply to connector resources both within the application and within stand-alone connector resources. |

There are five embedded symbols provided to specify the path and name for `java.io.FilePermission`. These symbols enable flexible permission specifications. The absolute file path is fixed after the installation of the application.

| Symbol | Meaning |
|-------------------------|---|
| \${app.installed.path} | Path where the application is installed |
| \${was.module.path} | Path where the module is installed |
| \${current.cell.name} | Current cell name |
| \${current.node.name} | Current node name |
| \${current.server.name} | Current server name |

Note: You cannot use the `${was.module.path}` in the `${application}` entry.

The `app.policy` file supplied by WebSphere Application Server resides at `install_root/config/cells/cell_name/nodes/node_name/app.policy`, which contains the following default permissions:

Attention: In the following code sample, the first two lines related to permission `java.io.FilePermission` were split into two lines each due to the width of the printed page.

```
grant codeBase "file:${application}" {
    // The following are required by Java mail
    permission java.io.FilePermission "${was.install.root}${/}java${/}
jre${/}lib${/}ext${/}mail.jar", "read";
    permission java.io.FilePermission "${was.install.root}${/}java${/}
jre${/}lib${/}ext${/}activation.jar", "read";
};

grant codeBase "file:${jars}" {
    permission java.net.SocketPermission "*", "connect";
    permission java.util.PropertyPermission "*", "read";
};
```



```

grant codeBase "file:${connectorComponent}" {
    permission java.net.SocketPermission "*", "connect";
    permission java.util.PropertyPermission "*", "read";
};
grant codeBase "file:${webComponent}" {
    permission java.io.FilePermission "${was.module.path}${/}-", "read, write";
    permission java.lang.RuntimePermission "loadLibrary.*";
    permission java.lang.RuntimePermission "queuePrintJob";
    permission java.net.SocketPermission "*", "connect";
    permission java.util.PropertyPermission "*", "read";
};

grant codeBase "file:${ejbComponent}" {
    permission java.lang.RuntimePermission "queuePrintJob";
    permission java.net.SocketPermission "*", "connect";
    permission java.util.PropertyPermission "*", "read";
};

```

If all of the WebSphere Application Server enterprise applications in a cell require permissions that are not defined as defaults in the `app.policy` file, the `server.policy` file and the `app.policyfile`, then update the `app.policy` file. The symptom of a missing permission is the exception, `java.security.AccessControlException`. The missing permission is listed in the exception data, for example, `java.security.AccessControlException: access denied (java.io.FilePermission C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar read)`.

When a Java program receives this exception and adding this permission is justified, add a permission to the `server.policy` file, for example:

```

grant codeBase "file:<user client installed location>" {
    permission java.io.FilePermission
"C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar", "read"; };

```

To decide whether to add a permission, refer to the article `AccessControlException`.

Restart all WebSphere Application Server enterprise applications to ensure that the updated `app.policy` file takes effect.

Configuring filter.policy files:

Java 2 security uses several policy files to determine the granted permission for each Java program. Java 2 security policy filtering is only in effect when Java 2 security is enabled. Refer to `Configuring Java 2 security`. The filtering policy defined in the `filter.policy` file is cell wide. Refer to the article, `Dynamic policy`, for the list of available policy files supported by WebSphere Application Server. The `filter.policy` file is the only policy file used when restricting the permission instead of granting permission. The permissions listed in the filter policy file are filtered out from the `app.policy` file and the `was.policy` file. Permissions defined in the other policy files are not affected by the `filter.policy` file.

When a permission is filtered out, an audit message is logged. However, if the permissions defined in the `app.policy` file and the `was.policy` file are compound permissions like `java.security.AllPermission`, for example, the permission is not removed. A warning message is logged. If the `Issue Permission Warning` flag is

enabled (default) and if the `app.policy` file and the `was.policy` file contain custom permissions (non-Java API permission, the permission package name begins with characters other than `java` or `javax`), then a warning message is logged and the permission is not removed. You can change the value of the Issue Permission Warning flag from the administrative console in the Global Security panel. It is not recommended that you use `AllPermission` for the enterprise application.

There are some default permissions defined in the `filter.policy` file. These permissions are the minimal ones recommended by the product. If more permissions are added to the `filter.policy` file, certain operations can fail for enterprise applications. Add permissions to the `filter.policy` file carefully.

Note:

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 there are no syntax errors in the `filter.policy` file. If there are any syntax errors in `filter.policy` file, it will not be loaded by the product security run time, which implies that filtering is disabled.

An updated `filter.policy` file is applied to all of the WebSphere Application Server enterprise application after the servers are restarted.

The `filter.policy` file is managed by configuration and file replication services. Changes made in the file are replicated to other nodes in the Network Deployment cell.

The `filter.policy` file supplied by WebSphere Application Server resides at `install_root/config/cells/cell_name/filter.policy`.

It contains these permissions as defaults:

```
filterMask {
permission java.lang.RuntimePermission "exitVM";
permission java.lang.RuntimePermission "setSecurityManager";
permission java.security.SecurityPermission "setPolicy";
permission javax.security.auth.AuthPermission "setLoginConfiguration"; };
runtimeFilterMask {
permission java.lang.RuntimePermission "exitVM";
permission java.lang.RuntimePermission "setSecurityManager";
permission java.security.SecurityPermission "setPolicy";
permission javax.security.auth.AuthPermission "setLoginConfiguration"; };
```

The permissions defined in `filterMask` are for static policy filtering. The security run time tries to remove the permissions from applications during application startup. Compound permissions are not removed but are issued with a warning, and application deployment is stopped if applications contain permissions defined in `filterMask`, and if scripting was used (`wsadmin` tool). The `runtimeFilterMask` defines permissions used by the security run time to deny access to those permissions to application thread. Do not add more permissions to the `runtimeFilterMask`. Application start failure or incorrect functioning might result. Be careful when adding more permissions to the `runtimeFilterMask`. Usually, you only need to add permissions to the `filterMask` stanza.

WebSphere Application Server relies on the filter policy file to restrict or disallow certain permissions that could compromise the integrity of the system. For instance, WebSphere Application Server considers the `exitVM` and

setSecurityManager permissions as those permissions that most applications should never have. If these permissions are granted, then the following scenarios are possible:

- **exitVM** -- A servlet, JSP file, enterprise bean, or other library used by the aforementioned could call the System.exit() API and cause the entire WebSphere Application Server process to terminate.
- **setSecurityManager** -- An application could install its own SecurityManager that could either grant more permissions or bypass the default policy the WebSphere Application Server SecurityManager enforces.

For the updated filter.policy file to take effect, restart related Java processes.

Configuring the was.policy file:

Java 2 security uses several policy files to determine the granted permission for each Java program. See Dynamic policy for the list of available policy files supported by WebSphere Application Server Version 5. The was.policy file is an application-specific policy file for WebSphere Application Server enterprise applications. It is embedded in the enterprise archive (EAR) file (META-INF/was.policy). The was.policy file is located in:

```
install_root/config/cells/cell_name/applications/  
ear_file_name/deployments/application_name/META-INF/was.policy
```

The union of the permission contained in the java.policy file, the server.policy file, the app.policy file, application was.policy file and the permission specification of the ra.xml file are applied to the WebSphere Application Server enterprise application. Configuration and file replication services manage was.policy files. Changes made in these files are replicated to other nodes in the Network Deployment cell.

Several product-reserved symbols are defined to associate the permission lists to a specific type of resources.

| Symbol | Definition |
|-----------------------------|--|
| file:\${application} | file:\${application} |
| file:\${jars} | Permissions apply to all utility Java archive (JAR) files within the application |
| file:\${ejbComponent} | Permissions apply to enterprise bean resources within the application |
| file:\${webComponent} | Permissions apply to Web resources within the application |
| file:\${connectorComponent} | Permissions apply to connector resources within the application |

Important: The Signed By and the Java Authentication and Authorization Service (JAAS) principal keywords are not supported in the was.policy file. The **Signed By** keyword is supported in the following policy files: java.policy, server.policy, and client.policy. The JAAS principal keyword is supported in a JAAS policy file when it is specified by the Java Virtual Machine (JVM) system property, java.security.auth.policy. You can statically set the authorization policy files in java.security.auth.policy with auth.policy.url.n=URL where URL is the location of the authorization policy.

Other than these blocks, you can specify the module name for granular settings. For example,

```
"file:DefaultWebApplication.war" {
    permission java.security.SecurityPermission "printIdentity";
};

grant codeBase "file:IncCMP11.jar" {
    permission java.io.FilePermission
    "${user.install.root}${/}bin${/}DefaultDB${/}-",
    "read,write,delete";
};
```

There are five embedded symbols provided to specify the path and name for the `java.io.FilePermission`. These symbols enable flexible permission specification. The absolute file path is fixed after the application is installed.

| Symbol | Definition |
|--------------------------------------|---|
| <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 (union of the permissions defined in the `java.policy` file, the `server.policy` file and the `app.policy` file) are enough, no action is required. If an application has specific resources to access, update the `was.policy` file. The first two steps assume that you are creating a new policy file.

Note: Syntax errors in the policy files cause the application server to fail. Use care when editing these policy files.

1. Create or edit a new `was.policy` file using the Policy Tool.
2. Package the `was.policy` file into the enterprise archive (EAR) file. The following instructions describe how to import a `was.policy` file. However, you also can use the Assembly Toolkit to create a new file by clicking **File > New > File**.
 - a. Start the Assembly Toolkit and open the J2EE Perspective by selecting **Window > Open Perspective > J2EE**.
 - b. Import the client EAR file by selecting **File > Import > EAR file**.
 - c. Click **Next**.
 - d. Enter the path name to the EAR file in the **EAR File** field or click **Browse** to locate the file.
 - e. Enter the project name in the **Project name** field.
 - f. Click **Finish**.
 - g. Open the Project Navigator view.
 - h. 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**.
 - i. At the bottom of the Project Navigator view, click **J2EE Hierarchy**.
 - j. Import the `was.policy` file by right-clicking the **Modules** directory and clicking **Import > File system**.

- k. Click **Next**.
 - l. Enter the path name to the `was.policy` file in the **From directory** field or click **Browse** to locate the file.
 - m. Verify that the path directory listed in the **Into directory** field lists the correct META-INF directory.
 - n. Click **Finish**.
 - o. To validate the EAR file, right-click the EAR file, which contains the Modules directory, and click **Run Validation**.
 - p. 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.
 - q. 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 installed `was.policy` file with Policy Tool.

The updated `was.policy` file is applied to the application after the application restarts.

If an application must access a specific resource that is not defined as a default in the `java.policy` file, the `server.policy` file and the `app.policy`, then delete the `was.policy` file for that application. The symptom of the missing permission is that the exception, `java.security.AccessControlException`. The missing permission is listed in the exception data, `java.security.AccessControlException: access denied (java.io.FilePermission C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar read)`.

When a Java program receives this exception and adding this permission is justified, add a permission to the `was.policy` file: `grant codeBase "file:<user client installed location>" { permission java.io.FilePermission "C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar", "read"; };`

To determine whether to add a permission, refer to the article, `AccessControlException`.

Restart all applications for the updated `app.policy` file to take effect.

Configuring spi.policy files:

Java 2 security uses several policy files to determine the granted permission for each Java program. See *Dynamic policy* for the list of available policy files supported by WebSphere Application Server Version 5.

Since the default permissions for Service Provider Interface (SPI) is `AllPermission`, the only reason to update the `spi.policy` file is a restricted SPI permission. When a change in the `spi.policy` is required, complete the following steps.

Syntax errors in the policy files cause the application server to fail. Edit these policy files carefully.

Important: Do not place the `codebase` keyword or any other keyword after the `filterMask` and `runtimeFilterMask` keywords. The Signed By and the Java Authentication and Authorization Service (JAAS) Principal

keywords are not supported in the `spi.policy` file. The Signed By keyword is supported in the following policy files: `java.policy`, `server.policy`, and `client.policy`. The JAAS Principal keyword is supported in a JAAS policy file that is specified by the Java Virtual Machine (JVM) system property, `java.security.auth.policy`. You can statically set the authorization policy files in `java.security.auth.policy` with `auth.policy.url.n=URL` where `URL` is the location of the authorization policy.

Modify the `spi.policy` file with the Policy Tool.

The updated `spi.policy` is applied to the SPI libraries after the Java process is restarted.

The `spi.policy` file is the template for SPIs (Service Provider Interface) or third-party resources embedded in the product. Example of SPIs are Java Message Services (JMS) (MQSeries) and Java database connectivity (JDBC) drivers. They are specified in the `resources.xml` file. The dynamic policy grants the permissions defined in the `spi.policy` file to the class paths defined in the `resources.xml` file. The union of the permission contained in the `java.policy` file and the `spi.policy` file are applied to the SPI libraries. The `spi.policy` files are managed by configuration and file replication services. Changes made in these files are replicated to other nodes in the Network Deployment cell.

The `spi.policy` file supplied by WebSphere Application Server resides at `install_root/config/cells/cell_name/nodes/node_name/spi.policy`. It contains the following default permission:

```
grant {  
    permission java.security.AllPermission;  
};
```

Restart the related Java processes for the changes in the `spi.policy` file to become effective.

Configuring library.policy files:

Java 2 security uses several policy files to determine the granted permission for each Java programs. See Dynamic policy for the list of available policy files supported by WebSphere Application Server Version 5. The `library.policy` file is the template for shared libraries (Java library classes). Multiple enterprise applications can define and use shared libraries. Refer to Managing shared libraries for information on how to define and manage the shared libraries.

If the default permissions for a shared library (union of the permissions defined in the `java.policy` file, the `app.policy` file and the `library.policy` file) are enough, no action is required. The default library policy is picked up automatically. If a specific change is required to share a library in the cell, update the `library.policy` file.

Syntax errors in the policy files cause the application server to fail. Edit these policy files carefully.

Important: Do not place the codebase keyword or any other keyword after the grant keyword. The Signed By keyword and the Java Authentication and Authorization Service (JAAS) Principal keyword are not supported in the `library.policy` file. The Signed By keyword is supported in the

following policy files: `java.policy`, `server.policy`, and `client.policy`. The JAAS Principal keyword is supported in a JAAS policy file when it is specified by the Java Virtual Machine (JVM) system property, `java.security.auth.policy`. You can statically set the authorization policy files in `java.security.auth.policy` with `auth.policy.url.n=URL` where `URL` is the location of the authorization policy.

Modify the `library.policy` file with the Policy Tool.

An updated `library.policy` is applied to shared libraries after the servers restart.

The union of the permission contained in the `java.policy` file, the `app.policy` file, and the `library.policy` file are applied to the shared libraries. The `library.policy` file is managed by configuration and file replication services. Changes made in the file are replicated to other nodes in the Network Deployment cell.

The `library.policy` file supplied by WebSphere Application Server resides at: `install_root/config/cells/cell_name/nodes/node_name/library.policy`, contains an empty permission entry as a default. For example,

```
grant {  
};
```

If the shared library in a cell requires permissions that are not defined as defaults in the `java.policy` file, `app.policy` file and the `library.policy` file, update the `library.policy` file. The missing permission causes the exception, `java.security.AccessControlException`. The missing permission is listed in the exception data, for example:

```
java.security.AccessControlException: access denied (java.io.FilePermission  
C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar read)
```

The previous lines are one continuous line.

When a Java program receives this exception and adding this permission is justified, add a permission to the `library.policy` file, for example: `grant codeBase "file:<user client installed location>" { permission java.io.FilePermission "C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar", "read"; };`

to decide whether to add a permission, refer to `AccessControlException`.

Restart the related Java processes for the changes in the `library.policy` file to become effective.

Adding the was.policy file to applications:

When Java 2 security is enabled for a WebSphere Application Server, all the applications that run on that WebSphere Application Server undergo a security check before accessing system resources. An application might need a `was.policy` file if it accesses resources that require more permissions than those granted in the default `app.policy` file. By default, the product security reads an `app.policy` file that is located in each node and grants the permissions in the `app.policy` file to all the applications. Include any additional required permissions in the `was.policy` file. The `was.policy` file is only required if an application requires additional permissions.

The default policy file for all applications is specified in the `app.policy` file. This file is provided by the product security, is common to all applications, and should not be changed. Add any new permissions required for an application in the `was.policy` file.

The `app.policy` file is located in the `install_root/config/cells/cell_name/nodes/node_name` directory. The contents of the `app.policy` file follow:

Attention: In the following code sample, the two permissions that are required by JavaMail were split into two lines each due to the width of the printed page.

```
// The following permissions apply to all the components under the application.
grant codeBase "file:${application}" {
    // The following are required by JavaMail
    permission java.io.FilePermission "
        ${was.install.root}${/}java${/}jre${/}lib${/}ext${/}mail.jar", "read";
    permission java.io.FilePermission "
        ${was.install.root}${/}java${/}jre${/}lib${/}ext${/}activation.jar", "read";
};

// The following permissions apply to all utility .jar files (other
// than enterprise beans JAR files) in the application.
grant codeBase "file:${jars}" {
    permission java.net.SocketPermission "*", "connect";
    permission java.util.PropertyPermission "*", "read";
};

// The following permissions apply to connector resources within the application
grant codeBase "file:${connectorComponent}" {
    permission java.net.SocketPermission "*", "connect";
    permission java.util.PropertyPermission "*", "read";
};

// The following permissions apply to all the Web modules (.war files)
// within the application.
grant codeBase "file:${webComponent}" {
    permission java.io.FilePermission "${was.module.path}${/}-", "read, write";
    // where "was.module.path" is the path where the Web module is
    // installed. Refer to Dynamic policy concepts for other symbols.
    permission java.lang.RuntimePermission "loadLibrary.*";
    permission java.lang.RuntimePermission "queuePrintJob";
    permission java.net.SocketPermission "*", "connect";
    permission java.util.PropertyPermission "*", "read";
};

// The following permissions apply to all the EJB modules within the application.
grant codeBase "file:${ejbComponent}" {
    permission java.lang.RuntimePermission "queuePrintJob";
    permission java.net.SocketPermission "*", "connect";
    permission java.util.PropertyPermission "*", "read";
};
```


If additional permissions are required for an application or for one or more modules of an application, use the `was.policy` file for that application. For example, use `codeBase` of `${application}` and add required permissions to grant additional permissions to the entire application. Similarly, use `codeBase` of `${webComponent}` and `${ejbComponent}` to grant additional permissions to all the Web modules and all the enterprise bean (EJB) modules in the application. You can assign additional permissions to each module (`.war` file or `.jar` file) as shown in the following example.

An example of adding extra permissions for an application in the `was.policy` file:

Attention: In the following code sample, the permission for the EJB module was split into two lines due to the width of the printed page.

```
// grant additional permissions to a Web module
grant codeBase " file:aWebModule.war" {
    permission java.security.SecurityPermission "printIdentity";
};

// grant additional permission to an EJB module
grant codeBase "file:aEJBModule.jar" {
    permission java.io.FilePermission "
        ${user.install.root}${/}bin${/}DefaultDB${/}-" ."read.write,delete";
    // where, ${user.install.root} is the system property whose value is
    // located in the <install_root> directory.
};
```

1. Create a `was.policy` file using the policy tool. For more information on using the policy tool, see *Using PolicyTool* to edit policy files
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 the Assembly Toolkit .
 - a. Verify that the `was.policy` file in the application is syntactically correct. In the Assembly Toolkit, right-click the enterprise application module and click **Run Validation**.

An application EAR file is now ready to run when Java 2 security is enabled.

This step is required for applications to run properly when Java 2 security is enabled. If the `was.policy` file is not created and it does not contain required permissions, the application might not access system resources.

The symptom of the missing permissions is the exception, `java.security.AccessControlException`. The missing permission is listed in the exception data, for example:

```
java.security.AccessControlException: access denied (java.io.FilePermission
C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar read)
```

The previous two lines are one continuous line.

When an application program receives this exception and adding this permission is justified, include the permission in the `was.policy` file, for example,

```
grant codeBase "file:${application}" { permission java.io.FilePermission
"C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar", "read"; };
```

The previous two lines are one continuous line.

Install the application.

Configuring static policy files:

Java 2 security uses several policy files to determine the granted permission for each Java program. See the Dynamic Policy article for the list of available policy files supported by WebSphere Application Server Version 5.

There are two types of policy files supported by WebSphere Application Server Version 5, dynamic policy files and static policy files. Static policy files provide the default permissions. Dynamic policy files provide application's permissions.

| Policy file name | Description |
|------------------|--|
| java.policy | Contains default permissions for all of the Java programs on the node. This file seldom changes. |
| server.policy | Contains default permissions for all of the WebSphere Application Server programs on the node. This files is rarely updated. |
| client.policy | Contains default permissions for all of the applets and client containers on the node. |

The static policy file is not a configuration file managed by the repository and the file replication service. Changes to this file are local and do not get replicated to the other machine.

1. Identify the policy file to update.
 - If the permission is required only by an application, update the dynamic policy file. Refer to Configuring Java 2 security policy files.
 - If the permission is required only by applets and client containers, update the client.policy file. Refer to Configuring client.policy files.
 - If the permission is required only by WebSphere Application Server (servers, agents, managers and application servers), update the server.policy file. Refer to Configuring server.policy files.
 - If the permission is required by all of the Java programs running on the Java virtual machine (JVM), update the java.policy file. Refer to Configuring java.policy files.
2. Stop and restart the WebSphere Application Server.

The required permission is granted for all of the Java programs running with the restarted JVM.

If Java programs on a node require permissions, the policy file needs updating. If the Java program that required the permission is not part of an enterprise application, update the static policy file. The missing permission causes the exception, java.security.AccessControlException. The missing permission is listed in the exception data, for example:

```
java.security.AccessControlException: access denied (java.io.FilePermission
C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar read)
```

When a Java program receives this exception and adding this permission is justified, add a permission to an adequate policy file, for example:

```
grant codeBase "file:<user client installed location>" {
    permission java.io.FilePermission
        "C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar",
        "read";
};
```

To decide whether to add a permission, refer to `AccessControlException`.

Configuring java.policy files:

Java 2 security uses several policy files to determine the granted permission for each Java program. See *Dynamic policy* for the list of available policy files supported by WebSphere Application Server Version 5. The `java.policy` file is a global default policy file shared by all of the Java programs running in the Java Virtual Machine (JVM) on the node. Modifying this file is not recommended. If a specific change is required to some of the Java programs on a node and the `java.policy` file requires updating, modify the `java.policy` file with `policy tool`. A change to the `java.policy` file is local for the node. The default Java policy is picked up automatically. Syntax errors in the policy files cause the application server to fail. Edit these policy files carefully.

An updated `java.policy` file is applied to all the Java programs running in all the JVMs on the local node. Restart the programs for the updates to take effect

The `java.policy` file is not a configuration file managed by the repository and the file replication service. Changes to this file are local and do not get replicated to the other machine. The `java.policy` file supplied by WebSphere Application Server is located at `install_root/java/jre/lib/security/java.policy`. It contains these default permissions.

```
// Standard extensions get all permissions by default
grant codeBase "file:${java.home}/lib/ext/*" {
    permission java.security.AllPermission;
};
// default permissions granted to all domains
grant {
    // Allows any thread to stop itself using the java.lang.Thread.stop()
    // method that takes no argument.
    // Note that this permission is granted by default only to remain
    // backwards compatible.
    // It is strongly recommended that you either remove this permission
    // from this policy file or further restrict it to code sources
    // that you specify, because Thread.stop() is potentially unsafe.
    // See "http://java.sun.com/notes" for more information.
    // permission java.lang.RuntimePermission "stopThread";

    // allows anyone to listen on un-privileged ports
    permission java.net.SocketPermission "localhost:1024-", "listen";

    // "standard" properties that can be read by anyone
```

```

permission java.util.PropertyPermission "java.version", "read";
permission java.util.PropertyPermission "java.vendor", "read";
permission java.util.PropertyPermission "java.vendor.url", "read";
permission java.util.PropertyPermission "java.class.version", "read";
permission java.util.PropertyPermission "os.name", "read";
permission java.util.PropertyPermission "os.version", "read";
permission java.util.PropertyPermission "os.arch", "read";
permission java.util.PropertyPermission "file.separator", "read";
permission java.util.PropertyPermission "path.separator", "read";
permission java.util.PropertyPermission "line.separator", "read";

permission java.util.PropertyPermission "java.specification.version", "read";
permission java.util.PropertyPermission "java.specification.vendor", "read";
permission java.util.PropertyPermission "java.specification.name", "read";

permission java.util.PropertyPermission "java.vm.specification.version", "read";
permission java.util.PropertyPermission "java.vm.specification.vendor", "read";
permission java.util.PropertyPermission "java.vm.specification.name", "read";
permission java.util.PropertyPermission "java.vm.version", "read";
permission java.util.PropertyPermission "java.vm.vendor", "read";
permission java.util.PropertyPermission "java.vm.name", "read";
};

```

If some Java programs on a node require permissions that are not defined as defaults in the `java.policy` file, then consider updating the `java.policy` file. Most of the time, other policy files are updated instead of the `java.policy` file. The missing permission causes the exception, `java.security.AccessControlException`. The missing permission is listed in the exception data, for example:

```

java.security.AccessControlException: access denied (java.io.FilePermission
C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar read)

```

The previous two lines are one continuous line.

When a Java program receives this exception and adding this permission is justified, add a permission to the `java.policy` file, for example:

```

grant codeBase "file:<user client installed location>" {
permission java.io.FilePermission
"C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar", "read"; };

```

To decide whether to add a permission, refer to `AccessControlException`.

Restart all of the Java processes for the updated `java.policy` file to take effect.

Configuring server.policy files:

Java 2 security uses several policy files to determine the granted permission for each Java program. See *Dynamic policy* for the list of available policy files supported by WebSphere Application Server Version 5. The `server.policy` file is a default policy file shared by all of the WebSphere servers on a node. The `server.policy` file is not a configuration file managed by the repository and the file replication service. Changes to this file are local and do not replicate to the other machine.

If the default permissions for a server (the union of the permissions defined in the `server.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 the policy tool article to edit policy files. Changes to the `server.policy` file are local for the node. Syntax errors in the policy files cause the application server to fail. Edit these policy files carefully.

An updated `server.policy` file is applied to all the server programs on the local node. Restart the servers for the updates to take effect.

If you want to add permissions to an application, use the `app.policy` file and the `was.policy` file.

When you do need to modify the `server.policy` file, locate this file at: `install_root/properties/server.policy`. This file contains these default permissions:

```
// Allow to use sun tools
grant codeBase "file:${java.home}/../lib/tools.jar" {
    permission java.security.AllPermission;
};

// WebSphere system classes
grant codeBase "file:${was.install.root}/lib/-" {
    permission java.security.AllPermission;
};
grant codeBase "file:${was.install.root}/classes/-" {
    permission java.security.AllPermission;
};

// Allow the WebSphere deploy tool all permissions
grant codeBase "file:${was.install.root}/deploytool/-" {
    permission java.security.AllPermission;
};
```

If some server programs on a node require permissions that are not defined as defaults in the `server.policy` file and the `server.policy` file, update the `server.policy` file. The missing permission causes the exception, `java.security.AccessControlException`. The missing permission is listed in the exception data, for example:

```
java.security.AccessControlException: access denied (java.io.FilePermission
C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar read)
```

The previous two lines are one continuous line.

When a Java program receives this exception and adding this permission is justified, add a permission to the `server.policy` file, for example:

```
grant codeBase "file:<user client installed location>" {
    permission java.io.FilePermission
"C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar", "read"; };
```

To decide whether to add a permission, refer to `AccessControlException`.

Restart all of the Java processes for the updated `server.policy` file to take effect.

Configuring client.policy files:

Java 2 security uses several policy files to determine the granted permission for each Java program. See *Dynamic policy* for the list of available policy files supported by WebSphere Application Server Version 5. The `client.policy` file is a default policy file shared by all of the WebSphere Application Server client containers and applets on a node. The union of the permissions contained in the `client.policy` file and the `client.policy` file are given to all of the WebSphere client containers and applets running on the node. The `client.policy` file is not a configuration file managed by the repository and the file replication service. Changes to this file are local and do not replicate to the other machine. The `client.policy` file supplied by WebSphere Application Server is located at `install_root/properties/client.policy`. It contains these default permissions:

```
grant codeBase "file:${java.home}/lib/ext/*" {
    permission java.security.AllPermission;
};
// IBM Developer Kit, Java Technology Edition classes
grant codeBase "file:${java.home}/lib/ext/-" {
    permission java.security.AllPermission;
};
grant codeBase "file:${java.home}/../lib/tools.jar" {
    permission java.security.AllPermission;
};
// WebSphere system classes
grant codeBase "file:${was.install.root}/lib/-" {
    permission java.security.AllPermission;
};
grant codeBase "file:${was.install.root}/classes/-" {
    permission java.security.AllPermission;
};
grant codeBase "file:${was.install.root}/installedConnectors/-" {
    permission java.security.AllPermission;
};
// J2EE 1.3 permissions for client container WAS applications
// in $WAS_HOME/installedApps
grant codeBase "file:${was.install.root}/installedApps/-" {
    //Application client permissions
    permission java.awt.AWTPermission "accessClipboard";
    permission java.awt.AWTPermission "accessEventQueue";
    permission java.awt.AWTPermission "showWindowWithoutWarningBanner";
    permission java.lang.RuntimePermission "exitVM";
    permission java.lang.RuntimePermission "loadLibrary";
    permission java.lang.RuntimePermission "queuePrintJob";
    permission java.net.SocketPermission "*", "connect";
    permission java.net.SocketPermission "localhost:1024-", "accept,listen";
    permission java.io.FilePermission "*", "read,write";
    permission java.util.PropertyPermission "*", "read";
};
// J2EE 1.3 permissions for client container - expanded ear file code base
grant codeBase "file:${com.ibm.websphere.client.applicationclient.archivedir}/-"
{
    permission java.awt.AWTPermission "accessClipboard";
    permission java.awt.AWTPermission "accessEventQueue";
};
```

```

permission java.awt.AWTPermission "showWindowWithoutWarningBanner";
permission java.lang.RuntimePermission "exitVM";
permission java.lang.RuntimePermission "loadLibrary";
permission java.lang.RuntimePermission "queuePrintJob";
permission java.net.SocketPermission "*", "connect";
permission java.net.SocketPermission "localhost:1024-", "accept,listen";
permission java.io.FilePermission "*", "read,write";
permission java.util.PropertyPermission "*", "read";
};
// For MQ Series
grant codeBase "file:${mq.install.root}/java/*" {
    permission java.security.AllPermission;
};

```

1. If the default permissions for a client (union of the permissions defined in the `client.policy` file and the `client.policy` file) are enough, no action is required. The default client policy is picked up automatically.
2. If a specific change is required to some of the client containers and applets on a node, modify the `client.policy` file with the policy tool. Refer to Using the policy tool, to edit policy files. Changes to the `client.policy` file are local for the node.

All of the client containers and applets on the local node are granted the updated permissions at the time of execution.

If some client containers or applets on a node require permissions that are not defined as defaults in the `client.policy` file and the default `client.policy` file, update the `client.policy` file. The missing permission causes the exception, `java.security.AccessControlException`. The missing permission is listed in the exception data, for example,

```

java.security.AccessControlException: access denied (java.io.FilePermission
C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar read)

```

The previous two lines of sample code are one continuous line, but extended beyond the width of the page.

When a client program receives this exception and adding this permission is justified, add a permission to the `client.policy` file, for example, grant codebase `"file:user_client_installed_location"` { permission java.io.FilePermission `"C:\WebSphere\AppServer\java\jre\lib\ext\mail.jar", "read"; }`};

To decide whether to add a permission, refer to `AccessControlException`.

Close and restart the browser. You also must restart the client application if you have one.

Migrating Java 2 security policy

Previous WebSphere Application Server releases

Starting from Version 3.x, WebSphere Application Server installed a Java 2 security manager in the server run time to prevent enterprise applications from calling the `System.exit()` and the `System.setSecurityManager()` methods. These two Java APIs have undesirable consequences if called by enterprise applications. The

`System.exit()` API, for example, causes the Java virtual machine (application server process) to exit prematurely, which is an undesirable operation for an application server.

However, Java 2 security was not a fully supported feature prior to Version 5. To support Java 2 security properly, all the server run time must be marked as privileged (with `doPrivileged()` API calls inserted in the correct places), and identify the default permission sets or policy. Application code is not privileged and subject to the permissions defined in the policy files. The `doPrivileged` instrumentation is important and necessary to support Java 2 security. Without it, the application code must be granted the permissions required by the server run time. This is due to the design and algorithm used by Java 2 security to enforce permission checks. Please refer to the Java 2 security check permission algorithm.

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

- `java.lang.RuntimePermission(exitVM)`
- `java.lang.RuntimePermission(setSecurityManager)`

Application code is denied access to these permissions regardless of what is in the Java 2 security policy. However, the server run time is granted these permissions. All the other permission checks are not enforced.

Partial support was introduced since the version 4.02 product release. Prior to version 4.0.2, Java 2 security was not supported. From version 4.02 and later, only two permissions are supported:

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

However, not all the product server run time is properly marked as privileged. You must grant the application code all the other permissions besides the two listed previously or the enterprise application can potentially fail to run. This Java 2 security policy for enterprise applications is liberal.

What changed

Java 2 Security is fully supported in version 5, which means all permissions are enforced. The default Java 2 security policy for enterprise application is the recommended permission set defined by the J2EE 1.3 specification. Refer to the `${install_root}/config/cells/cell_name/nodes/node_name/app.policy` file for the default Java 2 security policy granted to enterprise applications. This is a much more stringent policy compared to previous releases.

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

Note: Enterprise applications that run on Version 4.0.x with Java 2 security enabled are not guaranteed to run successfully when migrating to Version 5 (when Java 2 security is enabled), even if the Java 2 security policy is migrated properly. The default Java 2 security policy for enterprise applications is much more stringent and all permissions are enforced in Version 5. It might fail because the application code does not have the

necessary permissions granted where system resources (such as file I/O for example) can be programmatically accessed and are now subject to the permission checking.

Migrating system properties

The following system properties are used in previous releases in relation to Java 2 security:

- **java.security.policy.** The absolute path of the policy file (action required). It contains both system permissions (permissions granted to the Java Virtual Machine (JVM) and the product server run time) and enterprise application permissions. Migrate the Java 2 security policy of the enterprise application to Version 5. For Java 2 security policy migration, see the steps for migrating Java 2 security policy.
- **enableJava2Security.** Used to enable Java 2 security enforcement (no action required). This is deprecated; a flag in the WebSphere common configuration model (WCCM) is used to control whether to enabled Java 2 security. Enable this option through the administrative console.
- **was.home.** Expanded to the installation directory of the WebSphere Application Server (action might be required). This is deprecated; superseded by `${user.install.root}` and `${was.install.root}` properties. If the directory contains instance specific data then `${user.install.root}` is used; otherwise `${was.install.root}` is used. Use these properties interchangeably for the WebSphere Application Server or the Network Deployment environments. See the steps for migrating Java 2 security policy.

Migrating the Java 2 Security Policy

There is no easy way of migrating the Java policy file from Version 4.0.x automatically because there is a mixture of system permissions and application permissions in the same policy file. Manually copy the Java 2 security policy for enterprise applications to a `was.policy` or `app.policy` file. However, migrating the Java 2 security policy to a `was.policy` file is preferable because symbols or relative codebase is used instead of absolute codebase. There are many advantages to this process. The permissions defined in the `was.policy` file should only be granted to the specific enterprise application, while permissions in the `app.policy` file apply to all the enterprise applications running on the node where the `app.policy` file belongs. Refer to the Dynamic policy article for more details on policy management.

The following example illustrates the migration of a Java 2 security policy from a previous release. The contents include the Java 2 security policy file (the default is `install_root/properties/java.policy`) for the `app1.ear` enterprise application and the system permissions (permissions granted to the JVM and product server run time). Default permissions are omitted for clarity:

```
// For product Samples
grant codeBase "file:${install_root}/installedApps/app1.ear/-" {
    permission java.security.SecurityPermission "printIdentity";
    permission java.io.FilePermission "${install_root}${/}temp${/}somefile.txt", "read";
};
```

For clarity of illustration, all the permissions are migrated as the application level permissions in this example. However, you can grant permissions at a more

granular level at the component level (Web, enterprise beans, connector or utility Java archive (JAR) component level) or you can grant permissions to a particular component.

1. Ensure that Java 2 security is disabled on the application server.
2. Create a new `was.policy` file (if one is not present) or update the `was.policy` for migrated applications in the configuration repository in `(config/cells/<cell_name>/applications/app.ear/deployments/app/META-INF/was.policy)` with the following contents:

```
grant codeBase "file:${application}" {  
    permission java.security.SecurityPermission "printIdentity";  
    permission java.io.FilePermission "  
        ${user.install.root}${/}temp${/}somefile.txt", "read";  
};
```

The third and fourth lines in the previous code sample are one continuous line, but extended beyond the width of the page.

3. **5.1+** Use the Assembly Toolkit to attach the `was.policy` file to the enterprise archive (EAR) file. You also can use the Assembly Toolkit to validate the contents of the `was.policy` file. For more information, see [Configuring was.policy](#).
4. Validate that the enterprise application does not require additional permissions to the migrated Java 2 Security permissions and the default permissions set declared in the `${was.install.root}/config/cells/cell_name/nodes/node_name/app.policy` file. This requires code review, code inspection, application documentation review, and sandbox testing of migrated enterprise applications with Java 2 security enabled in a pre-production environment. Refer to developer kit APIs protected by Java 2 security for information about which APIs are protected by Java 2 security. If you use third party libraries, consult the vendor documentation for APIs that are protected by Java 2 security. Verify that the application is granted all the required permissions, or it might fail to run when Java 2 security is enabled.
5. Perform pre-production testing of the migrated enterprise application with Java 2 security enabled. **Hint:** Enable trace for the WebSphere Application Server Java 2 security manager in the pre-production testing environment (with trace string: `com.ibm.ws.security.core.SecurityManager=all=enabled`). This can be helpful in debugging the `AccessControlException` exception thrown when an application is not granted the required permission or some system code is not properly marked as *privileged*. The trace dumps the stack trace and permissions granted to the classes on the call stack when the exception is thrown.

Note: Because the Java 2 security policy is much more stringent compared with previous releases, it is strongly advised that the administrator or deployer review their enterprise applications to see if extra permissions are required before enabling Java 2 security. If the enterprise applications are not granted the required permissions, they fail to run.

Troubleshooting security configurations

Refer to [Security components troubleshooting tips](#) for instructions on how to troubleshoot errors related to security.

The following topics explain how to troubleshoot specific problems related to configuring and enabling security configurations:

- Errors when configuring or enabling security
- Errors or access problems after enabling security
- Errors after enabling Secure Sockets Layer (SSL) or SSL-related error messages

Tuning security configurations

Performance issues typically involve trade-offs between function and speed. Usually, the more function and the more processing involved, the slower the performance. Consider what type of security is necessary and what you can disable in your environment. For example, if your application servers are running in a Virtual Private Network (VPN), consider whether you must disable Single Sockets Layer (SSL). If you have a lot of users, can they be mapped to groups and then associated to your J2EE roles? These questions are things to consider when designing your security infrastructure.

Complete the following steps for general security tuning:

1. Consider disabling Java 2 Security Manager if you know exactly what code is put onto your server and you do not need to protect process resources. Remember that in doing so, you put your local resources at some risk.
2. Disable security for the specific application server that does not require resource protection because some application servers do not have protected resources. If the application server needs to go downstream with credentials, however, this action might not be feasible.
3. Consider propagating new security settings to all nodes before restarting the deployment manager and node agents to change the new security policy. If your security configurations are not consistent across all servers, you get access denied errors. Therefore, you must propagate new security settings when enabling or disabling global security in a Network Deployment environment.

Configuration changes are generally propagated using configuration synchronization. If auto-synchronization is enabled, you can wait for the automatic synchronization interval to pass, or you can force synchronization before the synchronization interval expires. If you are using manual synchronization, you must synchronize all nodes.

If the cell is in a configuration state (the security policy is mixed with nodes that have security enabled and disabled) you can use the `syncNode` utility to synchronize the nodes where the new settings are not propagated.

Refer to the article, [Enabling and disabling global security in the WebSphere Application Server Network Deployment package](#) for more detailed information about enabling security in a distributed environment.

4. Consider increasing the cache and token time-out if you feel your environment is secure enough. By doing so, you have to re-authenticate less often. This action supports subsequent requests to reuse the credentials that already are created. The downside of increasing the token time-out is the exposure of having a token hacked and providing the hacker more time to hack into the system before the token expires. You can use security cache properties to determine the initial size of the primary and secondary hashtable caches, which affect the frequency of rehashing and the distribution of the hash algorithms. See the article [Security cache types and sizes](#) for a list of these properties.
5. Consider changing your administrative connector from Simple Object Access Protocol (SOAP) to Remote Method Invocation (RMI) because RMI uses stateful

connections while SOAP is completely stateless. Run a benchmark to determine if the performance is improved in your environment.

6. Use the `wsadmin` script to complete the access IDs for all the users and or groups to speed up the application startup. Complete this action if applications contain many users, or groups, or if applications are stopped and started frequently.
7. Consider tuning the Object Request Broker (ORB) because it is a factor in enterprise bean performance with or without security enabled. Refer to the article, ORB tuning guidelines.

Tuning CSiv2

1. Consider using SSL client certificates instead of a user ID and password to authenticate Java clients. Since you are already making the SSL connection, using mutual authentication adds little overhead while removing the service context containing the user ID and password completely.
2. If you send a large amount of data that is not very security sensitive, reduce the strength of your ciphers. The more data you have to bulk encrypt and the stronger the cipher, the longer this action takes. If the data is not sensitive, do not waste your processing with 128-bit ciphers.
3. Consider putting just an asterisk (*) in the trusted server ID list (meaning trust all servers) when you use Identity Assertion for downstream delegation. Use SSL mutual authentication between servers to provide this trust. Adding this extra step in the SSL handshake performs better than having to fully authenticate the upstream server and check the trusted list. When an asterisk is used, we simply trust the identity token. The SSL connection trusts the server by way of client certificate authentication.
4. Ensure that stateful sessions are enabled for Common Secure Interoperability Version 2 (CSiv2). This is the default, but only requires authentication on the first request and any subsequent token expirations.
5. If you are only communicating with WebSphere Application Server Version 5 servers, make the Active Authentication Protocol CSI, instead of CSI and SAS. This action removes an interceptor invocation for every request on both the client and server sides.
6. For a pure Java client, you can disable the creation of server sockets used for Object Request Broker (ORB) callbacks. Do this only if you are communicating with servers running WebSphere Application Server, Version 5 or later.
 - a. In the `sas.client.props` file, add
`com.ibm.CSI.claimTransportAssocSSLTLSRequired=false` and
`com.ibm.CSI.claimTransportAssocSSLTLSSupported=false`.
 - b. Set the active protocol to `csiv2` instead of both in the `sas.client.props` file. The protocol property changes to `com.ibm.CSI.protocol=csiv2`.

Tuning LDAP authentication

1. Select the **Ignore Case** check box in the WebSphere Application Server LDAP User Registry configuration, when case-sensitivity is not important.
2. Select **Reuse Connections** in the WebSphere Application Server LDAP User Registry configuration.
3. Check to see which caches your LDAP server has and take advantage of them. This action is best with LDAP servers that do not change frequently.
4. Choose the directory type of either `IBM_Directory_Server` or `SecureWay`, if you are using an IBM Directory Server. The IBM Directory Server yields improved performance because it is programmed to use the new group membership

attributes to improve group membership searches. However, it is required that authorization is case insensitive to use IBM Directory Server.

5. Choose either iPlanet Directory Server (also known as Sun ONE) or Netscape as the directory if you are an iPlanet Directory user. Using the iPlanet Directory Server directory increases performance in group membership lookup. However, only use **Role** for group mechanisms.

Tuning Web authentication

1. Consider increasing the cache and token time-out if you feel your environment is secure enough. The Web authentication information is stored in these caches and as long as the authentication information is in the cache, the login module is not invoked to authenticate the user. This supports subsequent requests to reuse the credentials already created. The downside of increasing the token time-out is the exposure of having a token stolen and providing the thief more time to hack into the system before the token expires. See the article Security cache types and sizes for a list of these properties.
2. Consider enabling single signon (SSO). SSO is only available when you select **LTPA** as the authentication mechanism in the **Global Security** panel. When you select SSO, a single authentication to one application server is enough to make requests to multiple application servers in the same SSO domain. There are some situations where SSO is not desirable and should not be used in those situations.

Tuning authorization

1. Consider mapping your users to groups in the user registry. Then associate the groups with your J2EE roles. This association greatly improves performance as the number of users increases.
2. Judiciously assign method-permissions for enterprise beans. For example, you can use an asterisk (*) to indicate all methods in the method-name element. When all the methods in enterprise beans require the same permission, use an asterisk (*) for the method-name to indicate all methods. This indication reduces the size of deployment descriptors and reduces the memory required to load the deployment descriptor. It also reduces the search time during method-permission match for the enterprise beans method.
3. Judiciously assign security-constraints for servlets. For example, you can use the URL pattern *.jsp to apply the same authentication data constraints to indicate all JSP files. For a given URL, the exact match in the deployment descriptor takes precedence over the longest path match. Use the extension match (*.jsp, *.do, *.html) if there is no exact match and longest path match for a given URL in the security constraints.

There is always a trade off between performance, feature and security. Security typically adds more processing time to your requests, but for a good reason. Not all security features are required in your environment. When you decide to tune security, you should create a benchmark before making any change to ensure the change is improving performance.

In a large scale deployment, performance is very important. Running benchmark measurements with different combinations of features can help you to determine the best performance versus the benefit configuration for your environment.

Continue to run benchmarks if anything changes in your environment, to help determine the impact of these changes.

Security cache properties

The following system properties determine the initial size of the primary and secondary hash table caches, which affect the frequency of rehashing and the distribution of the hash algorithms. The larger the number of available hash values, the less likely a hash collision occurs, retrieval time might be slower. If several entries compose a hash table cache, creating the table in a larger capacity supports more efficient hash entries than letting automatic rehashing determine the growth of the table. Rehashing causes every entry to move each time.

com.ibm.websphere.security.util.authCacheSize

This cache stores basic authentication credentials at the security server. Whenever a Lightweight Third Party Authentication (LTPA) token expires, a new token generates from the basic authorization credentials in this cache. If basic authorization credentials do not exist, the requesting browser must send the basic authorization credentials to the security server. The browser prompts the user for a user ID and password if a cookie containing the credentials does not exist.

com.ibm.websphere.security.util.tokenCacheSize

This cache stores LTPA credentials in the cache using the LTPA token as a lookup value. When using an LTPA token to log in, the LTPA credential is created at the security server for the first time. This cache prevents the need to go to the security server on subsequent logins using an LTPA token.

com.ibm.websphere.security.util.CredentialCacheSize

Given the user ID and password for login, this cache returns the concrete credential object (either LocalOS or LTPA) without the need to repeat authentication at the security server. If the credential object has expired, repeat authentication is required.

com.ibm.websphere.security.util.LTPAValidationCacheSize

Given the credential token for login, this cache returns the concrete LTPA credential object, without the need to revalidate at the security server. If the token has expired, revalidation is required.

Secure Sockets Layer performance tips

The following are two types of Secure Sockets Layer (SSL) performance:

- Handshake
- Bulk encryption and decryption

When an SSL connection is established, an SSL handshake occurs. After a connection is made, SSL performs bulk encryption and decryption for each read-write. The performance cost of an SSL handshake is much larger than that of bulk encryption and decryption.

To enhance SSL performance, decrease the number of individual SSL connections and handshakes.

Decreasing the number of connections increases performance for secure communication through SSL connections, as well as non-secure communication through simple TCP/IP connections. One way to decrease individual SSL connections is to use a browser that supports HTTP 1.1. Decreasing individual SSL connections can be impossible if you cannot upgrade to HTTP 1.1.

Another common approach is to decrease the number of connections (both TCP/IP and SSL) between two WebSphere Application Server components. The following

guidelines help to verify the HTTP transport of the application server is configured so that the Web server plug-in does not repeatedly reopen new connections to the application server:

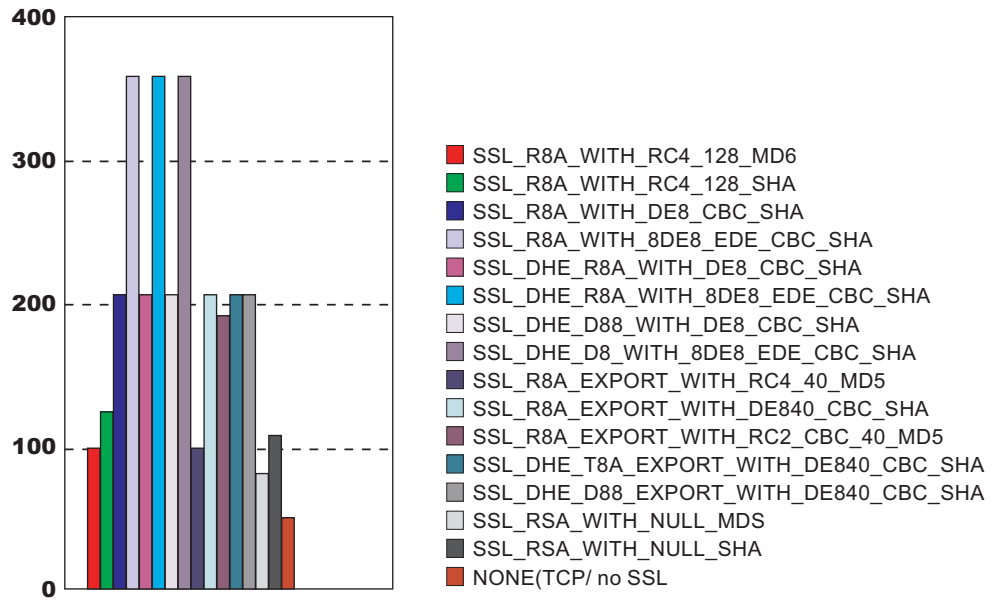
- Verify that the maximum number of keep alives are, at minimum, as large as the maximum number of requests per thread of the Web server (or maximum number of processes for IBM HTTP Server on UNIX). Make sure that the Web server plug-in is capable of obtaining a keep alive connection for every possible concurrent connection to the application server. Otherwise, the application server closes the connection after a single request is processed. Also, the maximum number of threads in the Web container thread pool should be larger than the maximum number of keep alives, to prevent the keep alive connections from consuming the Web container threads.
- Increase the maximum number of requests per keep alive connection. The default value is 100, which means the application server closes the connection from the plug-in after 100 requests. The plug-in then has to open a new connection. The purpose of this parameter is to prevent denial of service attacks when connecting to the application server and preventing continuous send requests to tie up threads in the application server.
- Use a hardware accelerator if the system performs several SSL handshakes. Hardware accelerators currently supported by WebSphere Application Server only increase the SSL handshake performance, not the bulk encryption and decryption. An accelerator typically only benefits the Web server because Web server connections are short-lived. All other SSL connections in WebSphere Application Server are long-lived.
- Use an alternative cipher suite with better performance.

The performance of a cipher suite is different with software and hardware. Just because a cipher suite performs better in software does not mean a cipher suite will perform better with hardware. Some algorithms are typically inefficient in hardware (for example, DES and 3DES), however, specialized hardware can provide efficient implementations of these same algorithms.

The performance of bulk encryption and decryption is affected by the cipher suite used for an individual SSL connection. The following chart displays the performance of each cipher suite. The test software calculating the data was Java Secure Socket Extension (JSSE) for both the client and server software, which used no crypto hardware support. The test did not include the time to establish a connection, but only the time to transmit data through an established connection. Therefore, the data reveals the relative SSL performance of various cipher suites for long running connections.

Before establishing a connection, the client enables a single cipher suite for each test case. After the connection is established, the client times how long it takes to write an integer to the server and for the server to write the specified number of bytes back to the client. Varying the amount of data had negligible effects on the

relative performance of the cipher suites.



An analysis of the above data reveals the following:

- Bulk encryption performance is only affected by what follows the WITH in the cipher suite name. This is expected since the portion before the WITH identifies the algorithm used only during the SSL handshake.
- MD5 and SHA are the two hash algorithms used to provide data integrity. MD5 is 25% faster than SHA, however, SHA is more secure than MD5.
- DES and RC2 are slower than RC4. Triple DES is the most secure, but the performance cost is high when using only software.
- The cipher suite providing the best performance while still providing privacy is SSL_RSA_WITH_RC4_128_MD5. Even though SSL_RSA_EXPORT_WITH_RC4_40_MD5 is cryptographically weaker than RSA_WITH_RC4_128_MD5, the performance for bulk encryption is the same. Therefore, as long as the SSL connection is a long-running connection, the difference in the performance of high and medium security levels is negligible. It is recommended that a security level of high be used, instead of medium, for all components participating in communication only among WebSphere Application Server products. Make sure that the connections are long running connections.

Tuning security

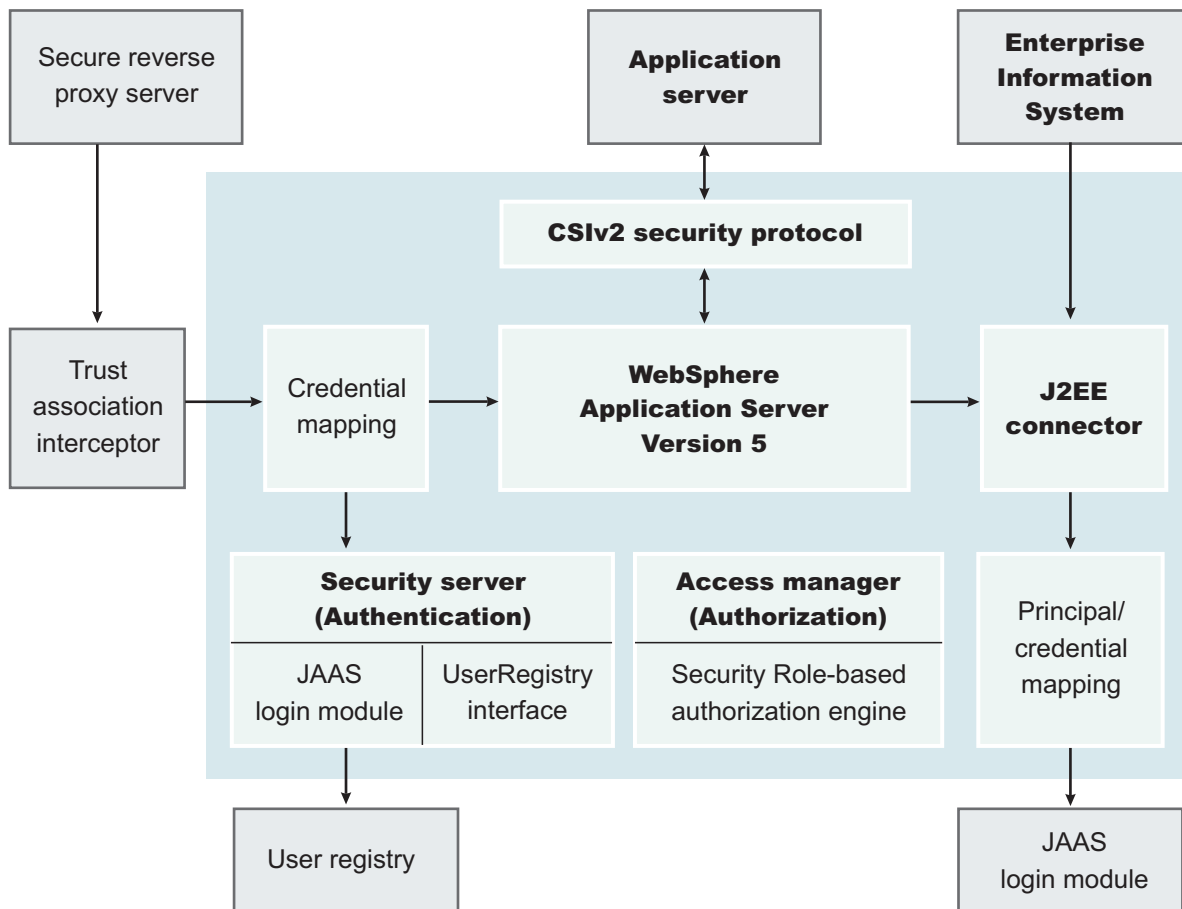
Enabling security decreases performance. The following tuning parameters give you considerations for increasing performance.

- **Disable security** in Global security settings.
- Fine tune **Cache time-out** in Global security settings.
- **Security cache properties.**
- **Tracking session timeout with SSL.**
- You can also read about Secure Sockets Layer performance tips and Tuning security configurations.

Chapter 3. Integrating IBM WebSphere Application Server security with existing security systems

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

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



A list of terms used in this discussion follows:

Protocol firewall

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

WebSphere Application Server plug-in

Redirects all the requests for servlets and JSP pages. Also referred to in

WebSphere Application Server literature as *Web server redirector* was introduced to separate Web server from application server. The advantage of using Web server redirector is that you can move an application server and all the application business logic behind the domain firewall.

Domain firewall

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

Directory

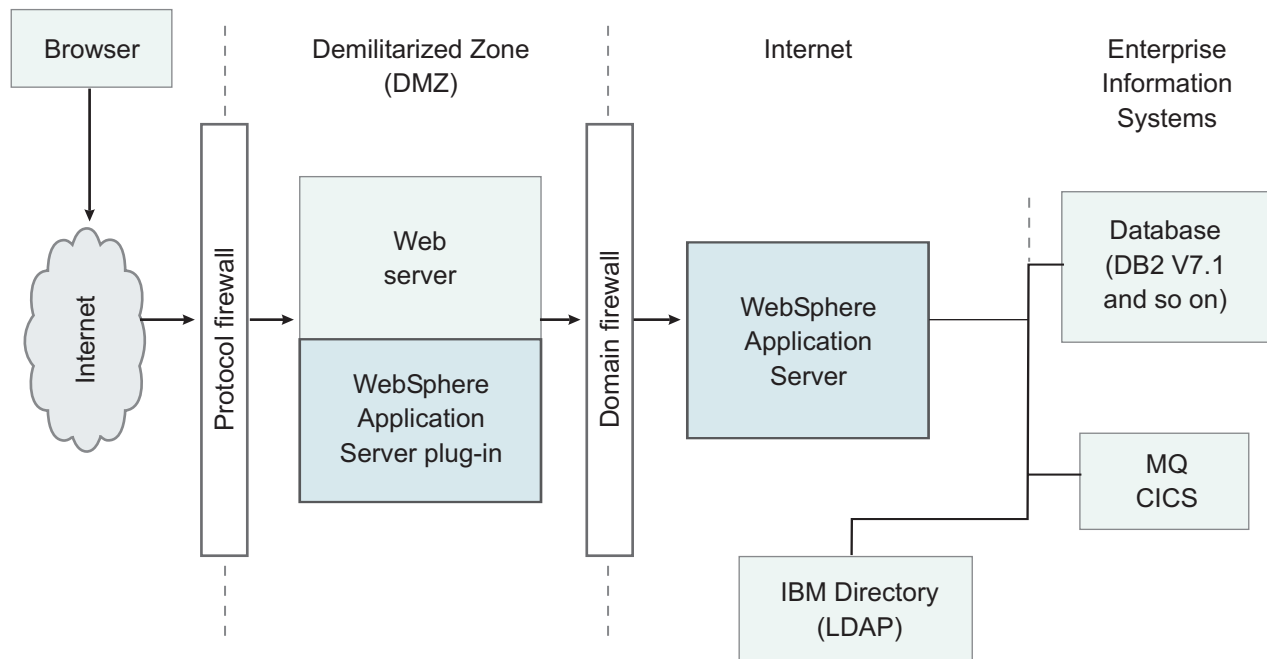
Provides information about the users and their rights in the Web application. The information can contain user IDs, passwords, certificates, access groups, and so forth. This node supplies the information to the security services like authentication and authorization service.

Enterprise information system

Represents existing enterprise applications and business data in back-end databases.

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

You can configure WebSphere Application Server and the Web server plug-in to communicate through secure SSL channels. You can configure a WebSphere Application Server HTTP server to open communication channels only with a restricted set of Web server plug-ins. You can configure the HTTP server to require client certificate authentication with self-signed certificates and to trust only the signer certificate. For instructions on how to generate self-signed certificates and how to set up secure communications channels between an HTTP server and the WebSphere Application Server plug-in, refer to Configuring IHS plug-in and the Internal Web Server for SSL and Configuring IHS for SSL Mutual Authentication.



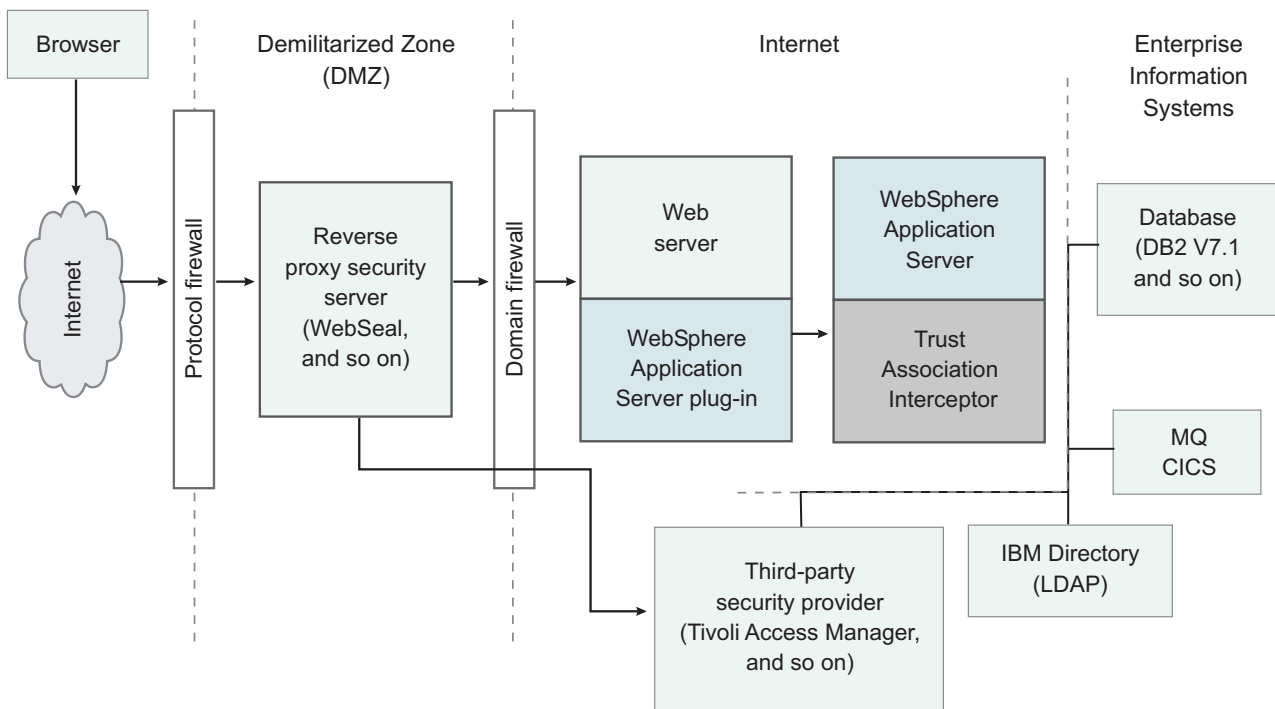
The WebSphere Application Server plug-in routes HTTP requests according to the virtual host and port configuration and the URL pattern matching. Client authentication and finer grained access control are handled by WebSphere Application Server behind the firewall.

In cases where the Web server can contain sensitive data and direct access is not desirable, the following configuration uses Tivoli WebSEAL to shield a Web server from unauthorized requests. WebSEAL is a Reverse Proxy Security Server (RPSS) that uses Tivoli Access Manager to perform coarse-grained access control to filter out unauthorized requests before they reach the domain firewall. WebSEAL uses Tivoli Access Manager to perform access control as illustrated in the picture. WebSphere Application Server supports various user registry implementations through the pluggable user registry interface. WebSphere Application Server ships a Local OS user registry implementation for Windows, AIX, AS/400, and Lightweight Directory Access Protocol (LDAP).

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

Basically, the RPSS performs authentication and adds proper authentication data into the request header and then redirects the request to Web server. A trust relationship is formed between an RPSS and WebSphere Application Server, and the RPSS can assert client identity to WebSphere Application Server to achieve single signon between RPSS and WebSphere Application Server. When the request is forward to WebSphere Application Server, WebSphere Application Server uses the TAI plug-in for the particular RPSS server to evaluate the trust relationship and to extract the authenticated client identity. WebSphere Application Server then

maps the client identity to a WebSphere Application Server security credential. For instructions on setting up a trust association interceptor, refer to Trust associations, Configuring trust association interceptors.



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

Each connector can be configured to use a different set of IDs and passwords. For a description on how to configure JCA principal mapping user IDs and passwords, refer to Managing J2C Authentication Data Entries. A principal mapping module is a special purpose Java Authentication and Authorization Service (JAAS) login module. You can develop your own principal mapping module to fit your particular business application environment. For detailed steps on developing and configuring a custom principal mapping module, refer to the articles, Developing

your own Java 2 security mapping module underneath JAAS Programmatic Login and Managing Java Authentication and Authorization Service (JAAS) Login Configuration.

Security and WebSphere MQseries

It is important to note that security logging information on UNIX systems is not protected because of the world-writable files in the /var file system of MQseries. MQseries ships the following files with its product:

- -rw-rw-rw- /var/mqm/errors/AMQERR01.LOG
- -rw-rw-rw- /var/mqm/errors/AMQERR02.LOG
- -rw-rw-rw- /var/mqm/errors/AMQERR03.LOG

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

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

To determine the size of the /var/mqm file system for a server installation, consider the following:

- Maximum number of messages in the system at one time
- Contingency for message buildups, if there is a system problem
- Average size of the message data, plus 500 bytes for the message header
- Number of queues
- Size of log files and error messages

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

Interoperability issues for security

To have interoperability of Security Authentication Service (SAS) between C++ and WebSphere Application Server, use the Common Secure Interoperability Version 2 (CSIV2) authentication protocol over Remote Method Invocation over the Internet Inter-ORB Protocol (RMI-IIOP). To have interoperability of SAS between WebSphere Application Server and WebSphere Application Server for z/OS use the zSAS authentication protocol over RMI-IIOP.

Interoperability with C++ common object request broker architecture client support and limitations

In addition to the WebSphere base installation, you can choose from two types of C++ common object request broker architecture (CORBA) client support, IBM WebSphere Application Server Enterprise, Version 5 or WebSphere Application Server Client Version 5. If you plan to develop or rebuild your own C++ client applications, then the Enterprise version is required. It installs tools, libraries, and include files for the build environment in selecting C++ CORBA client software development kit (SDK). Otherwise, a client installation suffices to run your C++

client applications with security. In Version 5, WebSphere Application Server supports the C++ CORBA client on the Windows 2000, Windows NT, Linux, and AIX operating systems and the Solaris operating environment.

Secure Sockets Layer Version 2 (SSLV2) cipher suites are not supported. In Version 5, only the most commonly used ciphers among Java Secure Socket Extension (JSSE) and Global Security Kit (GSKit) are supported.

Since the WebSphere Enterprise CORBA C++ Client has only implemented security on the transport layer, other authentication mechanisms such as user ID and password (Basic Authentication) are not supported.

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

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

Security authentication from non-Java based C++ client to enterprise beans. WebSphere Application Server supports security in the CORBA C++ client to access protected enterprise beans. If configured, C++ CORBA clients can access protected enterprise bean methods using client certificate to achieve mutual authentication on WebSphere Application Server Enterprise applications.

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

- Create an environment file for the client, such as `current.env`. Set the variables listed below (`security_sslKeyring`, `client_protocol_user`, `client_protocol_password`) in the file.
- Using `security_sslKeyring`, specify the SSL keyring to a keyring that was created for the client.
- Using `client_protocol_user` and `client_protocol_password`, specify a user ID and password if using zSAS Basic Authentication.
- Point to the environment file using the fully qualified pathname through the environment variable `WAS_CONFIG_FILE`. For example, in the test shell script `test.sh`, export
`WAS_CONFIG_FILE=/WebSphere/V5R0M0/AppServer/bin/current.env`.

| C++ security setting | Description |
|---------------------------------------|---|
| <code>client_protocol_password</code> | Specifies the password for the user ID. |
| <code>client_protocol_user</code> | Specifies the user ID to be authenticated at the target server. |

| C++ security setting | Description |
|----------------------|--|
| security_sslKeyring | Specifies the name of the RACF keyring the client will use. The keyring must be defined under the user ID that is issuing the command to run the client. |

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

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

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

Use the Key Management Utility from the Global Security Kit (GSKit) to extract the public key from the personal certificate and save it in the .arm format. For details, see the related information about how to extract the personal certificate of the public key.

2. Prepare a truststore file for WebSphere Application Server.

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

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

For details, see the article on Adding truststore files.

3. Configure WebSphere Application Server to support SSL as the authentication mechanism.

- a. Start the administrative console.

- b. Locate the application server that has the target enterprise bean deployed and configure it to use SSL client certificate authentication.

If it is a base installation, go to **Security > Authentication Protocol > CSIv2 Inbound Authentication** then select **Supported** for Basic Authentication and Client Certificate Authentication and leave the rest as defaults. Go to the CSIv2 Inbound Transport and make sure **SSL-Supported** is selected.

If it is a Network Deployment setting, go to **Server > Application Server > *server_name_where_EJB_resides* > Server Security > CSI Authentication Inbound**. Then select **Supported** for Basic Authentication and Client Certificate Authentication. Leave the rest as defaults. Go to **CSI Transport > Inbound** to make sure **SSL-Supported** is selected.

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

- c. Restart the application server.

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

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

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

| C++ security setting | Description |
|----------------------|-------------|
|----------------------|-------------|

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

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

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

Interoperating with previous product versions

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

1. Enable security with the LTPA authentication mechanism and the LDAP user registry. Make sure that the same LDAP user registry is shared by all the product versions.
2. Extract and add Version 5 server certificates into the server key ring file of the previous version.
 - a. Open the Version 5 server key ring file using the key management utility (iKeyman) and extract the server certificate to a file.
 - b. Open the server key ring of the previous product version, using the key management utility and add the certificate extracted from product Version 5.
3. Extract and add Version 5 trust certificates into the trust key ring file of the previous product version.
 - a. Open the Version 5 trust key ring file using the key management utility and extract the trust certificate to a file.
 - b. Open the trust key ring file of the previous product version using the key management utility and add the certificate extracted from Version 5.
4. If single signon (SSO) is enabled, export keys from the Version 5 product and import them into the previous product version. The Version 4 product requires the fix, PQ61779, and the Version 3.5 product requires the fix, PQ59667, for SSO to function.
5. Verify that the application uses the correct JNDI name. In Version 5, the enterprise beans are registered with long JNDI names like,

(top)/nodes/node_name/servers/server_name/HelloHome. Whereas in previous releases, enterprise beans are registered under a root like, (top)/HelloHome. Therefore, EJB applications from previous versions perform a lookup on the Version 5 enterprise beans.

You can also create EJB name bindings in Version 5 that are compatible with the previous version. To create an EJB name binding at the root Version 5, start the administrative console and click **Environment > Naming > Naming Space Bindings > New > EJB > Next**. Complete all the fields and enter a short name (for example, -HelloHome) as the JNDI Name. Click **Next** and **Finish**.

6. Stop and restart all the servers.
7. Make sure that the correct naming bootstrap port is used to perform naming lookup. In previous product versions, the naming bootstrap port is **900**. In Version 5, the bootstrap port is **2809**.

Security: Resources for learning

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

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

View links to additional information about:

- Planning, business scenarios and IT architecture.
- Programming model and decisions
- Programming specifications
- Administration

Planning, business scenarios and IT architecture

- WebSphere Application Server Library
- WebSphere Application Server Support
- WebSphere Application Server Version 5 Security Redbook

Programming model and decisions

- JSSE Documentation.

Refer to *install_dir/web/docs/jsse/jssedocs.jar* for the Javadoc of the APIs.

- For other JSSE information refer to *install_dir/web/docs/jsse/API_users_guide.html* and *install_dir/web/docs/jsse/readme.jsse.ibm.html*.
- For sample JSSE applications refer to *install_dir/web/docs/jsse/samplejsse.jar*.

- iKeyman Documentation.

Look in {was_install_root}/web/docs/ikeman/ikmuserguide.pdf for the SSL Introduction and iKeyman.

- JCE Documentation.

- For the JCA spec and JCE API usage refer to *install_dir/web/docs/jce/api_users_guide.html*.
- For JCE sample applications refer to *install_dir/web/docs/jce/SampleJCE.jar*.
- For Java Cryptography Architecture Reference refer to *install_dir/web/docs/jce/CryptoSpec.html*.

- For how to implement a JCE provider refer to *install_dir/web/docs/jce/HowToImplAProvider.html*.
- For the Javadoc of JCE APIs refer to *install_dir/web/docs/jce/jcedocs.jar*.
- For overview of IBM JCE refer to *install_dir/web/docs/jce/readme.jce.ibm.html*.
- IBM SDK for z/OS, Java 2 Technology Edition, Version 1.4
 - Refer to Java 2 Security check permission algorithm.

Programming specifications

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

For programming and usage in JAAS, refer to the specification located at *{was_install_root}/web/docs/jaas/JaasDocs.jar*. This document contains the following when unpacked:

- *login.html* - LoginModule Developer's Guide
- *api.html* - Developer's Guide (JAAS Javadoc)
- *HelloWorld.tar* - Sample JAAS Application
- JavaTM 2 Platform, Standard Edition, v 1.4.2 API Specification

Administration

- WebSphere Application Server Version 4.0 Security Redbook: WebSphere Security Model.
- IBM HTTP Server Support and Documentation
- IBM Directory Server Support and Documentation
- IBM Application Developer Kit Readme
 - For IBM Application Developer Kit refer to *{was_install_root}/java/docs/readme.devkit.ibm.html*
 - For IBM Application Developer Kit Installation and Configuration Readme refer to *{was_install_root}/java/docs/readme.install.ibm.html*
- IBM cryptographic hardware devices
- Supported hardware, software and APIs prerequisite Web site

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, New York 10594 USA

Trademarks and service marks

The following terms are trademarks of IBM Corporation in the United States, other countries, or both:

- AIX
- AS/400
- CICS
- Cloudscape
- DB2
- DFSMS
- Domino
- Everyplace
- iSeries
- IBM
- IMS
- Informix
- iSeries
- Language Environment
- Lotus
- MQSeries
- MVS
- OS/390
- RACF
- Redbooks
- RMF
- SecureWay
- SupportPac
- Tivoli
- ViaVoice
- VisualAge
- VTAM
- WebSphere
- z/OS
- zSeries

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.