Security -- table of contents

Development and special topics

Administration

5: Securing applications -- special topics

IBM WebSphere Application Server provides security components thatprovide or collaborate with other services to provide authentication,authorization, delegation, and data protection. WebSphere ApplicationServer also supports the security features described in the Java 2Enterprise Edition (J2EE) specification. Security elements in yourWebSphere environment are discussed in article 5.1.

Security is established at two levels. The first level is globalsecurity. Global security applies to all applications running in theenvironment and determines whether security is used at all, the type of registry against which authentication takes place, andother values, many of which act as defaults.

The second level is application security. Application security, whichcan vary with each application, determines the requirements specificto the application. In some cases, these values can overrideglobal defaults. Application security includes settings likemechanisms for authenticating users and authorization requirements.

Security information is supplied in one of two places. Securityinformation is classified as global, which applies to all applicationsrunning in the environment, or application-specific, which is tailoredto individual applications. Global security is administered byusing the WebSphere administrative console; application securityis administered during the assembly phase by using the applicationassembly tool (AAT) and during the deployment phase by using theadministrative console and the **wscp** tool.

Information about the standard security tasks appearsin 6.6.18: Securing applications.General administrative tasks, including standard global-security tasks,are described in 6.6.0.1: Using the Java administrative console.The application assembly tool is covered in6.3: Using the application assembly tool.

The rest of the material in this section concentrates on more specializedissues related to security. Some of these are programmatic innature, and some are administrative. The discussions assume familiaritywith general security procedures in the WebSphere Application Serverenvironment.

Article 5.1, The WebSphere security componentsgives an overview of WebSphere Application Server security.

Article 5.2, Using a custom registry describeshow to use a custom registry within WebSphere Application Server for authentication of users. This allows sitesto provide support for user registries not explicitly supported byWebSphere itself.

Article 5.3, Changes to security describeschanges in security since the previous version of WebSphere Application Server.

Article 5.4, Using programmatic and custom logindescribes the use of programmatic client and server login routines that work with the authentication policies and other settings specified by the administrator of WebSphere Application Server. This allows sitesto customize the way in which authentication information is collectedfrom users.

Article 5.5, Certificate-based authenticationprovides an introduction to the concepts ofcertificate-based authentication and its use in the WebSphereenvironment. This includes a discussion of general cryptographicconcepts like public-key encryption and digital signatures as well asinformation on the use of certificates in the WebSphere environment,tools for managing certificates and keys, and other related topics:

- 5.5.1: Introduction to public-key cryptography is the first article in a sequence that explainsencryption, signatures, certificates, and other related topics.
- 5.5.6: Tools for managing certificates and keys documents WebSphere Application Server's command-line and GUI certificate and key management tools. It also includes common procedures for managing certificates and keys with the tools.
- 5.5.7: Setting up an LDAP connection over SSL describes how to establish an SSL connection between

WebSphere Application Server and an LDAP server.

Article 5.6, Establishing trust association with a reverse proxy serverdescribes how to use a reverse proxy server to perform authentication for applications within WebSphereApplication Server.

Article 5.7, The Secure Association Servicedescribes the Secure Association Service (SAS), which plays a crucial role in security for WebSphere ApplicationServer. It also provides reference material on security-relatedproperties.

Article 5.8, Single sign-on support between WebSphere Application Server and Lotus Domino,describes the single sign-on (SSO) capability and describes how to configure it between WebSphere Application Serverand Lotus Domino.

5.1: The WebSphere security components

Security for WebSphere Application Server is managed as a collaborativeeffort by several components:

- Security collaborators
- Security policies
- The Secure Association Service (SAS)
- The user registry
- Secure Sockets Layer (SSL)

The security collaborators

The security collaborators reside in the application server process and arethe key run-time components for enforcing the security constraints andattributes specified in the deployment descriptors. There is a collaboratorfor Web resources in the Web container and another collaborator in the enterprise-bean container.

The Web collaborator performs authentication and authorization. Theenterprise-bean collaborator performs authorization, but not authentication,and sets the run-as identity for delegated request. The enterprise-beancollaborator relies on the Secure Association Service (SAS) to authenticateJava client requests to enterprise beans.Both collaborators do the following when a client request is made fora Web or enterprise-bean resource:

- Perform an authorization check.
- Log security tracing information.

The Web collaborator can perform an additional authentication operationbefore the two above: If the client has not already authenticated, theWeb collaborator can challenge the user, to collect a user ID and password.The challenge mechanism is specified as the login-configurationelement in the Web archive's web.xml deployment descriptor.

The enterprise-bean collaborator performs an additional operation afterthe two mentioned above. It sets the run-as identity, based on thedelegation policy. The delegation policy determines the identity to useif the enterprise bean invokes methods on any other enterprise beans.The delegation policy or run-as mode is specified in the ejb-jar.xmldeployment descriptor.

For example, when a client makes an HTTP request to a protected Webresource such as a JSP file, the request is dispatched to the Webcollaborator for the security check. The collaborator determinesif the client should be authenticated and, if so, challengesthe client to collect a user ID and password. The Web collaboratorauthenticates the user ID and password supplied by the client againsta user registry, for example, the local operating-system registry.If the client is successfully authenticated, the collaboratorthen consults an internal authorization table to determine whetherthe user is in one of the roles protecting the resource and, if so,permits access.

Security policies

Security attributes for enterprise and Web applications are specified inXML deployment descriptors, typically using a tool like the applicationassembly tool (AAT). The deployment descriptors contain much more thansecurity attributes, but only those related to security are discussed here.

The security attributes include roles, method permissions, the run-as modeor delegation policy, login-configuration or challenge type, and data-protection(confidentiality and integrity) settings.

When an application is deployed, the roles are mapped to users or groups.This combination of the users and

groups is mapped to roles and to the enterprise beans and Web methods protected by the roles. This mapping formsthe authorization table. There is an authorization table for each enterpriseapplication, and it is consulted by the collaborators during theauthorization check.

For more information on security-related attributes for deployment, see:

- The Servlet 2.2 specification, for Web resources
- The Enterprise JavaBeans 1.1 specification, for enterprise-bean resources
- 6.6.0.5: Using the Application Assembly Tool interface

The Secure Association Service (SAS)

SAS performs authentication for Java clients of enterprise beans andhelps to provide message protection or encryption between such clientsand WebSphere application servers using RMI/IIOP over SSL for communication.SAS also provides message protection between WebSphere application services.

User Registry

In environments that enforce security restrictions on applications, one ofthe first steps toward meeting such restrictions is to require users to*authenticate*--to prove their identities--in order toaccess applications. To prove an identity, a user submits a piece ofinformation, for example, a password or a certificate, to the security system,and the system checks the information against a database of knownusers. If the submitted information matches the information in thedatabase, the user has successfully authenticated.

The database of known users is a *registry*. WebSphereApplication Server supports the following types of registries:

- Local registries, which are limited to environments with a singleapplication server and single node or Windows NT domaincontroller.
- Centralized registries, which use the Lightweight Third PartyAuthentication (LTPA) protocol to access a supported Lightweight DirectoryAccess Protocol (LDAP) service
- Customer-defined registries, which use a WebSphere interface thatfacilitates access to custom registries

SSL

Secure Sockets Layer (SSL) is a public-key network-security protocol thatcan perform both authentication and message encryption. SSL is used betweenWeb browsers, Web servers, and WebSphere application servers to encryptmessage data.

For instructions on how to configure SSL in WebSphere ApplicationServer, see article 6.6.18, Securing Applications.

5.1.1: Security features

This section briefly describes some of the features of WebSphereApplication Server that you can use to secure your applications.

The security system has two facets. First, it enables administratorsto define security *policies* to establish control of resources.Administrators use security policies to tell WebSphere ApplicationServer how security is to be handled. The security system also provides built-in security *services* to enforce the policies.

The IBM WebSphere Application Server security system provides a numberof features, including the following:

Authentication policies and services

Authentication is the process of verifying that users are who they say they are. You can indicate how you want WebSphere Application Server to verify the identity of users who try to access your resources. You can choose a supported directory service, the operating system registry, or a custom registry to verify the identity of users and groups.

Authorization policies and services

Authorization is the process of determining what a user is allowed to do with a resource. You can specify policies that give different users differing levels of access to your resources. If you define authorization policies, WebSphere Application Server will enforce them for you.

Delegation policies

Delegation allows an intermediary to do work initiated by a client under an identity based on the associated delegation policy. Therefore, enforcement of delegation policies affect the identity under which the intermediary performs downstream invocations, that is, the calls made to complete the current request. When making downstream requests, the intermediary uses the client's credentials by default; other choices are also possible. The result is that the downstream resources do not know the identity of the intermediary; they see the identity under which the intermediary is operating. There are three possibilities for the identity under which the intermediary operates are when making the downstream requests:

- The client's identity (default)
- Its own identity
- An identity specified by configuration

A unified security administration model

The different components of WebSphere Application Server use the same model for security, so after you learn how to set up security for one type of resource, you can apply that knowledge to other resources. Enterprise beans, servlets, JSP files, and Web pages are all administered similarly in terms of security. You can combine all of these resources into an application for which you also establish security.

Single sign-on support

Application Server supports third-party authentication, a mechanism for achieving single sign-on across the Internet domain that contains your resources. You can use single sign-on to allow users to log on once per session rather than requiring them to log on to each resource or application separately.

Password encoding in configuration files

Several of the WebSphere configuration files contain user IDs and passwords. These are needed at run time to access external secure resources such as databases. Passwords are encoded, not encrypted, to deter casual observation of sensitive information. Password encoding combined with proper operating system file system security is intended to protect the passwords stored in these files.

5.1.2: The WebSphere authentication model

Authenticationis the process of determining if a user is who the userclaims to be. WebSphere Application Server authenticates usersby using one of several authentication mechanisms.J2EE does not specify how toauthenticate to an enterprise-bean container. However, WebSphere usesthe Secure Association Service (SAS) to authenticate Java clientsto enterprise beans.

The authentication mechanism for Web resources is specifiedby using the `login-config` element of the web.xml deploymentdescriptor for the Web application. Each Web application in an enterpriseapplication can have a different `login-config` value specified.Here is an example of a `login-config` element where formlogin is specified:

```
      <login-config>         <auth-method>FORM</auth-method>         <realm-name>Example Form-Based
Authentication</realm-name>         <form-login-config>
<form-login-page>/login.html</form-login-page>
<form-error-page>/error.jsp</form-error-page>         </form-login-config>      </login-config>
```

The servlet specification identfies the following authentication methods:

- **Basic authentication:**

  This is the familiar style of authentication in which the Web browser presents a dialog window requesting the user to enter a user ID and password when the user attempts to access a protected Web resource.

  After the user provides the identifier and password, the security service validates them against a database of known users, the user registry. If the user-provided information is valid, the security system considers the user authenticated.

  The registry can take the form of a local registry, a distributed directory service, or a custom registry.
- **Digest authentication**

  This authentication mechanism is not supported by WebSphere. You must specify one of the other authentication mechanisms.
- **Client-certificate**

  This authentication mechanism requires the client to use a digital certificate. The identity in the digital certificate is mapped to an entry in either the LDAP registry specified when LTPA was configured or to a custom registry.
- **Form-based authentication**

  This authentication mechanism permits a site-specific login through an HTML page or a JSP form.

See 5.4.2.3: Accessing secured resources from Javaclients for information on authenticating Java clients to enterprisebeans.

5.1.3: The WebSphere authorization model

Authorization information is used todetermine if a caller has the necessary privilege to request aservice. Authorization information can be stored in many ways. Forexample, with each resource, you can store a list of users and whatthey are permitted to do. Such a list is called an access-controllist. Another way to store the information is to associate with eachuser a list of resources and the corresponding privilege held by theuser. This is called a capability list.

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

- During the assembly of an application, permission to execute 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, for example, the withdraw and deposit methods. The Teller role is not granted permission to close accounts; that 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.

  Role-to-method mapping

		AccountBean methods					AccountServlet methods	
		getBalance	**setBalance**	**deposit**	**withdraw**	**closeAccount**	**HTTP_GET**	**HTTP_DELETE**
Roles	**Teller**	yes	-	yes	yes	-	-	-
	Clerk	yes	-	-	-	-	-	-
	Supervisor	-	yes	-	-	yes	-	yes
	WebTeller	-	-	-	-	-	yes	-

  There are two special subjects that are not defined by J2EE but are worth mentioning, AllAuthenticatedUsers and Everyone, and a special role, DenyAllRole. A *special subject* is Websphere-defined entity that is 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 allows access to protected methods as if the resources are unprotected.
  - **DenyAllRole** is a special role that is assigned by default to a partially protected resource. For instance, if an enterprise bean has four methods and only three are explicitly protected, the fourth method is associated with the DenyAllRole. This role denies everyone access to the methods it is associated with. The DenyAllRole is never mapped to any users or groups; it is always empty.

- 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 that 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:

  - It improves performance during the authorization check. There are typically far fewer groups than users.
  - For AEs, it can greatly improve application server startup time.
  - It provides greater flexibility, by using group membership to control resource access.
  - Users can be added to and deleted from groups outside of the WebSphere environment. This is preferred to adding and removing them to WebSphere roles; the enterprise application must be stopped and

restarted for such changes to take effect, and this can be very disruptive in a production environment.

Subject-to-role mapping

		Roles			
		Teller	**Clerk**	**Supervisor**	**WebTeller**
Subjects	**TellerGroup**	yes	-	-	yes
	Bob	yes	yes	-	yes
	ClerkGroup	-	yes	-	-
	Supervisor	-	-	yes	-

- At execution 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 alsorecognizes that not all situations can be dealt with declaratively. For those situations,methods are provided for determining user and role information programmatically. ForEnterprise JavaBeans, the following two methods are supported by WebSphere ApplicationServer:

- getCallerPrincipal: This method retrieves the user's identification information.
- isCallerInRole: This method checks the user's 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.

5.1.3.1: Securing applications and resources

WebSphere supports the J2EE model for creating, assembling, securing, anddeploying applications. This document provides a high-level descriptionof what is involved in securing resources in a J2EE environment.Resources are secured by doing the following:

- Specifying roles and defining method permissions in deployment descriptors.
- Assigning users and groups to roles during application deployment.
- Enabling global security in the WebSphere environment.

The J2EE specifications should be consulted for complete details.

Applications are often created, assembled and deployed in differentphases and by different teams.

Application-component providers

Component providers create enterprise beans, servlets, JSP files,HTML files, and related components. These components are packaged intoJ2EE modules for containers that can support them.

Enterprise-bean modules contain enterprise-bean class filesand a deployment descriptor. These modules are packaged asstandard JAR files, using the .jar extension.

Web modules contain servlets, JSP pages, HTML pages, GIFs, andother, and also include a deployment descriptor. These modulesare packaged as Web archive files, JAR files with a .war extension.

Enterprise bean and Web modules can be assembled into enterprise-applicationmodules. These modules are packaged as enterprise archive files, JAR fileswith a .ear extension.

The component provider specifies most of the configuration meta-informationfor the components, including the security attributes, in the deploymentdescriptors. These attributes identify roles, specify the methods that areassociated with the roles, the `login-config` method, and soforth. A tool like the WebSphere application assembly tool (AAT) is usedto create J2EE modules and to set the attributes in the deployment descriptors.

Application assemblers

Application assemblers combine J2EE modules, resolve references betweenthem, and create from them a single deployment unit, typically a .ear file.A tool like AAT is also used to accomplish these tasks. Componentproviders and application assemblers can be the same people, but theydo not have to be.

Deployers

Deployers links entities referred to in an enterprise application tothe run-time environment. One of the important tasks the deployerperforms is mapping actual users and groups to the application's roles.The deployer installs the enterprise application into the environmentand makes the final adjustments needed to run the application.

Most of the steps in creating J2EE applications involve deploymentdescriptors; the deployment descriptors play a central role inapplication security in a J2EE environment.

5.1.3.2: Role-based authorization scenarios

This article describes the steps taken by WebSphere ApplicationServer to authorize requests. The two scenarios are based on a bankingapplication that includes both an enterprise bean called AccountBeanand a servlet called AccountServlet. The following tables definethe application's role-to-method mapping and user-to-role mapping:

		AccountBean methods					AccountServlet methods	
		getBalance	**setBalance**	**deposit**	**withdraw**	**closeAccount**	**HTTP_GET**	**HTTP_DELETE**
Roles	**Teller**	yes	-	yes	yes	-	-	-
	Clerk	yes	-	-	-	-	-	-
	Supervisor	-	yes	-	-	yes	-	yes
	WebTeller	-	-	-	-	-	yes	-

Role-to-method mapping

		Roles			
		Teller	**Clerk**	**Supervisor**	**WebTeller**
Subjects	**TellerGroup**	yes	-	-	yes
	Bob	yes	yes	-	yes
	ClerkGroup	-	yes	-	-
	Supervisor	-	-	yes	-

Subject-to-role mapping

Authorizing a request to an enterprise bean

When a client attempt to execute a method on the home or remote interfaceof an enterprise bean, WebSphere Application Server must determine whetherthe user ID, or principal, of the client is in a role that is authorizedto execute the method.

Scenario: A request attempts to execute the getBalance method on theenterprise bean AccountBean. To authorize this request, WebSphereApplication Server does the following:

1. Determines the calling client's principal. If the principal cannot be determined, the request is rejected. Suppose that the user Bob is identified as the calling principal.
2. Determines the set of roles permitted to invoke the getBalance method. The role-to-method mapping table indicates that both the Teller and the Clerk roles are authorized to execute the getBalance method.
3. Determines if the calling principal is in at least one of the authorized roles. The user-to-role mapping table indicates that Bob is in the Teller, Clerk, and WebTeller roles, so the authorization requirements are met.
4. Determines whether the security policy specifies a different identity to use for invoking the method and any subsequent methods it calls.
5. Invokes the requested method.

Authorizing a request to a Web resource

When a Web browser attempts to execute a method on a Web resource,WebSphere Application Server must determine whether the user ID, or principal,of the client is in a role that is authorized to execute the requeston the Web resource.

Scenario: A request attempts to execute the HTTP_GET method for theservlet AccountServlet. To authorize this request, WebSphere ApplicationServer does the following:

1. Challenge the user for authentication information. Suppose that the user ID and password for Bob are successfully authenticated.
2. Determine the set of roles permitted to invoke the HTTP_GET method. The role-to-method mapping table indicates that the WebTeller role is authorized to execute the HTTP_GET method.
3. Determine if the calling principal is in at least one of the authorized roles. The user-to-role mapping table indicates that Bob is in the Teller, Clerk, and WebTeller roles, so the authorization requirements are met.
4. Invoke the requested method.

5.1.4: The WebSphere delegation model

The WebSphere delegation model is an extension the Enterprise JavaBeans1.1 specification; delegation is fully addressed in Enterprise JavaBeans2.0 specification. Enterprise beans can have delegation policies;Web resources cannot.

Delegation allows an intermediary to perform a task initiated by a clientunder an identity determined by the associated policy. Therefore,enforcement of delegation policies affects the identity under whichthe intermediary performs downstream invocations, that is, invocation madeby the intermediary in order to complete the current request, on other objects.By default, if no delegation policy is set, the intermediary will use theidentity of the the requesting client while making the downstream calls.Alternatively, the intermediary can perform the downstream invocations underits own identity or under an identity specified by configuration.

When the intermediary operates under an identity other than its own,downstream resources do not know the identity of the intermediary. Therefore,they make their access decisions based on the privileges associated with theidentity being used.

The administrator specifies a delegation policy bysetting the run-as mode for each enterprise-bean method. For each,the administrator can choose among three policies:

- The client identity
- The system identity, the identity of the intermediary
- A specified identity, based on a particular role, named in the delegation policy

For example, suppose that a client invokes a session bean thatinvokes an entity bean. If the delegation policy states thatmethods are invoked under the client's identity, the session beanmakes its invocations under the client's identity. Therefore,it is the client, rather than the session bean, that must havepermission to invoke the entity-bean methods. If the delegationpolicy requires the system identity, the session bean makes itsinvocation under the identity of the server in whichthe session bean resides; it is this server that must have permissionon the entity-bean methods. Finally, if the delegation policyrequires a specified identity, the session bean invokes themethods under this identity, so the specified identity must havepermission on the entity-bean methods.

In WebSphere Application Server, the application assembler determinesthe use of delegation by using the application-assembly tool (AAT) toset the `SecurityIdentity` value in the deployment descriptor.If this value is not set, no special instructions about security identitiesare used, and the intermediary uses the caller identity for any downstreaminvocations. The `SecurityIdentity` value be associated withany of the following types:

- `UseCallerIdentity` (cannot be used for message-driven beans)
- `UseSystemIdentity`
- `RunAsSpecifiedIdentity`

Use of `UseCallerIdentity` means that the intermediarywill use its client's credentials for downstream invocations. Use of`UseSystemIdentity` means that the intermediary willuse its own credentials for downstream infocations. Use of `RunAsSpecifiedIdentity` means that credentials determined elsewhere will be used.

The application assembler does not typically know the makeup ofthe run-time environment, including the specific user identitiesthat are available. Therefore, it can be impossible for anassembler to have a concrete value to specify for an intermediary that is to run as a specified identity. Therefore, the run-asidentity is designated as a logical role name, which correspondsto one of the security roles defined in the deployment descriptor.That is, if the type of identity is specified as the`RunAsSpecifiedIdentity` type, the deployment descriptoralso contains a `runAsSpecifiedIdentity` element witha `roleName` attribute. Thus, to establish a delegationpolicy under which a resource runs as an administrator, that is, amember of the **admin** role, the `runAsSpecifiedIdentity`element looks like this:

```
   ...      <runAsSpecifiedIdentity         xmi:id="Identity_1"         roleName="admin"
description=""      />   ...
```

At deployment time, a particular user is assigned to that role andbecomes the run-as identity by indirection. This allows you to usethe specified-identity delegation policy to run beans under theidentity of a user who has been associated with the role.

5.1.5 Using Windows NT or Windows 2000 with Local authorization

When enabling security on Windows NT or Windows 2000 systems, if Local Operating System (LocalOS) is selected as theauthentication mechanism, keep the following in mind:

- WebSphere Application Server dynamically determines whether the machineis a member of a Windows domain.
- WebSphere Application Server does not support Windows NT trusteddomains.
- If a machine is a member of a Windows domain, both the domain userregistry and the machine's local user registry participate inauthentication and security role mapping.
- The domain user registry takes precedence over the machine's localuser registry and may have undesirable implications if users with thesame password exist in both user registries.

When **LocalOS** is selected as the authentication mechanism, theuser registry used for authentication depends on whether the machineis a member of a Windows domain. When WebSphere is started, the securityruntime initialization process dynamically attempts to determine ifthe local machine is a member of a Windows domain. WebSphereApplication Server relies on the Windows computer browser service tohelp determine which domain the machine is a member of.

If the machine is not a member of a Windows domain, the userregistry local to that machine is used for authentication.

If the machine is a member of a Windows domain, both the domain userregistry and the local user registry can be used for authorization.The Windows domain registry is used for authentication first. If theuser cannot be authenticated there, authentication will be attemptedat the machine's local user registry.

Authorizing with the domain user registry first can cause problemsif a user exists in both the domain and local user registries with thesame password. Role-based authorization can fail in this situationbecause the user is first authenticated within the domain userregistry. This authentication produces a unique domain security IDthat is used in WebSpere Application Server during the authorizationcheck. However, the local user registry is used for role assignment.The domain security ID will not match the unique security IDassociated with the role. To avoid this problem, map security rolesto domain users instead of local users.

5.1.6: Relationship to the operating environment

This section discusses how Application Server security relatesto the security provided by your operating system and by Java.

WebSphere Application Server security sits on top of your operatingsystem security and the security features provided by other components,including the Java language.

The types of security involved include:

- Operating-system security support, for example, authentication against, the local user registry.
- Java-language security, provided through the Java Virtual Machine (JVM) used by WebSphere and the programmatic security classes.
- CORBA security, in applications involve interprocess communication between secure ORBs.
- EJB security, in applications involving Enterprise Java Beans.
- WebSphere security, which relies on and enhances all of the above.

See the Sun Microsystems Enterprise JavaBeans specification, Version1.1, for a description of enterprise bean security in general.

5.2: Introduction to custom registries

WebSphereApplication Server supports the following types of registries:

- **Local registries**. Local registries are limited to single-machine or Windows NT domain-controller environments and a single application server. WebSphere Application Server does not support multiple node, multiple application servers or secure delegation when the Local registry is used as the user registry.
- **Centralized registries**, which use the Lightweight Third Party Authentication (LTPA) protocol to access a supported Lightweight Directory Access Protocol (LDAP) service. Centralized registries are limited to the set of WebSphere-supported LDAP directory services. The interface for custom registries allows WebSphere applications to take advantage of new or existing registries that are not otherwise accessible.
- **Customer-defined registries**, by using a WebSphere interface that facilitates access to custom registries.

For the custom-registry choice, WebSphere Application Server provides an interface thatdefines a set of methods that WebSphere Application Server calls to performsecurity operations for applications configured to use the customregistry. A developer must implement the methods in this interface byusing calls to the desired registry. This layer of code allows thedesired registry to be plugged into the WebSphere environment. Theinterface defines a very general set of methods, so it can be used toencapsulate a wide variety of registries.

5.2.1: The CustomRegistry interface

Developers can use a WebSphere interface to encapsulate registries that areotherwise unsupported. To encapsulate such registries, developers mustimplement the methods in the CustomRegistry interface, which is located in theJava package com.ibm.websphere.security. The source code is available fromCustom-registry source code. The structure of theCustomRegistry interface is shown in Figure 1.

Figure 1. The CustomRegistry interface

```
package com.ibm.websphere.security; import java.util.*;import java.security.cert.X509Certificate;
public interface CustomRegistry{   // General methods   public void initialize(java.util.Properties
props)      throws CustomRegistryException;    public String getRealm()      throws
CustomRegistryException;    // User-related methods   public boolean isValidUser(String userName)
throws CustomRegistryException;    public List getUsers()      throws CustomRegistryException;
public List getUsers(String pattern)      throws CustomRegistryException;    public String
getUniqueUserId(String userName)      throws CustomRegistryException,
EntryNotFoundException;    public String getUserSecurityName(String uniqueUserId)      throws
CustomRegistryException,             EntryNotFoundException;    public String
getUserDisplayName(String securityName)      throws CustomRegistryException,
EntryNotFoundException;    public List getUsersForGroup(String groupName)      throws
CustomRegistryException,             EntryNotFoundException;    public List getUniqueUserIds(String
uniqueGroupId)      throws CustomRegistryException,             EntryNotFoundException;    //
Group-related methods   public boolean isValidGroup(String groupName)      throws
CustomRegistryException;    public List getGroups()      throws CustomRegistryException;    public
List getGroups(String pattern)      throws CustomRegistryException;    public String
getUniqueGroupId(String groupName)      throws CustomRegistryException,
EntryNotFoundException;    public String getGroupSecurityName(String uniqueGroupId)      throws
CustomRegistryException,             EntryNotFoundException;    public String
getGroupDisplayName(String groupName)      throws CustomRegistryException,
EntryNotFoundException;    public List getGroupsForUser(String userName)      throws
CustomRegistryException,             EntryNotFoundException;    public List getUniqueGroupIds(String
uniqueUserId)      throws CustomRegistryException,             EntryNotFoundException;    //
Authentication methods   public String checkPassword(String userId, String password)      throws
PasswordCheckFailedException,             CustomRegistryException;    public String
mapCertificate(X509Certificate cert)      throws CertificateMapNotSupportedException,
CertificateMapFailedException,             CustomRegistryException;}
```

The CustomRegistry interface supports authentication of individual users bypassword and by digital certificate. It also contains a set of methodsfor retrieving information about users and a set for retrieving thecorresponding information about groups.

The CustomRegistry interface operates on the basis of the several pieces ofinformation. When implementing the methods in the interface, you mustdecide how to map the information manipulated by the CustomRegistry interfaceto the information in your registry. The methods in the CustomRegistryinterface operate on the following information for users:

- *User name:* an identifier for a user. The CustomRegistry interface requires user names to be unique. For most registries, the user name logically maps to an identifier that is meaningful to the user; some common terms for this identifier include login name, account name, user name, and principal.
- *Unique identifier:* a unique identifier for a user. The CustomRegistry interface requires this identifier to be unique. For most registries, the unique identifer logically maps to a numeric counterpart of a user name. For example, UNIX systems assign a user ID (UID) to each user name.
- *Display name:* an optional string that describes a user. Display names are used by the CustomRegistry interface to provide a way to describe user names, which are typically single-word identifiers. Display names can be used to hold full names or other descriptive information. Some common terms for this kind of information in registries include annotations, full-name fields, string fields, and others. Some registries do not support this kind of information at all. The CustomRegistry implementation uses display names for informational purposes only; display names are not required to exist or be unique. Display names are shown, along with user names, in the administrative console when a search is done for users or groups. Although the display names are used only as annotations within the registry, the getRemoteUser and getUserPrincipal methods, used by servlets and JSPs, and the getCallerPrincipal method, used by enterprise beans, use the information differently; see The getUserDisplayName and getGroupDisplayName methods for more information.

The CustomRegistry interface also operates on parallel information for groups:

- *Group name:* an identifier for a group.
- *Unique identifier:* a unique identifier for a group.
- *Display name:* an optional string that describes a group.

5.2.2: Implementing the CustomRegistry interface

To use a registry that is not natively supported by WebSphere ApplicationServer, you must provide a class that implements the CustomRegistry interfaceby providing code for each method in the interface. This code does thework necessary to retrieve and manipulate the information from the desiredregistry. Most of the methods in the CustomRegistry interface returneither strings or lists. When you implement these methods, indicatefailure to retrieve the desired information by returning null strings or nulllists.

To illustrate the structure of an implementation of the CustomRegistryinterface, this document describes a class that uses a UNIX-like localregistry. The class implements every method in the interface, andbecause the backing registry is so simple, the methods are simple. Animplementation using a realistic registry will use more complex,registry-specific code, but the structure will be the same. The sourcecode is available from Custom-registry source code.

5.2.2.1: Structure of the example registry

The registry used in this example consists of two text files. Thesefiles are variants of the UNIX /etc/password and /etc/group files. Thefile containing user information is called users.props, and the filecontaining group information is called groups.props.

The user-information file

Entries in the users.props file consist of the following fields,separated by the colon (:) character:

- User name: the unique name associated with a user's account; maps to the user name in the CustomRegistry interface
- Password: the password associated with the user name
- User ID (UID): a single, unique number associated with the user name; maps to the unique identifier in the CustomRegistry interface
- Group IDs (GIDs): a comma-delimited list of numeric identifiers indicating the groups to which the user belongs
- Annotation: an optional string of information used for description; maps to the display name in the CustomRegistry interface

In this simple registry, the passwords are simply stored as cleartextfields; the passwords are not encrypted. Any lines that begin withthe hash (#) character are considered comments and ignored.Figure 3 shows a sample user-information file.

Figure 3. The example users.props file

```
# User-information file# Format: username:password:UID:GID[,
GID]*:annotationbob:bob1:123:567:bobdave:dave1:234:678:jay:jay1:345:678,789:Jay-Jayted:ted1:456:678:Teddy
Gjeff:jeff1:222:789:Jeffvikas:vikas1:333:789:vikasbobby:bobby1:444:789:
```

The group-information file

Entries in the groups.props file consist of the following fields,separated by the colon (:) character:

- Group name: the unique name associated with the group; maps to the group name in the CustomRegistry interface
- Group ID (GID): a single, unique number associated with the group name; maps to the unique identifier in the CustomRegistry interface
- User names: a comma-delimited list of the names of the members of the group
- Annotation: an optional string of information used for description; maps to the display name in the CustomRegistry interface

Any lines that begin with the hash (#) character are considered commentsand ignored. Figure 4 shows a samplegroup-information file.

Figure 4. The example groups.props file

```
# Group-information file# Format: groupname:GID:username[,
username]*:annotationadmins:567:bob:Administrative groupoperators:678:jay,ted,dave:Operators
groupusers:789:jay,jeff,vikas,bobby:
```

5.2.2.2: Writing the sample application

To enable WebSphere applications to use the registry described in Structure of the example registry, you must provide a class thatimplements the methods in the CustomRegistry interface, described in TheCustomRegistry interface. This section describes the structure and methods of a classthat accesses the example registry.

5.2.2.2.1: Structure of the implementation class

The class implementing the CustomRegistry interface is calledFileRegistrySample. It primarily contains implementations of themethods in the CustomRegistry interface, but it also contains privatevariables representing the user- and group-information files making up theregistry, private file-manipulation methods for accessing the registry files,and an empty constructor. Figure 5 shows the structure and content of the class, excluding themethods in the CustomRegistry interface, which are describedseparately.

Figure 5. Code example: The structure of the FileRegistrySample class

```
import java.util.*;import java.io.*;import java.security.cert.X509Certificate;import
com.ibm.websphere.security.*; public class FileRegistrySample implements CustomRegistry{   private
static String USERFILENAME = null;   private static String GROUPFILENAME = null;    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 is an
auxiliary class for regular expressions      RegExpSample regexp = new RegExpSample(pattern);
boolean matches = false;      if(regexp.match(name))          matches = true;      return matches;
}   public FileRegistrySample() {}    // Methods from the CustomRegistry interface   ...}
```

This sample implementation also includes an auxiliary class, RegExpSample,that implements basic regular-expression handling.

5.2.2.2.2: The getRealm and initialize methods

The CustomRegistry interface defines the getRealm method for determiningthe name of the security realm. The name of the realm identifies thesecurity domain for which the registry authenticates users. EachWebSphere Application Server resides in a specific realm, and access to itsapplications is restricted by the security requirements of the realm. Ifthis method returns a null value, a default name of `customRealm` isused. For the sample implementation, the string `customRealm`is simply coded into the getRealm method.

The CustomRegistry interface also defines the initialize method forinitializing the custom registry. This method is used for establishingcontact with the registry and performing any initial work. For theexample registry, the intialize method retrieves the names of the registryfiles containing the user and group information.

WebSphere Application Server expects both the getRealm method and theinitialize method to throw the CustomRegistryException exception in case ofany problems. Figure 6 showsthe methods as implemented in the FileRegistrySample class.

Figure 6. Code example: The getRealm and initialize methods in the FileRegistrySample class

```
public String getRealm()   throws CustomRegistryException{   String name = "customRealm";   return name;} public void initialize(java.util.Properties props)      throws CustomRegistryException{   try {      // Get the files containing the user and group information.      // The properties "usersFile" and "groupsFile" are set in      // the GUI when the registry is configured.      if (props != null) {         USERFILENAME = props.getProperty("usersFile");         GROUPFILENAME = props.getProperty("groupsFile");      }   }   catch (Exception ex) {      throw new CustomRegistryException(ex.getMessage());   }   if (USERFILENAME == null || GROUPFILENAME = null) {   throw new CustomRegistryException(         "users/groups information missing);   }}
```

5.2.2.2.3: The isValidUser and isValidGroup methods

The isValidUser and isValidGroup methods are used to determine whether aprovided user or group name appears in the registry. Theimplementations of both methods must check that the name supplied as anargument appears in the registry as the name of a user or group and returneither a value of TRUE if the name appears or FALSE if it doesn't.WebSphere Application Server expects both the isValidUser and isValidGroupmethods to throw the CustomRegistryException exception in case of anyproblems.

To validate users and groups against the sample registry, each methoditerates through the entries in the appropriate file and examines the value inthe field for the user or group name. When a match is found, the methodstops looking and returns a TRUE value. If the entire file is traversedwithout a match, the method returns a FALSE value.Figure 7 shows the isValidUsermethod--and the structure of the isValidGroup method--as implemented inthe FileRegistrySample class. The only difference between the two methodsis the file over which they iterate.

Figure 7. Code example: The isValidUser and isValidGroup methods in the FileRegistrySample class

```
public boolean isValidUser(String userName)   throws CustomRegistryException{   String s;   boolean
isValid = false;   BufferedReader in = null;    try {      in = fileOpen(USERFILENAME);      while
((s=in.readLine())!=null) {         if (!s.startsWith("#")) {            int index = s.indexOf(":");
if ((s.substring(0,index)).equals(userName)) {               isValid=true;               break;
}         }      }   }   catch (Exception ex) {      throw new
CustomRegistryException(ex.getMessage());   }   finally {      fileClose(in);   }   return isValid;}
public boolean isValidGroup(String userName)   throws CustomRegistryException{   String s;   boolean
isValid = false;   BufferedReader in = null;   try {      in = fileOpen(GROUPFILENAME);      ...   }
catch (Exception ex) { ... }   finally { ... }    return isValid;}
```

5.2.2.2.4: The getUsers and getGroups methods

The getUsers and getGroups methods are used to retrieve lists of user orgroup names in the registry. The CustomRegistry interface defines twoof each method: a version that takes no arguments and returns the namesof all users or groups, and a version that takes a string and returns thenames of the users or groups that match a string:

- getUsers()
- getUsers(pattern)
- getGroups()
- getGroups(pattern)

WebSphere Application Server expects these methods to return null values ifno users or groups, or none matching the pattern, are found. All themethods are expected to throw the CustomRegistryException exception for anyother conditions.

The getUsers(pattern) and getGroups(pattern) methods must be able to handlearguments consisting of the full name of an existing user or group, whichmatches a single user or group, and of the asterisk (*) character, whichmatches all users or groups. At a minimum, these methods must behave asfollows:

- If the argument is the complete name of a user or group, that user or group must be returned.
- If the argument is the asterisk (*) character, the names of all users or groups must be returned. This can be implemented by calling either getUsers() or getGroups().

Developers can introduce more sophisticated pattern-matchingtechniques. The techniques implemented here determine the retrievalstrategies available on the administrative console. For registriesinvolving thousands of users or groups, the ability to retrieve names based onpartial matches, common endings, and so forth can greatly enhance theusability of the console.

Figure 8 shows theimplementation of the getUsers() method for theexample registry. The method iterates through the user-information fileand collects the user name from each entry. When the file is exhausted,the method returns the list of user names. The getGroups() method doesthe same work on the group-information file.

Figure 8. Code example: The getUsers() and getGroups() methods in the FileRegistrySample class

```
public List getUsers()   throws CustomRegistryException{   String s;   List allUsers = new
ArrayList();   BufferedReader in = null;    try {      in = fileOpen(USERFILENAME);      while
((s=in.readLine())!=null) {         if (!s.startsWith("#")) {           int index = s.indexOf(":");
allUsers.add(s.substring(0,index));         }      }   }   catch (Exception ex) {      throw new
CustomRegistryException(ex.getMessage());   }   finally {      fileClose(in);   }    return
allUsers;} public List getGroups()   throws CustomRegistryException{   String s;   List allGroups =
new ArrayList();   BufferedReader in = null;    try {      in = fileOpen(GROUPFILENAME);      ...
}   catch (Exception ex) { ... }   finally { ... }    return allGroups;}
```

The behavior of the methods that retrieve the names of all users or groupsis straightforward, but the definition of what constitutes a match, used bythe pattern-matching methods, varies with the registry used and the types ofinformation it stores. Matching on complete user or group name yields atmost one match. Partial and wildcard matches can be implemented to increasethe number of matches. For other registries, different characteristics canmake different matching strategies useful.

Figure 9 shows theimplementation of the getUser(pattern) method forthe example registry. The method iterates through the user-informationfile and attempts to match the user name with the provided string. Ifthey match, the user name is added to a list. When the file isexhausted, the method returns the list of user names. ThegetGroups(pattern) method does the same work on the group-informationfile. The match method is a local, private method that returns TRUE ifthe two arguments match. This method makes use of the RegExpSample class.

Figure 9. Code example: The getUsers(pattern) and getGroups(pattern) methods in the FileRegistrySample class

```
public List getUsers(String pattern)   throws CustomRegistryException{   String s;   List allUsers =
new ArrayList();   BufferedReader in = null;   try {      in = fileOpen(USERFILENAME);      while
((s=in.readLine())!=null) {         if (!s.startsWith("#")) {           int index = s.indexOf(":");
String user = s.substring(0,index);           if (match(user, pattern))
allUsers.add(user);         }      }   }   catch (Exception ex) {      throw new
CustomRegistryException(ex.getMessage());   }   finally {      fileClose(in);   }    return
allUsers;} public List getGroups(String pattern)   throws CustomRegistryException{   String s;
List allGroups = new ArrayList();   BufferedReader in = null;    try {      in =
fileOpen(GROUPFILENAME);      ...   }   catch (Exception ex) { ... }   finally { ... }    return
allGroups;}
```

5.2.2.2.5: The getUniqueUserId and getUniqueGroupId methods

The getUniqueUserId and getUniqueGroupId methods allow the retrieval of aunique identifier for a named user or group.

WebSphere Application Server expects the methods to throw theEntryNotFoundException exception if the user or group name does not exist inthe registry and to throw the CustomRegistryException exception for any otherconditions.

Figure 10 shows theimplementation of the getUniqueUserId method forthe example registry. The method iterates through the user-informationfile and attempts to locate an entry with the specified user name. Ifthe name is located, the corresponding UID field is extracted andreturned. If the user name is not found, the EntryNotFoundExceptionexception is thrown. The getUniqueGroupId method does the same work onthe group-information file.

Figure 10. Code example: The getUniqueUserId and getUniqueGroupId methods in the FileRegistrySample class

```
public String getUniqueUserId(String userName)   throws CustomRegistryException,
EntryNotFoundException{   String s, uniqueUsrId = null;   BufferedReader in = null;    try {      in
= fileOpen(USERFILENAME);       while ((s=in.readLine())!=null) {         if (!s.startsWith("#")) {
int index = s.indexOf(":");            int index1 = s.indexOf(":", index+1);            if
((s.substring(0,index)).equals(userName)) {               int index2 = s.indexOf(":", index1+1);
uniqueUsrId = s.substring(index1+1,index2);               break;            }         }      }   }
catch(Exception ex) {      throw new CustomRegistryException(ex.getMessage());   }   finally {
fileClose(in);   }    if (uniqueUsrId == null) {      EntryNotFoundException nsee =         new
EntryNotFoundException(userName);      throw nsee;   }    return uniqueUsrId;} public String
getUniqueGroupId(String userName)   throws CustomRegistryException, EntryNotFoundException{   String
s, uniqueGrpId = null;   BufferedReader in = null;    try {      in = fileOpen(GROUPFILENAME);
...   }   catch(Exception ex) { ... }   finally { ... }   if (uniqueGrpId == null) { ... }
return uniqueGrpId;}
```

5.2.2.2.6: The getUserSecurityName and getGroupSecurityName methods

The getUserSecurityName and getGroupSecurityName methods allow theretrieval of the name of a user or group from a unique identifier.

WebSphere Application Server expects the methods to throw theEntryNotFoundException exception if the unique identifier does not exist inthe registry and to throw the CustomRegistryException exception for any otherconditions.

Figure 11 shows theimplementation of the getUserSecurityName methodfor the example registry. The method iterates through theuser-information file and attempts to locate an entry with the specifiedUID. If the UID is located, the corresponding name field is extractedand returned. If the UID is not found, the EntryNotFoundExceptionexception is thrown. The getGroupSecurityName method does the same workon the group-information file.

Figure 11. Code example: The getUserSecurityName and getGroupSecurityName methods in the FileRegistrySample class

```
public String getUserSecurityName(String uniqueId)   throws CustomRegistryException,
EntryNotFoundException{   String s, usrSecName = null;   BufferedReader in = null;    try {      in
= fileOpen(USERFILENAME);       while ((s=in.readLine())!=null) {         if (!s.startsWith("#")) {
int index = s.indexOf(":");            int index1 = s.indexOf(":", index+1);            int index2 =
s.indexOf(":", index1+1);           if ((s.substring(index1+1,index2)).equals(uniqueId)) {
usrSecName = s.substring(0,index);               break;           }         }      }   }   catch
(Exception ex) {      throw new CustomRegistryException(ex.getMessage());   }   finally {
fileClose(in);   }   if (usrSecName == null) {      EntryNotFoundException ex =          new
EntryNotFoundException(uniqueId);   }    return usrSecName;} public String
getGroupSecurityName(String uniqueId)   throws CustomRegistryException, EntryNotFoundException{
String s, grpSecName = null;   BufferedReader in = null;    try {      in = fileOpen(GROUPFILENAME);
...   }   catch (Exception ex) { ... }   finally { ... }   if (grpSecName == null) { ... }    return
grpSecName;}
```

5.2.2.2.7: The getUserDisplayName and getGroupDisplayName methods

The getUserDisplayName and getGroupDisplayName methods allow the retrievalof the display name, a descriptive field, associated with the name of a useror group. In the example registry, the annotation field is returned asthe display name.

WebSphere Application Server expects the methods to throw theEntryNotFoundException exception if the specified user or group name is notfound in the registry and to throw the CustomRegistryException exception forany other conditions. The display name is an optional value, so themethods must return NULL when no display name is found for named user or group.

 The getRemoteUser or getUserPrincipal method in a servlet or JSP, andthe getCallerPrincipal method in an enterprise bean, also use the displayname. These methods return the display name if one exists and the user nameif a display name does not exist. Group display names are not an issue.

Figure 12 shows theimplementation of the getUserDisplayName methodfor the example registry. The method calls the isValidUser method,described in Figure 7,to verify that the name appears in the registry. Ifit does not, the method throws the EntryNotFoundException exception. Ifthe user name is valid, the corresponding annotation field is extracted andreturned. The getGroupSecurityName method does the same work on thegroup-information file.

Figure 12. Code example: The getUserDisplayName and getGroupDisplayName methods in the FileRegistrySample class

```
public String getUserDisplayName(String userName)   throws CustomRegistryException,
EntryNotFoundException{   String s, displayName = null;   BufferedReader in = null;
if(!isValidUser(userName)) {      EntryNotFoundException nsee =         new
EntryNotFoundException(userName);      throw nsee;   }    try {      in = fileOpen(USERFILENAME);
while ((s=in.readLine())!=null) {         if (!s.startsWith("#")) {            int index =
s.indexOf(":");            int index1 = s.lastIndexOf(":");            if
((s.substring(0,index)).equals(userName)) {               displayName = s.substring(index1+1);
break;            }         }      }   }   catch(Exception ex) {      throw new
CustomRegistryException(ex.getMessage());   }   finally {      fileClose(in);   }    return
displayName;} public String getGroupDisplayName(String userName)   throws CustomRegistryException,
EntryNotFoundException{   String s,displayName = null;   BufferedReader in = null;
if(!isValidGroup(userName)) { ... }      try {      in = fileOpen(GROUPFILENAME);      ...   }
catch(Exception ex) { ... }   finally { ... }    return displayName;}
```

5.2.2.2.8: The getGroupsForUser and getUsersForGroup methods

The getGroupsForUser returns a list of the names of the groups to which thenamed user belongs, and the getUsersForGroup method returns a list of the usernames in a group. These methods must return lists of names, not UIDs orGIDs.

WebSphere Application Server expects the methods to throw theEntryNotFoundException exception if the specified user or group name is notfound in the registry and to throw the CustomRegistryException exception forany other conditions.

Figure 12 showsthe implementation of the getGroupsForUser method forthe example registry. Unlike the other user methods, which returninformation about users from the user-information file, this method iteratesover the group-information file, collecting the name of every group that liststhe named user as a member. The group-information file stores eachmember list as a set of names, and the user-information file stores each grouplist as a list of GIDs. By using the group-information file, the methodcan create the list of user names directly. If the method had beenimplemented by iterating over the user-information file, the method would haveto call the getGroupSecurityName method on each GID to construct the list ofgroup names.

In the event of an exception, the method calls the isValidUser method,described in Figure 7,to verify that the user name appears in theregistry. If it does not, the method throws the EntryNotFoundExceptionexception. If the user name is valid, the CustomRegistryExceptionexception is thrown. The getGroupSecurityName method does similar workon the group-information file.

Figure 13. Code example: The getGroupsForUser and getUsersForGroup methods in the FileRegistrySample class

```
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("#")) {            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(userName);      }      throw new CustomRegistryException(ex.getMessage());
}   finally {      fileClose(in);   }    return grpsForUser;} public List getUsersForGroup(String
userName)   throws CustomRegistryException, EntryNotFoundException{   String s;   List usrsForGroup
= new ArrayList();   BufferedReader in = null;    try {      in = fileOpen(GROUPFILENAME);      ...
}   catch (Exception ex) { ... }   finally { ... }    return usrsForGroup;}
```

5.2.2.2.9: The getUniqueUserIds and getUniqueGroupIds methods

The getUniqueUserIds and getUniqueGroupIds methods allow the retrieval ofthe unique identifiers for all members of a group and all the identifiers forthe groups to which a user belongs. These methods are similar infunction to the getGroupsForUser and getUsersForGroup methods; thedifference is that these methods take unique identifiers are arguments andreturn lists of unique identifiers, and the getGroupsForUser andgetUsersForGroup methods work with names. The similarly namedgetUniqueUserId and getUniqueGroupId methods also take user and group namesare arguments.

WebSphere Application Server expects these methods to return null values ifno matches are found, to throw the EntryNotFoundException exception if therequested user or group identifier does not exist in the registry, and tothrow the CustomRegistryException exception for any other conditions.

Figure 14 shows the implementationof the getUniqueUserIds method forthe example registry. The method iterates through the group-informationfile and attempts to locate an entry with the specified groupidentifier. If the identifier is located, the identifiers of allmembers are extracted and returned. If the group identifier is notfound in the file, the EntryNotFoundException exception is thrown. ThegetUniqueGroupIds method does similar work on the user-informationfile.

Figure 14. Code example: The getUniqueUserIds and getUniqueGroupIds methods in the FileRegistrySample class

```
public List getUniqueUserIds(String uniqueGroupId)   throws CustomRegistryException,
EntryNotFoundException{   String s = null;   List uniqueUserIds = new ArrayList();   BufferedReader
in = null;    try {      in = fileOpen(GROUPFILENAME);      while ((s=in.readLine())!=null)      {
if (!s.startsWith("#")) {          int index = s.indexOf(":");     int index1 = s.indexOf(":",
index+1);           if ((s.substring(index+1,index1)).equals(uniqueGroupId)) {
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())
uniqueUserIds.add(getUniqueUserId(st1.nextToken()));              break;       }    }      }   }
catch(Exception ex) {      throw new CustomRegistryException(ex.getMessage());   }   finally {
fileClose(in);   }    return uniqueUserIds;} public List getUniqueGroupIds(String uniqueUserId)
throws CustomRegistryException,     EntryNotFoundException{   String s,uniqueGrpId = null;   List
uniqueGrpIds=new ArrayList();   BufferedReader in = null;    try {      in = fileOpen(USERFILENAME);
while ((s=in.readLine())!=null)      {        ...      }   }   catch(Exception ex) { ... }   finally
{ ... }    return uniqueGrpIds;}
```

5.2.2.2.10: The mapCertificate and checkPassword methods

The mapCertificate and checkPassword methods allow users to beauthenticated against the custom registry. Both methods return a username, which is typically the name of the authenticated user. In somecases, however, it is desirable to authenticate a user but return a differentvalid user name. For example, consider a Web site that offers usersdifferent services depending on their subscription level. When a userenters the site, he or she is prompted for login information, which is used toauthenticate the user and determine the subscription level. All usersat one subscription level can then be assigned the same user name, and usersat another subscription level can be assigned a different one. Becauseauthorization is based on the subscription level rather than a user'sidentity, and there are fewer subscription levels than individual users, thisapproach simplifies the authorization procedures for the application.

The mapCertificate method takes a X.509 certificate as an argumentand returns a valid user name as the return value. Typically, thecertificate holder's name is extracted from the certificate,authenticated against the registry, and returned. WebSphere ApplicationServer expects the method to throw the CertificateMapNotSupportedExceptionexception if the registry does not support mapping to certificates, to throwthe CertificateMapFailedException is expected if the certficate does notrepresent a valid user in the registry, and to throw theCustomRegistryException exception for any other conditions.

Figure 15 shows theimplementation of the mapCertificate method forthe example registry. The method extracts the user name from thecertificate and returns it.

Figure 15. Code example: The mapCertificate method in the FileRegistrySample class

```
public String mapCertificate(X509Certificate cert)   throws CertificateMapNotSupportedException,
CertificateMapFailedException,          CustomRegistryException{   String name=null;   try {      //
Extract the SubjectDN from the certificate.      name = cert.getSubjectDN().getName();   }
catch(Exception ex) {      throw new CertificateMapNotSupportedException(ex.getMessage());   }   //
Determine if the SubjectDN represents a valid user.   if(!isValidUser(name)) {      throw new
CertificateMapFailedException(name);   }   return name;}
```

The checkPassword method verifies that the password submitted for a username matches the password recorded in the registry for that user.WebSphere Application Server expects the method to throw thePasswordCheckFailedException exception if the supplied password does not matchthe recorded password and to throw the CustomRegistryException exception forany other conditions.

Figure 16 shows theimplementation of the checkPassword method for theexample registry. The method locates the entry for the user in theuser-information file and matches the supplied password againt the value ofthe password field. If the passwords do not match, thePasswordCheckFailedException exception is thrown; otherwise, the methodreturns the name of the authenticated user.

Figure 16. Code example: The checkPassword method in the FileRegistrySample class

```
public String checkPassword(String userId, String passwd)    throws PasswordCheckFailedException,
CustomRegistryException{   String s, userName = null;   BufferedReader in = null;    try {      in =
fileOpen(USERFILENAME);       while ((s=in.readLine())!=null)      {         if (!s.startsWith("#"))
{            int index = s.indexOf(":");            int index1 = s.indexOf(":",index+1);
// Check existence of the username/password pair.            if
((s.substring(0,index)).equals(userId) &&
s.substring(index+1,index1).equals(passwd)) {               // The username and password match the
registry,               // so authentication succeeds.               userName = userId;
break;            }         }      }   }   catch(Exception ex) {      throw new
CustomRegistryException(ex.getMessage());   }   finally {      fileClose(in);   }       if (userName
== null)   {      throw new PasswordCheckFailedException(userId);   }     return userName;}
```

5.2.3: Building and configuring the sample user registry application

To use the sample custom registry, perform the following steps:

1. Build the FileRegistrySample application.
2. Configure WebSphere Application Server to use the FileRegistrySample registry.

This section describes these procedures.

Building the FileRegistrySample application

This section describes how to build the sample described in thisarticle. This sample has been designed more for simplicity thanperformance and is intended only to familiarize you with the custom-registryfeature. An implementation intended for production use requires muchbetter scalability and performance.

The sample consists of the following files:

- FileRegistrySample.java: the sample implementation itself
- users.props: the users information in the registry
- groups.props: the groups information in the registry

The complete source code is provided elsewhere in this package.

To run this sample, you must first build it and then configure it foruse. This discussion assumes that:

- WebSphere Application Server is installed in the C:\WebSphere\AppServer directory.
- The sample is being run under Windows. The main difference between Windows and other platforms is where the files are located.

To build the sample, follow these steps:

1. Copy the FileRegistrySample.java file to a directory, for example, C:\temp.
2. Add the C:\WebSphere\AppServer\lib\websphere.jar file to the classpath.
3. Compile the sample by using the Java compiler that is shipped with WebSphere Application Server. After compilation, you will have two class files:
    - FileRegistrySample.class
    - RegExpSample.class
4. Copy the two class files to a directory that is on the classpath. For this sample, the C:\WebSphere\AppServer\classes directory is used because it is already on the classpath. Alternatively, you can add the directory in which the files reside, or a JAR file containing the files, to the classpath by modifying the value of the classpath in the appropriate configuration files, for example, on Windows platforms, admin.config and adminserver.bat.

Configuring the custom registry

Setting up security for a custom registry is very similar to setting upsecurity for LDAP. If you are unfamiliar with the configuration ofsecurity in WebSphere Application Server, see Administering applications for more information about the process.

A custom registry is enabled by using the Security Center panel in theadministrative console. On the Authentication panel, choose**Lightweight Third Party Authentication (LTPA)** as the authenticationmechanism. Select the **Custom User Registry** button and fillin the required values for the

following in the Custom User Registry Settingssection:

- Security Server ID
- Security Server Password
- Custom User Registry classname

The server ID and password combination must exist in the customregistry. The class name is the file in which you have implemented theCustomRegistry interface, for example,com.myCompany.mySample. This class file must be in theclasspath environment variable of WebSphere Application Server. For the FileRegistrySampleapplication, use the following values:

- Security Server ID: dave
- Security Server Password: dave1
- Custom User Registry classname: FileRegistrySample

You can use also the **Special custom settings** button to createproperties that are specific to your custom registry. All propertiesset here are provided to your implementation class during run time when theinitialize method is called.

For the FileRegistrySample application, two additional properties areneeded; they are used for locating the files that make up the registry. Setthe usersFile property to the location of the users.props file; set thegroupsFile property to the location of the groups.props file. For example,if these files are stored in the C:\temp directory, insert the followingcustom settings:

- usersFile -- C:\temp\users.props
- groupsFile -- C:\temp\groups.props

When the required information has been entered, click the **OK**button. Restart WebSphere Application Server. When it restarts,the custom registry is in use. The information in the users.propsand groups.props files is now the information against whichauthentication and authorization requests are checked.

You can also use the XMLConfig tool to update the configurationinformation. When properties are entered using the **Specialcustom settings** button on the administrative console,the properties are stored with the prefix `Custom_`in the database; this way, the administrative console candistinguish properties associated with the custom registryfrom other properties. The prefix is stripped off and the restof the name is passed to the implementation. When using the XMLConfigtool to update the configuration, the string `Custom_` must beprefixed to the name of the property as it appears in theadministrative console. For example, the usersFile and groupsFileproperties described for the sample application must bereferred to as Custom_usersFile and Custom_groupsFile if youuse the XMLConfig tool to modify them.

5.2.4: Custom-registry source code

The files collected here comprise the source code forthe sample implementation of a custom registry, the FileRegistrySample,and the source code for the custom registry component.

5.2.4.1: Source code for the FileRegistrySample application

The files collected here comprise the FileRegistrySample implementationof a custom registry. The material is organized as follows:

- The FileRegistrySample.java file
- A file containing the two properties files:
  - users.props
  - groups.props

5.2.4.1.1: The FileRegistrySample.java file

```
////  5639-D57 (C) COPYRIGHT International Business Machines Corp. 2001////  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//----------------------------------------------------------------------// The main purpose of this sample is to demonstrate the use of the// Custom Registry feature available in WebSphere. This sample is a very // simple 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
etc.//----------------------------------------------------------------------import
java.util.*;import java.io.*;import java.security.cert.X509Certificate;import
com.ibm.websphere.security.*;public class FileRegistrySample implements CustomRegistry {   private
static String USERFILENAME = null;          private static String GROUPFILENAME = null;   public
FileRegistrySample() {}    // Default Constructor   /**    * Initializes the registry.    * @param
props the registry-specific properties with which to     * initialize the registry object.    *
@exception CustomRegistryException if the registry is "bad".    **/   public void
initialize(java.util.Properties props)         throws CustomRegistryException {      try {
/* try getting the USERFILENAME and the GROUPFILENAME from           * properties that are passed in
(i.e from GUI).          * These values should be set in the security center GUI in 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());      }
if (USERFILENAME == null || GROUPFILENAME == null) {         throw new
CustomRegistryException("users/groups information missing");      }   }   /**    * Checks the
Password of the user.    * @param userId the user name data to authenticate.    * @param passwd the
password of the user.    * @return the userId that will be used for authentication.    * @exception
WrongPasswordException if passwd is not valid.    * @exception CustomRegistryException if this
Registry is "bad".    **/   public String checkPassword(String userId, String passwd)       throws
PasswordCheckFailedException,             CustomRegistryException {      String s,userName = null;
BufferedReader in = null;          try {         in = fileOpen(USERFILENAME);         while
((s=in.readLine())!=null)         {            if (!s.startsWith("#")) {              int index =
s.indexOf(":");               int index1 = s.indexOf(":",index+1);              // check if the
userId:passwd combination exists               if ((s.substring(0,index)).equals(userId) &&
s.substring(index+1,index1).equals(passwd)) {                  // Authentication successful, return
the userId.                  userName = userId;                  break;               }           }
}      } catch(Exception ex) {         throw new CustomRegistryException(ex.getMessage());      }
finally {         fileClose(in);      }             if (userName == null) {         throw new
PasswordCheckFailedException(userId);      }       return userName;   }  /**   * Maps a Certificate
(of X509 format) to a valid userId in the Registry.   * @param cert the certificate that needs to be
mapped.   * @return the mapped name of the user (userId).   * @exception
CertificateMapNotSupportedException if the particular   * certificate is not supported.   *
@exception CertificateMapFailedException if the mapping of the   * certificate fails.   * @exception
CustomRegistryException if the registry is "bad".   **/   public String
mapCertificate(X509Certificate cert)      throws CertificateMapNotSupportedException,
CertificateMapFailedException,             CustomRegistryException {      String name=null;      try
{         // map the SubjectDN in the certificate to a userID.         name =
cert.getSubjectDN().getName();      } catch(Exception ex) {         throw new
CertificateMapNotSupportedException(ex.getMessage());      }      if(!isValidUser(name)) {
throw new CertificateMapFailedException(name);      }      return name;   }   /**    * Returns the
realm of the registry.    * @return the realm. The realm is a registry-specific string indicating
the    * <i>realm</i> or <i>domain</i> for which this registry applies.  E.g. 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".    * @exception
CustomRegistryException if the registry is "bad".    **/   public String getRealm()      throws
CustomRegistryException {      String name = "customRealm";      return name;   }  /**   * Returns
names of all the users in the registry.   * @return a List of the names of all the users.   *
@exception CustomRegistryException if the registry is "bad".   **/   public List getUsers()
throws CustomRegistryException {      String s;      BufferedReader in = null;      List allUsers =
new ArrayList();      try {         in = fileOpen(USERFILENAME);         while
((s=in.readLine())!=null)         {            if (!s.startsWith("#")) {               int index =
s.indexOf(":");               allUsers.add(s.substring(0,index));            }         }      }
catch (Exception ex) {         throw new CustomRegistryException(ex.getMessage());      } finally {
fileClose(in);      }      return allUsers;   }  /**   * Returns names of the users in the registry
```

```
that match a pattern.   * @param pattern the pattern to match. (For e.g., a* will match all   *
userNames starting with a). At a minimum when a full name is used   * as the pattern the full name
should be returned back if it is a   * valid user.   * @return a List of the names of all the users
that match the pattern.   * @exception CustomRegistryException if the registry is "bad".   **/
public List getUsers(String pattern)      throws CustomRegistryException {      String s;
BufferedReader in = null;      List allUsers = new ArrayList();      try {         in =
fileOpen(USERFILENAME);         while ((s=in.readLine())!=null)         {            if
(!s.startsWith("#")) {               int index = s.indexOf(":");               String user =
s.substring(0,index);               if (match(user,pattern))               allUsers.add(user);
}         }      } catch (Exception ex) {         throw new
CustomRegistryException(ex.getMessage());      } finally {         fileClose(in);      }      return
allUsers;   }  /**   * Returns the names of the all the users in a group.   * @param groupName the
name of the group.   * @return a List of all the names of the users in the group.   * @exception
EntryNotFoundException if groupName does not exist.   * @exception CustomRegistryException if the
registry is "bad".   **/   public List getUsersForGroup(String groupName)      throws
CustomRegistryException,             EntryNotFoundException {      String s;      BufferedReader in
= null;      List usrsForGroup = new ArrayList();      try {         in = fileOpen(GROUPFILENAME);
while ((s=in.readLine())!=null)         {            if (!s.startsWith("#")) {               int
index = s.indexOf(":");               if ((s.substring(0,index)).equals(groupName))               {
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())
usrsForGroup.add(st1.nextToken());               }            }         }      } catch (Exception
ex) {         if (!isValidGroup(groupName)) {            throw new
EntryNotFoundException(groupName);         }         throw new
CustomRegistryException(ex.getMessage());      } finally {         fileClose(in);      }      return
usrsForGroup;   }   /**   * Returns the display name for the user specified by userName.   * @param
userName 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.   * @exception EntryNotFoundException if userName
does not exist.   * @exception CustomRegistryException if the registry is "bad".   **/   public
String getUserDisplayName(String userName)      throws CustomRegistryException,
EntryNotFoundException {      String s,displayName = null;      BufferedReader in = null;
if(!isValidUser(userName)) {         EntryNotFoundException nsee = new
EntryNotFoundException(userName);         throw nsee;      }      try {         in =
fileOpen(USERFILENAME);         while ((s=in.readLine())!=null)         {            if
(!s.startsWith("#")) {               int index = s.indexOf(":");               int index1 =
s.lastIndexOf(":");               if ((s.substring(0,index)).equals(userName)) {
displayName = s.substring(index1+1);                  break;               }            }         }
} catch(Exception ex) {         throw new CustomRegistryException(ex.getMessage());      } finally {
fileClose(in);      }      return displayName;   }   /**   * Returns the UniqueId for a userName.
* @param userName the name of the user.   * @return the UniqueId of the user. The UniqueId for an
user is   * the stringified form of some unique, registry-specific, data that   * serves to
represent the user.  E.g. for the UNIX user registry, the   * UniqueId for a user can be the UID.
* @exception EntryNotFoundException if userName does not exist.   * @exception
CustomRegistryException if the registry is "bad".   **/   public String getUniqueUserId(String
userName)      throws CustomRegistryException,             EntryNotFoundException {      String
s,uniqueUsrId = null;      BufferedReader in = null;      try {         in = fileOpen(USERFILENAME);
while ((s=in.readLine())!=null)         {            if (!s.startsWith("#")) {               int
index = s.indexOf(":");               int index1 = s.indexOf(":", index+1);               if
((s.substring(0,index)).equals(userName)) {                  int index2 = s.indexOf(":", index1+1);
uniqueUsrId = s.substring(index1+1,index2);                  break;               }            }
}      } catch(Exception ex) {         throw new CustomRegistryException(ex.getMessage());      }
finally {         fileClose(in);      }      if (uniqueUsrId == null) {
EntryNotFoundException nsee = new EntryNotFoundException(userName);         throw nsee;      }
return uniqueUsrId;   }  /**   * Returns the UniqueIds for all the users that belong to a group.   *
@param uniqueGroupId the uniqueId of the group.   * @return a List of all the user Unique ids that
are contained in the   * group whose Unique id matches the uniqueGroupId.   * The Unique id for an
entry is the stringified form of some unique,   * registry-specific, data that serves to represent
the entry.  E.g. 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 uniqueGroupId
does not exist.   * @exception CustomRegistryException if the registry is "bad".   **/   public List
getUniqueUserIds(String uniqueGroupId)      throws CustomRegistryException,
EntryNotFoundException {      String s = null;      List uniqueUserIds = new ArrayList();
BufferedReader in = null;      try {         in = fileOpen(GROUPFILENAME);         while
((s=in.readLine())!=null)         {            if (!s.startsWith("#")) {               int index =
s.indexOf(":");               int index1 = s.indexOf(":", index+1);               if
((s.substring(index+1,index1)).equals(uniqueGroupId)) {                  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())
uniqueUserIds.add(getUniqueUserId(st1.nextToken()));                  break;               }
}         }      } catch(Exception ex) {         throw new CustomRegistryException(ex.getMessage());
```

```
} finally {         fileClose(in);      }      return uniqueUserIds;   }  /**   * Returns the name
for a user given its uniqueId.   * @param uniqueUserId the UniqueId of the user.   * @return the
name of the user.   * @exception EntryNotFoundException if the uniqueUserId does not exist.   *
@exception CustomRegistryException if the registry is "bad".   **/   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("#")) {               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());      } finally {
fileClose(in);      }      if (usrSecName == null) {         EntryNotFoundException ex =
new EntryNotFoundException(uniqueUserId);         throw ex;      }      return usrSecName;   }  /**
* Determines if a user exists.   * @param userName the name of the user.   * @return true if the
user exists; false otherwise.   * @exception CustomRegistryException if the registry is "bad".   **/
public boolean isValidUser(String userName)      throws CustomRegistryException {      String s;
boolean isValid = false;      BufferedReader in = null;      try {         in =
fileOpen(USERFILENAME);         while ((s=in.readLine())!=null)         {            if
(!s.startsWith("#")) {               int index = s.indexOf(":");               if
((s.substring(0,index)).equals(userName)) {                  isValid=true;                  break;
}            }         }      } catch (Exception ex) {         throw new
CustomRegistryException(ex.getMessage());      } finally {         fileClose(in);      }      return
isValid;   }  /**   * Returns names of all the groups in the registry.   * @return a List of the
names of all the groups.   * @exception CustomRegistryException if the registry is "bad".   **/
public List getGroups()      throws CustomRegistryException {      String s;      BufferedReader in
= null;      List allGroups = new ArrayList();      try {         in = fileOpen(GROUPFILENAME);
while ((s=in.readLine())!=null)         {            if (!s.startsWith("#")) {               int
index = s.indexOf(":");               allGroups.add(s.substring(0,index));            }         }
} catch (Exception ex) {         throw new CustomRegistryException(ex.getMessage());      } finally
{         fileClose(in);      }      return allGroups;   }  /**   * Returns names of the groups in
the registry that match a pattern.   * @param pattern the pattern to match. (For e.g., a* will match
all   * group names starting with a). At a minimum when a full name is used   * as the pattern the
full name should be returned back if it is a   * valid group.   * @return a List of the names of the
groups.   * @exception CustomRegistryException if the registry is "bad".   **/   public List
getGroups(String pattern)      throws CustomRegistryException {      String s;      BufferedReader
in = null;      List allGroups = new ArrayList();      try {         in = fileOpen(GROUPFILENAME);
while ((s=in.readLine())!=null)         {            if (!s.startsWith("#")) {               int
index = s.indexOf(":");               String group = s.substring(0,index);               if
(match(group,pattern))                  allGroups.add(group);            }         }      } catch
(Exception ex) {         throw new CustomRegistryException(ex.getMessage());      } finally {
fileClose(in);      }      return allGroups;   }  /**   * Returns the names of the groups to which
userName belongs.   * @param userName the username of the user.   * @return a List of the names of
all the groups that the user belongs to.   * @exception EntryNotFoundException if userName does not
exist.   * @exception CustomRegistryException if the registry is "bad".   **/   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("#")) {               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(userName);         }
throw new CustomRegistryException(ex.getMessage());      } finally {         fileClose(in);      }
return grpsForUser;   }  /**   * Returns the display name for a group.   * @param groupName 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.   * @exception EntryNotFoundException if the groupName does not exist.   * @exception
CustomRegistryException if the registry is "bad".   **/   public String getGroupDisplayName(String
groupName)      throws CustomRegistryException,            EntryNotFoundException {      String
s,displayName = null;      BufferedReader in = null;      if(!isValidGroup(groupName)) {
EntryNotFoundException nsee = new EntryNotFoundException(groupName);         throw nsee;      }
try {         in = fileOpen(GROUPFILENAME);         while ((s=in.readLine())!=null)         {
if (!s.startsWith("#")) {               int index = s.indexOf(":");               int index1 =
s.lastIndexOf(":");               if ((s.substring(0,index)).equals(groupName)) {
displayName = s.substring(index1+1);                  break;               }            }         }
} catch(Exception ex) {         throw new CustomRegistryException(ex.getMessage());      } finally {
fileClose(in);      }         return displayName;   }  /**   * Returns the Unique id for a group.
* @param groupName 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 entry.  E.g. for the   * Unix user registry, the Unique id could be the GID for the
```

```
entry.   * @exception EntryNotFoundException if groupName does not exist.   * @exception
CustomRegistryException if the registry is "bad".   **/   public String getUniqueGroupId(String
groupName)      throws CustomRegistryException,            EntryNotFoundException {      String
s,uniqueGrpId = null;      BufferedReader in = null;      try {         in =
fileOpen(GROUPFILENAME);         while ((s=in.readLine())!=null)         {            if
(!s.startsWith("#")) {               int index = s.indexOf(":");               int index1 =
s.indexOf(":", index+1);               if ((s.substring(0,index)).equals(groupName)) {
uniqueGrpId = s.substring(index+1,index1);                  break;             }           }
}      } catch(Exception ex) {         throw new CustomRegistryException(ex.getMessage());      }
finally {         fileClose(in);      }      if (uniqueGrpId == null) {
EntryNotFoundException nsee = new EntryNotFoundException(groupName);         throw nsee;      }
return uniqueGrpId;   }  /**   * Returns the Unique id for a group.   * @param groupName 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 entry.  E.g. for the
* Unix user registry, the Unique id could be the GID for the entry.   * @exception
EntryNotFoundException if groupName does not exist.   * @exception CustomRegistryException if the
registry is "bad".   **/   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("#")) {               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());      } finally {
fileClose(in);      }      return uniqueGrpIds;   }   /**   * Returns the name for a group given its
uniqueId.   * @param uniqueGroupId the UniqueId of the group.   * @return the name of the group.   *
@exception EntryNotFoundException if the uniqueGroupId does not exist.   * @exception
CustomRegistryException if the registry is "bad".   **/   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("#")) {               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());      } finally {
fileClose(in);      }      if (grpSecName == null) {         EntryNotFoundException ex =
new EntryNotFoundException(uniqueGroupId);         throw ex;      }      return grpSecName;   }  /**
* Determines if a group exists.   * @param groupName the name of the group.   * @return true if the
group exists; false otherwise.   * @exception CustomRegistryException if the registry is "bad".
**/   public boolean isValidGroup(String groupName)      throws CustomRegistryException {
String s;      boolean isValid = false;      BufferedReader in = null;      try {         in =
fileOpen(GROUPFILENAME);         while ((s=in.readLine())!=null)         {            if
(!s.startsWith("#")) {               int index = s.indexOf(":");               if
((s.substring(0,index)).equals(groupName)) {                  isValid=true;                  break;
}            }         }      } catch (Exception ex) {         throw new
CustomRegistryException(ex.getMessage());      } finally {         fileClose(in);      }      return
isValid;   }   private BufferedReader fileOpen(String fileName)      throws FileNotFoundException {
try {         return new BufferedReader(new FileReader(fileName));      }
catch(FileNotFoundException e) {         throw e;      }   }   // private methods   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[];}
```

5.2.4.1.2: Properties files for FileRegistrySampleapplication

The users.props file

```
# Here is the format for the users.props file# 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:bobdave:dave1:234:678:jay:jay1:345:678,789:Jay-Jayted:ted1:456:678:Teddy
Gjeff:jeff1:222:789:Jeffvikas:vikas1:333:789:vikasbobby:bobby1:444:789:
```

The groups.props file

```
# Here is the format for the groups.props file# 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 groupoperators:678:jay,ted,dave:Operators
groupusers:789:jay,jeff,vikas,bobby:
```

5.2.4.2: Source code for the custom-registry component

The files collected here comprise the custom-registry component.This includes the interface, CustomRegistry, that must be implemented,as well as several exception classes. The material is organized as follows:

- The CustomRegistry.java file
- The CustomRegistryException.java file
- The PasswordCheckFailedException.java file
- The EntryNotFoundException.java file
- The CertificateMapNotSupportedException.java file
- The CertificateMapFailedException.java file

5.2.4.2.1: The CustomRegistry.java file

```
// IBM Confidential OCO Source Material// 5648-C83, 5648-C84 (C) COPYRIGHT International Business Machines Corp. 2001// The source code for this program is not published or otherwise divested// of its trade secrets, irrespective of what has been deposited with the// U.S. Copyright Office.package com.ibm.websphere.security;import java.util.*;import java.security.cert.X509Certificate;/** * The CustomRegistry interface provides an API that supports the following registry entry types: *
```

- user *
- group *

```
 * Implementation of this interface must provide implementations for: *
```

- initialize *
- checkPassword *
- mapCertificate *
- getRealm *
- getUsers *
- getUsers(String) *
- getUsersForGroup *
- getUserDisplayName *
- getUniqueUserId *
- getUniqueUserIds *
- getUserSecurityName *
- isValidUser *
- getGroups *
- getGroups(String) *
- getGroupsForUser *
- getGroupDisplayName *
- getUniqueGroupId *
- getUniqueGroupIds *
- getGroupSecurityName *
- isValidGroup *

```
**/public interface CustomRegistry {   /*   *  In all of the methods in this interface if the return type is a String    *  then an empty String or a failure in the method should return null.   *  If the return type is a List, return null for a failure or for a list   *  with no entries.   */  /** * Initializes the registry.   * @param props the registry-specific properties with which to initialize the   * registry object.   * @exception CustomRegistryException if the registry is "bad". **/   public void initialize(java.util.Properties props)      throws CustomRegistryException;  /** * Checks the password of the user.   * @param userId the username whose password needs to be checked.   * @param password the password of the userId.   * @return a valid username (this can be the same userId whose password   * was checked or it could be some other userId in the registry if the   * implementation was to do so).   * @exception CheckPasswordFailedException if userId/password * combination does not exist in the registry.   * @exception CustomRegistryException if the registry is "bad".   **/   public String checkPassword(String userId, String password)      throws PasswordCheckFailedException,            CustomRegistryException;  /**   * Maps a Certificate (of X509 format) to a valid userId in the Registry.   * @param cert the certificate that needs to be mapped.   * @return the mapped name of the user (userId).    * @exception CertificateMapNotSupportedException if the particular   * certificate is not supported.   * @exception CertificateMapFailedException if the mapping of the    * certificate fails.   * @exception CustomRegistryException if the registry is "bad".   **/   public String mapCertificate(X509Certificate cert)      throws CertificateMapNotSupportedException, CertificateMapFailedException,            CustomRegistryException;  /**   * 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.  E.g. 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".   * @exception CustomRegistryException if the registry is "bad".   **/   public String getRealm()      throws CustomRegistryException;  /** * Returns names of all the users in the registry.   * @return a List of the names of all the users. * @exception CustomRegistryException if the registry is "bad".   **/   public List getUsers() throws CustomRegistryException;  /**   * Returns names of the users in the registry that match a pattern.   * @param pattern the pattern to match. (For e.g., a* will match all   * userNames starting with a)   * @return a List of the names of all the users that match the pattern.   *
```

```
@exception CustomRegistryException if the registry is "bad".   **/   public List getUsers(String
pattern)      throws CustomRegistryException;  /**   * Returns the names of the all the users in a
group.   * @param groupName the name of the group.   * @return a List of all the names of the users
in the group.   * @exception EntryNotFoundException if groupName does not exist.   * @exception
CustomRegistryException if the registry is "bad".   **/   public List getUsersForGroup(String
groupName)      throws EntryNotFoundException,             CustomRegistryException;  /**   * Returns
the display name for the user specified by userName.   * @param userName 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.   * @exception EntryNotFoundException if userName does not exist.   *
@exception CustomRegistryException if the registry is "bad".   **/   public String
getUserDisplayName(String userName)      throws EntryNotFoundException,
CustomRegistryException;  /**   * Returns the UniqueId for a userName.   * @param userName the name
of the user.   * @return the UniqueId of the user. The UniqueId for an user is   * the stringified
form of some unique, registry-specific, data that    * serves to represent the user.  E.g. for the
UNIX user registry, the    * UniqueId for a user can be the UID.   * @exception
EntryNotFoundException if userName does not exist.   * @exception CustomRegistryException if the
registry is "bad".   **/   public String getUniqueUserId(String userName)      throws
EntryNotFoundException,             CustomRegistryException;  /**   * Returns the UniqueIds for all
the users that belong to a group.   * @param uniqueGroupId the uniqueId of the group.   * @return a
List of all the user Unique ids that are contained in the   * group whose Unique id matches the
uniqueGroupId.   * The Unique id for an entry is the stringified form of some unique,   *
registry-specific, data that serves to represent the entry.  E.g. 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 uniqueGroupId does not exist.   * @exception
CustomRegistryException if the registry is "bad".   **/   public List getUniqueUserIds(String
uniqueGroupId)      throws EntryNotFoundException,             CustomRegistryException;  /**   *
Returns the name for a user given its uniqueId.   * @param uniqueUserId the UniqueId of the user.
* @return the name of the user.   * @exception EntryNotFoundException if the uniqueUserId does not
exist.   * @exception CustomRegistryException if the registry is "bad".   **/   public String
getUserSecurityName(String uniqueUserId)      throws EntryNotFoundException,
CustomRegistryException;  /**   * Determines if a user exists.   * @param userName the name of the
user.   * @return true if the user exists; false otherwise.   * @exception CustomRegistryException
if the registry is "bad".   **/   public boolean isValidUser(String userName)      throws
CustomRegistryException;  /**   * Returns names of all the groups in the registry.   * @return a
List of the names of all the groups.   * @exception CustomRegistryException if the registry is
"bad".   **/   public List getGroups()      throws CustomRegistryException;  /**   * Returns names
of the groups in the registry that match a pattern.   * @param pattern the pattern to match.   *
@return a List of the names of the groups.   * @exception CustomRegistryException if the registry is
"bad".   **/   public List getGroups(String pattern)      throws CustomRegistryException;  /**   *
Returns the names of the groups to which userName belongs.   * @param userName the username of the
user.   * @return a List of the names of all the groups that the user belongs to.   * @exception
EntryNotFoundException if userName does not exist.   * @exception CustomRegistryException if the
registry is "bad".   **/   public List getGroupsForUser(String userName)      throws
EntryNotFoundException,             CustomRegistryException;  /**   * Returns the display name for a
group.   * @param groupName 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.   * @exception EntryNotFoundException if the groupName does not exist.   *
@exception CustomRegistryException if the registry is "bad".   **/   public String
getGroupDisplayName(String groupName)      throws EntryNotFoundException,
CustomRegistryException;  /**   * Returns the Unique id for a group.   * @param groupName 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 entry.
E.g. for the   * Unix user registry, the Unique id could be the GID for the entry.    * @exception
EntryNotFoundException if groupName does not exist.   * @exception CustomRegistryException if the
registry is "bad".   **/   public String getUniqueGroupId(String groupName)      throws
EntryNotFoundException,             CustomRegistryException;  /**   * Returns the Unique ids for all
the groups that contain the UniqueId of   * a user.    * @param uniqueUserId the uniqueId of the
user.   * @return a List of all the group Unique ids that uniqueUserId belongs to.   * The Unique id
for an entry is the stringified form of some unique,    * registry-specific, data that serves to
represent the entry.  E.g. 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
uniqueUserId does not exist.   * @exception CustomRegistryException if the registry is "bad".   **/
public List getUniqueGroupIds(String uniqueUserId)      throws EntryNotFoundException,
CustomRegistryException;  /**   * Returns the name for a group given its uniqueId.   * @param
uniqueGroupId the UniqueId of the group.   * @return the name of the group.   * @exception
EntryNotFoundException if the uniqueGroupId does not exist.   * @exception CustomRegistryException
if the registry is "bad".   **/   public String getGroupSecurityName(String uniqueGroupId)
throws EntryNotFoundException,             CustomRegistryException;  /**   * Determines if a group
exists.   * @param groupName the name of the group.   * @return true if the group exists; false
otherwise.   * @exception CustomRegistryException if the registry is "bad".   **/   public boolean
isValidGroup(String groupName)      throws CustomRegistryException;}
```

5.2.4.2.2: The CustomRegistryException.java file

```
// IBM Confidential OCO Source Material// 5648-C83, 5648-C84 (C) COPYRIGHT International Business Machines Corp. 2001// The source code for this program is not published or otherwise divested// of its trade secrets, irrespective of what has been deposited with the// U.S. Copyright Office.package com.ibm.websphere.security;/** * Thrown to indicate that a error occurred while using the * specified custom registry. */public class CustomRegistryException extends Exception {   /**    * Create a new CustomRegistryException with an empty description string.    */   public CustomRegistryException() {     super();   }   /**    *  Create a new CustomRegistryException with the associated string description.    *    *  @param message the String describing the exception. */   public CustomRegistryException(String message) {     super(message);   }}
```

5.2.4.2.3: The PasswordCheckFailedException.java file

```
// IBM Confidential OCO Source Material// 5648-C83, 5648-C84 (C) COPYRIGHT International Business Machines Corp. 2001// The source code for this program is not published or otherwise divested// of its trade secrets, irrespective of what has been deposited with the// U.S. Copyright Office.package com.ibm.websphere.security;/** * Thrown to indicate that the userId/Password combination does not exist * in the specified custom registry. */public class PasswordCheckFailedException extends Exception {   /**    *  Create a new PasswordCheckFailedException with an empty description string.    */   public PasswordCheckFailedException() {     super();   }   /**    *  Create a new PasswordCheckFailedException with the associated string description.    *    *  @param message the String describing the exception.    */   public PasswordCheckFailedException(String message) {   super(message);   }}
```

5.2.4.2.4: The EntryNotFoundException.java file

```
// IBM Confidential OCO Source Material// 5648-C83, 5648-C84 (C) COPYRIGHT International Business Machines Corp. 2001// The source code for this program is not published or otherwise divested// of its trade secrets, irrespective of what has been deposited with the// U.S. Copyright Office.package com.ibm.websphere.security;/** * Thrown to indicate that the specified entry is not found in the * custom registry. */public class EntryNotFoundException extends Exception {   /**    *  Create a new EntryNotFoundException with an empty description string.    */   public EntryNotFoundException() { super();   }   /**    *  Create a new EntryNotFoundException with the associated string description. *    *  @param message the String describing the exception.    */   public EntryNotFoundException(String message) {     super(message);   }}
```

5.2.4.2.5: The CertificateMapNotSupportedException.java file

```
// IBM Confidential OCO Source Material// 5648-C83, 5648-C84 (C) COPYRIGHT International Business
Machines Corp. 2001// The source code for this program is not published or otherwise divested// of
its trade secrets, irrespective of what has been deposited with the// U.S. Copyright Office.package
com.ibm.websphere.security;/** * Thrown to indicate that the certificate mapping for the * specified
certificate is not supported. */public class CertificateMapNotSupportedException extends Exception {
/**    *  Create a new CertificateMapNotSupportedException with an empty description string.    */
public CertificateMapNotSupportedException() {     super();   }   /**    *  Create a new
CertificateMapNotSupportedException with the associated string description.    *    *  @param
message the String describing the exception.    */   public
CertificateMapNotSupportedException(String message) {     super(message);   }}
```

5.2.4.2.6: The CertificateMapFailedException.java file

```
// IBM Confidential OCO Source Material// 5648-C83, 5648-C84 (C) COPYRIGHT International Business Machines Corp. 2001// The source code for this program is not published or otherwise divested// of its trade secrets, irrespective of what has been deposited with the// U.S. Copyright Office.package com.ibm.websphere.security;/** * Thrown to indicate that a error occurred while mapping the * specified certificate. */public class CertificateMapFailedException extends Exception {   /**    * Create a new CertificateMapFailedException with an empty description string.    */   public CertificateMapFailedException() {     super();   }   /**    *  Create a new CertificateMapFailedException with the associated string description.    *    *  @param message the String describing the exception.    */   public CertificateMapFailedException(String message) { super(message);   }}
```

5.3: Changes to security since Version 3

With version 4.0, WebSphere Application Server adopts the security model described inthe Java 2 Enterprise Edition (J2EE) specification. This specification describestechniques for creating, assembling, deploying, and securing enterprise applications. Thesecurity-related aspects of J2EE are now supported by WebSphere and include the following:

- The use of J2EE deployment descriptors to declaratively specify various security constraints for Web and enterprise-bean resources. This change is important because many of an application's security attributes are now specified during the creation and assembly phases instead of during the deployment phase. In Version 3.x, most application-level security attributes are specified during the deployment phase.
- The use of role-based authorization.

Many security features have changed with respect to the security offered by IBMWebSphere Application Server Version 3. This table summarizes the differences.

Version 4	Version 3.x
When global security is enabled, only the resources of the administrative application are protected. All other resources are unprotected.	When global security is enabled, enterprise beans are protected by default.
WebSphere no longer secures or protects URIs, for example, HTML files and CGI scripts, that are served by an external Web server, for example, Apache or IHS. WebSphere secures or protects only URIs served by WebSphere. URIs not served by WebSphere can be protected with IBM's WebSeal security solution, or the URIs and the resources they represent can be restructured and packaged in a Web application archive (a WAR file) so that WebSphere can serve them.	WebSphere can protect URIs served by an external Web server.
Deployment descriptors are provided in XML. The web.xml, ejb-jar.xml, and application.xml deployment-descriptor files are used to declare security contraints. Security constraints include the identification of the methods belonging to roles, the login configuration or challenge mechanism, whether HTTPS/SSL is required, and so forth. The application assembly tool (AAT) is used to create and manipulate deployment descriptors and the various archive (EAR, WAR, and JAR) files that contain them.	Most of application-specific security attributes are defined by using the administrative console during the application's deployment phase.
The login configuration and challenge type apply to individual Web applications, not to individual enterprise applications.	The challenge type applies to an entire enterprise application.

The local operating-system user registry now supports J2EE form-based login configuration. This means that AEs can now supports the form-based login configuration. J2EE form-based login replaces AbstractLoginServlet, CustomLoginServlet, and SSOAuthenticator, which are now deprecated. Although these features still exist in version 4.0, they are intended to be used for migration purposes only until the application can be modified to use J2EE form-based login.	AbstractLoginServlet, CustomLoginServlet, and SSOAuthenticator are features used to create custom or form based login mechanisms for web applications. CustomLogin servlets are supported only with the LTPA authentication mechanism, which is available only in Advanced Edition.
Passwords are encoded with a simple masking alogorithm in various ASCII WebSphere configuration files to deter casual observation.	Passwords are in plain text.

5.4: Overview: Using programmatic and form logins

This section describes the use of login specifications (including the use of Single Sign-On)in WebSphere Application Server.

When Java enterprise-bean client applications require the user to provide identifying information,the writer of the application must collect that informationand authenticate the user. The work of the programmer can be broadlyclassified in terms of where the actual user authentication is performed:

1. In a client program
2. In a server program

Users of Web applications can be prompted for authentication data in many ways. The login-config element in the Web application's deployment descriptor defines the mechanism used to collectthis information. Programmers who want to customize login procedures,rather than relying on general-purpose devices like a 401 dialog windowin a browser, can use a form based login to provide an application-specificHTML form for collecting login information.

No authentication occurs unless WebSphere global security isenabled. Additionally, if you want to use form-based login forWeb applications, you must specify "FORM" in the `auth-method`tag in the `login-config` element in the deploymentdescriptor of each Web application.

5.4.1: Client-side login

Use a client-side login when a pure Java client needs to log usersinto the security domain but does not need to use the authenticationdata itself.

Client-side login works in the following manner:

1. The user makes a request to the client application.
2. The client presents the user with a login form for collecting authentication data. The user inserts his or her user ID and password into the form and submits it.
3. The client programmatically places the user's authentication data into an ORB-related data structure called the *security context*.
4. The client program invokes a method on a server.
5. The server processes the request, extracting the authentication data from the context and performing authentication.
6. If the authentication was successful, the server grants the request and returns the security credentials for further use. If the authentication fails, the server denies service.

The client programmer is responsible for writing the code toextract the authentication data and insert it into the CORBAdata structures. WebSphere provides a utility class, the LoginHelperclass, that can be used to simplify the CORBA programming needed todo this kind of programmatic login. The TestClient applicationillustrates the use of the LoginHelper class.

In order to use the LoginHelper class, the client needs to knowthe security properties of the ORB, so you must load a propertiesfile containing those values when you start the client program.The file sas.client.props file installed with WebSphere containsvalid values. Specify the properties file on the command lineas follows:

```
-Dcom.ibm.CORBA.ConfigURL=URL of properties file
```

For example, to load the sas.client.props file and run the TestClientprogram, issue the following command:

```
java -Dcom.ibm.CORBA.client.ConfigURL=file://<install_root>/properties/sas.client.props TestClient
```

Because the JDK which requires a call to System.exit()any time the AWT is activated, the client programmerneeds to call System.exit() at the end to exitthe program.

5.4.1.1: The TestClient program

The TestClient program illustrates the use of the LoginHelper class,a utility class provided to help simplify programming client-sidelogin. The excerpt below shows the performLogin method.

TestClient class

```
public class TestClient {   ...   private void performLogin()    {      // Get the user's ID and
password.      String userid = customGetUserid();      String password = customGetPassword();
// Create a new security context to hold        // authentication data.   LoginHelper loginHelper =
new LoginHelper();    try {           // Provide the user's ID and password for authentication.
org.omg.SecurityLevel2.Credentials credentials =                    loginHelper.login(userid,
password);                // Use the new credentials for all future invocations.
loginHelper.setInvocationCredentials(credentials);               // Retrieve the user's name 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.        }   }   ...}
```

5.4.1.2: The LoginHelper class

The LoginHelper class is a WebSphere-provided utility class thatprovides wrappers around CORBA security methods. It can be usedby pure Java clients that need the ability to programmaticallyauthenticate users but don't need to use the authentication data onthe client side.

The methods in this class give a client program a way tocollect authentication information from a user and packageit to be sent to a server. The server authenticates the userand returns security credentials to the client.

The following list summarizes the public methods in the LoginHelper class.The source file is installed at:

`<installation_root>/installedApps/sampleApp.ear/default_app.war/WEB-INF/classes/LoginHelper.java`

and the class file is installed at:

`<installation_root>/installedApps/sampleApp.ear/default_app.war/WEB-INF/classes/LoginHelper.class`

LoginHelper()

The constructor obtains a new security-context object from the underlying ORB. This object is used to carry authentication information and resulting credentials for the client.

Syntax:

```
LoginHelper() throws IllegalStateException
```

login()

This method takes the user's authentication data (identifier and password), authenticates the user (validates the authentication data), and returns the resulting Credentials object.

Syntax:

```
org.omg.SecurityLevel2.Credentials login(String userID, String password)      throws
IllegalStateException
```

setInvocationCredentials()

This method sets the specified credentials so that all future methods invocations will occur under those credentials.

Syntax:

```
void setInvocationCredentials(org.omg.SecurityLevel2.Credentials invokedCreds)      throws
org.omg.Security.InvalidCredentialType,             org.omg.SecurityLevel2.InvalidCredential
```

getInvocationCredentials()

This method returns the credentials under which methods are currently being invoked.

Syntax:

```
org.omg.SecurityLevel2.Credentials getInvocationCredentials()       throws
org.omg.Security.InvalidCredentialType
```

getUserName()

This method returns the user name from the credentials in a human-readable format.

Syntax:

```
String getUserName(org.omg.SecurityLevel2.Credentials creds)       throws
org.omg.Security.DuplicateAttributeType,              org.omg.Security.InvalidAttributeType
```

5.4.2: Server-side login

Use a server-side login when a program needs to log users into the securitydomain and to use the authentication data itself. A client-side logincollects the authentication data and sends it to another programfor actual authentication; a server-side login does both tasks.

Server-side login works in the following manner:

1. The user makes a request that triggers a servlet.
2. The servlet presents the user with a login form for collecting authentication data. The user inserts his or her user ID and password into the form and submits it.
3. The servlet presents the request to the server.
4. The server processes the request, extracting the authentication data from the context and performing authentication.
5. If the authentication was successful, the server grants the request. If the authentication fails, the server denies service.

The server programmer is responsible for writing the code toextract the authentication data, insert it into the CORBAdata structures, and authenticate the user. WebSphere provides autility class, the ServerSideAuthenticator class, that can be usedto simplify the CORBA programming needed to do this kind ofprogrammatic login. This class extends the LoginHelper classused for client-side login. The TestServer applicationillustrates the use of the ServerSideAuthenticator class.

5.4.2.1: The TestServer program

The TestServer program illustrates the use of the ServerSideAuthenticatorclass, a utility class provided to help simplify programming server-sidelogin. The excerpt below shows the performLoginAndAuthentication method.

TestServer class

```
public class TestServer{   ...   private void performLoginAndAuthentication()   {       // Get the
user's ID and password.      String userid = customGetUserid();      String password =
customGetPassword();  // Ensure immediate authentication.     boolean forceAuthentication = true;
// Create a new security context to hold        // authentication data.      ServerSideAuthenticator
serverAuth = new ServerSideAuthenticator();     try     {          // Perform authentication based
on supplied data.       org.omg.SecurityLevel2.Credentials credentials =
serverAuth.login(userid, password, forceAuthentication);          // Retrieve the user's name from
the credentials           // so we can tell the user that login succeeded.    String username =
serverAuth.getUserName(credentials);          System.out.println("Authentication successful for
user: "+username); }       catch (Exception e)     {              // Handle exceptions.   }   }
...}
```

5.4.2.2: The ServerSideAuthenticator class

The ServerSideAuthenticator class is a WebSphere-provided utility class thatprovides wrappers around CORBA security methods. It extends the LoginHelperclass for use by servers.

The following list summarizes the public methods in theServerSideAuthenticator class. The source file is installed at:

`<installation_root>/installedApps/sampleApp.ear/default_app.war/WEB-INF/classes/ServerSideAuthenticator.java`

and the class file is installed at:

`<installation_root>/installedApps/sampleApp.ear/default_app.war/WEB-INF/classes/ServerSideAuthenticator.class`

ServerSideAuthenticator()

The constructor obtains a new security-context object from the underlying ORB. This object is used to carry authentication information and resulting credentials.

Syntax:

```
ServerSideAuthenticator() throws IllegalStateException
```

login()

This method takes the user's authentication data (identifier and password), authenticates the the user (if the force_authn argument is set to TRUE), and returns the resulting Credentials object.

Syntax:

```
org.omg.SecurityLevel2.Credentials login(String userID, String password,
boolean force_authn)      throws org.omg.SecurityLevel2.LoginFailed,
com.ibm.IExtendedSecurity.RealmNotRegistered,             com.ibm.IExtendedSecurity.UnknownMapping,
com.ibm.IExtendedSecurity.MechanismTypeNotRegistered,
com.ibm.IExtendedSecurity.InvalidAdditionalCriteria
```

authenticate()

This method does the actual authentication work.

Syntax:

```
org.omg.SecurityLevel2.Credentials authenticate(String userID, String password)      throws
org.omg.SecurityLevel2.LoginFailed,             org.omg.SecurityLevel2.InvalidCredential,
org.omg.Security.InvalidCredentialType,             com.ibm.IExtendedSecurity.RealmNotRegistered,
com.ibm.IExtendedSecurity.UnknownMapping,
com.ibm.IExtendedSecurity.MechanismTypeNotRegistered,
com.ibm.IExtendedSecurity.InvalidAdditionalCriteria
```

5.4.2.3: Accessing secured resources from Java clients

A Java client that needs to access a secured resource must knowthat resource is secured. This page describes how to provide clientswith the information they need.

1. Create a text file. In it, specify the following property-value pairs:
    - `com.ibm.CORBA.securityEnabled=true`
    - Configure SSL as described in 5.7.3: ORBSSL Configuration.

    You can use the properties file sas.client.props installed with WebSphere Application Server as a model.
2. When you start the client, load the properties file you just created. Specify the properties file on the command line as follows:
   `-Dcom.ibm.CORBA.ConfigURL= <URL of properties file>`

   For example, to load a properties file called my.client.props located in the product installation directory for a client called MyClient App:

   ```
   java -Dcom.ibm.CORBA.client.ConfigURL=file://install_root/properties/my.client.props MyClientApp
   ```

5.4.3: Form-based login

Applications can present site-specific login forms by making use of WebSphere'sform-login type. The J2EE specification defines form login as one of the authenticationmethods for Web applications. However, the Servlet 2.2 specification does not define amechanism for logging out. WebSphere extends J2EE by also providing a form-logoutmechanism.

Form login

A form login works in the following manner:

1. An unauthenticated user attempts to use a resource secured with a form-login authentication method.
2. The user is redirected to the form-login page, which takes the user to an HTML form that collects authentication information.
3. The user enters his or her user ID and password into the form and submits it.
4. The submission triggers a special WebSphere servlet that authenticates the user.
5. If the user authenticates successfully, the orginally requested secure resource can be accessed.

If you select LTPA as theauthentication mechanism under global security settings and use form login in any Webapplications, you must also enable single sign-on (SSO). If SSO is not enabled,authentication during form login fails with a configuration error. SSO is required becauseit generates an HTTP cookie that contains information representing the identity of theuser at the web browser. This information is needed to authorize protected resources whena form login is used.

Configuring form login

Form login is one of the possible values for the `auth-method` tag in the `login-config`element in the deployment descriptor of a Web application. For example:

```
      <login-config>         <auth-method>FORM</auth-method>         <realm-name>Example Form-Based
Authentication</realm-name>         <form-login-config>
<form-login-page>/login.html</form-login-page>
<form-error-page>/error.jsp</form-error-page>         </form-login-config>      </login-config>
```

The `form-login-page` element above specifies the form to display when arequest is made to a protected Web resource in the Web application. The form-login page isusually an HTML or JSP file, but it can also be a servlet. The page named in the `form-error-page`element is displayed if an error occurs during login.

The form-login page

The form-login page is usually an HTML form with text-entry fields for a user ID andpassword. The HTML file is included in the Web application archive (WAR) file. However,there several key requirement:.

- The text-entry field for the user ID must be named j_username
- The field for the password must be named j_password.
- The post action must be j_security_check.

The j_security_check post action is a special action recognized by the web container;it dispatches the action to a special WebSphere servlet that authenticates the user.

Here is an example of a form-login HTML page:

```
        <!DOCTYPE HTML PUBliC "-//W3C/DTD HTML 4.0 Transitional//EN">        <html>        <META
HTTP-EQUIV = "Pragma" CONTENT="no-cache">         <title>Form Login Page </title>
<body>                <h2>Sample Form Login</h2>                  <FORM METHOD=POST
ACTION="j_security_check">        <p>        <font size="2">  <strong> Please 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>
```

Form logout

Form logout is a mechanism to log out without having to close all Web-browser sessions.After logging out with form logout, access to a protected Web resource requiresreauthentication.

Suppose that it is desirable to log out after logging into a Web application andperforming 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 on the submit button of the form to logout.
3. The WebSphere security code logs the user out.

4. Upon logout, the user is redirected to a logout exit page.

Configuring form logout

Form logout does not require any attributes in any deployment descriptor. It is simplyan HTML or JSP file that is included with the Web application.

The form logout page

The form-logout page is like most HTML forms except that, like the form-login page, ithas a special post action that is recognized by the Web container, which dispatches it toa special internal WebSphere form-logout servlet.

The post action in the form-logout page must be ibm_security_logout.

A logout-exit page can be specified in the logout form, and the exit page can be a HTMLor JSP file within the same Web application that the user is redirected to after loggingout. The logout-exit page is simply specified as a parameter in the form-logout page. Ifno 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 toredirect 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 logout: </strong></font>                 <input type="submit" name="logout"
value="Logout">                <INPUT TYPE="HIDDEN" name="logoutExitPage" VALUE="/login.html">
</p>                </form>        </body></html>
```

5.5: Certificate-based authentication

Certificates and keys are part of an authorization mechanismsupported in WebSphere Application Server. Instead of requiringeach component of an application to log users in, acertificate-based authentication mechanism centralizes thelogin process. In such a system, users need to explicitlyprove their identities only to a *certificate authority* (CA).A CA is a trusted third party; components of a system agree totrust the CA to do the necessary authentication for them.

When the CA authenticates a user, it issues the user a certificatethat contains a variety of data, including the identity of theissuing CA, how much the CA trusts the user, and an expirydate for the certificate. Other components of the system canread the user's certificate to determine if the certificate(and thus the identity it represents) is valid.

To use certificates for authentication in WebSphere ApplicationServer, choose Lightweight Third-Party Authentication (LTPA) or custom user registry as your authentication mechanism.

Certificate-based authentication relies on several relatedtechnologies:

- Public-key encryption
- Digital signatures
- Certificate- and key-management systems

In order for certification to work, a system requires three things:

- Trustworthy certificate authorities
- A way to protect certificates from tampering or forgery
- A way to guarantee that the holder of the certificate is the owner of the certificate.

Trust

In order to accept third-party certificates from users,the components of the system need some way to know which CAs to trust.This is handled by creating a *trust base*, a collection ofcertificates authenticating the CAs themselves. Certificate authoritiescan be commercial ventures--companies that offer certificationas their business--or they can be local entities. Creating thetrust base is part of the work of the system administrator,who must contact commercial CAs (if used), configure local CAs (ifused), and build the trust base.

Each certificate issued to a user identifies the CA that issuedthe certificate. The component examining the certificate decideswhether the certificate is trustworthy by determining ifthe issuing CA is in the trust base. Maintaining the integrityof the trust base is a crucial part of third-party authentication.

As with any authentication mechanism, a user's ability topresent a valid certificate from a valid CA proves only thatthe user was able to meet the CA's requirements for proving identity.It does not prove that the user is not malicious, usinga stolen identity, or otherwise undesirable. Procedures forestablishing trust in those scenarios are application- and site-specific.A site with stringent requirements can choose to pay a commercialcertification company that agrees to impose requirements onthose who request certificates, and a site doing testing cancreate certificates that impose no requirements at all. Administratorsfor each application must determine how thorough the CAs must be.

Protection from forgery

Even if all the certificates in a system appear to be issued by trustedCAs, the certificates are worthless if they can be easily forged(for example, to create certificates for unauthorized users) ortampered with (for example, to give users "better" certificatesthan they are permitted to have). To preserve their contents,certificates are protected using digital signatures based on apublic-key encryption strategy, making the forgery of and

tamperingwith certificates (or any other data) impossible in practice.

Use of certificates by owners

If an intact certificate issued by a trusted CA can be used bysomeone other than the rightful owner of the certificate, theauthentication system has failed. The system of digital signaturesbased on public-key encryption provides not only a way to ensurethat certificates are intact; it also guarantees that thecertificate can be used only by its rightful owner. The mechanicsof public-key encryption ensure that a stolen certificate is useless.

5.5.1: Introduction to public-key cryptography

All encryption systems rely on the notion of a key. A key is the basisfor a transformation, usually mathematical, of an ordinarymessage into a unreadable one. For centuries, most encryption systemshave relied on what is called private-key encryption. Only withinthe last 30 years has a challenge to private-key encryptionappeared: public-key encryption.

Private-key encryption

Private-key encryption systems use a single key. This requires thesender and the receiver to share the key. Both must have the key; the sender encrypts the message by using the key, and the receiverdecrypts the message with the same key. Both must keep the key privateto keep their communication private. This kind of encryption hascharacteristics that make it unsuitable for widespread, general use:

- It 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 have made arrangements to communicate by sharing keys.

Private-key encryption is also called *symmetric* encryption, becausethe same key is used to encrypt and decrypt the message.

Public-key encryption

In the 1970s, a mathematical breakthrough led to the development ofanother major cryptographic system, public-key encryption. Public-keyencryption uses a pair of mathematically related keys. A messageencrypted with the first key must be decrypted with the second,and a message encrypted with the second key must be decrypted withthe first. Each participant in a public-key system has a pairof keys. One of these keys is kept secret; this is the *private key*.The other is distributed to anyone who wants it; this is the*public key* .

To send an encrypted message to you, the sender encrypts themessage by using your public key. When you receive it, you decrypt itby using your private key. When you wish to send a message to someone,you encrypt it by using the recipient's public key. The message canbe decrypted only with the recipient's private key. This kind ofencryption has characteristics that make it very attractive for general use:

- Public-key encryption requires only two keys per participant. The total number of keys rises much less dramatically as the number of participants increases than it does in private-key encryption.
- The need for secrecy is more easily met. The only thing that needs to be kept private is the private key, and since it does not need to be shared, it is less vulnerable to theft in transmission than the shared key in a private-key system.
- Public keys can be published. This 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, becausethe 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 ApplicationServer uses the RSA public/private key-encryption algorithm.

With private-key encryption, you have to be careful of stolenor intercepted keys. In public-key encryption, where anyone cancreate a key pair and publish the public key, the challenge isin verifying that the owner of the

public key really is the personyou think it is. There is nothing to stop a user from creatinga key pair and publishing the public key under a false name.The person listed as the owner of the public key will notbe able to read messages encrypted with that key because heor she will not have the private key. If the creator of the falsepublic key can intercept these messages, that person candecrypt and read messages intended for someone else.To counteract the potential for forged keys, public-key systemsprovide mechanisms for validating public keys (and otherinformation) with digital signatures and digital certificates.

5.5.2: Introduction to digital signatures

A digital signature is a number attached to a document. For example,in an authentication system that uses public-key encryption, digitalsignatures are used to sign certificates. This signature establishestwo different things for you:

- 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 consistsof distilling the document down into a large number. This number isthe *digest code* or *fingerprint*. The digest codeitself is then encrypted, resulting in the digital signature. Thedigital signature is appended to the document from which thedigest code was generated.

There are several ways of generating the digest code--WebSphere ApplicationServer supports the MD5 message digest function and the SHA1 secure hashalgorithm--but all of them reduce a message to a number. This process is*not* encryption; rather, it is a sophisticated checksum. The message*cannot* be regenerated from the resulting digest code.The crucial aspect of distilling the document down to a number is this:if the message is changed, even in trivial way, a different digest coderesults. This means that when the recipient gets a message and verifiesthe digest code by recomputing it, any changes in the document willresult in a mismatch between the stated and the computed digest codes.If a message is changed, the resulting digest code changes as well.

So far, there is nothing to stop someone from intercepting a message,changing it, recomputing the digest code, and retransmitting themodified message and code. We need a way to verify the digest code as well.This is done by reversing the use of the public and private keys.For private communication, it makes no sense to encrypt messages withyour private key; these can be decrypted by anyone with your public key.But this technique can be useful for proving that a message must havecome from you. No one else could have created it, since no one elsehas your private key. If some meaningful message results from decryptinga document by using someone's public key, it verifies the fact thatthe holder of the corresponding private key did, in fact, encryptthe message.

The second step in creating a digital signature takes advantageof this reverse application of public and private keys. After a digestcode has been computed for a document, the digest code itself is encryptedwith the sender's private key. The result is the digital signature,which is simply attached to the end of the message.

When the message is received, the recipient follows these steps to verifythe signature:

- Recompute the digest code for the message.
- Decrypt the signature by using the sender's public key. This yields the original digest code for the message.
- Compare the original and recomputed digest codes. If they match, the message is both intact and authentic. If not, something has changed and the message is not to be trusted.

5.5.3: Introduction to digital certificates

A digital certificate is equivalent to an electronic ID card. Itserves two purposes:

- To establish the identity of the owner of the certificate
- To distribute the owner's public key

Certificates provide a way of authenticating users, referred to asauthentication by trusted third parties. Instead of requiring eachparticipant in an application to authenticate every user, third-partyauthentication relies on the use of certificates, electronic ID cards.

Certificates are issued by trusted parties, called *certificateauthorities* (CAs). These authorities can be commercial venturesor they can be local entities, depending on the requirements of yourapplication. Regardless, the CA is trusted to adequately authenticateusers before issuing certificates to them. Also, when a CA issuescertificates, it digitally signs them. When a user presents a certificate,the recipient of the certificate validates it by using the digitalsignature. If the digital signature validates the certificate,the certificate is known to be intact and authentic. Participantsin an application need only to validate certificates; they do not needto authenticate users themselves. The fact that a user can present avalid certificate proves that the CA has authenticated the user.The descriptor *trusted third-party* indicates that the systemrelies on the trustworthiness of the CAs.

Contents of a digital certificate

A certificate contains several pieces of information, includinginformation 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'spublic key, signs the public key with the its own private key, andreturns this to the owner as a certificate. When the owner distributesthe certificate to another party, it signs the certificate with itsprivate key. The receiver can extract the certificate (containingthe CA's signature) with the owner's public key. By using theCA's public key and the CA's signature on the extractedcertificate, the receiver can validate the CA's signature. If it isvalid, the public key used to extract the certificate is known to be good.The owner's signature is then validated, and if the validationsucceeds, the owner has successfully authenticated to the receiver.

The additional information in a certificate allows an application todecide if it should honor the certificate. With the expiration date, theapplication can determine if the certificate is still valid.With the name of the issuing CA, the application can check thatthe CA is considered trustworthy by the site.

A process that uses certificates must be able to provide its *personalcertificate*, the one containing its public key, and the certificateof the CA that signed its certificate, called a *signing certificate*.In cases where chains of trust are established, several signingcertificates may be involved.

Requesting certificates

To get a certificate, you must send a certificate request to theCA. The certificate request includes the following:

- The distinguished name of the owner (the user for whom the certificate is being 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 requestto ensure that the request is intact and authentic. The CA thenauthenticates the owner. Exactly what the authentication consists ofdepends on a prior agreement between the CA and the requestingorganization. If the owner in the request is successfully authenticated,the CA issues a certificate for that owner.

Using certificates: Chains of trust and self-signed certificates

To verify the digital signature on a certificate, you must have thepublic key of the issuing CA. Since public keys are distributed incertificates, you must have a certificate for the issuing CA. Thatcertificate will be signed by the issuer. One CA can certifyother CAs, so there can be a chain of CAs issuing certificates forother CAs, all of whose public keys you need. Eventually, though,you reach a starting point. The starting point is a *root CA*that issues itself a *self-signed certificate*. In order tovalidate a user's certificate, you need certificates for all interveningparticipants, back to the root CA. Then you have the public keysyou need to validate each certificate, including the user's.

A self-signed certificate contains the public key of theissuer and is signed with the private key. The digital signatureis validated like any other, and if the certificate is valid,the public key it contains can be used to check the validityof other certificates issued by the CA. However, anyone cangenerate a self-signed certificate. In fact, you will probablygenerate self-signed certificates for testing purposes beforeinstalling production certificates. The fact that a self-signedcertificate contains a valid public key does not mean that theissuer is really a trusted certificate authority. In order toensure that self-signed certificates are generated by trustedCAs, such certificates must be distributed by secure means(hand-delivered on floppy disks, downloaded from secure sites,and so forth).

Applications that use certificates store those certificatesin *key*, or *keyring*, files. This file typically containsthe necessary personal certificates, its signing certificates,and its private key. The private key is used by the applicationto create digital signatures. Servers will always have personalcertificates in their key files. A client requires a personal certificateonly if the client must authenticate to the server, that is, whenmutual authentication is enabled.

To allow a client to authenticate to a server, a server's keyringfile contains the server's private key and certificate and thecertificates of its CA. A client's keyring must contain thecertificates of the CAs of each server to which the client mustauthenticate.

If mutual authentication is needed, the client's keyring must contain the client's private key and certificate and thecertificates of any CAs. The server's keyring needsa copy of the certificate of the client's CA as well.

5.5.4: Requesting certificates

When you request a certificate from a certificate authority,you need to take into account:

- The time it takes to get a certificate
- Requirements the CA imposes on the format of information

Time requirements

Because of the diligence expected of a commercial CA, the authenticationprocess for principals can take a significant amount of time. CommercialCAs often require up to a week to complete their authentication process.Even on-site CAs can take between minutes and days to complete theirauthentication process.

As a result, when planning to add a new application server or host (nameserver) to your enterprise, you must take into account the time ittakes to get a certificate. Although primarily of concern for productioncertificates, it can also be a concern in getting test certificates aswell.

Note that if your server's certificate is compromised, or if someother server in its trust-base is compromised, you must acquirea replacement certificate. This involves similar time requirements.

Requirements on the format of information

When you create a certificate request, you need to provide the informationabout the owner of the certificate. The required information and itsformat vary across certificate authorities. Also, the WebSphere ApplicationServer graphical tool and command-line tools vary in the way they representthe name.

Certificates use names in the X.500 format. A name in this styleconsists of many components. The entire name is called a *distinguishedname* (DN). It consists of a set of components, which often includesa *common name* (CN), and organization (O), an organizationunit (OU), a country (C), a locality (L) and many others. Forexample, an X.500 name for a server called PolicyServer1 aspart of the Accounting division of the US-based AccountingCorpcan look like this:

```
"CN=PolicyServer1, OU=Accounting, O=AccountingCorp, c=US"
```

Certificates are often used to represent server principals, so a typicalconvention is to create CNs of the form*host_name*/*server_name*, for example,for the server PolicyServer1 on the host centralops.acctcorp.com, thecommon name is centralops.acctcorp.com/PolicyServer1.

Some CAs require the use of fully-qualified host names in commonnames. For example, VeriSign does not sign your certificate unlessthe domain portion of the host name is owned by your organization.Check with the CA for any requirements on common-name fields.

The distinguished name can include other information as well. Some certificateauthorities, including VeriSign, require that you spell out completelythe state or province fields. For example, you need to specify "New York"rather than "NY." Check with the CA for any such requirements before generatingyour certificate requests.

5.5.4.1: Getting a test certificate from acertificate authority

To obtain a certificate from a certificate authority, youmust create file containing a certificate signing request (CSR).You then send the file to the CA. The procedure for gettingthe file to the CA varies with the CA and with the type ofcertificate, test or production, being requested. It is oftenhelpful to request a test certificate from a CA before requestinga production certificate.

This file describes how to get a test certificate from a specificcommercial CA, VeriSign, which offers a test certificate for free.The test certificate is a legitimate certificate, fully signedand endorsed for actual use, and it can be used to validateyour configuration before you acquire a production certificate.However, the test certificate is only good for two weeks afterreceipt, so it is not useful for production use.

After you have created file containing a certificate signing request,request a test certificate by following these steps:

1. Start your Web browser and link to VeriSign's home page at http://www.verisign.com.
2. Choose the free trial SSL trial ID option. This displays a page where you can request a free trial of a secure server ID.
3. Follow the instructions for requesting a free trial ID. Be sure to read the frequently asked questions (FAQ) list, the legal agreement for VeriSign trial subscribers, and the information comparing Trial Secure Server IDs to Secure Server Digital IDs. VeriSign also provides online help for each step of the process.
4. When you get to the page on which you submit the CSR file, scroll down to the edit box. This is where you insert the CSR.
5. Open the file containing the CSR; use any text editor that supports cut-and-paste actions.
6. In your editor window, select all of the text, including the header

   `-----BEGIN NEW CERTIFICATE REQUEST-----`

   and the corresponding trailer.
7. Paste the test into the edit box on the Enrollment page in your browser.
8. Click the Continue button.
9. On the resulting page, verify and complete the following information:
   - **Verify Distinguished Name:** Check all of the information displayed about your certificate. In particular, ensure that the Common Name is correct and unique.
   - **Enter Technical Contact Information:** Enter the requested information about you. VeriSign needs this information to send you your signed certificate. In particular, make sure that your e-mail address is correct. VeriSign will e-mail your certificate to this address.
   - **Read the Digital ID Subscriber Agreement:** Read the terms and conditions stipulated by VeriSign about the Test ID you are requesting.
     If you do not accept these conditions, do not continue.
10. When the information is complete, and if you accept the VeriSign's Subscriber Agreement, click the Accept button.

You will recieve an acknowledgement, usually by e-mail, that you havesuccessfully completed your request. You will probably be instructedto download the certificate and to install it in your browser.

 Do *not* install the certificate in your browser. For use withWebSphere, the certificate must be installed in a keyring,not in your browser.

5.5.4.2: Getting a production certificate from a certificate authority

To obtain a certificate from a certificate authority, youmust create file containing a certificate signing request (CSR).You then send the file to the CA. The procedure for gettingthe file to the CA varies with the CA and with the type ofcertificate, test or production, being requested.

This file describes how to get a production certificate from a specificcommercial CA, VeriSign. Getting a production certificate can beexpensive, depending on the type of certificate and its strength.It is often instructive to request a test certificate from a CAbefore requesting a production certificate.

After you have created file containing a certificate signing request,request a production certificate by following these steps:

1. Start your Web browser and link to VeriSign's home page at http://www.verisign.com.
2. Choose Web Server Certificates --> Buy Now --> [Buy] Global Site Services. This begins a series of pages that collect the information VeriSign needs to process your certificate request. Read each page carefully. When you complete a page, display the next page by clicking the Continue button.

   The page titled Before You Start lists the things you should do before beginning this process, including installing web server software, setting up your Internet proxies, determining how you will pay for the certificate, reviewing the legal agreement and, if necessary, printing the enrollment guide. You should treat any references to "web server software" as references to the WebSphere software.
3. The page titled Step 1: Obtain Proof of Right provides instructions on one of the authentication steps that VeriSign performs. In this case, you must prove that your enterprise has the right to operate under the Organization name that you specified in your CSR. The VeriSign process is optimized to using D-U-N-S numbers for this purpose. If you take this approach, you must provide your D-U-N-S number or, if you are a U.S. company, VeriSign can look it up for you.

   If you don't have a D-U-N-S number, or if you don't want to use this to prove your right to the Organization name, you can provide alternate proof of right. For example, if you have a letter of incorporation or similar article, you can fax a copy to VeriSign. Using an alternate proof of right will slow the process down, because you will not be able to continue until VeriSign has received and processed the alternative proof.
4. The page titled Step 2: Confirm Domain Name informs you that you (your enterprise) must own the domain name indicated in the common name of your certificate. These domain names are registered with NIC, and VeriSign will verify that the domain name you specified belongs to your enterprise; this is part of the authentication process completed by certificate authorities.
5. The page titled Step 3: Generate CSR instructs you to create your CSR. If you have already created a CSR file, you can skip this step.
6. The page titled Step 4: Submit CSR provides you with an edit box. This is where you will insert the CSR.
7. Open the file containing the CSR; use any text editor that supports cut-and-paste actions.
8. In your editor window, select all of the text, including the header

   `-----BEGIN NEW CERTIFICATE REQUEST-----`

   and the corresponding trailer.
9. Paste the test into the edit box on the Submit CSR page in your browser.
10. The page titled Step 5: Complete Application page requires you to enter a lot of information. Verify your distinguished name and enter the following:
    - Server information

        - Vendor of the server software: Click the pull-down button and select IBM.
        - A challenge phrase: A text string. This can be anything you like, and you should treat it like a password. You will be asked to present this same challenge phrase when you submit a renewal request or if you ask to have the certificate revoked (for example, if the certificate is compromised). You may also be asked to supply this challenge phrase when speaking with VeriSign.
    - Technical contact information: This should identify you. Your e-mail address is particularly important; VeriSign will e-mail the certificate to this address.
    - Organizational contact information: This should be someone other than yourself who is a member of your enterprise. VeriSign will contact this person during the authentication process, to verify the legitimacy of your request.
    - Billing contact information: Enter the person in your organization who is responsible for payment.
    - The type of Secure Server ID that you are requesting
    - Payment information
    - Organizational information (your D-U-N-S number): If you use an alternate proof of right, then VeriSign will instruct you on how to fill out this information.
11. Review the Server Certificate Agreement. To accept the conditions and submit your request, click the Accept button. If reject the conditions, click the Decline button.

VeriSign will send you an e-mail message containing your signedproduction certificate. The certificate must be installed ina keyring class.

5.5.4.3: Using test certificates

If you need to start using a server before you get a productioncertificate from a CA -- for example, to test your installation --you can do either of the following, less secure, alternatives:

- You can use the test certificate (in the DummyServerKeyFile, see 5.7.3: ORB SSL Configuration) provided with WebSphere to perform some early tests. However, you should replace it with a certificate that legitimately represents your server as soon as possible. For this, you do can either of the following:
  - Acquire production (or test) certificates from the CA
  - Create your own test CA and issue test certificates
- You can configure the server initially without its certificate keyring. This means that clients cannot access the server securely. Again, this situation is acceptable only for testing purposes.

When you receive the certificate from the CA, you can modify theconfiguration of the server to use the new certificate. Clients canthen access the server with the security provided by the certificate.

5.5.5: Mapping certificates to users for client authentication and authorization

Client-side certificates allow access to secured resources from Webclients. A client presents an X.509-compliant digital certificateto perform mutual authentication with a Web server. The WebSpheresecurity run time attempts to map the certificate to a known user inthe associated LDAP directory. If the certificate is successfullymapped to a user, then the holder of the certificate is believedto be the user in the registry and is authorized as this user.

After the Web server gets the client's certificate, there mustbe a way to map the certificate to a user. WebSphere ApplicationServer supports two techniques for mapping certificates to entriesin LDAP registries:

- By exact distinguished name
- By matching attributes in the certificate to attributes of LDAP entries

Mapping by exact distinguished name

This approach attempts to map the distinguished name (DN) associatedwith the Subject in the certificate to an entry in the LDAP directory.If the mapping is successful, the user is authenticated and isauthorized according to the privileges granted to the identity in the LDAPdirectory.

The mapping is case insensitive. For example, the following twoDNs 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, and if nomatch is found, authentication fails.

Mapping by filtering certificate attributes

This approach maps certificate attributes to attributes of entries in anLDAP directory. For example, you can specify that the common name (CN)attribute of the Subject field in the certificate is to be matched againstthe uid attribute of your LDAP entry. If the mapping is successful, the useris authenticated and is authorized according to the privileges granted to theidentity in the LDAP directory.

If you are matching the Subject CN field in the certificate to theuid 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 CertificateMapping and set up the certificate filter in the administrative console.

1. Click Task --> Configure Application Security
2. Set the Challenge Type to "Certificate"
3. Click Task --> Global Security Settings --> User Registry
4. Click the Advanced button
5. Set the Certificate Mapping choice to "Certificate Filter"
6. Enter the certificate filter you want to implement. For example, to match the CN attribute of the Subject in the certificate to the uid attribute in the LDAP entry, enter `(uid=${SubjectCN})`

This specification extracts the CN field from the Subject attribute in thecertificate ("Smith") and creates a filter ("uid=Smith") from it.The LDAP directory is searched for a user entry that matches thefilter. If an entry matches the filter, authentication succeeds.Note that the search and match of the LDAP directory arebased in part on how your LDAP directory is configured.

5.5.6: Tools for managing certificates and keys

WebSphere Application Server, Advanced Edition provides utilities for managing certificatesand keys:

- A graphical tool, called iKeyman, the IBM Key Management tool.
- The standard Java command-line tool, keytool.

The graphical tool is easier to use than the command-linetools, which makes it ideal for occasional or casual use. However,command-line tools support scripting of certificate management,which is useful for administrators who do a lot of this work or whowant to automate the work.

5.5.6.2: The IBM Key Management tool

WebSphere provides a graphical tool, the IBM Key Management tool (iKeyman)for managing keys and certificates. The graphical tool is easierto use than the command-line tools, which makes it ideal for occasionalor casual use.

Using the tool

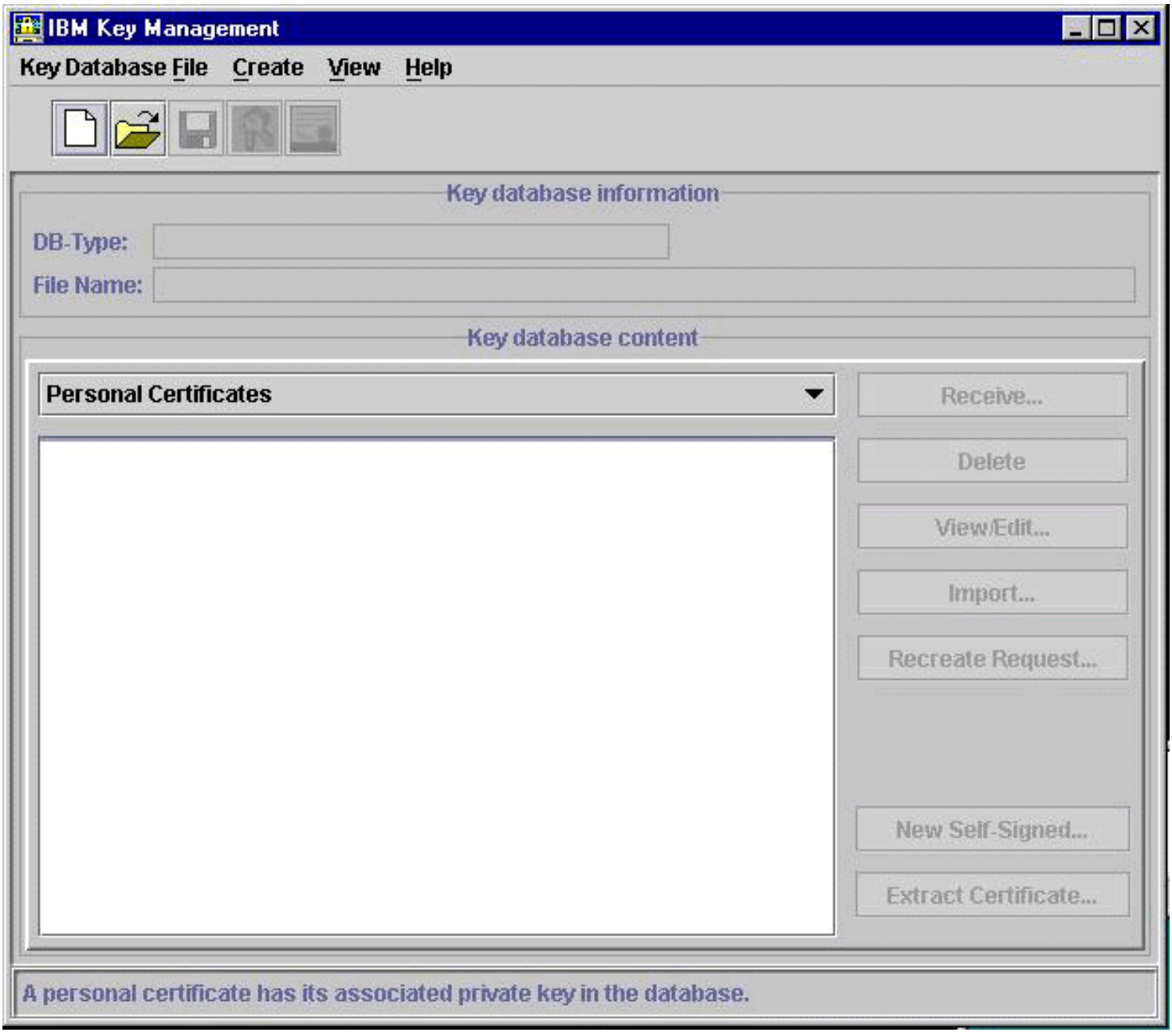
To start the iKeyman tool:

1. Move to the product_installation_root/bindirectory.
2. Issue one of the following commands:
    - On Windows systems:

      `ikeyman`
    - On Unix systems:

      `ikeyman.sh`

The iKeyman window appears as shown below.



5.5.6.2.1: Creating a self-signed test certificate

For test purposes, you can create a self-signed certificatespecifically for a server and its Secure Sockets Layer (SSL) basedJava clients. You can also set up a temporary certificateauthority by creating a self-signed certificate and using it to signother certificates.

This procedure is useful when the WebSphere test certificate hasexpired, or if you want a self-signed test certificate thatspecifically recognizes your server. If you need a test certificatethat has been signed by a Certificate Authority (CA), follow theprocedure in article 5.5.6.2.2, Creating acertification request.

To create your own self-signed test certificate, complete the followingsteps:

1. Create a server key store file. See article 5.5.6.2.1.1, Creating a serverkey store, for details.
2. Create a client trust store file. See article 5.5.6.2.1.2, Creating aclient trust store, for details.
3. Enable Websphere Application Server to access the client andserver keyring files. See article5.5.6.2.5, Making client and server key store and trust store filesaccessible, for details.

5.5.6.2.1.1 Creating a server key store

The first step in creating a self-signed test certificate is tocreate a server key store file. It contains a private key for the serverfor which the test certificate is being requested and a public key forcertificate requests. You can optionally create a trust store filewhich contains additional trusted signers. To create a server key store, complete the following steps:

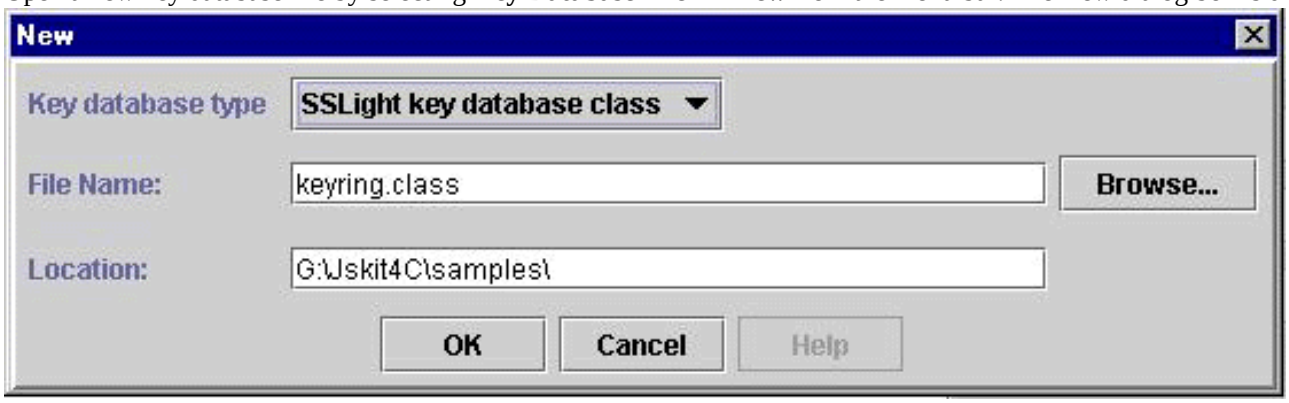
1. Start the IBM Key Management tool. See article 5.5.6.2, The IBM Key Management tool, for instructions.
2. Create a server key store file.
3. Create a new self-signed personal certificate.
4. Export the public key from the server key store file. This key is required by the client trust store file.

The rest of this article describes how to complete these steps.

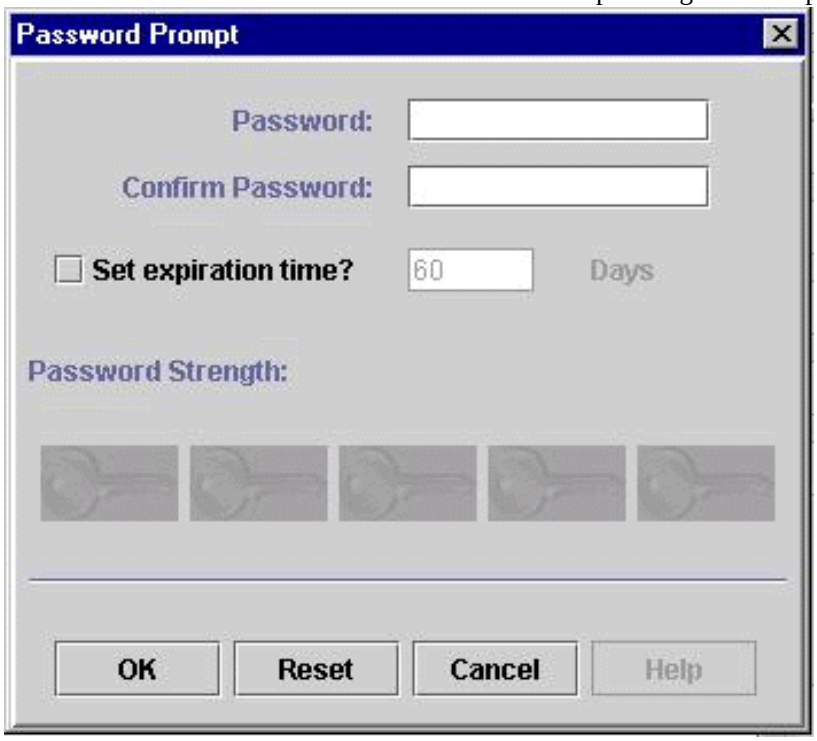
Create a server keyring file

To create a server key store file, do the following:

1. Open a new key database file by selecting **Key Database File --> New** from the menu bar. The New dialog box is displayed.



2. Set **Key Database Type** to JKS.
3. Enter the name and location of the server key store file. In this example, the file name is ServerKeyStoreFile.jks and the location is product_installation_root/etc
4. Click the **OK** button to continue. The Password Prompt dialog box is displayed.



5. Enter a password to restrict access to the key database. In this example, the password is WebAS.
The server keyring password is stored in the administrative console. The client keyring password is stored in the

sas.client.props file using the property com.ibm.CORBA.SSLClientKeyRingPassword. You need to set the keyring-password properties to this password so that the keyring file can be opened by iKeyman during runtime. See article 5.5.6.2.5, Making client and server key store and trust store files accessible, for details.

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 iKeyman during runtime.

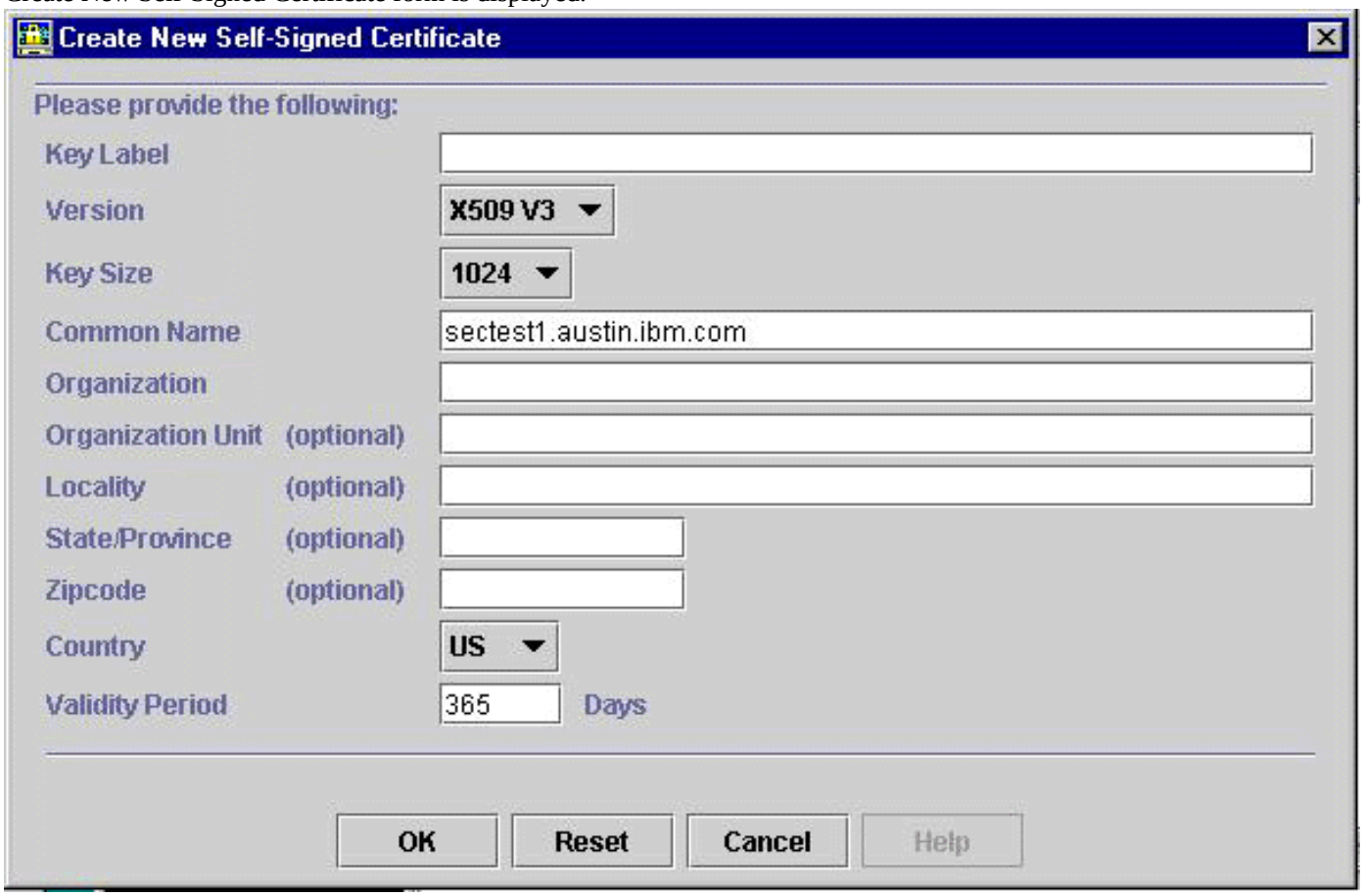
6. Click the **OK** button to continue. The tool now displays all of the available default signer certificates. These are the public keys of the most common CAs. You can add, view or delete signer certificates from this screen.

Create a new self-signed personal certificate

Creating a self-signed personal certificate creates a private keyand public key within the server key store file. A server key store filecontains both a private and public key. A client trust store file onlycontains the public key of the self-signed certificate, but as atrusted signer. A client key store file is optional. It is usuallyonly necessary when client authentication is used. WebSphereApplication Server does not support SSL mutual authentication.

To create a self-signed certificate, do the following:

1. Click the **New Self-Signed...** button on the tool bar or select **Create --> New Self-Signed Certificate...** from the menu. The Create New Self-Signed Certificate form is displayed.



2. Enter the appropriate information for your self-signedcertificate.

Key Label

Give the certificate a key label, which is used to uniquely identify the certificate within the key store. If you have only one certificate in each key store, 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 server's common name. This 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 CNs of the form *host_name*/*server_name*.

Organization

Enter the name of your organization.

Other X.500 fields

Enter the organization unit (a department or division), location (city), state/province (if applicable), zipcode (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. Commercial CAs may require them.

Validity period

Specify the lifetime of the certificate in days, or accept the default.

3. Click the **OK** button to continue. The ServerKeyStoreFile.jksfile now contains a self-signed personal certificate. You must copythe key store file to the designated directory on the server'shost.

If you have onlyone personal certificate, it is automatically set as the defaultcertificate for the database. If you have more than one, you mustselect one as the default certificate. You can change the defaultcertificate as follows:

1. Highlight the certificate
2. Click the **View/Edit...** button
3. Check the box on the resulting screen to make the chosen certificate the default
4. Click the **OK** button

Export the public certificate

The client trust store file needs to reference the publiccertificate created for the self-signed personal certificate. Toenable the client trust store file to use the public certificate, exportthe public certificate from the server key store file as follows:

1. Click **Extract Certificate**.
2. Under **Data type**, select Base64-encoded ASCII data.
3. Enter the certificate file name and location. In this case, thename is cert.arm and the location is product_installation_root/etc.
4. Click OK to export the public certificate

5.5.6.2.1.2 Creating a client trust store

The second step in creating a self-signed test certificate is tocreate a client trust store file. It is a trusted signer to the publickey for the self-signed test certificate. You can optionally createa client key store file if client authorization is desired. Key storefiles store private keys and personal certificates; trust store filescontain public keys.

To create a client trust store file, complete the following steps:

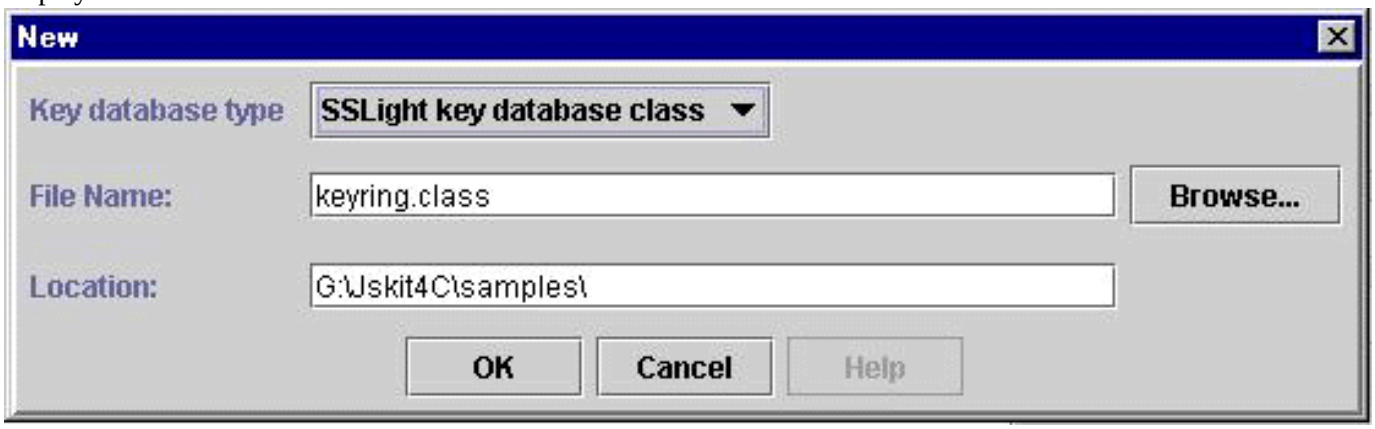
1. Start the IBM Key Management tool if you have not already done so. See article 5.5.6.2, The IBM Key Management tool, for instructions.
2. Create a client keyring file.
3. Import the public key that was exported from the server keyring file.
4. Set the certificate as a trusted root.
5. Exit the IBM Key Management tool.

The rest of this article describes how to complete these steps.

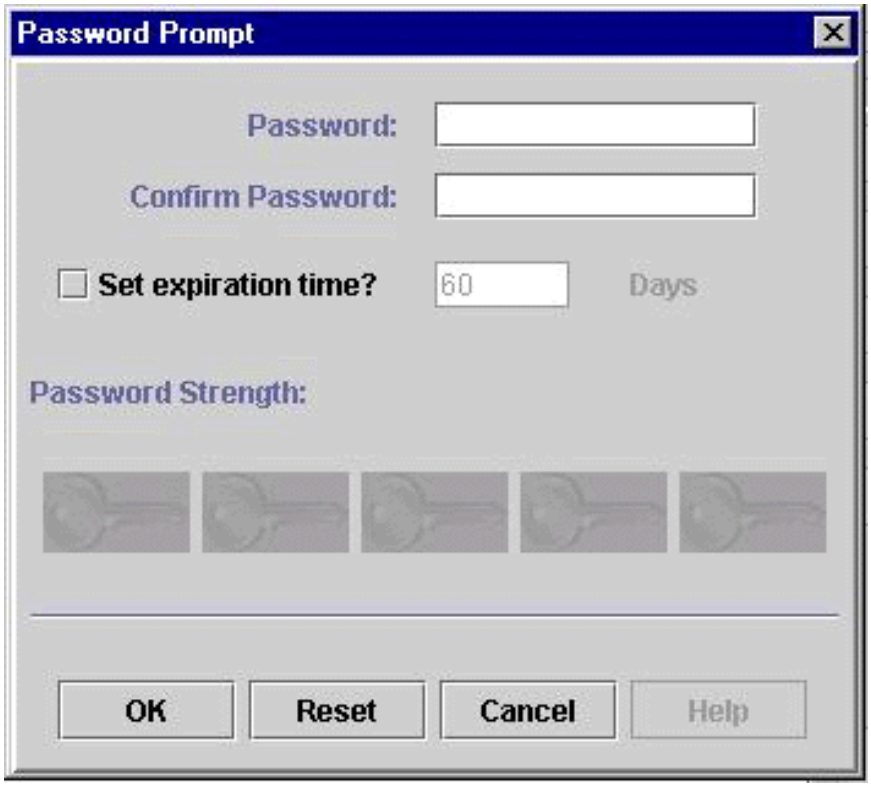
Create a client trust store file

To create a client keyring file, do the following:

1. Open a new key database file by selecting **Key Database File --> New** from the menu bar. The New dialog box is displayed.



2. Set **Key Database Type** to JKS.
3. Enter the name and location of the client keyring file. In this example, the file name is ClientTrustStoreFile.jks and the location is product_installation_root/etc
4. Click the **OK** button to continue. The Password Prompt dialog box is displayed.



5. Enter a password to restrict access to the key database. In this example, the password is WebAS.
   The server key store password is stored in the administrative console. The client trust store password is stored in the sas.client.props file using the property com.ibm.CORBA.trustStorePassword. You need to set the trust store password properties to this password so that the trust store file can be opened by iKeyman during runtime. See article 5.5.6.2.5, Making client and server key store and trust store files accessible, for details.

   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 iKeyman during runtime.
6. Click the **OK** button 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.

Import the public key from the serverkey store file

Next, you need to import the public key certificate that wasexported from the server keyring. (See article 5.5.6.2.1.1, Creating a serverkey store.) To import the public key, do the following:

1. Choose *Signer Certificates -->Add*.
2. Specify the data type of the exported key. In this case, the data type is **Base64-encoded ASCII data**.
3. Specify the name and location of the public key that was exported from the server keyring. In this case, the key name is cert.arm and the location is product_installation_root/etc.
4. Click **OK**.
5. Enter a unique label for the key. In this example, the label is **Server CA**.
6. Click **OK**. The certificate label appears in the list of certificates.

Verify that the certificate is a trustedroot

The client certificate must be a trusted root of the public keycertificate that you just created. To verify this, do the following:

1. Select the name of the certificate you just created. In this case, the certificate name is **Server CA**.
2. Select **View-->Edit**. The **Key information** dialog box appears.
3. Make sure that the box beside **Set the certificate as a trusted root** is checked.

4. Click **OK**.

Exit the IBM Key Management tool

Exit the Ikeyman tool by closing the IBM Key Management window.

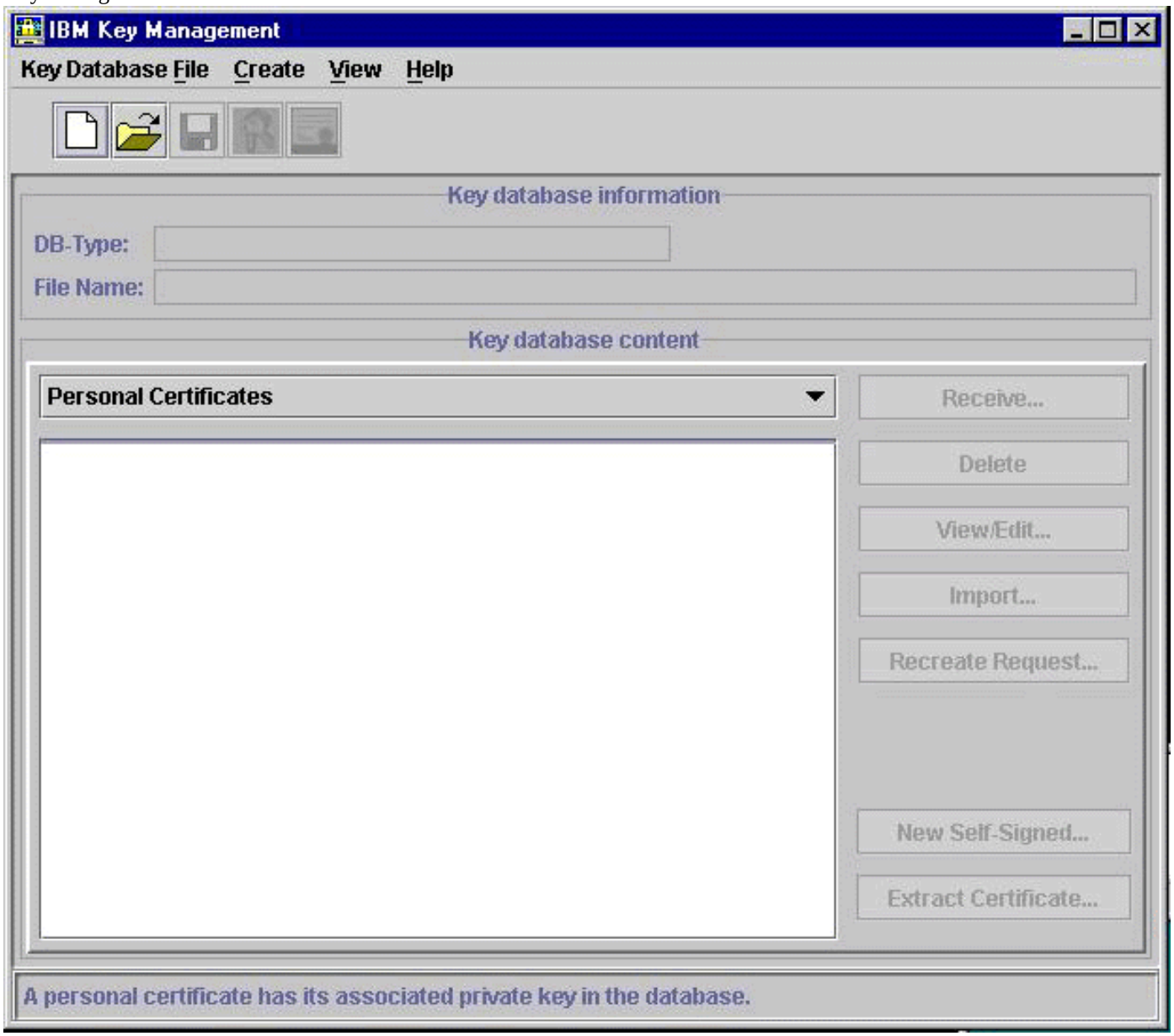
5.5.6.2.2: Creating a certification request

To obtain a certificate from a certificate authority, you mustsubmit a certificate signing request (CSR). You can request eitherproduction or test certificates from a CA with a CSR.

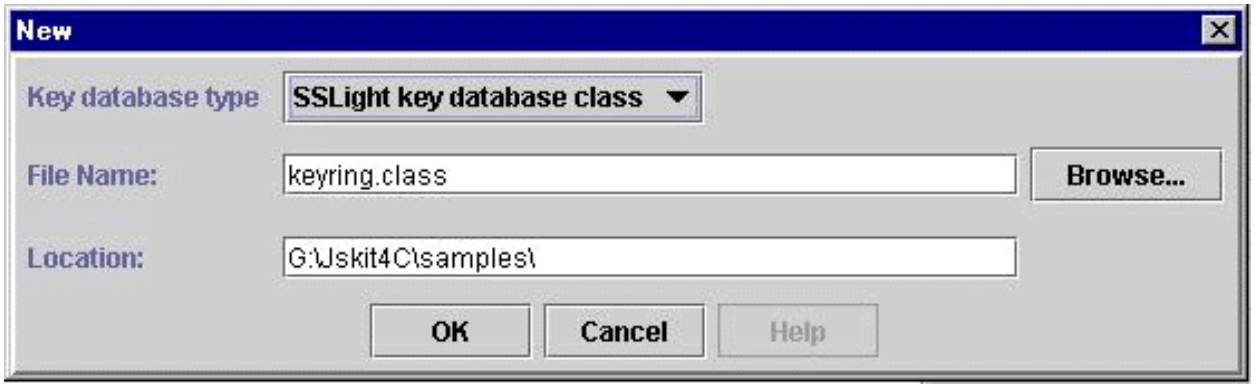
With iKeyman, generating a certificate signing request also generatesa private key for the server for which the certificate is beingrequested. The private key remains in the server's keyring class,so it stays private: the public key is included in the CSR.

To create a certificate signing request (CSR), complete the followingsteps:

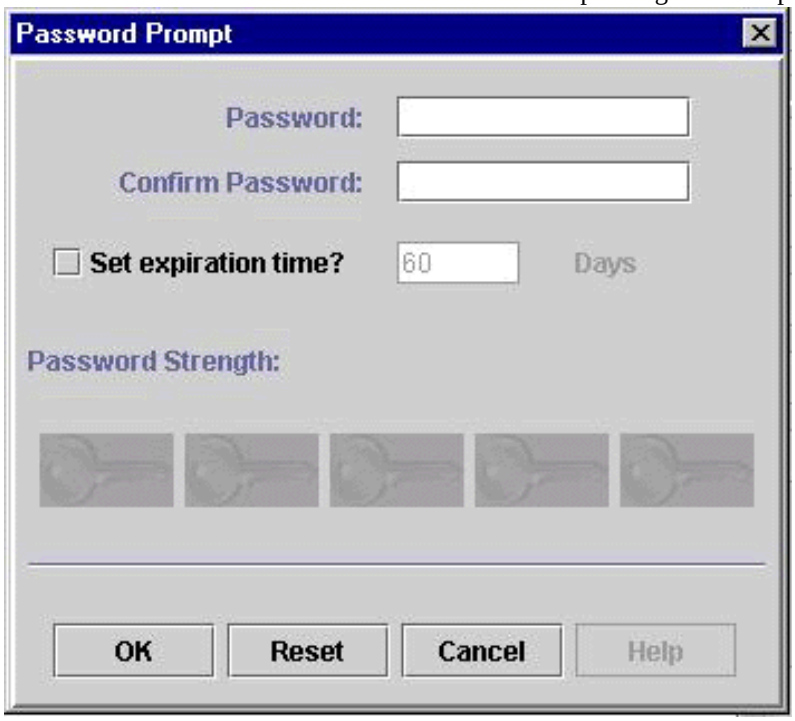
1. Start the IBM Key Management tool. See article 5.5.6.2, The IBM Key Management tool,for instructions. This displays the IBM Key Management window.



2. Open a new key database file by selecting **Key Database File --> New** from the menu bar. The New dialog box is displayed.
3. Set **Key Database Type** to JKS.
4. Enter the name and location of the new key file.



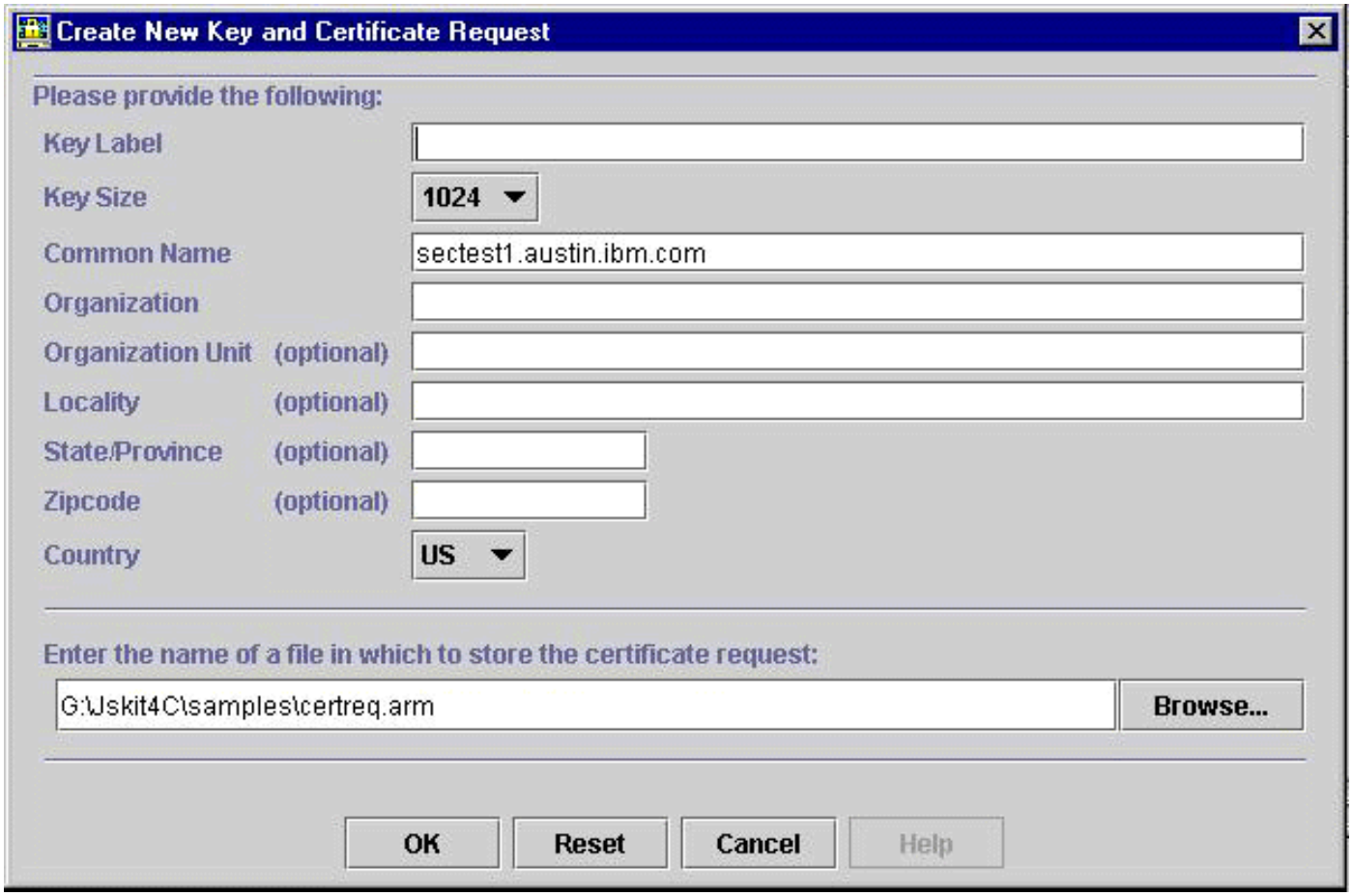
5. Click the **OK** button to continue. The Password Promptdialog box is displayed.



6. Enter a password to restrict access to the key database. In this example, the defaultpassword is WebAS.
The server key store password is stored in the administrative console. The client trust store password is stored in the sas.client.props file using the property com.ibm.ssl.trustStorePassword. You need to set the key store-password properties to this password so that the key store file can be opened by iKeyman during runtime. See article 5.5.6.2.5, Making client and server key store and trust store files accessible, for details.

   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 iKeyman during runtime.

7. Click the **OK** button to continue.
8. Locate the Key database content portion in the center of the main window Select **Key Database Content --> Personal Certificate Requests**. This updates the IBM Key Management window with any existing personal certificate requests.
9. Click the **New...** button.
10. The Create New Key and Certificate Request dialog box is displayed. Enter the necessary information to complete your request. The information certificate authorities require varies; be sure to determine the necessary fields and formats before sending your request.



Key Label

Give the certificate a key label, which is used to uniquely identify the certificate within the key store. If you have only one certificate in each key sotre, you can assign any value to the label, but it is good practice to use a unique label, related to the server name.

Common Name

Enter the server's common name. This 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 CNs of the form <*host_name*>/<*server_name*>.

Organization

Enter the name of your organization.

Other X.500 fields

Enter the organization unit (a department or division), location (city), state/province (if applicable), zipcode (if applicable), and select the two-letter identifier of the country in which the server belongs.

File name for the certificate request

Enter the name of the file for the request. CSR files are typically named for the server, with a .arm extension.

11. Click the **OK** button.
12. An Information panel is displayed to indicate that the request file has been successfully created. Click the **OK** button to dismiss the panel.
13. Exit the Ikeyman tool by closing the IBM Key Management window.

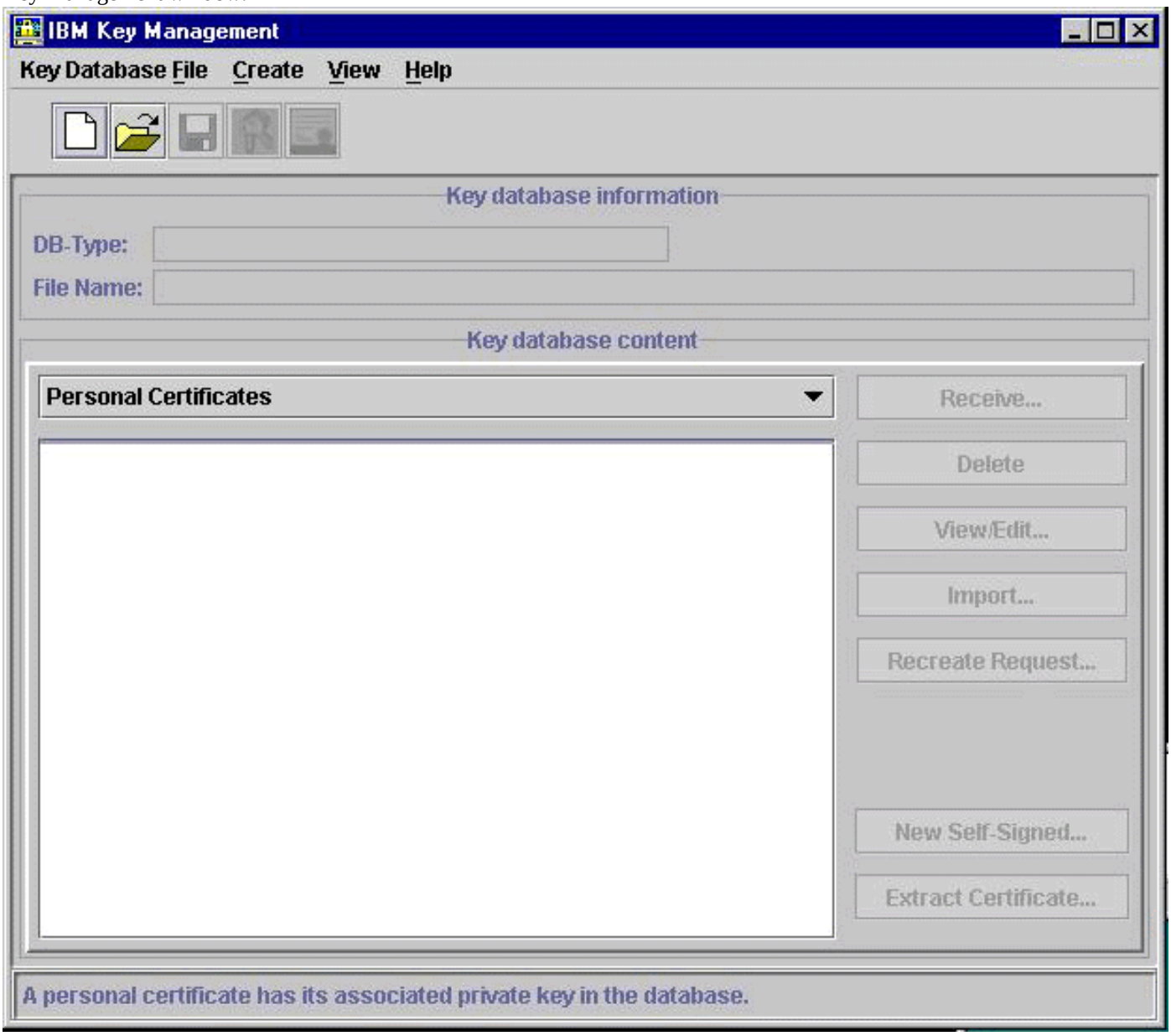
You must now submit the certificate-request file to the CA. Theprocedure will vary with the CA and with the type of certificate(test or production) being requested.

5.5.6.2.3: Placing a signed digital certificate intoa key store file

When a certificate authority issues you a signed certificate for aserver, you need to place that certificate in that server's key store file.The certificate is used by the server to authenticate its identityand to distribute its public key. This article describes howto place a new certificate (either a test or a production certificate)into a key store file using the iKeyman tool.

To place a signed certificate into a server's key store file, complete thefollowing steps:

1. When you receive e-mail from the CA containing your certificate, save the message into a file. In this example, the certificate was saved to a file called PolicyServer1.responseMail.arm.
2. Start the IBM Key Management tool. See article 5.5.6.2, The IBM Key Management tool,for instructions. This displays the IBM Key Management window.



3. Open a destination key database file by selecting **Key Database File --> Open** from the menu bar.
4. Enter the name and location of the key store file at the prompt and click **Open**. The password prompt dialog box is displayed.
5. Enter the key store file's password and click **OK** to continue. The IKeyman window is displayed. The title bar shows the name of the key database file you selected, indicating that the file is open.
6. Click on the certificate types pull-down list beneath **Key Database Context**, and select **Personal Certificates** (the default).
7. Click the **Receive** button. The Receive Certificate from a File dialog window is displayed.
8. Click **Data Type** and select the data type of the signeddigital certificate. Emailed certificates are generally**Base64-encoded ASCII**.
9. Enter the name of the file containing the saved e-mail. You can also use the **Browse** button to find and select the file.

10. Click the **OK** button to continue to add the certificate in the file to the previously selected key store file. The Enter a Labeldialog box is displayed.
11. Type a label for the new signed digital certificate and click**OK**. The IBM Key Management window is displayed. The PersonalCertificates field shows the label of the signed digital certificateyou just added.

At this point, the server's key store file contains both its private key(which was generated as part of requesting the certificate) and thecertificate.

5.5.6.2.5: Making client and server keystore and trust store files accessible

After you have created key store and trust store files and insertedthe necessary certificates, you need to make the key store and truststore files accessible to the client and server programs.

To use created server and client key store and trust store files inyour WebSphere environment, you must first copy them to the client andserver machines.

- Copy the client trust store file (ClientTrustStoreFile.jks) to the following location on the client machine:

  `product_installation_root/etc/ClientTrustStore.jks`
- Optionally, copy the client key store file (ClientKeyStoreFile.jks) to the following location on the client machine:

  `product_installation_root/etc/ClientKeyStore.jks`
- Copy the server key store file (ServerKeyStoreFile.jks) to thefollowing location on the server machine:

  `product_installation_root/etc/ServerKeyStoreFile.jks`
- Copy the server trust store file (ServerTrustStoreFile.jks) to thefollowing location on the server machine:

  `product_installation_root/etc/ServerTrustStoreFile.jks`

Managing the server SSL key store and trust store files

The administrative model in WebSphere Application Server allows theSSL settings for each WebSphere component to be centrally andindividually managed. SSL settings are centrally managed in theadministrative console through the default SSL Settings panel. Inaddition, any of the default settings can be overridden for anindividual component by using the HTTPS, ORB, and LDAPS SSL settingspanels. See article 6.6.18, Securingapplications, for more detailed information about using theadministrative console to configure WebSphere security.

Always use theadministrative console to manage the server key store and trust storefiles. Changes made in the console overwrite any manual changes tothe sas.server.props file. Client key store and trust store files aremanaged in the sas.client.props file because clients can be located ona remote machine.

The Default SSL Settings panel can be used to configure WebSphereApplication Server components using SSL. Parameters that are setthrough the ORB SSL Settings panel override the default SSL settingsfor the ORB. Regardless of which settings are in effect, the ORB usesthese settings as follows. (Additionally, the ORB requires the SASproperties files on the client and server to be configured asdescribed below.)

Key file name

The path of the SSL key file used by server connections. For the server key store file generated in this document, add the following to this field:

`product_installation_root/etc/ServerKeyStoreFile.jks`

Key file password

The password for the SSL key file for server connections. On the server, the key file password is configured in the administrative console.

Key file format

The key file formats supported by the ORB are **JKS**, **PKCS12**, and **JCEK**. **JKS** is the default key file format. The client and server key file format is set through the `com.ibm.ssl.keyStoreType`

property.

Trust file name

The path of the SSL trust file used by clients. On the server, the trust file name is configured in the administrative console. For the client keyring file generated in this document, add the following to this field:

`product_installation_root/etc/ClientTrustStoreFile.jks`

Trust file password

The password for the SSL trust file. On the server, the trust file password is configured in the administrative console.

Client Authentication

The WebSphere AEs ORB does not currently support SSL client authentication using digital certificates.

Managing the client SSL key store and trust store files

You need to modify the sas.client.props file, which is located inthe product installation root/propertiesdirectory. If you used WebAS as the password when yougenerated the client and server keyrings, you need to make thefollowing changes to the sas.client.props file:

- com.ibm.ssl.keyStore=product_installation_root/etc/ClientKeyStoreFile.jks
- com.ibm.ssl.keyStorePassword=WebAS
- com.ibm.ssl.trustStore=product_installation_root/etc/ClientTrustStoreFile.jks
- com.ibm.ssl.trustStorePassword=WebAS

You can now start your WebSphere application using the newlycreated key store and trust store files.

5.5.6.3: Understanding how the Keytool utility works

The *Keytool* utility is a Java-based key-and-certificatemanagement utility. The following categories cover the administrationtasks that are handled by the utility:

- Administering a keystore database discusses tasks that apply to a keystore database.
- Administering key pair entries discusses tasks that apply to key pair entries in a keystoredatabase.
- Administering trusted certificates discusses tasks that apply to trusted certificate entries ina keystore database.
- Administering both certificate and key pair entries discusses tasks that apply to both key pair and trustedcertificate entries.

Options used with the keytool command provides reference information about the options used withthe **keytool** command, and this article covers the followingconceptual and overview topics:

- Rules for using the keytool commands
- Files that are used by the Keytool utility
- Default values
- Standards
- Security considerations

Rules for using the keytool commands

Options are used in combination with the **keytool** command toperform the administration tasks needed to implement and maintain a keystoredatabase. See Options used with the keytool command for the full list of options.

The following rules apply to all options:

- All options are preceded by the minus sign (-).
- The options are case insensitive, so aliases of *ruth* and*Ruth* refer to the same entry.
- Commands must be entered on a single line. (When a command examplein these topics is shown on multiple lines, it is done only to accommodatelimitation in the width of the screen or page.
- The order in which the option occurs in the command string isirrelevant.
- If no password is provided on the command line, the Keytool utility issuesa prompt for the password when it is required to complete the**keytool** command.
- If the value for an option contains a blank space, the value must beenclosed in quotation marks (" ").
- When the **keytool** command is issued with no options, the**keytool** help is activated. (The **-help** option alsoactivates the help facility.)

Files that are used by the Keytool utility

The Keytool utility interacts with several files while it accomplishes itssecurity functions. This topic examines these files and the functionthey serve when used with the Keytool utility.

The .keystore file

The Keytool utility stores its key pair entries and trusted certificateentries in a keystore database. The *keystore database* is afile that has the default name of .keystore and is located by defaultin the user's home directory. The keystore database uses otherfiles to interact with certificate authorities (CAs) and to hold its trustbase, which is its list of trusted certificates.

See Administering a keystore database for more information on the keystore database.

The cacerts files

The *cacerts file* holds the CA certificates, which are the listof trusted certificates managed by the Keytool utility. This fileresides in the JDK security properties directory in the run-time environmentdirectory.

When a new certificate is imported into the keystore, the Keytool utilityverifies that the certificate has integrity (that is, the contents areintact), and that it is authentic (that is, the entity claiming to have sentthe data is actually the entity it claims to be). The Keytool utilityattempts this verification by building a chain of trust from that certificateto the self-signed certificate that belongs to the root CA. Because thelist of trusted certificates held in the cacerts file are already trusted, theKeytool utility uses the certificates in that file as its basis forcomparison.

The Keytool utility supplies five VeriSign root certificates in the cacertsfile. The Distinguished Names associated with the VeriSign root CAcertificates are as follows:

- OU=Class 1 Primary Certification Authority, O="VeriSign, Inc.",C=US
- OU=Class 2 Primary Certification Authority, O="VeriSign, Inc.",C=US
- OU=Class 3 Primary Certification Authority, O="VeriSign, Inc.",C=US
- OU=Class 4 Primary Certification Authority, O="VeriSign, Inc.",C=US
- OU=Secure Server Certification Authority, O="RSA Data Security,Inc.", C=US

See Security considerations for maintaining the cacerts file for information on keeping the cacerts file secure.

See Administering trusted certificates for more information on certificate management by theKeytool utility.

Keytool files used by a CA

The Keytool utility uses the **-certreq** option to generate anauthentication request for a self-signed certificate from a CertificateAuthority (CA). The **-certreq** option creates a CertificateSigning Request (CSR) for the certificate and places the CSR in a file namedcertreq_file.csr, where certreq_file.csr is the name of the filethat is to be sent to the CA for authentication. If a CA considers thecertificate to be valid, it issues a certificate reply and places the reply ina file named cert_reply.cer, where cert_reply.cer is the filereturned by the CA which holds the results of the CSR authorizations that weresubmitted in the certreq_file.csr file. The Keytool utility usesthe **-import** option to read the *.cer file into thekeystore.

Default values

The Keytool utility supplies default values with many of itsoptions. Table 1 identifies the default value when the option has adefault associated with it.

In addition to the option-related default values, the Keytool utility takesits implementation type from the keystore.type property which islocated in the security properties file. Java supplies JKS as thedefault implementation type for use with the Keytool utility. Customizing a keystore implementation type discusses how to enable the JKS type or how to specify acustomized type.

Standards

The Keytool utility uses the following certificate standards:

- X.509 Certificates
- X.500 Distinguished Names
- Internet RFC 1421 printable encoding standard

X.509 Certificates

The Keytool utility uses the *X.509 certificate* standardto define what information is to be included in a certificate and what dataformat is to be used for the information. The information in theX.509 certificate is encoded using Abstract Syntax Notation1(ASN.1) standard to describe data and the Definite Encoding Rules (DER)standard to identify how the information is to be stored andtransmitted. The X.509 certificates takes the values for its Thevalues subject and issuer fields from the X.500 Distinguished Name (DN)standard.

X.500 Distinguished Names

The Keytool utility uses **-dname** option to supply the followingsubcomponents of the *X.500 Distinguished Name*standard:

- CN (common name)
- OU (organization unit)
- O (organization name)
- L (city)
- S (state)
- C (country code)

The choice of including the subcomponent optional; however, if asubcomponent is included, its order of occurrence is mandatory. Theutility is case insensitive to the abbreviations used for thesubcomponents; so, for example, `CN`, `cn`,`Cn`, and `cN` are all identified as the common namesubcomponent for the X.500 DN. The Keytool utility prompts formissing subcomponents when a DN is required.

Internet RFC 1421 printable encoding standard

The Keytool utility uses the *Internet RFC 1421* standard todefine its printable encoding format. This certificate format is alsoknown as *Base 64 encoding*. This format is enclosed by beginand end tagging. However, the **-export** option defaults todisplaying the output in binary encoding. If the printable encodingformat is desired, include the **-rfc** option with the**-export** command.

Security considerations

The security provided by the Keytool utility relies on passwords andcertificate authentication. This section provides suggestions forensuring security.

Security considerations for passwords

Passwords can be specified on the command line or in a script when the**-storepass** or **-keypass** option is supplied.However, prudent security procedures discourage this practice, unless you arein a testing environment

or on a secure system.

When a required password is not supplied, a prompt is issued. Takecare when supplying the password at the prompt because the entry is echoed(displayed as typed) on the screen.

When an identity database is migrated into a keystore database, all privatekeys are encrypted to the same password. The system administrator mustreassign a unique password to each entry. See Migrating an identity database into a keystore database for instructions on performing this task.

Security considerations for importing trusted certificates

Before importing a trusted certificate into your list of trustedcertificates, view its fingerprint by using the **-printcert** optionand compare the output with a secure source. A *fingerprint*is a hash value that is calculated by using a message digest function toencrypt a digital signature. By making a visual comparison between thefingerprint of the received certificate with that of the sent certificate, youcan ensure that the certificate was not tampered with in transit.Unless the **-import** option is issued with the **-noprompt**option included, the **-printcert** option is automatically invoked toensure verification prior to including the certificate in your list of trustedcertificates. (If the **-noprompt** option is issued, nointeraction with the user occurs.)

Security considerations for maintaining the cacerts file

The cacerts keystore file has an initial password of*changeit*. Administrators need to change thispassword. In addition, the JDK installation grants default accesspermission to the cacerts file. Administrators need to change theaccess permission for this file.

5.5.6.3.1: Administering a keystore database

The Keytool utility administrates the storage of keys and certificates in a*keystore* file. A password protects access to the keystore,and within the keystore each private key has its own password. TheKeyStore class, which is provided in the java.security package,contains well-defined interfaces to access and modify multiple types ofkeystore implementations. See Understanding how the Keytool utility works for conceptual information on the use of the Keytoolutility. Options used with the keytool command provides reference information for the options usedwith the **keytool** command.

The administration tasks that you perform using the Keytool utility fallinto the following categories:

- Tasks that apply to the keystore database, which is the focus of thisarticle.
- Tasks that apply to key pair entries. (See Administering key pair entries.)
- Tasks that apply to trusted certificate entries. (See Administering trusted certificates.)
- Tasks that apply to both key pair and trusted certificate entries.(See Administering both certificate and key pair entries.)

Managing a keystore involves the following tasks:

- Creating a keystore
- Adding entries to a keystore
- Deleting a keystore database
- Customizing the name or location of a keystore
- Changing the password for a keystore
- Customizing a keystore implementation type
- Accessing and displaying keystore entries
- Migrating an identity database into a keystore database

Creating a keystore

Use the **keytool** command with the **-keystore** option toexplicitly create a keystore. See Customizing the name or location of a keystore for information on this option.

In addition, to create a default keystore, issue the **keytool**command in combination with the **-genkey**, **-import**, or-**identitydb** options, without including the **-keystore**option. Using the options in this way creates a default file named.keystore and places it in the user's home directory.

For example,

- On a Windows NT system, if a user's ID is `sandra`, then theuser.home system property value is:
  `C:\Winnt\Profiles\sandra`
- On a UNIX system, the default .keystore file is user.homeproperty value translates to the user's home directory.

Adding entries to a keystore

An entry in a keystore can be either of two types:

- A *key entry*. Typically, this is an entry which consistsof a private key and a *certificate chain*. A certificatechain holds a linked set of certified authorizations that connect the publickey back to its corresponding private key.
- A *trusted certificate entry*. This is a certificate whichholds the public key of another entity. The holder trusts in theauthenticity of the certificate because the entity has vouched for thecertificate by signing it.

For more information on keys, certificates and digital signatures, see 5.5: Introduction to certificate-based authentication.

Use the **keytool** command in combination with a**-genkey**, **-import**, or **-identitydb** option toadd an entry to the keystore. See the following topics for informationon these options:

- Generating a key pair entry
- Importing certificates
- Migrating an identity database into a keystore database

Deleting a keystore database

To remove a keystore, use operating system commands to delete the keystorefile.

See Deleting a keystore entry for information on removing an entry from thekeystore.

Customizing the name or location of a keystore

When you include the **-keystore** option with the**-genkey**, **-import**, or **-identitydb** options, The**keytool** command uses the name and location supplied with**-keystore** option to override the default keystore name andlocation.

See Generating a key pair entry for an example of the **-keystore** option combinedwith **-genkey** option.

Changing the password for a keystore

To change the keystore password, combine the **-storepasswd** optionwith the **keytool** command. A prompt is issued for theexisting password, if it is not provided. For example:

```
keytool -storepasswd -new newpassword -storepass oldpassword
```

In this example, the password for the default keystore is changed from*oldpassword* to *newpassword*.

Customizing a keystore implementation type

The KeyStore class, which is provided in the java.security package,contains well-defined interfaces to access and modify multiple types ofkeystore implementations. A *keystore type* defines theformat of the data that is stored in the keystore. It also identifiesthe algorithms used to protect the private keys in the database. SunMicrosystems supplies a proprietary keystore format, JKS, for use as abuilt-in default keystore implementation type. The JKS type usesindividual passwords to protect private keys. It also protects thekeystore database with a password. The default type is identified bythe following line in the security property file:

```
keystore.type=jks
```

Keystore type designations are case insensitive; so JKS is consideredto be the same as jks.

In addition to the default JKS implementation type, thejava.security package contains an abstract KeystoreSpi class, whichenables other keystore formats to be implemented using Service ProviderInterfaces (SPI). When an implementation type other than the defaulttype is used to create the keystore, the client must provide an SPI and supplya KeystoreSpi subclass implementation type.

Each application that uses the keystore retrieves the value for thekeystore.type property and compares the value to each installedprovider until a match is located. Applications use a static methodcalled getDefaultType, which is part of the KeyStore class, to retrieve thevalue of the keystore.type property. An instance of the defaultkeystore type is created by the following line of code:

```
KeyStore keystore = KeyStore.getInstance(Keystore.getDefaultType())
```

Keystores having different implementation types are not compatible.Applications can choose different types of keystore implementations fromdifferent providers. The Keytool utility treats the keystore locationthat is passed to it on the command line as a file name. It reads inthe keystore information and provides access to the file by converting thefile name into a FileInputStream class object.

For information on implementing customized keystore types, see the SunMicrosystems web site:

```
http://java.sun.com/
```

Accessing and displaying keystore entries

The Keytool utility uniquely identifies a keystore entry by itsalias. To access a specific entry, include the **-alias** optionwhen issuing **keytool** commands.

Listing keystore entries

To display keystore entries, combine the **-list** option when youissue the **keytool** command. Include the **-alias**option with the **-list** option to display the entry associated withthat alias. If the entry associated with the alias is a key pair, thefirst certificate in the certificate chain, which is the public key for theentry, is displayed. If the entry associated with the alias is atrusted certificate, then the MD5 fingerprint, in the default binary codeformat is displayed. (A fingerprint is a hash value that is calculatedby using a message digest function to encrypt a digital signature.) Youcan display the output in printable encoding format, as defined by theInternet RFC 1421 standard, by including the **-rfc** option.

If you combine the **-list** option with the **keytool**command and do not include an alias, the entire content of the keystore isdisplayed.

Printing a keystore certificate

The **-printcert** option outputs the fingerprint of the certificateentry, using the MD5 binary code format. If the **-rfc** optionis used with the **-printcert** option, the output is displayed inprintable encoding format. The **-printcert** option enables acertificate's fingerprint to be compared to an entry from a trustedsource.

The contents of a file can be sent to the **-printcert** option bysupplying the file name with the **-file** option.

The **-printcert** option is automatically invoked when the**-import** option is issued. (The **-noprompt** optionsuppresses the **-printcert** output.)

Migrating an identity database into a keystore database

The **-identitydb** option reads the information from a JDK1.1.x-style identity database and migrates it in to thekeystore. The **-file** option is used to supply the file nameof the identity database. If no file name is given, it reads theidentity database from standard input. If a keystore does not alreadyexist, it is created.

Only identities (database entries) labeled as trusted are migrated in tothe keystore. An identity that is rejected is ignored. Thetrusted identity's name is used as the alias for the keystoreentry. All private keys are encrypted under the same password, which is`storepass`. If a default keystore is being created to holdthe entries from the identity database, this same password is automaticallyassigned to the keystore also. When the migration is complete, thesystem administrator must use the **-keypasswd** option to assignindividual passwords to the private keys and the **-storepass** optionto change the default password applied to the keystore.

In an identity database, it is possible to have multiple certificatesassociated with the same public key. In a keystore, each entry has aprivate key and a corresponding public key, which is stored in the first linkof the certificate chain. When identities are migrated from theidentity database into a keystore, only the first certificate in the identityis stored in the keystore. The name of the identity in the firstcertificate becomes the alias in the keystore, and an alias must beunique.

The following command is an example combining the **-identitydb**option with other options:

```
keytool -identitydb -file idb_file -storepass storepass -v
```

This command does the following:

- It reads the information in the file named `idb_file`, stores itas a keystore entry that is identified by an alias, which is created by thename of the identity in the first certificate, and assigns the password`storepass` to all private keys in the identity database and also tothe keystore itself.
- The **-v** option provides a more detailed output.

The **-identitydb** option is combined with the followingoptions:

- **-file**
- **-J**
- **-keystore**
- **-storepass**
- **-storetype**
- **-v**

These options are described in Options used with the keytool command.

5.5.6.3.2: Administering key pair entries

Administrators use the Keytool utility to perform tasks that apply thekeystore database or to the keystore entries: key pairs and trustedcertificates. Administering a keystore database discusses the tasks that apply to the keystoredatabase; Administering trusted certificates discusses tasks that only apply to trustedcertificates entries, and Administering both certificate and key pair entries discusses the tasks that are common to both entrytypes. Understanding how the Keytool utility works provides conceptual information about the Keytoolutility. This article discusses the administrative tasks that applyonly to managing key pair entries in a keystore:

- Generating a key pair entry
- Modifying a key pair entry

Options used with the keytool command provides reference information for the options that are usedwith the Keytool utility.

Generating a key pair entry

The **-genkey** option adds data to a keystore or creates thekeystore if one does not already exist. It generates a key pair (publickey and associated private key) and places the public key in an X.509v1 self-signed certificate. That certificate is stored as asingle-element certificate chain, which is placed, along with the private key,into a new keystore entry. The keystore entry is identified by analias.

The following command is an example of the use of the **-genkey**option in combination with other options:

```
keytool -genkey -dname "cn=Sandra Smith, ou=IBMPITT, o=IBM, c=US"  -alias sandra -keypass acc100
-keystore C:\Winnt\Profiles\sandra   -storepass PITTNV -validity 180
```

Note that the command must be entered as single line. Multiplelines are used in the example due to space constraints.

This command does the following:

- It creates a keystore file named `sandra` in`C:\Winnt\Profiles` directory and assigns the password`PITTNV` to the keystore.
- It generates a public/private key pair for the entity having theDistinguished Name values of `Sandra Smith` for the common name,`IBMPITT` for the organizational unit, `IBM` for theorganization. The password `acc100` is assigned to the privatekey.
- It uses the default DSA key-generation algorithm and creates two keys of1024 bits, the default length.
- It uses a default signature algorithm, SHA1withDSA, to create aself-signed certificate that is valid for 180 days.

The **-genkey** option is combined with the following options:

- **-alias**
- **-dname**
- **J**
- **-keyalg**
- **-keypass**
- **-keysize**
- **-keystore**
- **-sigalg**
- **-storepass**
- **-storetype**
- **v**
- **-validity**

See Options used with the keytool command for a description of these options.

Modifying a key pair entry

Changes can occur that affect the Distinguished Name of a keystore entry,for example, an employee can change departments within the sameorganization. In such a case, the organization unit (OU) subcomponentof the employee's Distinguished Name is changed. It can bedesirable to update an entry's Distinguished Name while still retainingits existing key pair. To do this, follow these steps:

1. Use the **-keyclone** option to create a copy of the existingentry.

   ```
   keytool -keyclone -alias jane -dest janenew
   ```

   In the command, the entry identified by the alias `jane` iscloned and assigned to the destination alias `janenew`.
2. Generate a new self-signed certificate with the new department indicatedin the Distinguished Name.

   ```
   keytool -selfcert -alias janenew   -dname "CN=Jane Brown, OU=Purchasing, O=IBM, C=US"
   ```

Issue this command on a single line; values for the **-dname**option must be specified in the order shown.

3. Generate a Certificate Signing Request (CSR) for the changed entry.

   ```
   keytool -certreq -alias janenew
   ```

4. Import the certificate reply from the Certificate Authority (CA).

   ```
   keytool -import -alias janenew -file VSSjanenew.cer
   ```

5. Remove the obsolete entry from the keystore.

   ```
   keytool -delete -alias jane
   ```

The combination of the **-keyclone** and **-dest** optionsalso can be used to establish multiple certificate chains for a key pair, orfor backup purposes.

5.5.6.3.3: Administering trusted certificates

Administrators use the Keytool utility to perform tasks that apply thekeystore database or to the keystore entries: key pairs and trustedcertificates. Administering a keystore database discusses the tasks that apply to the keystoredatabase; Administering key pair entries discusses tasks that only apply to key pair entries,and Administering both certificate and key pair entries discusses the tasks that are common to both entrytypes. Understanding how the Keytool utility works provides conceptual information about the Keytoolutility and Options used with the keytool command provides reference information for the options used with the**keytool** command. This article discusses the administrativetasks that apply only to managing trusted certificate entries in akeystore:

- Managing trusted certificates
- Adding a trusted certificate to the cacerts file
- Regenerating a self-signed certificate
- Generating a Certificate Signing Request
- Importing certificates
- Exporting certificates

Managing trusted certificates

When the **-genkey** option is used with the **keytool**command to generate a new key pair entry, the public key is automaticallywrapped into a self-signed certificate. A *self-signedcertificate*is one in which the same entity acts as both the issuer(signer) of the certificate and as the authentication subject of thecertificate. This self-signed certificate, containing the public key,takes the first position in the certificate chain that is associated with thecorresponding private key.

Further authentication can be obtained by submitting a certificate signingrequest (CSR) for the self-signed certificate to a certificate authority(CA).

Adding a trusted certificate to the cacerts file

Combine the **-trustcacerts** option with the **-import**option when the **keytool** command is issued to add a new certificateto the list of trusted certificates (the cacerts file).

See Generating a key pair entry for an example of how the **-trustcacerts** option iscombined with the **keytool** command.

See Security considerations for importing trusted certificates for security considerations related to trustedcertificates.

Regenerating a self-signed certificate

Certain circumstances, for example, when an employee transfers to adifferent department within the same company, can necessitate the regenerationof a self-signed certificate in order to assign the same key pair to adifferent X.500 Distinguished Name. The procedure for this taskfollows:

1. Use the **-keyclone** option to copy the original key entry.
2. Use the **-selfcert** option to generate a new self-signedcertificate that uses the new Distinguished Name.
3. Use the **-certreq** option to generate a CSR for the clonedentry.
4. Use the **-import** command to accept the certificate retuned bythe CA.
5. Use the **-delete** option to delete the original (now obsolete)entry.

The certificate is stored in the keystore as a single-element certificatechain. It is identified by the specified alias, and it replaces theoriginal (obsolete) entry.

The following command is an example combining the **-selfcert**option with other options:

```
keytool -selfcert -alias PUB900 -keypass r82Rij     -dname "cn=Barbara Brown, ou=purchaing, o=IBM
c=US"
```

Note that the command must be entered as single line. Multiplelines are used in the example due to space constraints. Also, thevalues for the **-dname** option must be specified in the ordershown.

This command generates a self-signed certificate for which the issuer andthe subject are the same entity.

The **-selfcert** option can be combined with the followingoptions:

- **-alias**
- **-dname**
- **-J**

- **-keypass**
- **-keystore**
- **-sigalg**
- **-storepass**
- **-storetype**
- **-v**

See Options used with the keytool command for descriptions of these options.

Generating a Certificate Signing Request

To generate a Certificate Signing Request (CSR), issue the**keytool** command in combination with the **-certreq**option.

The following command is an example combining the **-certreq**option with other options:

```
keytool -certreq -alias PUB700 -file csrFile
```

This command does the following:

- It generates a CSR to be submitted to a CA. The CSR is held in thecsrFile file.
- It compares the certification returned from the CA with the trustedcertificate for that entry in the cacerts file. If the certificate isaccepted, the **-import** option can be used to place it in thekeystore database.

The **-certreq** option can be combined with the followingoptions:

- **-alias**
- **-file**
- **-J**
- **-keypass**
- **-keystore**
- **-storepass**
- **-storetype**
- **-v**

See Options used with the keytool commandfor a description of these options.

Importing certificates

The **-import** option reads the certificate from the*cert_file* file (or from standard input, if no file is given) andstores it in the **keystore** entry that is identified by thealias. The **-import** option can be used with the**keytool** command to import X.509 v1, v2, or v3 certificatesand PKCS#7-formatted certificate chains. The data to be importedcan be stored in binary encoding format or in printable encoding format(Base64 encoding). If printable encoding format is used, it must adhereto the Internet RFC 1421 standard, as shown:

```
 "- - - - -BEGIN CERTIFICATE- - - - -" certificate information- bounded by Begin-End string  "- - - - -END CERTIFICATE- - - - -"
```

The following command is an example combining the **-import** optionwith other options:

```
keytool -import -alias PUB500 -file foreign.cer -keypass changeit -trustcacerts
```

Note that the command must be entered as single line.

This command does the following:

- It reads the certificate in the file named `foreign.cer`,stores it as a keystore entry that is identified by the alias`PUB500`, and assigns the password `changeit` to the privatekey.
- It gives consideration to including the certificate in the cacerts file(located in the JDK security properties directory) into its chain oftrust.
- It creates a default keystore file using the default type. Itprompts for the keystore password. If the certificates are rejected bythe chain of trust, it prints out the fingerprint of the rejected certificateto enable a manual comparison with a trusted source. (If the**-noprompt** option has been included with the command, there is nointeraction with the user.)
- Its certificate is valid for the default period of 90 days.

See The cacerts files for more information on how the keytool utility uses thecacerts file.

See Security considerations for maintaining the cacerts file for information on keeping the cacerts file secure.

The **-import** option can be combined with the followingoptions:

- **-alias**

- **-file**
- **-J**
- **-keystore**
- **-rfc**
- **-storepass**
- **-storetype**
- **-v**

See Options used with the keytool command for a description of these options.

Exporting certificates

The **-export** option reads the certificate associated with thespecified alias from the keystore and places it in a file, which is suppliedby the **-file** option (or by standard output, if no file isgiven).

If the specified alias is associated with a trusted certificate, thedefault output is in binary code format. The **-rfc** option canbe added to change the output to printable encoding format (Internet RFC1421). If the specified alias is associated with a key pair entry, thefirst certificate in the chain, which authenticates the public key, isreturned.

The following command is an example combining the **-export** optionother options:

```
keytool -export -alias joebrown  -file joebrown.cer
```

This command reads the entry associated with the alias `joebrown`and places it in binary format into the file named`joebrown.cer`. A prompt is issued for the keystorepassword because the **-storepass** option was not included with thecommand.

The **-export** option can be combined with the followingoptions:

- **-alias**
- **-file**
- **-J**
- **-keystore**
- **-rfc**
- **-storepass**
- **-storetype**
- **-v**

See Options used with the keytool command for a description of these options.

5.5.6.3.4: Administering both certificate and key pair entries

Administrators use the Keytool utility to perform tasks that apply to thekeystore database or to the keystore entries: key pairs and trustedcertificates. Administering a keystore database discusses the tasks that apply to the keystoredatabase; Administering key pair entries discusses tasks that apply to key pair entries, and Administering trusted certificates discusses the tasks that apply totrusted certificate entries. Understanding how the Keytool utility works provides conceptual information about the Keytool utilityand Options used with the keytool command provides reference information for the options used with the**keytool** command. This article discusses the administrativetasks that apply both keystore entry types and covers the followingtopics:

- Assigning an alias
- Deleting a keystore entry
- Setting an expiration period
- Changing a password for a keystore entry

Assigning an alias

All keystore entries, whether key pair entries or trusted certificateentries, are identified by a unique alias. The alias is assigned to theentry when you generate a new public-private key pair (**-genkey**option), when you import a certificate to the list of trusted certificates(**-import** option), or when you migrate an identity database(**-identitydb** option).

Subsequent **keytool** commands use the alias to identify the entryon which the operation is to be performed.

Deleting a keystore entry

To delete a keystore entry, identify the entry by its alias and issue the**keytool** command in combination with the **-delete**option. For example:

```
keytool -alias fred -delete
```

This command removes the entry associated with the alias `fred`from the keystore.

Setting an expiration period

The default expiration period for a keystore entry is 90 days. Tochange this value, identify the entry by its alias and issue the**keytool** command in combination with the **-validity**option. For example:

```
keytool -alias sally -validity 180
```

In addition, when the entry is initially created, the expiration period canbe changed by using the **keytool** command with a**-genkey**, **-import**, or**-identitydb** option and adding the **-validity**option.

Changing a password for a keystore entry

To change the password associated with an keystore entry, issue the**keytool** command in combination with the

-keypasswdoption for an entry, which is identified by its alias. Forexample:

```
keytool -keypasswd  -alias sally oldpassword -new newpassword
```

This command changes the password for the entry identified as`sally` from *oldpassword* to *newpassword*.A prompt is issued for the existing password associated with the specifiedalias, if no password is supplied with the command.

See Changing the password for a keystore for information on changing the password for the keystoredatabase.

5.5.6.3.5: Options used with the keytool command

Administrators use the Keytool utility to perform tasks that apply thekeystore database or to the keystore entries: key pairs and trustedcertificates. Administering a keystore database discusses the tasks that apply to the keystoredatabase; Administering key pair entries discusses tasks that apply to key pair entries;Administering trusted certificates discusses tasks that apply to trusted certificate entries,and Administering both certificate and key pair entries discusses the tasks that are common to both entrytypes. Understanding how the Keytool utility works provides conceptual information about the Keytoolutility. This article provides reference information about the optionsthat are used with the **keytool** command.

Table 1 lists the options that can be combined with the**keytool** command. The columns provide the followinginformation:

- **Options**-- Specifies the option that can be combined withthe keytool command
- **Function**--Briefly describes the administrative taskaccomplished by the option
- **Values**--Lists valid data entries for the option
- **Components**--Identifies the Keytool components (keystore,key pair entries, trusted certificate entries) with which the option can beused
- **Use**--Provides additional information about using theoption

Table 1. Options used with the keytool utility

Option	Function	Values	Components	Use
-alias	Assigns an identity to a keystore entry	User supplied	• Key pair entries • Trusted certificate entries	• Case insensitive • mykey (Default)
-certreq	Generates a certificate signing request	Requires a **-file** option supplying the .csr file name	• Key pair entries	Submitted to a certificate authority
-delete	Removes an entry from the keystore	Requires a **-alias** option to identify the entry	• Key pair entries • Trusted certificate entries • Keystores	Case insensitive
-dest	Identifies the destination alias for a cloned entry	User supplied	• Key pair entries • Trusted certificate entries	

-dname	Assigns an X.500 Distinguished Name to an entry	User supplied	• Key pair entries • Trusted certificate entries	• Order of subcomponents matters • Inclusion of subcomponents is optional
-export	Outputs a certificate in binary code	Requires a **-file** option to supply the output file	• Key pair entries • Trusted certificate entries	
-file *name*	Identifies files to be used for import or export	User supplied • Input: an identity database • Input: a certificate reply from a certificate authority • Output: certificate signing request	• Key pair entries • Trusted certificate entries • Keystores	• Standard input (default for reads) • Standard output (default for writes)
-genkey	• Creates a new key pair entry • Creates a keystore, if none exists	User supplied	• Key pair entries	
-help	Displays help for the Keytool utility			Issuing the **keytool** command with no options also displays help
-identitydb	Migrates an identity database to a keystore database	Requires the **-file** option to supply the identity database name	• Keystores	Only trusted entries are imported
-import	Brings the contents of a file into the keystore	Requires the **-file** option to identify the file source	• Trusted certificate entries	Automatically invokes the **-printcert** option (unless the**-noprompt** option is included)
-J *command*	Passes a Java command to the interpreter			
-keyalg	Signifies the algorithm to be used for key pair creation	• DSA (default) • RSA	• Key pair entries • Trusted certificate entries	Entry for this option determines the value for the **-sigalg**option

-keysize	Specifies a key size	Requires a value in multiples of 64 bits	• Key pair entries • Trusted certificate entries	• 1024 bits (default) • Range is from 512 to 1024 bits
-keypass	Assigns a password to a key pair	User supplied	• Key pair entries • Trusted certificate entries	Case insensitive
-keystore	Customizes the name and location of a keystore	User supplied	• Key pair entries • Trusted certificate entries • Keystores	The **-genkey**, **-import**, or **-identitydb**options create a keystore if none exists
-keypasswd	Changes a password for a keystore entry	User supplied	• Key pair entries • Trusted certificate entries	Case insensitive
-keyclone	Clones a key store entry	Requires a **-dest** option to identify the destination alias	• Key pair entries • Trusted certificate entries	
-list	• Display an entry if an alias is supplied • Display the contents of a keystore if no alias is supplied		• Key pair entries • Trusted certificate entries • Keystores	MD5 fingerprint (default)

-new	Identifies the new password	User supplied	• Key pair entries • Trusted certificate entries • Keystores	Combined with the **-keypasswd** and **-storepasswd**options
-noprompt	Indicates that no prompts are to be issued during an import operation		• Trusted certificate entries	Suppresses the default **-printcert** option associated with a**-import** option
-printcert	Prints a certificate fingerprint		• Trusted certificate entries	Binary code format (default)
-rfc	Converts output display to printable encoding format	Combined with the **-printcert** and **-list** options	• Trusted certificate entries	Uses Internet RFC 1421 standard
-selfcert	Generates a new self-signed certificate	• If **-dname** option is supplied, issuer and subject take theX.500 Distinguished Name • If no **-dname** option is supplied, issuer and subject takeX.500 Distinguished Name of alias	• Key pair entries • Trusted certificate entries	• Output: X.509 v1 self-signed certificate
-sigalg	Specifies the algorithm to be used to sign the certificate	• SHA1withDSA • MD5withRSA	• Key pair entries • Trusted certificate entries	Correlates with the value for the **-keyalg** option
-storetype	Assigns a type to a keystore or an entry into a keystore	A Service Provider Interface format	• Key pair entries • Trusted certificate entries • Keystores	• JKS (Default) • Case insensitive
-storepass	Assigns a password to a keystore	User supplied		Case insensitive

-trustcacerts	Indicates that the certificate is to be considered for inclusion in thelist of trusted certificates (the cacerts file)		• Trusted certificate entries	
-v	Designates verbose output			
-validity	Identifies an expiration period		• Key pair entries • Trusted certificate entries	90 days (default)

5.5.7: Introduction: Setting up an LDAP connection over SSL

This topic describes how to establish an SSL connection between WebSphereApplication Server and an LDAP server. This page gives an overview; referto the linked pages for more details.

Setting up an SSL connection between WebSphere Application Server andan LDAP server requires two logical tasks:

1. Establishing a WebSphere-to-LDAP connection without SSL
2. Enabling SSL over the WebSphere-to-LDAP connection

To establish a connection between WebSphere and an LDAP server, you must:

1. Create certificates and keys for the WebSphere server to use in authentication, and create a trust store that will also hold a certificate used for validating certificates for the LDAP server.
2. Configure the LDAP server of your choice.

After you have established the WebSphere-to-LDAP connection, you canadd the SSL constraint to the connection. To do this, you must

1. Configure your LDAP server to use SSL.
2. Get the necessary certificates for authenticating the LDAP server and add them to your WebSphere trust store.
3. Configure WebSphere to use SSL.

5.5.7.1: Establishing connections betweenapplication servers and LDAP servers

1. Disable WebSphere security before shutting down the administrativeserver and client. This is not strictly necessary, but it makesrecovery easier if something goes wrong.
2. To use SSL between WebSphere Application Server and the LDAPserver, create your own key and trust store files (if youhave not done so already). Put the LDAP server's certificate inthe trust store file, as this is used for most public keys. The keystore is used for a server's or client's (in the case of clientauthentication) private keys.

   The same trust store file can be used for LDAP as is used for theORB and HTTPS. Add the LDAP server's public key or root CAcertificate to the trust store specified in the Default SSLConfiguration in the Security Center of the administrative console.See the articles under section 5.5.6, Tools formanaging certificates and keys, for instructions on how to createkey and trust stores with the WebSphere Application Server key tools.

   The key and trust store files you create are used to configureglobal security. They are also used to enable an SSL connectionbetween WebSphere and the LDAP server.
3. Place your server key and trust store files in the appropriatedirectories on the server machine. See Making client and server key store and truststore files accessible for details.
4. WebSphere determines which key and trust store files to use andtheir passwords based on the settings in the Default SSL Configurationpanel in the Security Center of the Administrative Console. You canalso override the default settings by changing the LDAP SSL Settingsin the Security Center.
5. Restart the administrative server and client and configure WebSphere Security including LDAP.
   1. Enable Security (under the **Security Center --> General**).
   2. Set the Default SSL Configuration (under **Security Center --> General --> Default SSL Configuration**).
   3. Set the Authentication Mechanism to Lightweight Third-Party Authentication (LTPA) (under **Security Center --> Authentication --> Authentication Mechanism**)
   4. Set up your LDAP settings (under **Security Center --> Authentication Tab --> LDAP Settings**)
      - Choose a Security Server ID from your LDAP user registry. This ID must be a valid user from the registry. Do not use the LDAP administrative ID because this is not a searchable ID and validation failures will occur.
      - Set the Security Server Password associated with the Security Server ID.
      - Set the host name or IP address of the LDAP server.
      - Set the port to 389 (or whatever the TCP/IP listener port is for your LDAP server).
      - Set the Base Distinguished Name of your LDAP directory.
      - Optionally, set the Bind Distinguished Name and Bind Password of your LDAP server.
      - Optionally, modify the Advanced settings as necessary for your LDAP server's directory configuration.
      - Do not select the **SSL** button and then **Enable SSL** yet.
   5. Click **Finish**.

The application server now communicates with the LDAP server and the Security Server ID will be authenticated. If the Security Server ID is not valid, you should receive an error message indicating this. Check your LDAP server's configuration to resolve any problems with the WebSphere LDAP Settings. You can verify

the communication with your LDAP server by monitoring its connections.

5.5.7.2: Enabling SSL connections between WebSphere ApplicationServer and an LDAP Server

1. Configure SSL in the LDAP server. The procedure varies with the LDAP server being used. Consult the documentation for your server for details. For example, with the SecureWay LDAP server, the following must be done:
    1. Set the SSL status to **SSL ON**.
    2. Set the Authentication Method to **Server Authentication**. The SSL protocol requires the server to be authenticated. In this case, the LDAP server is the server and WebSphere Application Server is the client. If you need mutual authentication, choose **Server and Client Authentication**.
    3. Make sure that the secure port is set to 636. (You can optionally choose a different port, but you must set this port correctly when configuring LDAP SSL in WebSphere Application Server.)
    4. Point the Key Database path and filename to the LDAP server's keyfile. In SSL, certificates are used for authentication. Therefore, the LDAP server requires a certificate, which must be included in its keyfile.
    5. Set the Key Label to the label used for the LDAP server's certificate.
2. Update your WebSphere Application Server trust store file. The trust store file is the repository for the WebSphere server's trust base. Because it needs to authenticate the LDAP server during SSL initialization, the trust store file must provide information about the LDAP server.

    In order to validate the LDAP server's certificate, your server needs the public key of the CA that issued the LDAP server's certificate. This key is found in that CA's certificate, so you need to add the certificate of the CA that issued the LDAP server's certificate to your trust store file on the server. (For more information on authentication by certificate, see 5.5: Certificate-based authentication.)

    To add the additional certificate to the trust store file, do the following:
    1. Run IKeyMan, as described in 5.5.6.2: The IBM Key Management tool
    2. Add the new certificate to the server's trust store file.
3. Enable the SSL connection in WebSphere.
    1. Modify your LDAP configuration (under **Security Center --> Authentication --> LDAP Settings**).
        1. Set the port to 636. (If you used a different port number, set the port to that numer.)
        2. Click **SSL**.
        3. Click **Enable SSL**.
        4. Select **Use Global SSL default configuration**, unless you want to use a different key and trust store file for LDAP.
    2. Click **OK**.
4. Stop and restart the administrative server and client. After they restart, you are prompted to login to the LDAP registry.

Tips

- If your SSL connection does not work, try the following:
    1. Verify that your LDAP server is listening to port 636 (orthe other port specified in the settings).
    2. Verify that the LDAP server's certificate is still valid.

- If you need to export the certificate for the LDAP server's CA from keyring or other type of file, look for an option that lets you export the certificate in DER binary format or Base64-encoded ASCII. The tools you have can vary with the LDAP server.
- If you transfer a certificate file from a remote host by using FTP, be sure to set the transfer mode to binary.
- Make sure that your place your updated keyring class in the correct location.

5.5.7.4: Example: Generating key andtrust store files for SSL

This procedure describes how to create key and trust store filesthat permit SSL communications between WebSphere Application Serverand an LDAP server. This require the creation of key and trust files,one set for the server and one set for the client. The server's keystore file contains the public and private keys for the server. Theserver's trust store file contains the certificate authority'scertificate. The client's key store file contains public and privatekey of the client (if client authentication is desired). The client'strust store file stores the server's public key and the CA's rootcertificate.

1. Download the external public certificate for the root certificate authority (root CA) and save it to a file. In this example, the file is called caroot.arm.
2. Generate the server-side key store and trust store files.
    1. Request a certificate for the server, if it doesn't already have one.
        1. Generate a certificate request from within the key store file and save it to a file. In this example, the file is called certreq.arm.
        2. Submit the request to the certificate authority.
        3. Save the newly obtained certificate to a file. In this example, the file is called newcert.arm.
    2. Place the certificate into a key store file. This can be done using either the keytool command-line tool or the graphical IBM Key Managment (Ikeyman) tool. For example, if you are using the Ikeyman tool, you must:
        1. Create a new key store file. In this example, the file is called ServerKeyStore.jks.
        2. Specify the the certificate in the newcert.arm file as the certificate to be received into the keyring file. This is done on the Personal Certificates panel in the Ikeyman tool.
        3.
        4. The client also needs access to the server's certificate, so extract the certificate and save it to a file. In this example, the file is called websphere.arm.
        5. Add the certificate of the signing CA (saved in the file caroot.arm) to the key store file. This is done on the Signer Certificates panel in the Ikeyman tool.
3. Generate the client-side key and trust store files. This can be done using either the keytool command-line tool or the graphical IBM Key Managment (Ikeyman) tool. For example, if you are using the Ikeyman tool, you must:
    1. Create a new trust store file. In this example, the file is called ClientTrustStoreI.
    2. Add the certificate of the signing CA, saved in the file caroot.arm, to the trust store file. This is done on the Signer Certificates panel in the Ikeyman tool.
    3. Add the certificate of the server, saved in the file websphere.arm, to the key sore file. This is also done on the Signer Certificates panel in the Ikeyman tool.
    4. Optionally, if client authentication is desired, create a new client key store file called ClientKeyStoreFile.jks. You can then request a certificate from a CA, submit the certificate request to the CA, and add the certificate to the client key store file.
4. Install the new keyring files into the WebSphere Application Server environment. Place all key and trust store files (ServerKeyStoreFile.jks, ServerTrustStoreFile.jks, ClientKeyStoreFile.jks and ClientTrustStoreFile.jks) on the server in the product_installation_root/etc directory.
5. Configure the server properties as follows:
    1. Start the administrative console.
    2. Open the **Security Center**.
    3. Select **Default SSL Configuration**.
    4. Modify the following SSL properties:
        - Key File Name: product_installation_root/etc/ServerKeyStoreFile.jks
        - Key file password: WebAS
        - Confirm password: WebAS
        - Key file format: JKS
        - Trust file name: product_installation_root/etc/ServerTrustStoreFile.jks
        - Trust file password: WebAS
        - Confirm password: WebAS
        - Security level: high (128 bit encryption)

    If you use the same file for key and trust stores, you can specify the same file name for both properties:
        - Key File Name: product_installation_root/etc/ServerKeyStoreFile.jks
        - Trust File Name: product_installation_root/etc/ServerKeyStoreFile.jks

    If you only specify a key file name, the trust file name is automatically set to the same name as the key file name.
6. The client side requires only the ClientKeyStoreFile.jks and ClientTrustStoreFile.jks files. Modify the following lines in the sas.client.props file:

```
com.ibm.ssl.trustStore=ClientTrustStoreFile.jkscom.ibm.ssl.trustStorePassword=WebAScom.ibm.ssl.trustStoreType=JKScom.ibm.ssl.keyStore=ClientKeyStoreFile.jkscom.ibm.ssl.keyStorePassword=WebAScom.ibm.ssl.keyStoreType=JKScom.ibm.ssl.protocol=SSLv3com.ibm.CORBA.standardPerformQOPModels=high
(128 bit encryption)
```

5.6: Establishing trust association with a reverse proxy server

WebSphere Application Server can authenticate incoming user requests, but in somescenarios, like Web-based applications, it is often desirable to delegate this work toanother process, typically a reverse proxy server. This delegation requires theestablishment of a trust relationship, or *trust association*, between WebSphereApplication Server and the proxy server. In this case, the proxy server authenticates theclients for WebSphere Application Server, which accepts the authentication because ittrusts the proxy. WebSphere Application Server applies its authorization policies to therequests.

To delegate authentication work to a third-party server, two things must be done:

- You must have an *interceptor*, that is, a Java class, which is used by WebSphere Application Server to receive requests from the proxy server.
- You must establish trust between the proxy server and WebSphere Application Server. This typically requires the proxy to authenticate to WebSphere Application Server.

WebSphere Application Server provides a ready-to-use interceptor for Tivoli WebSealVersion 3.6, but you can also write your own; see Writing a custominterceptor for more information. The other related information discusses theconfiguration of WebSphere Application Server and WebSeal.

When the interceptor is in place and a trust relationship is established, WebSphereApplication Server is able to accept and process HTTP requests that come through the proxyserver rather than directly from the HTTP client. The proxy server authenticates the HTTPclients and passes authenticated requests to WebSphere Application Server. WebSphereApplication Server authorizes access to the requested resources based on the application'sauthorization policies.

Before the authorization of clients can be delegated to a proxy server, the followingWebSphere prerequisites must be met:

- Security must be enabled in WebSphere Application Server. If it security is disabled, incoming requests cannot be selectively authorized and refused.
- The authentication mechanism used by WebSphere Application Server must be Lightweight Third-Party Authentication (LTPA). You cannot delegate authentication to a proxy if you are using the local operating system as your authentication mechanism.
- If you are using WebSeal Version 3.6 as your reverse proxy server, certificates are not supported as a challenge mechanism. Only the basic authentication, that is, a user ID and password combination, is supported.
- Trust Association must be enabled in the **Authentication** tab of the Security Center in the administrative console.

5.6.1: Configuring trust association between WebSphereApplication Server and WebSeal Version 3.6

To enable use of a trust association between WebSphere ApplicationServer and WebSeal, you must perform configuration work for each ofthe following:

- WebSphere Application Server
- The interceptor for WebSeal (configuration is optional)
- WebSeal

This file describes the configuration for each component andprovides a sample configuration.

Configuring WebSphere Application Server to run in trust association

Configuring WebSphere Application Server to run in trustassociation is a two-step process:

1. Enable trust association in the Security Center console.
2. Set up the trust-association interceptors that are going to receive HTTP requests from the trusted proxy server.

Enabling trust association

To enable trust association in the Security Center console, do thefollowing:

1. Start the administrative server for the domain, if necessary.
2. Start the administrative console, if necessary.
3. Click on the **Console** action bar and then choose **Security Center** from the drop-down menu.
4. Click the **Authentication** tab in the Security Center.
5. Select the **Enable Web Trust Association** check box inthe LTPA settings group.
6. Complete the LDAP registry information, if necessary, by selecting **LDAP**. See 6.6.18.0.7: Properties for configuring LDAP support for more information.
7. Click **OK** to save the changes and close the SecurityCenter console.

Setting up trust-association interceptors

Create a file called named trustedservers.properties, and place the filein the *product_installation_root*/properties directory.

The trustedservers.properties file for WebSeal must include the followingthree lines and an optional fourth line:

```
com.ibm.websphere.security.trustassociation.enabled=truecom.ibm.websphere.security.trustassociation.types=webseal36com.ibm.websphere.security.trustassociation.webseal36.interceptor=com.ibm.ejs.security.web.WebSealTrustAssociationInterceptorcom.ibm.websphere.security.trustassociation.webseal36.config=webseal36
```

The following describes each of the property-value pairs:

- `com.ibm.websphere.security.trustassociation.enabled=true`
  This property-value pair enables the use of trust assocation.
- `com.ibm.websphere.security.trustassociation.types=webseal36`
  This property-value pair specifies the types of the servers with which you are establishing trust. If you are using multiple proxy servers, you can specify a comma-delimited list as the value.
- `com.ibm.websphere.security.trustassociation.webseal36.interceptor= com.ibm.ejs.security.web.WebSeal36TrustAssociationInterceptor`
  This property-value pair specifies the name of the Java class implementing the interceptor for the proxy. When specifying this class, note the following:
  - The class must be locatable from the information on the class path.
  - You only need to specify the implementation class for an interceptor once, even if multiple proxy servers use the same implementation class for the interceptor.
- `com.ibm.websphere.security.trustassociation.webseal36.config=webseal36`
  OPTIONAL. This property-value pair specifies a configuration file for the WebSeal36 interceptor. The contents of this file are described under "Configuring the WebSeal interceptor."

Each property-value pair must appear on a single line in the file. Pairsappearing on more than one line in this example have been broken forreadability.

Configuring the WebSeal interceptor (optional)

WebSphere Application Server provides a Java class,`com.ibm.ejs.security.web.WebSeal36TrustAssociationInterceptor`,that implements the essential interceptor for enabling trust associationbetween WebSeal 3.6 and WebSphere Application Server.

By default, the interceptor processes all HTTP requests it receives.You can configure the interceptor to restrict the requests that itprocesses locally. The restrictions can be specified by identifier,originating host, and originating port, and by combinations.This configuration is optional.

To configure the interceptor, create a property file for theoptional configuration-file property, and place the file in the*<product_installation_root>*/properties directory. In this example, we create a file calledwebseal36.properties to correspond to the the optional property-value pair`com.ibm.websphere.security.trustassociation.webseal36.config=webseal36`specified in the trustedservers.properties file.

Use this file to set properties restricting the requests thatinterceptor will process. The properties act as requirements onrequests, and each request must meet all of the requirements.Requests not meeting all of the requirements are not processedby the interceptor; they are passed on to WebSphere ApplicationServer for processing.

The file can set values for any of the following WebSeal properties, forexample:

- `com.ibm.websphere.security.webseal36.id=iv-user, iv-creds`
  This property-value pair tells the interceptor to filter incoming HTTP requests by identifier. The value is a comma-delimited list of identifiers. Every HTTP request is examined by the interceptor. Only those requests that contain *all* of the listed IDs as request-header names are be considered for processing by the interceptor. All other requests are passed on to WebSphere Application Server for processing in the usual way. By default, all HTTP requests are considered by the interceptor for processing.
  Because the WebSeal36 interceptor should process *only* HTTP requests from WebSeal, the recommended value for use with WebSphere Application Server sets this property to one or both of these values:
  - iv-user
  - iv-creds

  The example property-value pair uses both.
- `com.ibm.websphere.security.webseal36.hostnames= <hostname1>,<hostname2>`
  This property-value pair specifies a list of names of the machines on which WebSeal servers run and from which the interceptor can accept HTTP requests. If this property is not set, the interceptor accepts requests from any host.
- `com.ibm.websphere.security.webseal36.ports=444`
  This property-value pair specifies the ports from which HTTP requests must originate in order to be processed. Requests originating from other ports are ignored. The list applies to all hosts from which the interceptor accepts requests. There is no way to specify a list of ports for one host and a different list for a different host. If this property is not set, requests originating from any port are considered for processing.

Configuring WebSeal

The last step is to configure Tivoli's WebSeal product. This product is notpart of WebSphere Application Server, so you should consult the WebSealdocumentation for details and in case of problems.

To enable communication between WebSeal and WebSphere Application Server,the the Web server being used by WebSphere Application Server must becomean SSL junction in the schema of the Tivoli Policy Director. If the Webserver is using the default SSL port, port 443, create an SSL junctionwith the following **junctioncp** command:

```
create -c -t ssl -h <hostname> /<junction-name>
```

where

- The `-c` flag directs WebSeal to pass its authentication information in the basic authentication header of every request that it sends to WebSphere Application Server. The authentication information is the user ID and password of the WebSeal server. This allows WebSphere Application Server to authenticate every request that it receives from the WebSeal server.
- The `-t ssl` option requests the creation of junction of the type SSL.
- The `-h <hostname>` option specifies the host machine of the Web server used by WebSphere Application Server.

For example, the command:

```
create -c -t ssl -h was_host.raleigh.ibm.com /myjunction
```

creates an SSL junction called myjunction for the machine was_host.raleigh.ibm.com.

If the Web server is not listening to the default SSL port, port 443,use the port option to the **junctioncp** command to indicate the portbeing used:

```
-p <port_number>
```

The WebSeal server must have a user ID and password it can use whenit authenticates to WebSphere Application Server. To set up this authenticationinformation, you must do the following:

1. Designate a ID from the WebSphere Application Server user registry for use by WebSeal. You can create a special WebSeal ID in WebSphere Application Server, or you can simply use an existing ID from the WebSphere Application Server registry.
2. Put this user ID and associated password in the WebSeal configuration file, iv.conf. In this file, you must have the following:

   ```
   basic_auth_username=<userId >        basic_auth_passwd=<password>
   ```

   where *<userId>* and *<password>* are valid account information from the WebSphere Application Server registry.

Because SSL is involved in the junction, you must ensure that the Webserver being used by WebSphere Application Server is configured with SSL usingserver authentication only. In this configuration, WebSeal plays a clientrole. Therefore, you must copy the certificate of the issuing CA of theWeb server into the WebSeal certificate directory.

Please consult the WebSeal Policy Director manual for detailed informationon setting up SSL connections between WebSeal and a junction server.During the procedure, be sure to update the configuration file for thesecurity manager, secmgrd.conf, to include the following line:

```
junction-ca-cert-file = <ca-certfile>
```

where *<ca-certfile>* is the aboslute path of the filecontaining the CA certificates of the junction servers, for example,

```
/opt/intraverse/lib/certs/junctioncacert.pem
```

.Without the line, basic authentication will not take place betweenWebSeal and WebSphere Application Server.

Finally, to access a resource through WebSeal, you need to use SSL. Therefore,you must ensure that WebSeal itself is configured for SSL.

Sample configuration

This section describes a sample configuration.

- WebSphere Application Server is installed on the machine was_host.raleigh.ibm.com.
- The Web server is Netscape Enterprise Server, also installed on the machine was_host.raleigh.ibm.com. The Web server is listening on port 4343 for SSL requests.
- The LTPA security mechanism is used, with the LDAP server residing on the machine ldap_host.raleigh.ibm.com.
- WebSeal is installed on the machine webseal_host.raleigh.ibm.com. It listens on port 444 for SSL requests.
- A junction was created using the following command:
  ```
  junctioncp create -c -t ssl -h was_host.raleigh.ibm.com -p 4343 /myjunction
  ```
- In the WebSeal iv.conf file, the following lines are included:

```
basic_auth_username=web_user       basic_auth_passwd=testpassword
```

where the ID web_user with password testpassword is registered in the WebSphere Application Server registry.

- In the Policy Director secmgrd.conf file, the following line is included:
  `junction-ca-cert-file=/opt/intraverse/lib/certs/junctioncacert.pem`
- The ID testuser1 with password sherlock is a valid WebSeal user. It is also a valid WebSphere Application Server user.

A user tests the system by logging in as testuser1 and attempting accessthe WebSphere Application Server servlet /servlet/snoop:

- To test access without WebSeal, the user enters the following in the Web browser:
  `https://was_host.raleigh.ibm.com/servlet/snoop`
- To test access through WebSeal, the user enters the following:
  `https://webseal_host.raleigh.ibm.com:444/aim/servlet/snoop`

In both cases, a prompt is displayed in which the user enters thetestuser1/sherlock combination and the snoop servlet is displayedon the Web browser.

5.6.2: Frequently asked questions about trust associationbetween WebSphere Application Server and WebSeal

Can I still submit requests directly to WebSphere Application Server,without passing through Web Seal?
Yes. WebSphere Application Server will behave in the usual manner when requests are not received from the WebSeal server. However, please review the above section about the WebSeal36 interceptor.

What happens if security is not enabled in WebSphere Application Server,and the HTTP request is given to the WebSeal server?
The WebSeal server will still try to authenticate the user. If authenticationis successful, WebSphere Application Server is going to serve the requestwhether or not the user has permissions to access the resource.

Can I have trust associations with several WebSeal servers, possiblyfrom different locations, at the same time?
Yes, to the extent that different WebSeal servers are allowed to createjunctions to the same Web server.

Will WebSphere Application Server single sign-on (SSO) work with WebSeal3.6 as a front-end?
Yes. If your setup is such that there is only one WebSeal server andseveral junctions to Web servers, SSO itself is taken care of by WebSeal,and in this case, the SSO domain name of WebSphere ApplicationServer installation might not even matter. WebSphere Application ServerSSO will work the usual way even for a setup consisting of several WebSealservers, each one having a junction to a Web server being used byWebSphere Application Server.

Can I use the same LDAP directory for my WebSeal server and WebSphereApplication Server?
Yes. However, users and groups that were created by the Policy Directoritself may not be shared with WebSphere Application Server as schema specificto the Policy Director might be in use.

What if I want to demand that all requests pass through my WebSeal server?
To have all requests pass through the WebSeal server, simplydo none of the optional configuration of the interceptor.In that case, every HTTP request is processed by the interceptor.

Can I use custom login with trust association?
No. There is no point in doing so. Remember that WebSeal does theauthentication. Therefore, when the request reaches WebSphere ApplicationServer, it ignores any challenge type declared for your application.

What happens if I disable trust association and access a WebSphereApplication Server resource through the WebSeal server?
The WebSeal server will still try to authenticate the user. However, because there is no interceptor involved, WebSphere Application Server will applywhatever challenge type is appropriate for the resource requested. If thechallenge type is basic, the WebSeal ID and password will alwaysbe used. Thus, the end user ID and password will be ignored.Certificate challenge type will not work. Custom login will notwork either.

5.6.3: Writing a custom interceptor

If you are using a third-part reverse proxy server other than TivoliWebSeal Version 3.6, you must provide an implementation class for theWebSphere interceptor interface for your proxy server. This filedescribes the interface you must implement.

Using the TrustAssociationInterceptor interface

WebSphere Application Server provides the interceptor Java interface,com.ibm.websphere.security.TrustAssociationInterceptor, whichdefines the following methods:

- `public boolean isTargetInterceptor(HttpServletRequest req) throws WebTrustAssociationException;`
- `public void validateEstablishedTrust(HttpServletRequest req) throws WebTrustAssociationException;`
- `public String getAuthenticatedUsername(HttpServletRequest req) throws WebTrustAssociationException;`

The isTargetInterceptor method is used to determine whether therequest originated with the proxy server associated with the interceptor.The implementation code must examine the incoming request objectand determine if the proxy server forwarding the request is avalid proxy server for this interceptor. The result of this methoddetermines whether the interceptor processes the request or not.

The validateEstablishedTrust method determines if the proxy serverfrom which the request originated is trusted or not. This methodis called after the isTargetInterceptor method. The implementationcode must authenticate the proxy server. The authentication mechanismis proxy-server-specific. For example, in the WebSphere-providedimplementation for the WebSeal server, this method retrieves thebasic-authentication information from the HTTP header and validatesthe information against the user registry used by WebSphere ApplicationServer. If the credentials are invalid, the code throws theWebTrustAssociationException exception, indicating that the proxyserver is not trusted and the request is to be denied.

The getAuthenticatedUsername method is called after trust hasbeen established between the proxy server and WebSphere ApplicationServer. WebSphere Application Server has accepted the proxy server'sauthentication of the request and must now authorize the request.To authorize the request, the name of the original requestor must be subjectedto an authorization policy to determine if the requestorhas the necessary privilege. The implementation code for thismethod must extract the user name from the HTTP request headerand determine if that user is entitled to the requested resource.For example, in the WebSphere-provided implementation for theWebSeal server, the method looks for an *iv-user*attribute in the HTTP request header and extracts the user IDassociated with it for authorization.

After the interceptor class has been created, WebSphere ApplicationServer must be configured to use it by setting properties in thetrustedservers.properties file. This procedure is described for the WebSealinterceptor in Configuring trustassociation between WebSphere and WebSeal, and the proceduredescribed there varies as follows:

- Establish a name for your proxy to use in the WebSphere Application Server configuration properties. Use this name when you set the property `com.ibm.websphere.security.trustassociation.types`. For example, if you call your proxy myProxy, then set the property as follows:
  `com.ibm.websphere.security.trustassociation.types=myproxy`
- Based on the name you specified as the type of the proxy, WebSphere Application Server looks for a property that names the implementation class. Set the value of this property to the name of your implementation class. The implementation class must be locatable from the information on the class path.
  The name of the property is based on the name you assigned to your proxy according to this pattern:
  `com.ibm.websphere.trustassociation.<proxyname>.interceptor`
  For example, for the proxy called myProxy, the property name is `com.ibm.websphere.trustassociation.myproxy.interceptor`, and for the proxy type webseal36, the property name is is `com.ibm.websphere.trustassociation.webseal36.interceptor`.

Making your custom interceptor configurable

To allow configuration of your custom interceptor by reading aconfiguration file, you can subclass the WebSphere-providedclass com.ibm.websphere.security.WebSphereBaseTrustAssociationInterceptorand provide implementations of the following methods:

- `abstract public int init(String propsfile);`

- `abstract public void cleanup();`

The init method reads the configuration file specified for theinterceptor. The configuration file is specified in the trustedservers.properties file by using a property, the name of which is determined by thispattern:
`com.ibm.websphere.trustassociation.<proxyname>.config`
For example, for the proxy called myProxy, the property name is`com.ibm.websphere.trustassociation.myproxy.config`, and forthe proxy type webseal36, the property name is`com.ibm.websphere.trustassociation.webseal36.config`.The value of the property is the name of the configuration file for theinterceptor.

The cleanup method does any necessary termination work for the interceptor.

5.7: The Secure Association Service (SAS)

When global security is enabled in WebSphere Application Server, allrequests from clients to Enterprise JavaBeans are sent as RMI/IIOPmessages via the Object Request Broker (ORB) to the server that hoststhe enterprise beans. As part of every such request and response, theORB invokes the Secure Association Service (SAS) on the client and the serversides. On the client side, SAS intercepts requests before they are sent,obtains the client's security credentials, attaches the credentialsto the request as part of the security context, and sends the request.On the server side, SAS intercepts the incoming request, extracts thesecurity context from the message, authenticates the client's credentials, and passes the request to the enterprise bean container, wherethe request is authorized. The response is also routed throughthe SAS interceptors.

This article discusses the work performed by the Secure AssociationService and describes the properties available to configure its behavior.

The business methods in the client do not need to be written tohandle security. Security policies are defined during the deploymentphase, and WebSphere Application Server automatically enforces thedefined security policy, which specifies authorization requirements,before invoking the requested methods. The only thing required of theuser of a client program is authentication information. In some cases,the client program uses the CORBA security interfaces to establishthe proper credentials programmatically, before methods are invoked.In applications that do not establish credentials programmatically,SAS automatically prompts the user to collect the necessary information.The information collected is determined by the settings configuredfor the `com.ibm.CORBA.loginSource` property. For example,if the value of this property is specified as `prompt`, SASprompts the user for a user ID and password combination.If the user does not enter the information within a specified periodof time, determined by the value of the `com.ibm.CORBA.loginTimeout`property, SAS removes the login prompt and the request is handledwith no security. If the requested method is protected, therequest will fail because the user does not have the necessarypermission. If a method allows everyone, authenticated or not,access, the request can succeed.

5.7.1: SAS on the client side

When an enterprise-bean client, for example, a Java client, a servlet,or another enterprise bean, invokes a remote method, SAS interceptors arecalled to do the following work on the client side:

1. Establish an SSL connection
2. Establish a secure association between the client and the server
3. Send the request to the server

The following sections describe these steps in detail.

Establishing an SSL connection

Establishing an SSL connection requiresinformation from both the client and the server prior. The clientobtains some of this information from the client-side property file,sas.client.props. Some of the information must come from the server,which stores the information with the naming service. To contact aserver, the client retrieves information about the server from thenaming service. The returned information includes an interoperableobject reference (IOR), which the client uses to determine the type ofconnection expected by the server. If global security is enabledwithin WebSphere Application Server, servers insert a structure ofsecurity information, called a *security tag* into their IORsbefore registering the IORs with the naming service.

Theinformation from the security tag in the IOR and from thesas.client.props file is sufficient for creating an SSL connection.If the necessary information for an SSL connection is not present, aTCP/IP connection is created instead. For example, if the client doesnot find a security tag in a server's IOR, an SSL connection cannot becreated. If the target method is secured, the request must come in ona secure connection. Requests coming in on a TCP/IP connection alwaysfail for a lack of permission provided the method being invoked isprotected; no credentials are created for a TCP/IP connection. Atypical error message that indicates this condition is:

```
authorization failed for / while invoking method A
```

If global security is enabled, RMI/IIOP connections are typicallymade using SSL. There are a few exceptions, for which TCP/IPconnections are automatically made. These exceptions includename-server lookups, is_a queries, and a few other specialmethods. SSL connections are always the default for business methods.

The pure Java client or server acting as a client (that is, bymaking an outgoing connection to another server) gets some of theinformation it needs from the object's IOR from the server. Additionalinformation is obtained from the client properties file.

For a pure Java client (one that executes in a separate processfrom the server), the properties file used is the one specified on thecom.ibm.CORBA.ConfigURL property on the Java command line. This isusually the sas.client.props file.

For a server acting as a client, the property file used is thesas.server.props file on the server system. Some of the informationin the sas.server.props file can only be changed by using theadministrative console. Other parts of the sas.server.props file canbe changed using a text editor.

Most of the SSL and login configuration is done by using theSecurity Center in the administrative console and written into theWebSphere Application Server repository. After the administrativeserver restarts, the configuration information is migrated from therepository to the sas.server.props.future file. It is then mergedinto the sas.server.props file, which is used when the adminstrativeserver restarts.

The property file for an applicationis specified as a Java property on the command line when the applicationis started. The property, `com.ibm.CORBA.ConfigURL`, requires avalid URL as a value. For example, the URL for the sas.client.props file,assuming a default installation, is specified as follows:

- For Windows NT systems:

  ```
  com.ibm.CORBA.ConfigURL=file:/c:/WebSphere/AppServer/properties/sas.client.props
  ```

- For UNIX systems:

  ```
  com.ibm.CORBA.ConfigURL=file:///usr/WebSphere/AppServer/properties/sas.client.props
  ```

You can verify the URL syntax by following the URL with a browseron the system where the file resides. If the browser can read thefile, the URL is valid. The com.ibm.CORBA.ConfigURL property istypically specified on the java command line of the client program byusing the -D option in front of the property.

The information required before SAS can make a secure connection isshown below.

Information obtained from the server's IOR

This section describes the information retrieved on the client sidefrom the server's IOR and lists possible server-side sources for thatinformation. For example, some of the information in the IOR comesfrom server-side properties.

- **Server TCP/IP address**: This is determined by the TCP/IP configuration.
- **Server TCP/IP port**: This is usually assigned dynamically, but it can be explicitly set by using the server-side property in the sas.server.props file `com.ibm.CORBA.ListenerPort`.
- **Server SSL port**: This is usually assigned dynamically, but it can be explicitly set by using the server-side property in the sas.server.props file `com.ibm.CORBA.SSLPort`.
- **Server security name**: This is configured using the Administrator's Console through the Security Center. It contains the realm and user ID of the target server. The realm typically describes the name of the authentication server. The format of the value varies with the authentication mechanism:
  - For Local OS:

    `DOMAIN/server_id`

    The *DOMAIN* attribute can be either a Machine Name or Domain Name depending upon whether the WebSphere server is configured on a domain (if your operating system supports the domain concept).
  - For Lightweight Third-Party Authentication (LTPA):

    `LDAP HOST AND PORT/server_id`

    The *server_id* must be a valid *user* in the LDAP registry. The LDAP administrative ID is not supported for use as the WebSphere Server Security ID. If you want to specify a user called "cn=root", you can add a valid LDAP user record where the UID has cn=root specified to make it searchable.
- **Quality of protection (QOP) required**: This is set by using the server-side property `com.ibm.CORBA.standardClaimQOPModels`. The value of this property determines the quality of the SSL connection required by the server. If a client attempts to connect at a value lower, it will automatically be bumped up to this value. However, if the client tries to make a connection at a higher quality of protection, the connection should be opened at the higher value. Valid values are:
  - `high`: 128-bit encryption and digital signing
  - `medium`: 40-bit encryption and digital signing
  - `low`: No encryption or digital signing

Information obtained from the client's properties

This section describes the information retrieved on the client sidefrom the client's properties files.

- **Quality of protection (QOP) offered**: This is set by using the client-side property `com.ibm.CORBA.standardPerformQOPModels`. The value indicates what the client expects to do in creating an SSL connection; however, the server's quality-of-protection value can require the client to exceed its expected level. Valid values are:
  - `high`: 128-bit encryption and digital signing
  - `medium`: 40-bit encryption and digital signing
  - `low`: No encryption or digital signing
- **Login information**: This is information needed to authenticate the user. It is set by using the following client-side properties:
  - `com.ibm.CORBA.loginSource`: This determines the source of the authentication information. Valid values include:
    - `prompt`: A graphical panel is presented for the user for collecting the user ID and password. Pure Java clients must call the JDK API `System.exit(0)` at the end of the program in order to properly end the Java process. This is because the JDK starts a backward AWT thread that is not killed when the login prompt disappears. If you choose not to use a `System.exit(0)` call, pressing Ctrl-C ends the process.
    - `stdin`: The user is prompted for user ID and password by using a non-graphical console prompt. Currently only supported by a pure Java client.
    - `properties`: The user ID and password are retrieved from the following two properties:
      `com.ibm.CORBA.loginUserid`
      `com.ibm.CORBA.loginPassword`

      If you are using a client-side property file to log in (for instance, `com.ibm.CORBA.loginSource=properties`), you must specify the *realm* where you are trying to

log in to. There are two ways to do this:

- Set the `com.ibm.CORBA.principalName` property in that file to *`realm/loginUserid`*, where the *loginUserid* is the same as the value of the `com.ibm.CORBA.loginUserid` property and the *realm* matches the realm specified for the server localos machine name or domain name depending on the type of registry used. Note that the realm name is case sensitive. For example:
  `com.ibm.CORBA.loginUserid=userid`
  `com.ibm.CORBA.principalName=REALM/userid`
- Specify the *realm* on the same line as *loginUserid*. For example:
  `com.ibm.CORBA.loginUserid=REALM/userid`

- `key file`: The user ID specified by using the property `com.ibm.CORBA.loginUserid` and the realm name retrieved from the IOR are used to extract a user ID and password for authentication from a key file. The name of the key file to use is specified by setting the `com.ibm.CORBA.keyFileName` property.

- `com.ibm.CORBA.authenticationTarget`: This value determines the authentication method used to establish credentials. The valid values are:
  - `basicauth`
  - `localos`
  - `ltpa`

  The only supported value for a pure Java client is `basicauth`. A server actingas a client performs a login by properties. This creates `basicauth` credentials, which are then authenticated by the target server. On the server side, `localos` and `ltpa` can be specified; the value you select determines the type of registry against which `basicauth` credentials are verified.

- **Client SSL Configuration Properties**: See 5.7.3: ORB SSL Configuration

This information is used by SAS to construct the SSL connectionto the server. During this process, the client uses the publickey in the key store file to secure messages.

WebSphere Application Server provides several dummy keyring filesfor use in test and development environments. These keyring filesshould *not* be used in a production environment where messageprotection is desired. The certificate in this keyring filecan be used to do valid encryption, but the private key neededfor decrypting the messages is readily available.

During the SSL handshake between the client and server, thequality-of-protection level for the connection is determined byevaluating the client and server settings; the result is called the*coalesced QOP*. If the server setting is higher than the clientsetting, the server setting is used for both. The server setting isthe minimum acceptable level for the connection. If the client settingis higher but the server supports the higher level, then the clientsetting is used. If the server does not support the higherlevel offered by the client, the client uses the server setting.

The coalesced QOP value is used to determine the cipher suite to usewhen creating the SSL connection. The value determines the characteristicsof the SSL connection as follows:

- If the coalesced QOP is the `high` value, the messages are encrypted with 128-bit algorithms and digitally signed.
- If the coalesced QOP is the `medium` value, the messages are encyrpted with 40-bit algorithms and digitally signed.
- If the coalesced QOP is the `low` value, only digital signing occurs.

In cases where client authentication is required but the logininformation is not specified, the message is sent over an insecureTCP/IP connection. Ensure that methods are protected usingauthorization if you do not want unauthorized users to access them.When a TCP/IP connection is used to access a protected method, anauthorization failure occurs.

Establishing a secure association between the client and server

Once a connection is created at the server, SAS requires that a secureassociation between the client and server be established. This entailsauthenticating the client on the server side and establishing a SAS securitysession on both the client and server sides. Most problems that occur with authentication will happen during this process. This is where the serverauthenticates the client and returns success or failure. In many cases wherea failure occurs, you can expect to receive a NO_PERMISSION exception. To getmore information from the exception, use the getMessage() method to get a textdescription about the failure.

Sending the request to the server

After the SSL connection is created and a secure association isestablished, the client's request is sent to the server.

Receiving a response from the server

Once the server processes the request it sends a response back to the client. The SAS client processes the responseto determine if it was successful or not. If not successful, it will throw an exception to the business client to handle.Some of the exceptions you can expect to see are:

The exception is usually one of the following:

- org.omg.CORBA.NO_PERMISSIONTypically received because the userid and password entered on the client failed to authenticate. This could be due to an incorrectuserid/password or an internal reason such as the user registry being unavailable.
- org.omg.CORBA.COMM_FAILURETypically received when a server is not listening on the host and port specified in the IOR of the business object. Forexample, if an application server has been stopped which was sharing a particular resource, access to that resource will return a COMM_FAILURE.
- org.omg.CORBA.INTERNALTypically received when the SAS code reaches a path that was unexpected or a message is out of sequence. This can happenunexpectedly and SAS tracing may be required.

5.7.2: SAS on the server side

When an RMI/IIOP request arrives at a server, SAS intercepts therequest and performs the necessary security tasks before the businessmethod is invoked on the server. After the method is invoked, a response is sent back to the client.

Configuring the Server

Configuring a server for security starts at the administrativeconsole in the Security Center. The properties specified there arepropagated into the WebSphere Application Server repository and theneventually to the sas.server.props file for use by the SAS runtime.Some of the properties in the sas.server.props file are from theSecurity Center configuration and some are defaults which are editablein the file. The sas.server.props file documents which properties canbe changed without getting overwritten and which will get overwrittenby the information in the repository. See 5.7.5: SAS properties reference for more infoabout these properties.

Authenticating the user

When a server first receives a request, a user must beauthenticated and authorized before the method can be invoked. Partof SAS's responsibility is to authenticate the user to the userregistry to validate that they are who they say. The SAS programming model has APIs forauthenticating users on both the client and server sides. Currently,the only client authentication supported is Basic Auth (i.e.,authenticating a userid and password). SSL client authentication isplanned for a future release.

Invoking the method

Once SAS authenticates the user, a credential is created withinformation about the user. This credential is associated with thethread of execution and the method is invoked in the container afterbeing authorized.

Sending a response back to the client

After the method is invoked, a response is sent back to the client.

Credential forwarding - support for multiple nodes

Local OS credentials are only supported on the same node they werecreated. Therefore, when using the Local OS registry, only a singlenode configuration is supported. If you need a multi-nodeconfiguration, LTPA is the only supported option as the credentialscan be validated with trust on another node (that is, a differentphysical machine).

Credential expiration for LTPA credentials

When using the LTPA authentication mechanism, authenticatedcredentials have a configurable expiration period. When users make arequest to a server, a credential is created on the server side andassociated with the user's SAS security session on the server side.After the initial user request is made to a server, the SAS securitysession is stateful (meaning it will remember the state of the user).Every subsequent user request will use the same SAS security sessionand the same user credentials. Make sure that you set a high enoughvalue for the LTPA credential expiration time to keep usercredentials from expiring after a series of requests.

5.7.3: ORB SSL Configuration

The SSL implementation used by the application server is the IBMJSSE (Java Secure Sockets Extension). Configuring JSSE is verysimilar to configuring most other SSL implementations (for example, GSKit);however, a few differences are worth noting.

JSSE allows both signer and personal certificates to be stored in aSSL key file, but it also allows a separate file to be specifiedcalled a trust file. A trust file can contain only signercertificates. Therefore, you could put all of your personalcertificates in an SSL key file and your signer certificates in atrust file. This may be desirable, for example, if an inexpensivehardware device used as the key file which has only enough memory tohold a single personal certificate. All of the signer certificatesare then held in a trust file on disk.

JSSE does not recognize the proprietary SSL key file format that is used by the plug-in (i.e. .kdb files); instead, it recognizesstandard file formats such as JKS (Java Key Store). As such, SSL keyfiles cannot be shared between the plug-in and application server, anda different implementation of the key management utility (IKeyMan)must be used in order to manage application server key and trustfiles. IKeyMan can be started on Windows systems from the WebSphere **Start**menu.

Configuring SSL through an SSL Settings panel

The administrative model in WebSphere allows the SSL settings forsuch of the WebSphere components to be both centrally or individuallymanaged. SSL settings are centrally managed through the default SSLSettings panel. Furthermore, any of the default settings can beoverridden for an individual component by using the HTTPS, ORB, and LDAPS SSL settings panels.

The default SSL settings can be used to configure the variouscomponents using SSL. ORB SSL settings can be specified, inaddition, to override the default SSL settings specifically for theORB.

To configure server SSL by using the Default SSL Configurationpanel, do the following:

1. Start the administrative console.
2. Select **Security Center**.
3. Select **Default SSL Configuration**.
4. Modify the following SSL properties:
   - Key file name: c:\WebSphere\AppServer\etc\DummyServerKeyFile.jks
   - Key file password: WebAS
   - Confirm password: WebAS
   - Key file format: JKS
   - Trust file name:c:\WebSphere\AppServer\etc\DummyServerTrustFile.jks
   - Trust file password: WebAS
   - Confirm Password: WebAS
   - Security Level: high (128 bit encryption)

   If you choose to use the same file for key and trust,you can specify the same file for both properties:
   - Key file name: c:\WebSphere\AppServer\etc\DummyServerKeyFile.jks
   - Trust file name: c:\WebSphere\AppServer\etc\DummyServerKeyFile.jks

   If you only specify a Key file name, the Trust file name automatically is set to the same name as the Key file name.

Next, modify the following lines in the sas.client.props file toconfigure client SSL:

- com.ibm.ssl.keyStore=DummyClientKeyFile.jks
- com.ibm.ssl.keyStorePassword=WebAS
- com.ibm.ssl.keyStoreType=JKS
- com.ibm.ssl.trustStore=DummyClientTrustFile.jks
- com.ibm.ssl.trustStorePassword=WebAS
- com.ibm.ssl.trustStoreType=JKS
- com.ibm.ssl.protocol=SSLv3
- com.ibm.CORBA.standardPerformQOPModels=high (128 bit encryption)

The ORB uses these settings in the following way.

Key store file name

The path of the SSL key store file. This file is typically usedto store personal certificates, including private keys. DefaultClient and server key store files (DummyServerKeyFile.jks andDummyClientKeyFile.jks) are automatically configured and are locatedin the product_installation_root/etcdirectory. These files are included to simplify test and developmentenvironments. For production environments, generate new SSL key storefiles using iKeyMan (located in the product_installation_root/bin directory). Aself-signed certificate can be generated or a CA can be used to creates personal certificate trusted by most clients.

On the server, the SSL configuration isconfigured in the administrative console through the Security Center.For pure Java clients, the key file name is configured in thesas.client.props file through the property`com.ibm.ssl.keyStore`.

Key store password

The password for the SSL key store file.

On the server, the key file password is configured in theadministrative console through the Security Center. On a pureJava client, the key file password is configured in the sas.client.propsfile through the property `com.ibm.ssl.keyStorePassword`.

The passwordis automatically encoded whenever the server starts. To change thepassword on the server side, re-enter it in the Security Center. Tochange the password on the client side (in the sas.client.props file),overwrite the encoded password. The next time the client starts, thenew password is encoded.

Key file format

The format of the SSL key store file. Possible values are**JKS** (the default), **PKCS12**, and **JCEK**. Thisproperty that controls the key file format for both the client andserver is `com.ibm.ssl.keyStoreType`.

Trust file name

The path of the SSL trust file. The trust file is typically usedto store signer certificates, which specify whether the signer of theserver's certificate is trusted. The trust file helps to manage whichsigners an organization allows connections to. This enables clientsand servers to store their personal certificates in the key store fileand all of their signers in the trust store file.

On the server, the trust file name is configured in theadministrative console through the Security Center. On a pure javaclient, the trust file name is configured in the sas.client.props filethrough the property `com.ibm.ssl.trustStore`.

Trust file password

The password for the SSL trust store file. On the server, thetrust file password is configured in the administrative consolethrough the Security Center. On a pure client, the trust filepassword is configured in the sas.client.props file through the property`com.ibm.ssl.trustStorePassword`.

The passwordis automatically encoded whenever the server starts. To change thepassword on the server side, retype it in the Security Center. Tochange the password on the client side (in the sas.client.props file),overwrite the encoded password. The next time the client starts, thenew password is encoded.

Client Authentication

The WebSphere Application Server ORB does not currently supportSSL client authentication (that is, SSL mutual authentication) usingdigital certificates. However, it does support basicauth clientauthentication, where the user is prompted for a userid and password.Protected methods require basicauth credentials from the user beforethey can be invoked at the server. These credentials are sent to theserver and are authenticated using the LocalOS user registry.

Quality of Protection (Security Level)

Two properties must be configured for SSL to determine thestrength of the connection.

- For clients (including servers acting as clients), specify the security level or quality of protection by setting the following property in the sas.client.props file:

  `com.ibm.CORBA.standardPerformQOPModels=level`

  where *level* can be `low`, `medium` or `high` (the default).
- For servers, specify the security level or quality of protection by setting the following property in the sas.server.props file:

  `com.ibm.CORBA.standardClaimQOPModels=level`

  where *level* can be `low`, `medium` or `high` (the default). This value is added to the IOR to let the client know what quality of protection is required.

RegardIless of the values specified on standardPerformQOPModels,clients always make a connection at least as strong as the valuespecified on the server for standardClaimQOPModels. If the clientstandardPerformQOPModels is stronger than the serverstandardClaimQOPModels, the higher value is honored.

If the security level is `high` and either a server orclient SSL configuration is enabled, the selected cipher suites are asfollows. These suites are 128 bit ciphers.

```
SSL_RSA_WITH_RC4_128_MD5SSL_RSA_WITH_RC4_128_SHASSL_RSA_WITH_DES_CBC_SHASSL
RSA_WITH_3DES_EDE_CBC_SHASSL_DHE_RSA_WITH_DES_CBC_SHASSL_DHE_RSA_WITH_3DES_EDE_CBC_SHASSL_DHE_DSS_WITH_DES_CBC_SHASSL_DHE_DSS_WITH_3DES_EDE_CBC_SHASSL
```

If the security level is `medium` and either a server orclient SSL configuration is enabled, the selected cipher suitesare as follows. These suites are 40 bit ciphers.

```
SSL_RSA_EXPORT_WITH_RC4_40_MD5SSL_RSA_EXPORT_WITH_DES40_CBC_SHASSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHASSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
```

If the security level is `low`and a server SSLconfiguration is enabled, the selected cipher suites are as follows.These suites do not encrypt messages, but do perform integrity checksto ensure that the message is not modified while being sent over thewire.

```
SSL_RSA_WITH_NULL_MD5SSL_RSA_WITH_NULL_SHASSL_DH_anon_WITH_RC4_128_MD5SSL_DH_anon_WITH_DES_CBC_SHASSL_DH_anon_WITH_3DES_EDE_CBC_SHASSL_DH_anon_EXPORT_WITH_RC4_40_MD5SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA
```

If the security level is `low`and a client SSLconfiguration is enabled, the selected cipher suites are asfollows. These seuites do not encrypt messages, but do performintegrity checks to ensure that the message is not modified whilebeing sent over the wire.

```
SSL_RSA_WITH_NULL_MD5SSL_RSA_WITH_NULL_SHA
```

Enable Crypto Token Support

A *crypto token* is a hardware or software device with abuilt-in key store implementation. If this option is selected, thevalues on the **Crypto Token** panel are used. To display thispanel, click on **Crypto Token** under **Advanced Settings** inthe administrative console Security Center. See the documentation forthe crypto token device for the values for the following fields.

- **Token Type.** The type of built-in key store that is implemented in the crypto token. The default is **PKCS#12**.
- **Library File.** The DLL or shared object that implements the interface to the crypto token device.
- **Password.** The password for the crypto token device.

The supported crypto devices include the following:

- Devices that are usable by SSL clients:
  - IBM Security Kit Smartcard
  - GemPlus Smartcards
  - Rainbow iKey 1000/2000 (USB smartcard device)
- Devices that are usable by SSL clients and servers:
  - IBM 4758-23
  - nCipher nForce
  - Rainbow Cryptonet8i

Dynamic Properties

Dynamic properties allow configuration of lessfrequently-used JSSE properties. The name, possible values, defaultvalue, and a brief description of each property follows.

com.ibm.ssl.protocol

The SSL protocol to be used, including the version. The possible values and the versions of SSL that each supports are asfollows:

- **SSL** (the default). Supports SSLv20, SSLv30, and SSLv31.
- **SSLv2**. Supports SSLv20.
- **SSLv3**. Supports SSLv30.
- **TLS**. Supports SSLv31.
- **TLSv1**. Supports SSLv31.
- **SSL_TLS**. Supports SSLv20, SSLv30 and SSLv31.

The default value of **SSL** is compatible with earlierWebSphere Application Server releases.

TLSdoes not interoperate with the previous WebSphere ApplicationServer SSL implementation. The previous implementation only supportsSSLv30 handshakes and cannot interoperate with aniplementation that supports only SSLv31.

com.ibm.ssl.keyStoreProvider

The name of the key store provider. The default value is**IBMJCE**. Possible values are any of the security providerslisted in your java.security file. Change the default value of thisproperty only if you intend to use another SSL implementation.

com.ibm.ssl.keyManager

The name of the key management algorithm. The default value is**IbmX509**. Possible values are key management algorithmsimplemented by one of the providers listed in your java.security file.Change the value of this property only if you intend to use anotherSSL implementation.

com.ibm.ssl.trustStoreProvider

The name of the trust store provider. The default value is**IBMJCE**. Possible values are any valid trust store provider names.Change the value of this property only if you intend to use another JSSEimplementation.

com.ibm.ssl.trustManager

The name of the trust management algorithm. The default value is**IbmX509**. Possible values are trust management algorithms implementedby one of the providers listed in your java.security file. Change thevalue of this property onlyif you intend to use another SSL implementation.

com.ibm.ssl.trustStoreType

The format of the trust store. Possible values are **JKS**,**PKCS12**, and **JCEK**.

com.ibm.ssl.enabledCipherSuites

A space-separated list of cipher suite names. If this property isset, it overrides the com.ibm.CORBA.standardPerformQOPModels propertysetting. Possible cipher suite names are:

```
SSL_RSA_WITH_RC4_128_MD5SSL_RSA_WITH_RC4_128_SHASSL_RSA_WITH_DES_CBC_SHASSL_RSA_WITH_3DES_EDE_CBC_SHASSL_DHE_RSA_WITH_DES_CBC_SHASSL_DHE_RSA_WITH_3DES_EDE_CBC_SHASSL_DHE_DSS_WITH_DES_CBC_SHASSL_DHE_DSS_WITH_3DES_EDE_CBC_SHASSL_RSA_EXPORT_WITH_RC4_40_MD5SSL_RSA_EXPORT_WITH_DES40_CBC_SHASSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHASSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHASSL_RSA_WITH_NULL_MD5SSL_RSA_WITH_NULL_SHASSL_DH_anon_WITH_RC4_128_MD5SSL_DH_anon_WITH_DES_CBC_SHASSL_DH_anon_WITH_3DES_EDE_CBC_SHASSL_DH_anon_EXPORT_WITH_RC4_40_MD5SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA
```

SSL interoperability issues

Keep the following interoperability issues in mind when configuringSSL:

- The client and the server must run at the same SSLprotocol. Currently, WebSphere Advanced Edition 3.5 and WebSphereEnterprise Edition 3.5 both use SSL version 3.0. Make sure that WebSphereAdvanced Edition 4.0 is running with com.ibm.ssl.protocol=SSLv3 orcom.ibm.ssl.protocol=SSL (since TLS or SSL 3.1 is not supported in3.5) when your target supports only SSL v3.0.
- The client and the server must communicate with a compatibleCipher suite at the same encryption level. In some cases, a server cannotly support 40 bit encryption. When 40 bit encryption can becosidered high on one server, 128 bit encryption can be considered high on another server. If a server's high is 40 bit encryption andthe server publishes high in the objects IOR, the client will tryto connect at the high level. If the client supports 128 bitencryption for high, the connection fails due to the cipher suitesmismatch. In this case, it is necessary to set thecom.ibm.ssl.enabledCipherSuites property with 40 bit ciphers onthe client side. This overrides the value specified by the serverin the object's IOR and manually matches the ciphers on the server side.

  Often,the best way to resolve SSL cipher incompatibilities is toadd all of the ciphers above to the com.ibm.ssl.enabledCipherSuitesproperty. This allows all possible combinations to be consideredduring the SSL handshake. The other property to consider changing inthe com.ibm.ssl.protocol. First try the **TLS_SSL** protocol for thehighest version support, then try the **SSL** protocol to step down toSSLv3. The **SSL** selection interoperates most often with other SSLservers.
- Remember to add the target server's public key or the root CAcertificate to WebSphere Advanced Edition 4.0's TrustFile.

Debugging SSL Handshake Problems

Gathering ORB trace information is the first troubleshooting step.The trace information shows you the exception at which JSSE returns tothe createSocket implementation.

To enable ORB trace at the administrative server, add thefollowing lines to the admin.config file:

```
com.ibm.CORBA.Debug=true
com.ibm.CORBA.CommTrace=true
com.ibm.ejs.sm.adminServer.traceString="ORBRas=all=enabled"
com.ibm.ejs.sm.adminServer.traceOutput=c:\your_trace_directory\adminorb.log
```

To enable ORB trace at the application server, do thefollowing:

1. Enter the following in the **JVM Arguments** field of theapplication server:

   `com.ibm.CORBA.Debug=true com.ibm.CORBA.CommTrace=true`
2. Select **Service** > **Trace Service Settings** and enter thefollowing:
   - **Trace specification string**: ORBRas=all=enabled
   - **Trace output file**: *C:\your_trace_directory*appsrb.log

Next, enable debugging support for JSSE by doing the following:

- Ensure the JSSE debug module, ibmjsse-debug.jar, is located in thejava/jre/lib/ext directory.
- Enable debugging by adding the following property to the server'sJava command line: `-Djavax.net.debug=true`

Debugging the server side is the best way to find the cause ofa handshake failure. Typically, all you can see on the clientside is the socket closing due to the server side refusing theconnection.

5.7.4: Tracing SAS

The Secure Association Service (SAS) uses a messaging model, so for everySAS request, there is a response. In a distributed environment, wherea client can call a server, which can then act as a client and callanother server, solving security-related problems often requirestracing multiple servers simultaneously.

Frequently, these servers reside on the same machine; the interactionbetween an administrative server and an application server is often whereproblems arise. The administrative server includes a component calledthe security server, which performs authentication work, andmessages are frequently exchanged between the application serverand the administrative server during authentication. Furthermore,the administrative server stores authorization information in arepository, so authorization requests result in additional trafficbetween the administrative server and the application server.

Collecting information about SAS messages is often crucial fordebugging security problems, and SAS provides a set of propertiesthat govern the collection of SAS messages, including the typesof messages and the destination of the collected messages. These propertiesare set in the property file used by each server; this istypically the sas.server.props file.

The SAS message and trace logging facility captures informationabout the following different types of events:

- Activity: indicates that a specific event has occurred
- Error: indicates that a run-time problem has occurred and suggests a potential solution
- Exception: indicates that a run-time problem has occurred and prints a corresponding stack trace
- Trace: tracks the path through the code so that, when an error occurs, you can determine the events preceding it

This behavior is determined by the value of the`com.ibm.CORBA.securityTraceLevel` property.

The value of the `com.ibm.CORBA.securityDebug` propertyis used to determine whether the collected messages can be displayedon the standard output stream.

In addition, you can selectively send the messages for each type of eventto a file. For each type of event, you set an output-mode property.The output mode determines determines where the messages collected forthe event, for example, activity, are collected. You can use any ofthe following output modes:

- File: output goes to the destination set in the `com.ibm.CORBA.securityTraceOutput` property, and a new file is created after each server restart.
- Fileappend: output goes to the destination in the `com.ibm.CORBA.securityTraceOutput` property, and new output is appended after each server restart.
- Console: output is redirected to the standard output stream.
- Both: output is redirected to both the standard output stream and to the destination set in the `com.ibm.CORBA.securityTraceOutput` property, and a new file is created after each server restart.
- None: no output occurs.

The output mode is set for each type of trace event. Each ofthese properties can take any of the output modes as values:

- `com.ibm.CORBA.securityActivityOutputMode`
- `com.ibm.CORBA.securityErrorsOutputMode`
- `com.ibm.CORBA.securityExceptionsOutputMode`
- `com.ibm.CORBA.securityTraceOutputMode`

To send all trace messages to the standard output stream, use thefollowing settings:

```
com.ibm.CORBA.securityDebug=consolecom.ibm.CORBA.securityTraceLevel=intermediate
```

5.7.5: SAS properties reference

This following describes the properties used in the configuration filessas.client.properties and sas.server.properties. These files containlists of property-value pairs, using the syntax<*property*>=<*value*>.

The property names are case sensitive, but the values are not; thevalues are converted to lower case when the file is read.

In WebSphere Application Server version 4.0, some propertiesdo not appear in the sas.server.props file. Instead, theseproperties must be configured by using the administrative console.The entry for each property indicates how it can be modified.

Authentication properties

com.ibm.CORBA.authenticationTarget

Specifies the mechanism for authenticating principals.

valid values: basicauth, localos, ltpa

default value: basicauth

client/server usage: can be directly edited in the sas.client.props file; the server-side value must be set by using the Security Center within the administrative console

com.ibm.CORBA.loginUserid

Holds the name of an authorized user of the user registry, used when the `loginSource` property is specified as `properties`. The corresponding password is stored in the `loginPassword` property.

valid values: a user name in the registry

default value: no default value

client/server usage: can be directly edited in the sas.client.props file; the server-side value must be set by using the Security Center within the administrative console

com.ibm.CORBA.loginPassword

Holds the password for the user named in the `loginUserid` property, use when the `loginSource` property is specified as `properties`.

valid values: the password for the user named in the `loginUserid` property

default value: no default value

client/server usage: can be directly edited in the sas.client.props file; the server-side value must be set by using the Security Center within the administrative console

com.ibm.CORBA.principalName

Specifies the principal under which the WebSphere administrative server runs. The format is **REALM/userID**.

valid values: a realm name and a user name in the registry

default value: no default value

client/server usage: can be directly edited in the sas.client.props file; the server-side value must be set

by using the Security Center within the administrative console

com.ibm.CORBA.loginSource

Indicates the source for the user IDs and passwords.

valid values: prompt, properties, stdin, key file, none

- The value `stdin` is supported only in the sas.client.props file.
- The value `none` is typically used for applications that perform programmatic logins before they require credentials on a thread of execution.

default value: prompt

client/server usage: sas.client.props and sas.server.props

com.ibm.CORBA.loginTimeout

Specifies the length of time (in seconds) for which the login window is displayed to a user for entering login information (realm, user ID, password).

valid values: 0 to 600 (0 to 10 minutes)

default value: 300 (5 minutes)

client/server usage: sas.client.props and sas.server.props

SSL Properties

For more information on configuring SSL, see5.7.3: ORB SSL Configuration.

Miscellaneous properties

com.ibm.CORBA.securityEnabled

Indicates whether security is enabled or not.

valid values: false, no, true, yes

default value: true

client/server usage: can be directly edited in the sas.client.props file; the server-side value must be set by using the Security Center within the administrative console

com.ibm.CORBA.bootstrapRepositoryLocation

Holds the full path of the bootstrap repository file, which contains information about security properties needed during the boot process.

valid values: the absolute path to the repository file

default value: <*server_root*>/etc/secbootstrap

client/server usage: sas.server.props only

Trace and message properties

com.ibm.CORBA.securityDebug

Specifies whether debugging messages are displayed on the console or not.

valid values: console, false, no, true

default value: false

client/server usage: sas.client.props and sas.server.props

com.ibm.CORBA.securityTraceLevel

Determines the level of tracing provided.

valid values: none, basic, intermediate, advanced

- Trace level `basic` reports basic messages and is rarely used
- Trace level `intermediate` is typically used to troubleshoot long-run problems to minimize tracing
- Trace level `advanced` is used in most cases for troubleshooting

default value: none

client/server usage: sas.client.props and sas.server.props

com.ibm.CORBA.securityTraceOutput

Determine the output file for SAS when `file`, `fileappend`, or both are chosen for the output mode properties (`securityActivityOutputMode`, `securityErrorsOutputMode`, `securityExceptionsOutputMode`, or `securityTraceOutputMode`).

valid values: a valid path and file name in the file system.

default value: <*server.root*>/logs/sas.log

client/server usage: sas.client.props and sas.server.props

com.ibm.CORBA.securityActivityOutputMode

Determines where to direct activity messages.

valid values: none, file, fileappend, console, both

- `file`: output goes to the destination set in the `com.ibm.CORBA.securityTraceOutput` property and a new file is created after each server restart.
- `fileappend`: output goes to the destination in the `com.ibm.CORBA.securityTraceOutput` property and new output is appended after each server restart.
- `console`: output is redirected to the standard output stream.
- `both`: output is redirected to both the standard output stream and to the destination set in the `com.ibm.CORBA.securityTraceOutput` property, and a new file is created after each server restart.
- `none`: no output occurs.

default value: file

client/server usage: sas.client.props and sas.server.props

com.ibm.CORBA.securityErrorsOutputMode

Determines where to direct error messages.

valid values: none, file, fileappend, console, both
(The values work as described for the `securityActivityOutputMode` property.)

default value: both

client/server usage: sas.client.props and sas.server.props

com.ibm.CORBA.securityExceptionsOutputMode

Determines where to direct exception messages.

valid values: none, file, fileappend, console, both
(The values work as described for the `securityActivityOutputMode` property.)

default value: file

client/server usage: sas.client.props and sas.server.props

com.ibm.CORBA.securityTraceOutputMode

Determines where to direct trace messages. Client and server side.

valid values: none, file, fileappend, console, both
(The values work as described for the `securityActivityOutputMode` property.)

default value: file

client/server usage: sas.client.props and sas.server.props

5.7.6: Introduction to SAS programming

A fundamental concern within distributed systems in general is the protection of data and business assets available through the information system. Thisis no less true in distributed, object-oriented systems. Valuable informationexists in business objects. This information can be manipulated and accessedremotely and therefore must be protected from unauthorized use. TheSecurity Service in WebSphere Application Server helps to protect theseassets.

The Security Service is used primarily to prevent end users from accessinginformation and resources that they are not authorized to use. Although theseresources are predominantly distributed objects, they can also includeresources, neither object-oriented nor distributed, used by business objects.In many cases, WebSphere Application Server is used to wraplegacy resources, such as existing business applications and enterprise data.Such resources are often centralized resources, held in a physically secureenvironments or in environments with restricted access over controlledchannels.

A key objective of object-oriented programming and business re-engineeringis to provide for the abstraction of business resources that enables themto be used more readily in new applications. This abstraction frequentlyhas the effect of increasing access to those legacy resources, resourcesthat have been traditionally, either by intent or because ofthe limitations of technology, more restricted. Thus, the object-orientedapproach has the potential for undermining the protection that legacyresources require and have traditionally enjoyed.

The Security Service must, therefore, compensate for any protections thatcan be otherwise lost due to the increased accessibility of business objects ina distributed object system. The Security Service must not limit anybenefit an application programmer receives by using WebSphere ApplicationServer, except by preventing unauthorized access to resources.When security policies for a set of legacy resources have been established forproduction systems, the Security Service uses these policies to protectresources in the object-oriented system. It is not necessary to specifyexisting security policies a second time or to keep two sets of policiesin synchronization.

Object systems tend to introduce many more independent objects thanequivalent procedural systems, which tend to collect individualobjects into larger-grained artifacts like resources managers and databasetables. The presence of so many objects can introduce issuesrelated to administrative scalability. These issues present their ownsecurity exposures: when administration becomes overwhelming,administrators just stop administering, and objects remain unprotected.The Security Service guards against this risk by factoring securitypolicies across a server, forming an administrative boundary forcontrolling unauthorized access to both the objects that are containedwithin a server and the resources that are used by the server.WebSphere security provides support for the authentication of users,which prevents unauthenticated users from accessing secure servers. It also guarantees the identity associated with a request to a businessobject, so that object can determine if it should grant access.The Security Service also provides support for protecting message trafficbetween clients and servers and between servers acting as clients andother servers.

The role of the Secure Association Service (SAS)

Users and processes can be authenticated to the system. They canhave identities, which means that they can be distinguished and thattheir access to resources can be controlled. Any entity that can beidentified and authenticated in the system is referred to as a*principal*. A principal can be the user of a client programor it can be a server process. Other entities can also be principalsif they can be associated with identities and have mechanisms fordemonstrating their identities.

When a principal is authenticated, the Security Service createsa credential object for that principal. The credential representsan authenticated principal; credentials are created *only* afterthe principals are authenticated.

In a secure server, all activities occur on behalf of a specific principal,typically the identity associated with the user of the client. When aprincipal is authenticated at a client (a client principal), a credentialis created for that

client and associated with the thread of executionwithin the process. The credential is passed to the server when the clientissues any requests to the server, and the thread of execution in theserver is tagged with the credentials of the client principal thatoriginated the request. If the server issues any subsequent requests asa result of the original request, the client's credential is passed alongwith any requests that originate from the server.

The Security Service is able to efficiently and safely communicate thecredentials for the client principal by establishing a secure associationbetween the client and the server. Each client and server pair forms aunique association, even when the server acts as a client to another server.The secure association is also used to protect any message traffic betweenthe client and the server processes.

When to use SAS programming

SAS programming is useful when applications must login programmaticallyor manipulate the credentials on the thread of execution for the purposeof controlling the identity which is executing specificmethods. (Examples of these uses are illustrated in this material.)SAS programming can be combined with other WebSphere ApplicationServer programming techniques, including the use of security and standards-based models, like servlets, enterprise beans,Java ServerPages, HTTP programming, and many others.

The SAS programming interfaces are based on CORBA Security Servicespecification from the Object Management Group (OMG). For moredetails, visit the OMG Web site and obtain the CORBA Security Servicespecification.

5.7.6.1: Getting a reference to a Current object

The Current class contains an implementation ofthe CORBA SecurityLevel2 Current object. The classprovides access to security-level 2 function as defined in theObject Management Group (OMG) CORBA Security Service specification.

A Current object allows you to obtain or manipulate thecredentials that you want to use in your program. You can obtain aCurrent object in either the client or the server. However,you can only get a Current object if the Security Servicerun time has been installed and the ORB has been initialized.

To obtain a Current object, using following steps:

1. Obtain a reference to the com.ibm.CORBA.iiop.ORB object. You can obtain a reference to the com.ibm.CORBA.iiop.ORB object by invoking the com.ibm.ejs.oa.EJSORB.getORBInstance() method, which is static.
2. Create a reference to the org.omg.SecurityLevel2.Current object, and then use the ORB.resolve_initial_references method to get access to the security Current object. Pass the string `"SecurityCurren` to the resolve_initial_references method.

Code sample: obtaining a Current object

```
   ...   // Get the current ORB instance.   com.ibm.CORBA.iiop.ORB orb =
com.ibm.ejs.oa.EJSORB.getORBInstance();   // Get the security Current object.   if (orb != null)
org.omg.SecurityLevel2.Current securityCurrent =
(org.omg.SecurityLevel2.Current)orb.resolve_initial_references("SecurityCurrent");   if
(securityCurrent == null)      System.out.println("Security has not been initialized");   ...
```

5.7.6.2: Extracting credentials from a thread

You can use a credential associated with the thread of executionto examine and manipulate the identity of the principal thatissued the request, the identity of the server, or the identity used forany outgoing requests.

Retrieving a credential from a thread of execution requires twogeneral steps:

1. Obtain a reference to the security Current object.
2. Extract the desired credential.

The technique for extracting the desired credential varies withthe credential. Any thread of execution in a client or a servercan be associated with one of the following credentials:

Received credential

The *received credential* identifies the principal for whom a request is being performed. In the server, the received credential is the credential that arrived with the currently executing request. In the client, the received credential is the same as the client's own credentia; there is no incoming request carrying an external credential with it.

Invocation credential

The *invocation credential* is the credential that accompanies any requests made from this thread of execution. In the server, when delegation is enabled, the invocation credential is automatically set to the received credential. Otherwise, the invocation credential is the server's own credential.

Own credential

The *own credential* is also known as the default credential of the process. This credential identifies the principal associated with the process. In the server, this is the server principal; in the client, it is the client principal. Note that a server's own credential can become its invocation credential when delegation is disabled.

When extracting a credential from the thread of execution,you must decide which credential you want. Additionally, thesecurity run time must be installed, and the ORB must be initialized.

Extracting the received credential

To extract the received credential from a thread of execution, use the followingsteps:

1. Obtain a reference to the security Current object.
2. Call the SecurityCurrent.received_credentials method. This method returns an list of Credentials; the received credential is in the first position.
3. Obtain the received credential from the first position in the list.

```
   ...   // Get a reference to the security Current object.   ...         // Obtain the received
credentials.   org.omg.SecurityLevel2.Credentials[] recvdCreds =
securityCurrent.received_credentials();   // Retrieve the received credential from the first
position.   org.omg.SecurityLevel2.Credentials recvdCred = recvdCreds[0];   ...
```

Extracting the invocation credential

To extract the invocation credential from a thread of execution,use the following steps:

1. Obtain a reference to the security Current object.
2. To retrieve the invocation credential, call the Current.get_credentials method with the attribute `org.omg.Security.CredentialType.SecInvocationCredentials` as the argument. This method returns a Credentials object.

The only difference between extracting invocation credentials andextracting own credentials is the value of the argument passed tothe get_credentials method.

```
   ...   // Get a reference to the security Current object.   ...   // Obtain the invocation
credentials.   try   {      org.omg.SecurityLevel2.Credentials invCred =
securityCurrent.get_credentials(org.omg.Security.CredentialType.SecInvocationCredentials);   }
catch (Security::InvalidCredentialType e)   {      e.printStackTrace();   }   ...
```

Extracting the own credential

To extract the own credential from a thread of execution, use the followingsteps:

1. Obtain a reference to the security Current object.
2. To retrieve the own credential, call the Current.get_credentials method with the attribute `org.omg.Security.CredentialType.SecOwnCredentials` as the argument. This method returns a Credentials object.

The only difference between extracting invocation credentials andextracting own credentials is the value of the argument passed tothe get_credentials method.

```
   ...   // Get a reference to the security Current object.   ...   // Obtain the own credentials.
try   {      org.omg.SecurityLevel2.Credentials ownCred =
```

```
securityCurrent.get_credentials(org.omg.Security.CredentialType.SecOwnCredentials);   }   catch
(Security::InvalidCredentialType e)   {      e.printStackTrace();   }   ...
```

5.7.6.2.1: Manipulating credentials

A credential object is an object that implements theorg.omg.SecurityLevel2.Credentials interface. This interface supportsmany operations on credentials. A specific credential object containsidentifying information about a principal for a session; this informationincludes the security name of the principal, the principal's hostname,and more. The Credentials interface defines methods for the following:

- Copying a credential
- Retrieving the information in the credential
- Determining if the credential has expired

5.7.6.2.2: Client-side programmatic login

Client-side programmatic login allows the programmer to control whena user is prompted for the user ID and password used in constructingbasic-authentication credentials. Without programmatic login, WebSphereApplication Server security automatically prompts the user when thefirst method is invoked at a secured server. Clients that can use thistechnique are Java clients and servlets that access enterprise beans onother servers.

On the client side, the basic-authentication credentials are maintainedin the Current object on the client's thread of execution.

The LoginHelper class is a WebSphere-provided utility class thatprovides wrappers around CORBA security methods. It can be usedby pure Java clients that need the ability to programmaticallyauthenticate users but don't need to use the authentication data onthe client side. It provides the request_login method, which is usedby the Security Service to get login information from the client(or server) if the required credentials are not available.

A LoginHelper object can be used to obtain the user informationwith which to perform a login; that is, it can be used to collectthe information needed for a basic-authorization credential.It is typically implemented to present a login pop-up.An instance of the LoginHelper object can be created at any time. TheSecurity Service can provide different implementations of this objectfor different conditions, but the actual implementation class usedby the Security Service is directly coded into the service, to preventtampering.

The example code illustrates how to get a reference to a LoginHelperobject from a Current object, how to create a basic-authorizationcredential, and how to set the credential onto the Current objectfor propogation to a server or other access. For more informationon programmatic login, see 5.4: Using programmatic and custom login.

```
...// Get the security Current object....if (current != null){   // Get a handle to LoginHelper from
the Current object.   com.ibm.IExtendedSecurity._LoginHelper loginHelper = current.login_helper();
// Construct a basic-authorization credential for   // later authentication by the server.
org.omg.SecurityLevel2.Credentials credentials =        loginHelper.request_login(security_name,
realm_name,                                  password,                                        new
org.omg.SecurityLevel2.CredentialsHolder(),                                  new
org.omg.Security.OpaqueHolder());   // Set the credentials for outbound requests.
current.set_credentials(org.omg.Security.CredentialType.SecInvocationCredentials, credentials);
...}
```

5.7.6.2.3: Server-side programmatic login

Server-side programmatic login will authenticate the basic-authorization dataor credential token and create a credential authenticated againstthe local registry or LTPA registry. The basic-authorization credentialcan be sent from a client or created in the server. After authentication,the authenticated credential is maintained by the security session and isset onto the Current object each time a method request gets executed.The credentials remain available on the Current object as long as therequest is being executed on the server.

There are two ways to create the authenticated credential object:

- Map the basic-authentication credential to the local or LTPA registry by calling the com.ibm.IExtendedSecurity.CredentialsOperations.get_mapped_credentials method. This method maps the information in the basic-authentication credential to the specified registry. If authentication fails, the get_mapped_credentials method returns an empty credential. (There is also a get_mapped_creds method; it throws an exception if authentication fails.)
- Call the PrincipalAuthenticator.authenticate method, which takes the user ID and password as arguments.

The code example illustrates a server that creates a basic-authenticationcredential using the LoginHelper class and then creates an authenticatedcredential by calling the get_mapped_credentials method.

```
...// Get the security Current object....if (current != null){   // Get a handle to LoginHelper from
the Current object.   com.ibm.IExtendedSecurity._LoginHelper loginHelper = current.login_helper();
// Construct a basic-authorization credential for   // later authentication by the server.
org.omg.SecurityLevel2.Credentials credentials =       loginHelper.request_login(security_name,
realm_name,                                  password,                                        new
org.omg.SecurityLevel2.CredentialsHolder(),                                  new
org.omg.Security.OpaqueHolder());   // Set the credentials for outbound requests.
current.set_credentials(org.omg.Security.CredentialType.SecInvocationCredentials, credentials);
...   // Map the basic-authentication credentials to the registry.
org.omg.SecurityLevel2.Credentials mapcreds = null;   mapcreds =
((com.ibm.IExtendedSecurity.CredentialsOperations)creds).get_mapped_credentials(null, "", null);
// Check to see if authentication succeeded.   if (mapcreds = null)      System.out.println("Login
failed");}
```

If you prefer to catch an exception when authentication fails, use theget_mapped_creds method and catch the org.omg.Security.LoginFailedexception.

```
try{   // Map the basic-authentication credentials to the registry.
org.omg.SecurityLevel2.Credentials mapcreds = null;   mapcreds =
((com.ibm.IExtendedSecurity.CredentialsOperations)creds).get_mapped_creds(null, "", null);}catch
(org.omg.Security.LoginFailed e){   System.out.println("Login failed");}
```

5.7.7: Disabling security on specific application servers

In some circumstances, it is useful to allow unrestricted accessto resources managed by WebSphere Application Server, but it isoften less desirable to leave the administration of those resourcesunrestricted. This article describes how to unprotect the resourcesmanaged by an application server while protecting the resources ofthe WebSphere Application Server administrative server. This meansthat users of the administrative console are authenticated beforethey can modify the resources, but use of the resources requiresno authentication or authorization.

Resources must be unprotected on a node-by-node basis. If you havemultiple nodes and want only some to offer unprotected resources, you mustunprotect each node individually. Use this procedure only to createunprotected nodes.

How the procedure works

During initialization of the administrative server, the IOR for each enterprisebean hosted in an application server is registered with the name server.The IOR for each enterprise bean contains a security tag if any of thefollowing properties is set to the value `true`, which isthe default value:

- `com.ibm.CORBA.SSLTypeIClientAssociationEnabled`
- `com.ibm.CORBA.LTPAClientAssociationEnabled`
- `com.ibm.CORBA.DCEClientAssociationEnabled`

When the client reads the IOR, the presence of the securitytag indicates to the client that the server expects the clientto use a secure connection for sending messages. As a result,the client must obtain authentication information from theuser so the server can authenticate the user.

If the property is set to `false`, the IOR does not contain asecurity tag, and the client creates a TCP/IP connection to theserver. Messages sent over a TCP/IP connection are not secured.The application server receives the request on the TCP/IP portand handles the request.

Authorization of requests is completely disabled when the`SSLTypeIClientAssociationEnabled` is set to `false`.This tells the application server not to enable security on inboundrequests. This applies only when the application server uses a differentset of configuration properties than the administrative server does.The technique for disabling security on selected application serversis to provide them with a different properties file.

Setup Steps

1. Ensure that you have enabled global security and have restarted the administrative server at least once. This ensures that you have the correct security settings in the sas.server.props file. By default, all the components use this file; in this procedure, the administrative server and any secured application servers continue to use this server, but unsecured application servers use a different file.
2. Delete the sas.server.props.future file. If this file is present, when a server restarts, information in the sas.server.props.future file is copied into the sas.server.props file, effectively rewriting the sas.server.props file. Changes made during this procedure can be lost.
3. Make a copy of the sas.server.props file; in this example, the copy is called sas.appserver.props. The administrative server and the secured application servers continue to use the original sas.server.props file.
4. Edit the sas.server.props file and modify the settings as described.

   ⚠ You must make these changes carefully; incorrect settings can result in unwanted security behavior, and it is possible to create a state in which the administrative server cannot start if security is enabled. Also, once security is enabled, do not change any values other than the ones listed here unless you are sure of the consequences.

   - If the value of the `com.ibm.CORBA.authenticationTarget property` is `localos`, set the following properties:
     - **Client-association properties**

       ```
       com.ibm.CORBA.SSLTypeIClientAssociationEnabled=true
       com.ibm.CORBA.LocalOSClientAssociationEnabled=true
       com.ibm.CORBA.LTPAClientAssociationEnabled=false
       ```

     - **Server-association properties**

       ```
       com.ibm.CORBA.SSLTypeIServerAssociationEnabled=true
       com.ibm.CORBA.LocalOSServerAssociationEnabled=true
       com.ibm.CORBA.LTPAServerAssociationEnabled=false
       ```

   - If the value of the `com.ibm.CORBA.authenticationTarget property` is `ltpa`, set the following properties:
     - **Client-association properties**

       ```
       com.ibm.CORBA.SSLTypeIClientAssociationEnabled=true
       com.ibm.CORBA.LocalOSClientAssociationEnabled=false
       com.ibm.CORBA.LTPAClientAssociationEnabled=true
       ```

     - **Server-association properties**

       ```
       com.ibm.CORBA.SSLTypeIServerAssociationEnabled=true
       com.ibm.CORBA.LocalOSServerAssociationEnabled=false
       com.ibm.CORBA.LTPAServerAssociationEnabled=true
       ```

5. Edit the new sas.appserver.props file and modify the settings as described.

   ⚠ Do *not* change any other values in the file except those indicated. In particular, do not set the `securityEnabled` property to `false`; an unsecured application server must still be a secure client of the administrative server. Also, each time a principal or password in the sas.server.props file is changed, make the corresponding changes in this file.

   - If the value of the `com.ibm.CORBA.authenticationTarget property` is `localos`, set the following properties:
     - **Client-association properties**

       ```
       com.ibm.CORBA.SSLTypeIClientAssociationEnabled=false
       com.ibm.CORBA.LocalOSClientAssociationEnabled=false
       ```

```
com.ibm.CORBA.LTPAClientAssociationEnabled=false
com.ibm.CORBA.DCEClientAssociationEnabled=false
```

- **Server-association properties**

```
                    com.ibm.CORBA.SSLTypeIServerAssociationEnabled=true
com.ibm.CORBA.LocalOSServerAssociationEnabled=true
com.ibm.CORBA.LTPAServerAssociationEnabled=false
```

- If the value of the `com.ibm.CORBA.authenticationTarget property` is `ltpa`, set the following properties:
  - **Client-association properties**

```
                    com.ibm.CORBA.SSLTypeIClientAssociationEnabled=false
com.ibm.CORBA.LocalOSClientAssociationEnabled=false
com.ibm.CORBA.LTPAClientAssociationEnabled=false
com.ibm.CORBA.DCEClientAssociationEnabled=false
```

  - **Server-association properties**

```
                    com.ibm.CORBA.SSLTypeIServerAssociationEnabled=true
com.ibm.CORBA.LocalOSServerAssociationEnabled=false
com.ibm.CORBA.LTPAServerAssociationEnabled=true
```

6. Ensure that the following five lines of the sas.server.props file and the new sas.appserver.props file are exactly the same. The following sample shows the structure you are looking for:

```
          com.ibm.CORBA.loginUserid=<userid>          com.ibm.CORBA.principalName=<DOMAIN/userid>
com.ibm.CORBA.loginPassword=<password>          com.ibm.CORBA.securityEnabled=true
com.ibm.CORBA.authenticationTarget=ltpa
```

7. Start the administrative console and add a command-line entry to the application server. Modify this entry so that the command-line property `com.ibm.CORBA.ConfigURL` is set to the new sas.appserver.props file; for example:
   - **Syntax for Windows NT**:
     `-Dcom.ibm.CORBA.ConfigURL=file:/C:/Websphere/appserver/properties/sas.appserver.props`
   - **Syntax for UNIX**:
     `-Dcom.ibm.CORBA.ConfigURL=file:///usr/WebSphere/AppServer/properties/sas.appserver.props`

   Repeat this step for any other application servers from which you want serve unprotected resources. For application servers from which you want to serve protected resources, do not modify the `ConfigURL` property; continue to use the sas.server.props file in the value.
8. Stop and restart the entire WebSphere Application Server domain to make the changes take effect.

If you are using a pure Java client against an application serverusing the sas.appserver.props configuration file, the Java clientno longer needs to use the sas.client.props file.

5.8: Single Sign-On

Single sign-on (SSO) support allows Web users to authenticate once whenaccessing both WebSphere Application Server resources, such as HTML, JSPs,servlets, and enterprise beans, and Domino resources, such as documentsin a Domino database, or when accessing resources in multipleWebSphere domains.

Web users can authenticate once to a WebSphere application server orDomino server and then access any other WebSphere application servers orDomino servers in the same DNS domain that are enabled for Single Sign-On (SSO)without logging on again. This is accomplished by configuring the WebSphereapplication servers and the Domino servers to share authentication information.

To enable SSO among WebSphere application servers, you must configureSSO for WebSphere. To enable SSO between WebSphere application serversand Domino servers, you must configure SSO for both WebSphere and forDomino.

This configuration is described in subsequent sections, but thereare prerequisites that applications must meet in order to support theuse of single sign-on.

Prerequisites and conditions

To take advantage of support for single sign-on between WebSphere applicationservers or between WebSpere and Domino, applications must meet thefollowing prerequisites and conditions:

- All servers must be configured as part of the same DNS domain. For example, if the DNS domain is specified as mycompany.com, then SSO will be effective with any Domino or WebSphere application server on a host that is part of the mycompany.com domain, for example, a.mycompany.com and b.mycompany.com.
- All servers must share the same user registry. This registry can be either a supported LDAP directory server or, if SSO is being configured between two WebSphere application servers, a custom user registry. Domino does not support the use of custom registries, but a Domino-supported registry can be used as a custom registry within WebSphere. For more information on custom registries, see Introduction to custom registries.

  A Domino Directory (configured for LDAP access) or other LDAP directory can be used for the user registry. The LDAP directory product must be supported by WebSphere Application Server. Supported products include both Domino and all IBM SecureWay LDAP directory servers. Regardless of the choice to use an LDAP or custom registry, the SSO configuration is the same. The difference is in the configuration of the registry.
- All users must be defined 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 is not supported.
- Users must enable HTTP cookies in their 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 Domino
  - Domino R5.0.6a for iSeries 400 (or later) and Domino R5.0.5 (or later) for other platforms are supported.
  - A Lotus Notes client R5.0.5 (or later) is required for configuring the Domino server for SSO.
  - Authentication information can be shared across multiple Domino domains.
- For WebSphere Application Server
  - WebSphere Application Server V3.5 (or later) for all platforms is supported.

- Any HTTP Web server supported by WebSphere Application Server can be used.
- Authentication information can be shared across multiple WebSphere administrative domains.
- Basic authentication (user ID and password) using the basic and form-login mechanisms is supported.
- Permissions for either all authenticated users or groups of users is supported. If you are using the Domino Directory for authentication and have not specified a Base Distinguished Name during setup, permissions for individual users is also supported.

5.8.1: Configuring SSO for WebSphere Application Server

To use SSO between WebSphere Application Server and Domino or betweentwo WebSphere application servers, you mustfirst configure SSO for WebSphere Application Server. SSO for WebSphere Application Server allows authentication information to be shared acrossmultiple WebSphere Application Server administrative domains and withDomino servers.

To provide SSO to WebSphere application servers in more than one WebSphereApplication Server administrative domain, you must configure each of theadministrative domains to use the same DNS domain, user registry (using LDAPor a custom registry), and a common set of LTPA keys as described in the detailed sections below:

- Modify WebSphere Application Server security settings.
- Stop and restart the administrative server.
- Export LTPA keys to a file.
- Authorize users.
- Verify the configuration.

 This section assumes that you have already installed WebSphereApplication Server and configured one or more application serversin one or more WebSphere Application Server administrative domains.

 This section assumes that you are using LDAP as the user registry.The SSO setup is the same, regardless of the use of an LDAPregistry or a custom registry. The difference is in theconfiguration of the registry itself.For more information on custom registries, see5.2: Introduction to custom registries.

Before attempting to configure SSO for WebSphere Application Server, youcan verify the accessibility of WebSphere Application Server by doing thefollowing:

- Verify that the application servers are configured correctly by using a Web browser to access application resources.
- Verify the LDAP directory you are going to use is available and configured with at least one user. Configuring SSO for WebSphere Application Server requires access to the LDAP directory. You can use the Domino Directory or another LDAP directory.

Modify WebSphere Application Server security settings

SSO configuration is included as part of the overall security configurationof a WebSphere Application Server administrative domain.

1. Start the WebSphere administrative server for the administrative domain.
2. Start the WebSphere administrative console.
3. On the administrative console, select **Security Center** from the console menu.
4. Select the **General** tab if it is not already selected. On this panel,
    1. Enable WebSphere Application Server security by checking the **Enable Security** check box.
    2. Verify that the **Security Cache Timeout** field is set to a reasonable value for your application. When the timeout is reached, WebSphere Application Server clears the security cache and rebuilds the security data. If the value is set too low, the extra processing overhead can be unacceptable. If the value is set too high, you create a security risk by caching security data for a

long period of time. The default value is 600 seconds.

5. Click the **Authentication** tab. In this window:
    1. Set the **Authentication Mechanism** field to Lightweight Third Party Authentication (LTPA), to use an LDAP directory as the user registry.
    2. Check the **Enable Single Sign On (SSO)** box to enable SSO and authentication information to be placed in HTTP cookies.
    3. Set the **Domain** field to the domain portion of your fully qualified DNS name for the system running your WebSphere Application Server administrative domain. For example, if your system's host name is myhost.mycompany.com, type mycompany.com in this field.

    Before closing this window, you also need to configure the LTPA keys to be used by the administrative domain that you are configuring. You must perform *one* of the following steps, based on the number of administrative domains you are configuring:

    - If you are configuring the first or only WebSphere Application Server administrative domain, generate the LTPA keys as follows:
        1. Click **Generate Keys** to generate keys for LTPA.
        2. When prompted, type the LTPA password to be associated with these LTPA keys. Then click **OK** to save the LTPA keys. You must use this password when importing these keys into other WebSphere Application Server administrative-domain configurations (if any) and when configuring SSO for Domino.
    - If you are configuring an additional WebSphere Application Server administrative domain, you must import the LTPA keys used during the configuration of the first administrative domain. Import the LTPA keys as follows:
        1. Click **Import From File** to import the LTPA keys from a file.
        2. When prompted, select the file that was generated previously during the configuration of the initial administrative domain.
        3. Click **Open**.
        4. When prompted, type the LTPA password you set when initially generating the keys. Then click **OK** to import the keys.
6. Click the **LDAP** button. (If you are using a custom registry, click the **Custom User Registry** button instead. This discussion assumes the use of an LDAP user registry.)
7. Fill in the LDAP fields as follows:
    - **Security Server ID**: The user ID of the administrator for the WebSphere administrative domain. Use the short name or user ID for a user already defined in the LDAP directory. Do not specify a Distinguished Name by using `cn=` or `uid=` before the value. This field is not case sensitive. When you start the WebSphere Application Server administrative console, you are prompted to log in with an administrative account. You must enter exactly the same value that you specify in this field.
    - **Security Server Password**: The password corresponding to the **Security Server ID** field. This field is case sensitive.
    - **Directory Type**: The type of LDAP server you are using. For example, you can select SecureWay for IBM SecureWay LDAP Directory or Domino 5.0 for Domino R5.05 from the list.
    - **Host**: The fully qualified DNS name of the machine on which the LDAP directory runs, for example myhost.mycompany.com.
    - **Port**: The port on which the LDAP directory server listens. By default, an LDAP directory server using an unsecured connection listens on port 389. If your server meets this description, you can leave this field blank.

- **Base Distinguished Name**: The Distinguished Name (DN) of the directory in which searches begin within the LDAP directory. For example, for a user with a DN of `cn=John Doe, ou=Rochester, o=IBM, c=US` and a base suffix of `c=US`, the base DN can be specified as any of:
  - `ou=Rochester, o=IBM, c=us`
  - `o=IBM, c=us`
  - `c=us`

  This field is not case sensitive.

  This field is required for all LDAP directories except the Domino Directory. If you are using the Domino Directory and you specify a Base Distinguished Name, you will *not* be able to grant permissions to individual Web users for resources managed by your WebSphere application server.
- **Bind Distinguished Name**: The DN of the user who is capable of performing searches on the directory. In most cases, this field is not required; typically, all users are authorized to search an LDAP directory. However, if the LDAP directory contents are restricted to certain users, you need to specify the DN of an authorized user, for example, an administrator, `cn=administrator`.
- **Bind Password**: The password corresponding to the Bind Distinguished Name field. This value is required only if you specified a value for the Bind Distinguished Name field. This field is case sensitive.

8. Click **Finish** to save the security settings.
9. Click **OK** to acknowledge the information dialog box that warns that changes do not take effect until the administrative server is restarted.

Stop and restart the administrative server

Whever changes are made to the global security settings, theWebSphere Application Server administrative server must be stoppedand restarted for the changes to take effect.

1. On the administrative console, expand the Nodes icon.
2. Click the node representing your administrative server.
3. Expand the Application Servers icon within your administrative server.
4. Click the **Default Server** icon or the icon for the appropriate applicaiton server.
5. Click either **Stop** or **Force Stop**, and wait for the server to stop.
6. Right-click the node representing the administrative server, and select **Stop**.
7. Click **Yes** on the confirmation dialog box.
8. Monitor the administrative server task (or job) to ensure that the server stops. Then restart the administrative server, monitoring the server task (or job) to determine when the server is running. As you watch the server job, notice that it starts, stops, and then starts again. This is normal behavior after global security settings have been changed.
9. Start the administrative console. Specify the user ID and password by using exactly the same values that you specified for the **Security Server ID** and **Security Server Password** fields in the Global Security Settings wizard.

Export the LTPA keys to a file

To complete the security configuration for SSO, you need to export theLTPA keys to a file. This file is subsequently used during theconfiguration of additional administrative domains and during theconfiguration of

SSO for Domino.

1. Stop the WebSphere administrative domain to insure that the security settings are stored in WebSphere Application Server's configuration files or repository.
2. Start the administrative server for the domain.
3. Start the administrative console.
4. On the administrative console, select **Security Center** from the console menu.
5. Select the **Authentication** tab.
6. Click the **Lightweight Third Party Authentication (LTPA)** button.
7. Click the **Export To File** tab to export the LTPA keys to a file.
8. When prompted, specify the name and location of the file to contain the LTPA keys. You can use any file name and extension. Note the name and extension you specify; you must use this file when you configure SSO for any additional WebSphere Application Server administrative domains and for Domino.
9. Click **Save** to save the file.
10. Click **Cancel** to close the wizard. (This procedure has not changed any global security setting, so there are no new settings to save.)

Authorize users

Before you can test the SSO configuration for WebSphere ApplicationServer, you must grant users permissions to resources so that their access canbe tested. These tasks are not specific to SSO configuration and are notcovered in detail here. See The WebSphereauthorization model for more information.

Verify the configuration of SSO for WebSphere

After configuring each administrative domain, restart the WebSphere administrative console and log onto each of theadministrative domains to verify that the LTPA security settings are correct.

To verify the SSO configuration, attempt to configure at leastone resource, such as the Hello servlet, to be protected by a WebSphere application server. Use the Role Mapping panel inthe security center of the administrative console to authorizeWeb users to the resource.

The discussion in Verifying SSO betweenWebSphere and Domino assumes that SSO is being setup betweenWebSphere and Domino. If you are setting up SSO between twoWebSphere application servers, the verification procedure canstill be used if you replace the references to theDomino server with references to the second WebSphere applicationserver. Be sure that the LTPA keys are being shared properlybefore running the test. The keys must be exported from oneWebSphere Application Server domain and imported into thesecond domain so that the LTPA token can be decrypted.

5.8.2: Configuring SSO for Lotus Domino

To use SSO with Domino and WebSphere Application Server, youmust first configure SSO for WebSphere Application Server andthen configure SSO for Domino.

Configuring SSO for Domino is accomplished by selecting a newMulti-server option in a Server document for session-based authentication, and by creating a new domainwide configuration document, calledthe Web SSO Configuration document, in the Domino Directory.The Web SSO Configuration document, which must be replicated to allDomino servers participating in the SSO domain, is encrypted forparticipating Domino servers and contains a shared secret used byDomino servers for authenticating user credentials.

To provide SSO to Domino servers, do the following:

- Create the Web SSO Configuration document.
- Configure the Server document.
- Finish the Domino configuration.
- Verify the SSO for Domino configuration.

In addition, you can optionally do the following:

- Configure additional Domino servers in the original domain.
- Configure Domino servers in different domains.

To complete this procedure, you need the following informationfrom the configuration of SSO for WebSphere Application Server:

- The path and name of the file containing the LTPA keys created during SSO configuration for WebSphere Application Server
- The password used to protect the LTPA keys from WebSphere Application Server
- The name of DNS domain in which WebSphere Application Server is configured

Create the Web SSO Configuration document

To create the Web SSO Configuration document, use a Lotus Notes ClientR5.0.5 (or later) and follow these steps:

1. In the Domino Directory, select the Servers view.
2. Click on the Web pull-down menu item.
3. Select the **Create Web SSO Configuration** option to create the document.
4. On the Web SSO Configuration document, click the Keys pull-down menu.
5. Select the **Import WebSphere LTPA Keys** option to import the LTPA keys previously created for WebSphere Application Server and stored in a file.
6. Type the path to the file containing the keys for WebSphere Application Server and click **OK**.
7. Type the password that was used when generating the LTPA keys. The SSO Configuration document is automatically updated to reflect the information in the imported file.
8. Fill in remaining fields in this document. Groups and wildcards are not allowed in the fields. The following list describes the fields and the expected values:
    - **Token Expiration**: The number of minutes a token can exist before expiring.

      A token does not expire based on inactivity; it is valid for only the number of minutes specified from the time of issue.

- **Token Domain**: The DNS domain portion of your system's fully qualified Internet name. This is a required field.

  All servers participating in SSO must reside in the same DNS domain; this value must be the same as the Domain value specified when configuring WebSphere Application Server. Also, WebSphere Application Server treats the DNS domain as case sensitive, so ensure that the DNS domain value is specified in exactly the same way, including the same case.

- **Domino Server Names**: The Domino servers that will be participating in SSO. This SSO Configuration document will be encrypted for the creator of the document, the members of the **Owners** and **Administrators** fields, and the servers specified in this field. These servers can be in different Domino domains; however they must be in the same DNS domain.

  You must specify a fully qualified Domino server name, for example, `MyDominoServer/MyOu`. The Domino server name that you specify here must also match the name of the corresponding server's Connection document in your client's Domino Directory.

- **LDAP Realm**: The fully qualified DNS host name of the LDAP server.

  This field is initialized from the information provided in the imported LTPA keys file. You need to change this value only if an port value for the LDAP server was specified for the WebSphere Application Server administrative domain. If a port was specified, a backslash character (\) must be inserted into the value before the colon character (:). For example, replace myhost.mycompany.com:389 with myhost.mycompany.com\:389.

9. Save the Web SSO Configuration document. It now appears in the Web Configurations view.

If you are configuring multiple Domino servers for SSO, refer to Configuring additional Domino servers.

Configure the Server document

To update the Server document for SSO, follow these steps:

1. In the Domino Directory, select the Servers view.
2. Edit the Server document.
3. Select the **Ports --> Internet Ports --> Web** tab
4. Click the **Enable Name & Password Authentication for the HTTP Port** box to enable basic authentication for Web users.
5. Select **Internet Protocols --> Domino Web Engine**.
6. Select Multi-server in the **Session Authentication** field to enable SSO for Domino.
7. Save the Server document.

If you are configuring multiple Domino servers for SSO, refer to Configuring additional Domino servers.

Finish the Domino configuration

Before continuing, finish configuring the Domino server foruse by Web users. The remaining configuration steps are not specificto SSO and are not covered here in detail. Refer to the Domino 5 Administration Help for information on the following:

- Configuring access to an LDAP directory when the Domino Directory is not being used
- Authorizing Web users to Domino resources

Verify the SSO for Domino configuration

To verify the SSO configuration for Domino, ensure that theDomino server is configured correctly and that Web users areauthorized to access Domino resources by performing the followingsteps:

- To verify that the Domino server is configured correctly, stop and restart the Domino HTTP Web server. If SSO is configured correctly, the following message appears on the Domino server console: `HTTP: Successfully loaded Web SSO Configuration.`

  If a Domino server enabled for SSO cannot find a Web SSO Configuration document or is not included in the Domino Server Names field and therefore cannot decrypt the document, the following message appears on your server's console: `HTTP: Error Loading Web SSO configuration. Reverting to single-server session authentication.`
- To verify that users are authorized, attempt to access a Domino resource, such as a Domino Directory, first as a user defined in the Domino Directory itself, for local authorization, and then as a user defined in the LDAP directory service, for authorization of WebSphere Application Server users.

Configure additional Domino servers in a single domain

If you are using SSO with multiple Domino servers, perform thefollowing steps for each additional server:

1. Replicate the initial Web SSO Configuration document to each additional Domino server.
2. Update the Server document for each additional Domino server.
3. Restart each of the Domino HTTP web servers.

Configure Domino servers in multiple Domino domains

If you are using SSO with Domino servers is multiple Domino domains,you must also set up cross-domain authentication among the Dominoservers. For example, assume there are Domino servers in twoDomino domains, X and Y. Use the following procedure to enablethe Domino servers to perform SSO between the domains:

1. A Domino administrator must copy the Web SSO Configuration document from the Domino Directory for Domain X and paste it into the Domino Directory for Domain Y. The Domino administrator needs rights to decrypt the Web SSO Configuration document in Domain X and to create documents in the Domino Directory for Domain Y.
2. Ensure that your Lotus Notes client's location home server is set to a Domino server in Domain Y.
3. Edit the Web SSO Configuration document for Domain Y.
4. In the **Participating Domino Servers** field, include only the Domino servers with Server documents in Domain Y that will participate in SSO.
5. Save the Web SSO Configuration document. It is now to be encrypted for the participating Domino servers in Domain Y, so these servers now have the same key information as the Domino servers in domain X. This shared information allows Domino servers in Domain Y to perform SSO with Domino servers in Domain X.

5.8.3: Verifying SSO between WebSphere and Domino

This document discusses the verification of SSO between Dominoand WebSphere Application Server. Before proceeding, verifythat the following conditions are met:

- The LDAP directory contains at least one user that is defined for testing purposes.
- The WebSphere Application Server administrative console can be started for each of the WebSphere Application Server administrative domains involved in SSO.
- A user can authenticate to each administrative domain using a security name defined in the LDAP directory.
- At least one user in the LDAP directory must be authorized to access at least one Domino resource, such as the Domino Directory.
- At least one user in the LDAP directory must be authorized to access at least one WebSphere Application Server resource, such as the Hello servlet.
- From a Web browser that is configured *not* to accept HTTP cookies, you are able to reach the following resources:
    - WebSphere-protected resources, like servlets, after being prompted for a user ID and password.
    - Domino-protected resources, like Lotus Notes databases, after being prompted for a user ID and password.

If all of the preliminary tests succeed, you are ready to verify thatSSO is working correctly. To test the SSO functionality, performthe following steps:

1. Restart the Web browser.
2. Configure the Web browser to accept HTTP cookies. (If you are using Internet Explorer, enable the per-session (not stored) type of cookies.
3. Configure the browser to notify you before accepting HTTP cookies. This will provide visual confirmation that Domino and WebSphere Application Server are generating and returning HTTP cookies to your browser after you authenticate. (You can suppress the cookie notifications after you verify that cookies are being exchanged.)
4. From the browser, specify the URL for a resource protected by the Domino server; for example, attempt to open a database that permits no access to anonymous users, as described in the following example:
    - Make sure to user a fully qualified DNS host name in the URL; for example, enter `http://myhost.mycompany.com/names.nsf` instead of `http://myhost/names.nsf`.
    - When prompted for a user ID and password, make sure that you specify a user ID that is authorized to resources for both the Domino and WebSphere application servers.

      The format of the name depends on the level of restriction Domino is using for Web users and whether Domino or another LDAP directory is being used. (For details on the options for basic authentication, refer to the Domino 5 Administrative Help; in particular, see the information on controlling the level of authentication for Web clients.) The level of restriction Domino uses for Web users is set in the Web server authentication field on the Security window of the Server document. If you are using the default configuration settings, you can specify the user's short name or user ID.
    - When prompted, accept the HTTP cookie.

    Successfully accessing such a resource verifies that the token generated by the Domino server is accepted by WebSphere Application Server.
5. From the same browser session, attempt to access a resource protected by WebSphere Application Server. If SSO is working correctly, access is granted without prompting you to log in. (If you are

prompted, refer to SSO fails when accessing protected resources for assistance.) Make sure to use the fully qualified DNS host name in the URL. For example, type `http://myhost.mycompany.com/webapp/examples/showCfg` instead of `http://myhost/webapp/examples/showCfg`.

6. From the same browser session, attempt to access resources managed by any additional Domino and WebSphere Application Server domains included in your SSO configuration.
7. Restart your browser session and perform the SSO-verification steps again, but this time, start by accessing a resource protected by WebSphere Application Server. This will verify that the token generated by WebSphere Application Server is accepted by the Domino server or servers. When prompted for a user ID and password, use the user's short name or user ID; this is the default naming convention for users in WebSphere Application Server.

5.8.4: Troubleshooting SSO configurations

This article describes common problems in configuring singlesign-on between WebSphere Application Server and Domino and suggestspossible solutions. The problems include the following:

- Failure to save the Domino Web SSO Configuration document
- Domino server console fails to load the Web SSO Configuration document upon Domino HTTP server start-up
- Authentication fails when accessing a protected resource
- Authorization fails when accessing a protected resource
- SSO fails when accessing protected resources

Failure to save the Domino Web SSO Configuration document

The client must be able to find Domino Server documents for theparticipating SSO Domino servers. The Web SSO Configurationdocument is encrypted for the servers that you specify, so the home server indicated by the client's location recordmust point to a server in the Domino domain where the participatingservers reside. This ensures that lookups can find the public keysof the servers.

If you receive a message that states that one or more of the participatingDomino servers cannot be found, then those servers will not be able todecrypt the Web SSO Configuration document or perform SSO.

When the Web SSO Configuration document is saved, the status bar indicateshow many public keys were used to encrypt the document by findingthe listed servers, authors, and administrators on the document.

Domino server console fails to load the Web SSOConfiguration document upon Domino HTTP server startup

During configuration of SSO, the Server document is configured forMulti-Server in the Session Authentication field. Therefore, theDomino HTTP server tries to find and load a Web SSO Configurationdocument during startup. The Domino server console reports thefollowing if a valid document is found and decrypted:

```
HTTP: Successfully loaded Web SSO Configuration.
```

If a server cannot load the Web SSO Configuration document, SSOdoes not work. Such a server reports the following message:

```
HTTP: Error Loading Web SSO configuration. Reverting tosingle-server session
authentication.
```

Make sure that there is only one Web SSO Configurationdocument in the Web Configurations view of the DominoDirectory and in the $WebSSOConfigs hidden view.You cannot create more than one, but additional documentscan be inserted during replication.

Check the hidden view $WebSSOConfigs as follows:

1. From a Lotus Notes client, select **File --> Database --> Open**.
2. In the Open Database dialog, either type the Domino server name and press Enter or select the Domino server from the list.
3. Type the value `names.nsf` for the **FileName** field, located at the bottom of the Open Database dialog box. Do *not* press Enter. Instead, hold the the shift and control keys down and click **Open** on the dialog box. This opens the Domino Directory with all the hidden views exposed.
4. At the bottom of the view list, click $WebSSOConfigs and ensure there is only one document in this view. If there are more than one, delete them all and re-create the Web SSO Configuration document.

If there is only one Web SSO Configuration document, anothercondition that can elicit the same error message is that thepublic key of the Server document does not match the public keyin the ID file. In this case, attempts to decrypt theWeb SSO Configuration document fail and the error message isgenerated.

This situation can occur when the ID file is created multiple times but the Server document is not updated correctly.Usually, there is an error message displayed on the Domino Server Consolethat states that the public key does not match the server ID. If this happens,then SSO does not work because the document is encrypted witha public key for which the server does not possess the correspondingprivate key.

To correct a key-mismatch problem, do the following:

1. Copy the public key from the server ID file and paste it into the Server document.
2. Re-create the Web SSO Configuration document.

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 WebSphere securityserver is not able to authenticate the user with the LDAP server. Check the following possibilities:

- Verify that the LDAP server can be accessed from the Domino server machine. Use the TCP/IP **ping** utility to verify TCP/IP connectivity and 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, the following command, entered as a single line, can be run 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 follow:
  - No such object: This error indicates that the directory entry referenced by either the user's 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.
  - Can't contact LDAP server: This error means 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, ensure that the directory entry is configured with the short name. For a Domino Directory, this is the **Short name/UserID** field of the Person document. For other LDAP directories, this is the userid property of the directory entry.
- If Domino authentication fails when using an LDAP directory other than 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 DN of the directory containing the directory entries of the users.

  You can trace the Domino server's requests to the LDAP server by adding the following line to the server's notes.ini file:

  ```
  webauth_verbose_trace=1
  ```

  After restarting the Domino server, trace messages are displayed in the Domino server's console as Web users attempt to authenticate to the Domino server.

Authorization fails accessing a protected resource

After authenticating successfully, if a Web user is shown an authorizationerror message, security is not configured correctly. Check the followingpossibilities:

- 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 URL 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 re-created.

SSO fails when accessing protected resources

If a Web user is prompted to authenticate each time he or sheaccesses a resource, SSO is not configured correctly. Check the following

possibilities:

1. Both WebSphere Application Server and Domino must be configured to use the same LDAP directory. The HTTP cookie used for SSO stores the full Distinguished Name (DN) of the user, for example, `cn=John Doe, ou=Rochester, o=IBM, c=US`, and the DNS domain.
2. If the Domino Directory is being used, Web users must be defined by hierarchical names. 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. URLs issued to Domino and WebSphere application servers configured for SSO must specify the full DNS server name, not just the host name or TCP/IP address. For browsers to be able 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 URL. (This is why cookies cannot be used across TCP/IP domains.)
4. Domino and WebSphere Application Server must be configured to use the same DNS domain. Verify that the DNS domain value is exactly the same, including capitalization. The DNS domain value can be found on the Configure Global Security Settings panel of the WebSphere 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 Domino servers participating in SSO.
5. Clustered Domino servers must have the host name populated with the full DNS server name in the Server document for Domino ICM (Internet Cluster Manager) to redirect to cluster members using SSO. If this field is not populated, by default, ICM redirects URLs to clustered Web servers by using only the host name. It cannot send the SSO cookie because the DNS domain is not included in the URL.
   To correct the problem, do the following:
   1. Edit the Server document.
   2. Select the **Internet Protocols -- > HTTP** tab.
   3. Enter the server's full DNS name in the **Host names** field.
6. If a port value for an LDAP server was specified for a WebSphere Application Server administrative domain, the Domino Web SSO Configuration document must be edited and a backslash character (\)must be inserted into the value of the **LDAP Realm** field before the colon character (:). For example, replace myhost.mycompany.com:389 with myhost.mycompany.com\:389.

5.9: Configuring security interoperation with WebSphereon z/OS

WebSphere Application Server Advanced Edition supports interoperabilitybetween application servers running on UNIX or NT platforms andapplication servers running on the z/OS platform. This supportallows application servers on the UNIX or NT side to authenticateto the application server on the z/OS side and communicate securely.Unauthenticated requests from the UNIX- or NT-based application serversare rejected. Authentication is supported between application servers,not individual applications.

To configure this support, serveral steps must be taken. WebSpheresecurity must be enabled on both sides. Information used forauthenticating to the z/OS-based application server must be collectedand stored in a key file for use by the UNIX- or NT-based application server.The Secure Sockets Layer (SSL) protocol, which is used to secure thecommunication channel, requires that the UNIX- or NT-based serveralso have a valid certificate for the z/OS-based application server.Finally, the UNIX- or NT-based applications must be configured to usethe appropriate identities so that they can communicate with the z/OS-basedapplication servers. The following describes the specific steps that mustbe taken:

1. Collect the login information for the z/OS-based application server and store it in a key file for use by the UNIX- or NT-based application server. See Creating the key file for more information.
2. Enable global security for WebSphere Application Server for the UNIX- or NT-based application servers. See 6.6.18: Securing applications for more information.
3. Enable global security for WebSphere Application Server for the z/OS-based application server. See *WebSphere Application Server V4.0 for z/OS and OS/390: Installation and Customization* for more information; this can be reached from the Library link on the main WebSphere Application Server page.
4. Create a certificate for use by the UNIX- or NT-based application server, as required by the SSL protocol. See Creating the certificate for more information.
5. Configure the UNIX- or NT-based applications to use the identity of the application server when communicating with z/OS-based applications. See Configuration for interoperation for more specific information or 6.6.18: Securing applications for general information.
6. Configure the z/OS-based application server to accept communications from the UNIX- or NT-based application servers. See *WebSphere Application Server V4.0 for z/OS and OS/390: Installation and Customization* for more information; this can be reached from the Library link on the main WebSphere Application Server page.

Creating the key file

The UNIX- or NT-based application servers must have accessto the information needed to authenticate to each z/OS-basedapplication server. The login information, which includes thetarget realm, user ID, and password, for every z/OS targetmust be stored in a local text file. The passwords in thisfile are encoded when the security service processes the file,but it is also suggested that access to the file itself be restrictedby storing the file in a securable file system and settingpermissions appropriately. For example, on a Windows-based system,NTFS partitions systems are securable, but DOS partitions are not.

The information in the key file must be formatted as follows:

- Each entry must contain these three pieces of information, in the order specified, separated by spaces:
    1. **Realm name**: The IP name of the Daemon Server in WebSphere for z/OS.
    2. **User ID**: The user ID defined for SSL-secured servers on the z/OS platform.
    3. **Password**: The password corresponding to the user ID defined for SSL-secured servers.
- The file must contain no blank lines.
- Use the hash (#) character to include comments and other informational lines.
- All comments must begin on new lines; they cannot appear after the authentication entries on the same line.

A sample file is provided with WebSphere Application Server. Thisfile, wsserver.key, is installed in the<*product_installation_root*>/propertiesdirectory. It can be copied or modified. The following also illustratesthe structure of the file:

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

Creating the certificate

The SSL protocol is used to protect communication between theUNIX- or NT-based application server and the z/OS-based applicationserver. To complete the SSL handshake between them, the UNIX- or NT-basedapplication server must hold a valid key certificate. To create thiscertificate, perform the following steps:

1. On the z/OS side, extract the public key of the z/OS-based application server by using the z/OS key-management tools. See *WebSphere Application Server V4.0 for z/OS and OS/390: Installation and Customization* for more information; this can be reached from the Library link on the main WebSphere Application Server page.
2. On the UNIX or NT side, open the certificate for the UNIX- or NT-based application server and add the public key of the z/OS-based application server as a signer certificate. See 5.5.6: Tools for managing certificates and keys for more information on the tools and techniques for managing certificates.

Configuration for interoperation

Before UNIX- or NT-based application servers and z/OS-basedapplications servers can interoperate, the application serversand applications must be configured for interoperation. On theUNIX or NT side, this involves the following:

- Configuring application resources, for example, enterprise beans, that must access the z/OS-based application server to run under the identity of the hosting application server. In the interoperability scenario, it is the application servers, not individual applications, that authenticate, so resources like enterprise beans must run under the identity of application server. For example, before deploying an enterprise bean that can contact the z/OS-based application server, the `RunAs` identity of the bean must be set to **System Identity**.
- Setting properties for the application server so that it can find the key file and key certificate containing the information about the z/OS-based application servers. The following properties must be set:
  - **com.ibm.CORBA.loginSource**: set to `key file`.
  - **com.ibm.CORBA.keyFileName**: set to the absolute path of the key file. For example, C:\WebSphere\AppServer\properties\wsserver.key.
  - **com.ibm.CORBA.SSLClientKeyRing**: set to the absolute path of the key certificate file containing the public key of the z/OS-based application server.
  - **com.ibm.CORBA.SSLClientKeyRingPassword**: set to the password protecting the file specified in the `com.ibm.CORBA.SSLClientKeyRing` property.
  - **com.ibm.CORBA.requestTimeout** and **com.ibm.CORBA.locateRequestTimeout**: set both properties to 0 in the sas.client.props and sas.server.props files. The reason for this is that, when a WebSphere application server on z/OS first starts, it has no regions available for processing work. Setting these timeout properties to zero prevents timeouts from occurring before the regions are established.

6.6.18: Securing applications

For purposes of security, Application Server categorizes assetsinto two classes: resources and applications.

- *Resources* are individual components, such as servlets and enterprise beans.
- *Applications* are collections of related resources.

Security can be applied to applications and to individual resources. Setting up security involves the following general steps:

1. Setting global values for use by all applications.
2. Refining settings for individual applications.

Securing applications with IBM WebSphere ApplicationServer product security involves a series of tasks. Completing thetasks results in a set of policies defining *which*users have access to *which* methods or operations in *which*applications.

For example, the security administrator establishes policies specifyingwhether the user *Bob* is permitted to use the company's Inventoryapplication to perform a write operation, such as changing the numberunits of merchandise recorded in the company's inventory database.

The product security server works withthe selected user registry or directory product to enforce thepolicies whenever a user tries toaccess a protected application. For example, *Bob* might beprompted for a digital certificate verifying his identity when hetries to use the Inventory application.

6.6.18.0: General security properties

Key:

 Applies to Java administrative console of Advanced Edition Version 4.0

 Applies to Web administrative console of Advanced Single Server Edition Version 4.0

 Applies to Application Client Resource Configuration Tool

Cache Timeout  **or Security Cache Timeout** 

Time after which the authentication cache will be refreshed. Caching can improve performance with respect to authentication lookups.

Specify this value in seconds, with a minumum of 30.

Default SSL Configuration  **or Use global SSL default configuration** 

Apply the default SSL configuration to the entire administrative domain.

For *Advanced Edition*, see Configuring SSL support instructions.

Enabled  **or Enable Security** 

Whether global security is enabled. When security is not enabled, all other security settings are not validated or used.

 For *Advanced Edition* (non-Single Server), when security is enabled for the first timewith the LTPA authentication mechanism selected, you will be prompted toenter a password for encrypting and decrypting LTPA keys. Makesure you remember the password! For more information about LTPA keys, refer tothe article about making LTPA-secured calls across WebSphere domains.

Security Cache Timeout 

See Cache Timeout

Use Domain Qualified User Names  

When the value of this setting is true, user names returned by calls such as getUserPrincipal() will be qualified with the security domain in which they reside

Use global SSL default configuration 

See the Default SSL Configuration field description

6.6.18.0.1: Properties for configuring Secure Socket Layer (SSL) support

Key:

Applies to Java administrative console of Advanced Edition Version 4.0

Applies to Web administrative console of Advanced Single Server Edition Version 4.0

Applies to Application Client Resource Configuration Tool

For Advanced Edition (non-Single Server), see Configuring SSL support instructions.

Client Authenticationor **Enable Client Authentication**

Whether the server and client should prove their identities through an exchange of keys

The SSL server is always authenticated to the client. If client authentication is enabled, the SSL client is also authenticated to the server. By default, client authentication is disabled.

Confirm Password

Type the password again, to confirm the correct spelling

Dynamic Properties

Name=value pairs that you can use to configure additional SSL settings beyondthose available in the administrative interface

com.ibm.ssl.protocol

This is the SSL protocol to be 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 key store 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 trust store 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 trust store. 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 are determined by the value of the SecurityLevel (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_MD5SSL_RSA_WITH_RC4_128_SHASSL_RSA_WITH_DES_CBC_SHASSL_RSA_WITH_3DES_EDE_CBC_SHASSL_DHE_RSA_WITH_DES_CBC_SHASSL_DHE_RSA_WITH_3DES_EDE_CBC_SHASSL_DHE_DSS_WITH_DES_CBC_SHASSL_DHE_DSS_WITH_3DES_EDE_CBC_SHASSL_RSA_EXPORT_WITH_RC4_40_MD5SSL_RSA_EXPORT_WITH_DES40_CBC_SHASSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHASSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHASSL_RSA_WITH_NULL_MD5SSL_RSA_WITH_NULL_SHASSL_DH_anon_WITH_RC4_128_MD5SSL_DH_anon_WITH_DES_CBC_SHASSL_DH_anon_WITH_3DES_EDE_CBC_SHASSL_DH_anon_EXPORT_WITH_RC4_40_MD5SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA

Enable Client Authentication

See Client Authentication

Enable Crypto Token Support

Whether cryptographic token support is enabled. If this is selected, then the values on the **Crypto Token** panel are used. After enabling Crypto Token support, stop your application serverand start it again for the change to take effect.

A crypto token is a hardware or software device which has a built-in key store implementation. The exact values for the following fields should be documented in the documentation of the supported cryptographic device.

Enable SSL

Whether to enable SSL support

Key File Format

The format of the key file. Possible values are JKS, PKCS12, and JCEK

Key File Name

The fully qualified path to the key file that contains public keys and perhaps private keys. See below for a note about the name.

An SSL key file can be created with the IKeyMan key management utility, or it may correspond to a hardware device if one is available. In either case, this specifies the source for personal certificates, as well as for signer certificates unless a trust file is specified.

The default Key File and the default Trust File contains a test certificate, and is only intended for use in a test environment. The default key files should never be used in a production environment because the private keys are same on all the WebSphere installations. Please refer to the the introduction to security certificates for information about creating and managing digital certificates for your WebSphere domain.

Key File Password

The password for accessing the key file

Library File

The DLL or shared object which implements the interface to the cryptographic device

Password

The password for the cryptographic device

Security Level

The security level can be HIGH, MEDIUM, or LOW and is a user-friendly way of enabling a certain set of cipher suites. The Security Level can be overridden by giving an explicit value to the dynamic property named **com.ibm.ssl.EnabledCipherSuites** (a **Dynamic Property** described previously). The mapping of security level to enabled cipher suites is as follows.

If the security level is HIGH, the enabled cipher suites are:

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

If the security level is MEDIUM, the enabled cipher suites are:

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

If the security level is LOW and a *server* SSL configuration, the enabled cipher suites are:

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

If the security level is LOW and a client SSL configuration, the enabled cipher suites are:

SSL_RSA_WITH_NULL_MD5
SSL_RSA_WITH_NULL_SHA

Token Type

The type of token, such as PKCS#11

Trust File Format

The format of the specified trust file

Trust File Name

The fully qualified path to a trust file containing the public keys. See below for a note about the name.

As with the SSL key file, this can be created with the IKeyMan utility, or it may correspond to a hardware device. Unlike the SSL key file, no personal certificates are referenced; only signer certificates are retrieved. If a trust file is not specified but the key file is specified, then the SSL key file is used for retrieval of signer certificates as well as personal certificates.

Trust File Password

A password for accessing the trust file

Note about key and trust file names: The Default SSL configuration data and the LDAP SSL configuration data managed by the WebSphere Application Server Security Center are shared by multiple nodes in the same security domain. Machines in the same security domain can host different operating systems, such as AIX and Windows 2000. Moreover, WebSphere Application Server installation path can be different on different host machines.

Hence it is not always possible to use absolute file path when specifying the location of the key store and the trust store. IBM WebSphere Application Server uses a symbolic link WAS_HOME (which equates to *product_installation_root*) to locate key store and trust store. For example, the key file name can be defined by

`${WAS_HOME}/etc/ServerKeyFile.jks`

The ServerKeyFile.jks must exist on all the host machine under the "etc" subdirectory of the *product_installation_root*. The contents in the key files can be different on different nodes, but the file names should match.

6.6.18.0.2: Properties for configuring security using local operating system

Key:

 Applies to Java administrative console of Advanced Edition Version 4.0

 Applies to Web administrative console of Advanced Single Server Edition Version 4.0

 Applies to Application Client Resource Configuration Tool

Authentication Mechanism 

Select how to authenticate users that try to access applications.

- Against the local operating system user registry, or
- Against an LTPA based LDAP registry or custom registry

Note that the local operating system user registry is intended for single machineand single application server environments. *Advanced Single Server Edition* supports only the local operating system mechanism.

 When form-based login is used with local operating system authentication, the user information is stored in the HTTP session. Using an HTTP connection is not very secure, meaning the information can be obtained by others. Using SSL connections (HTTPS) between the browser and the Web server will improve security.

 When security is enabled for the first timewith the LTPA authentication mechanism selected, you will be prompted toenter a password for encrypting and decrypting LTPA keys. Makesure you remember the password! For more information about LTPA keys, refer tothe article about making LTPA-secured calls across WebSphere domains.

Security Server ID  **or Server ID** 

The user ID under which the server runs, for security purposes. This ID is not associated with the system process. This ID refers to the application security context within the WebSphere Application Server product.

If using local operating system authentication, the following conditions apply:

- On UNIX operating systems, the ID must be root or have root authority.
- On Windows operating systems, the account must be a member of the Administrators group and must have the rights to "Log on as a service" and "Act as part of the operating system." If the Windows machine is a member of an NT domain, then the ID must also be an administrator in the NT domain. Do not use an account whose name matches the name of your machine or Windows Domain.

If using LDAP or custom registry authentication (not available for *Advanced Single Server Edition*), the following conditions apply:

- The user should be a valid user in the LDAP or custom registry
- The user should *not* be a root DN or administrator DN because those users are not always in the directory in all LDAP implementations.

Security Server Password  **or Server Password** 

The password corresponding to the server ID

6.6.18.0.3: Properties for configuring security using Lightweight Third Party Authentication (LTPA)

Key:

 Applies to Java administrative console of Advanced Edition Version 4.0

 Applies to Web administrative console of Advanced Single Server Edition Version 4.0

 Applies to Application Client Resource Configuration Tool

Domain 

Restrict SSO to servers in the domain you specify in this field. This domain name is used when creating HTTP cookies for Single Sign On. It determinesthe scope to which Single Sign On applies.

For example, a domain of austin.ibm.com would allow Single Sign On to work between WebSphere application server A at serverA.austin.ibm.comand WebSphere application server B at serverB.austin.ibm.com. Note that cross-domainSingle Sign On is not supported. That is, a server at austin.lotus.com, and anotherat austin.ibm.com cannot partipicate in WebSphere Single Sign On.

Enable Single Sign On 

Causes your LTPA directory service to store extra information in the tokens so that other applications can accept clients as already authenticated by WebSphere Application Server. When clients try to access the other applications, they will not be interrupted and asked to log in.

When you enable Single Sign On, the **Domain** field will be enabled. You must enter a DNS domain name. See the **Domain** field description for more information. The **Limit to SSL connections only**check box will also be enabled. The **Import Keys** and **Export Keys** button will also be enabled.

Enable Web Trust Associations 

When enabled, one or more trust associations will be active. Trust associations enable a third party reverse proxy server to perform authentication on behalf of the WebSphere Application Server security component. To do so, you need to create a corresponding interceptor for the reverse proxy server and determine how "trust" will be established between them. See the security documentation in the InfoCenter for additional information.

Limit to SSL connections only 

Specifies to use a connection with SSL for Single Sign On, to prevent the SSO token from flowing over non-secure connections. When this is set, form-based authentication will not work when resources are accessed over HTTP. The resources can be accessed only over HTTPS.

If this property is set and form-based login is used for authentication, the resources can be accessed only using secure connections (HTTPS). Connections that are not secure (HTTP) will not work. If basic login for authentication is used and the access is through an connection that has not been secured, then SSO will not work. The user will be prompted to log in again.

Token Expiration 

How many minutes can pass before a client using an LTPA token must authenticate again. LTPA uses tokens to store the authenticated status of a client.

A positive integer indicates the token life, in minutes

6.6.18.0.4: Properties for mapping security roles and "run as" roles to users and groups

Key:

 Applies to Java administrative console of Advanced Edition Version 4.0

 Applies to Web administrative console of Advanced Single Server Edition Version 4.0

 Applies to Application Client Resource Configuration Tool

Note, clicking **Cancel** in the Security Center will not undo the changes made to the roles.

Roles 

Roles to which you want to map users and groups in order to give the users and groups permissionto run as those roles.

Users 

Users to which you want to map roles. The users must be defined in your chosen authentication mechanism.

Groups 

Groups to which you want to map roles. The groups must be defined in your chosen authentication mechanism.

6.6.18.0.5: Properties for configuring using custom user registry (pluggable user registry)

Key:

 Applies to Java administrative console of Advanced Edition Version 4.0

 Applies to Web administrative console of Advanced Single Server Edition Version 4.0

 Applies to Application Client Resource Configuration Tool

Display these settings by selecting the **Custom User Registry** radio buttonlocated in the middle of the **Authentication** tabbed page when LTPA is the selected authentication mechanism.

To add or remove custom settings, besides thoseavailable in the administrative console, click the **SpecifyCustom Settings** button.

Custom User Registry Classname 

The name of the custom user registry implementation class file. This should be a dot separated class name.

For example, if the implementation file is com/myCompany/sampleRegistry.java, then enter com.myCompany.sampleRegistry. The class file should be in the WebSphere Application Server classpath. (See InfoCenter article 6.4.1 about setting classpaths.)

Security Server ID 

The user ID under which the server runs, for security purposes. This user should be a valid user in the custom user registry.

Security Server Password 

The password corresponding to the Security Server ID.

6.6.18.0.6: Custom properties for custom user registry

Key:

 Applies to Java administrative console of Advanced Edition Version 4.0

 Applies to Web administrative console of Advanced Single Server Edition Version 4.0

 Applies to Application Client Resource Configuration Tool

Use the **Add** button to enter new name-value pairs. Use **Remove** to remove a selectedsetting.

Name 

The name of any user defined custom registry properties.

Value 

The value for the corresponding property.

6.6.18.0.7: Properties for configuring LDAP support

Key:

 Applies to Java administrative console of Advanced Edition Version 4.0

 Applies to Web administrative console of Advanced Single Server Edition Version 4.0

 Applies to Application Client Resource Configuration Tool

Display these settings by selecting the **LDAP** radio buttonlocated in the middle of the **Authentication** tab when LTPA is the selected authentication mechanism.

Click the **Advanced** button to set advanced LDAP properties. Click the **SSL Configuration** button to set SSL properties for LDAP.

Base Distinguished Name 

The base distinguished name of the directory service, indicating the starting point for LDAP searches of the directory service. (See RFC 1779 for a discussion of this technique).For example, for a user with a DN of`cn=John Doe, ou=Rochester, o=IBM, c=US`, the base DNcan be specified as any of (assuming a suffix of c=us):

- `ou=Rochester, o=IBM, c=us`
- `o=IBM, c=us`
- `c=us`

This field is not case sensitive.

 This field is required for all LDAP directories except theDomino Directory. If you are using the Domino Directory andyou specify a Base Distinguished Name, you will *not* beable to grant permissions to individual Web users for resourcesmanaged by your WebSphere application server.

Bind Distinguished Name 

The distinguished name for application server to use to bind 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 

The password for the application server to use to bind to the directory service

Directory Type 

The directory service product to use to locate information against which to authenticate users and groups.

Modifications to the default values in the advanced LDAP properties will cause this field valueto change to Custom.

Host 

The host ID (IP address or DNS name) of the LDAP server

Port 

The host port of the LDAP server. The port number will default to 389 if none is specified.

If multiple WebSphere application servers are installed and configured to run in the sameSingle Sign On domain, or if the WebSphere application server will inter-operate with a previousversion of WebSphere application server, then it is important that the port number match inall configurations.

For example, if the LDAP port is explicitly specified as 389 in a Version 3.5.x configuration, and a Version 4.0 application server is going to inter-operate with the V3.5.xserver, then port 389 should also be specified explicitly for the Version 4.0 server. Notethat this is true even though the default port number is 389 -- if the port is specifiedexplicitly in one server configuration, it should be specified explicitly in allserver configurations.

Security Server ID 

The user ID under which the server runs, for security purposes

If using LDAP or custom registry authentication (not available for *Advanced Single Server Edition*), the following conditions apply:

- The user should be a valid user in the LDAP or custom registry
- The user should *not* be a root DN or administrator DN because those users are not always in the directory in all LDAP implementations.

Security Server Password 

The password corresponding to the Security Server ID

6.6.18.0.8: Properties for Select Users/Groups window

Key:

 Applies to Java administrative console of Advanced Edition Version 4.0

 Applies to Web administrative console of Advanced Single Server Edition Version 4.0

 Applies to Application Client Resource Configuration Tool

The following three options can be selected in any combination. See belowfor important usage notes.

Everyone 

Grants anyone and everyone the access to the role. This choice basically provides no security protection.

All Authenticated Users 

Grants users who are authenticated access to the resource.

Select Users/Groups 

Grants users or groups whom you select access to the role.

Generally, it is preferable to grant groups rather than individual users access to a role. It is easier to manage roles mapped to groups because there are typically fewer groups than users, users can be added to or removed from groups outside of WebSphere, and the authorization table has fewer entries, which can improve performance.

Usage notes

- If "Everyone" is selected then any other selections will be ignored.
- If "All authenticated users" is selected, but "Everyone" is not, then "Select users/groups" will be ignored.
- When "Select users/groups" is selected, the search button can be used to select users and groups using a pattern.

  For better performance, avoid using general wildcard search (* for example) if the target registry contains a large number of users or groups. Currently, only the first 1000 users and the first 1000 groups will be displayed. The display name is attached to the security name in the "Available Users/Groups" panel.

6.6.18.0.9: Advanced properties for configuring LDAP support

Key:

 Applies to Java administrative console of Advanced Edition Version 4.0

 Applies to Web administrative console of Advanced Single Server Edition Version 4.0

 Applies to Application Client Resource Configuration Tool

 If any of the user and group filters are modified from their default value, the **Directory Type** field value on the **Authentication** tabbed page will change to Custom.

Certificate Filter 

If you specified the filter Certificate Mapping, use this property to specifythe LDAP filter to use to map attributes in the client certificate to entries in LDAP.Note that if more than one LDAP entry matches the filter specification at runtime,then authentication will fail because it results in an ambiguous match.

The syntax or structure of this filter is:

```
LDAP attribute=${Client certificate attribute}
```

For example:

```
uid=${SubjectDN}
```

The left side of the filter specification is an LDAP attribute that depends on theschema that your LDAP server is configured to use. The right side of the filter specificationis one of the public attributes in your client certificate. Note that the right side mustbegin with ${ and end with }.

The following certificate attribute values may be used on the right side of the filterspecification. Note that 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.

Certificate Mapping 

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.

Group Filter 

An LDAP filter clause for searching the registry for groups. It is typically used for Security Role to Group assignment. 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.

Group ID Map 

An LDAP filter that maps the short name of a group to an LDAP entry. Specifies the piece of information that should represent groups when groups are displayed.

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 (";").

Group Member ID Map 

An LDAP filter that identifies User to Groups memberships. Specifies 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.

Initial JNDI Context Factory 

Java classname of the initial context factory of a provider

User Filter 

An LDAP filter clause for searching the registry for users. It is typically used for Security Role to User assignment. 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.

User ID Map 

An LDAP filter that maps the short name of a user to an LDAP entry. Specifies the piece of information that should represent users when users are displayed.

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 (";").

6.6.18.0.10: Properties for mapping "Run As" roles to users

Key:

 Applies to Java administrative console of Advanced Edition Version 4.0

 Applies to Web administrative console of Advanced Single Server Edition Version 4.0

 Applies to Application Client Resource Configuration Tool

Security Name 

For LDAP, a security name is the full distinguished name, such as CN=Bob Smith, o=austin.ibm.com.

For the Windows operating system, it is the user name with the hostname or domain name attached, such as myDomain\user1.

Short Name 

For LDAP, a short name can be the uid, such as bob.

For the Windows operating system, it is the user name without the hostname or domain name attached, such as user1.

Password 

The password corresponding to the User

User 

The user **Short Name** or **Security Name** as entered in other fields

The user name entered here depends on the selection in the **Select Users/Groups/Group** panel under the **Role Mapping** tabbed page of the Security Center.

If "Everyone (no authentication)" is selected, the user name defined in this panel is optional. Any user name in the current registry is valid.

If "Everyone (no authentication)" is not selected but the "All authenticated users" is selected then the user name is required. Any user name in the current registry is valid.

If the "Select users/groups" is the only selection then the user name is required. This user name must have been assigned to the same role in the **Role Mapping** panel or belong to a group that has been assigned to the same role.

6.6.18.0.11: Properties for encrypting and decrypting LTPA keys

Key:

 Applies to Java administrative console of Advanced Edition Version 4.0

 Applies to Web administrative console of Advanced Single Server Edition Version 4.0

 Applies to Application Client Resource Configuration Tool

Password 

The password to encrypt and decrypt the LTPA keys. This password should be used when importing these keys into other WebSphere Application Server administrative domain configurations (if any) and when configuring SSO for Domino Server.

6.6.18.1: Securing applications with the Java administrative console

To configure security, use the Security Center. Access the Security Centerby clicking **Console -> Security Center** on the console menu bar.

With it, you can complete the following security tasks:

- Enable product security
- Define a security realm and set of valid users
- Specify how to authenticate users seeking access to applications
- Grant users permissions to access applications

6.6.18.1.1a: Specifying global settings with the Java administrative console

1. Start the Security Center by clicking **Console -> Security Center** from the console menu bar.
2. Complete the task, referring to the information below for assistance.
3. Stop the administrative server and start it again for the changes to takeeffect.

   The next time the administrator opens the WebSphere AdministrativeConsole, the administrator will be prompted to log in (if security has been enabled), using an ID and passwordspecified during Security Centerconfiguration.

General

Use the General tab to specify whether to enable security. If the check box is *not*selected, any other security settings you specify will be disregarded.

This page also contains an option for setting a security cache timeout. The securitysystem caches authentication lookup information it receives from the user registry ordirectory service. Use this field to specify how long to cache the information (inseconds). Caching can improve lookup performance.

Authentication

Use the Authentication tabbed page to specify how to authenticate theinformation presented by users trying to access an application or resources.

The administrator can have users or groups authenticated against either the local operating system user registry (such as Windows NT User Manager program) or an LDAP or custom user registry.

Role Mapping

Use the Role Mapping page to assign users in particular groups to specific roles.Role mapping gives particular users or groups authorization to access one ormore applications defined by a role.

The users, groups and roles were defined when the application was installed or configured.

Run As Role Mapping

Use the Run As Role Mapping page to assign only one user to a specific role.The application is delegated to that user. Any user who knows the assigned user'sID and password can access the application.

Administrative Roles

Use the Administrative Roles page to map an administrative role to at least one user or group.

6.6.18.1.2: Securing cloned applications

In an environment containing server groups (formerly called *models*)and clones, each server group and clone must be secured individually. Securing a server group does not automatically secure its clones.

For example, if you clone an application server that contains secure enterprise applications, then you need to secure those same enterpriseapplications (if you want to) on the cloned application servers.

Secure a cloned application as you would secure any new application.

6.6.18.1.4a.4.1: Supported directory services

For a list of supported directory services, see the prerequisites Web site discussed in the article about the site. An additional Custom option is available for tailoring any of the default filters to fit a *supported* LDAP directory service.

6.6.18.1a: Summary of security settings with the Java administrative console

Use the Security Center task wizard to specify global and default security settings for all applications:

- Global settings apply to existing and future applications and cannot be customized.
- Default settings apply only to future applications and can be customized.

The default settings are used as a template or starting point for configuring individual applications. The administrator should still explicitly configure security settings foreach application.

Task	Wizard page description	Global or default?
Enable security; specify how long to cache authentication lookup results	6.6.18.1a.1: General	Global
Specify how to authenticate users	6.6.18.1a.2: Authentication	Default
Select users and groups for roles	6.6.18.1a.3: Role Mapping	Global
Assign one user to each role	6.6.18.1a.4: Run As Role Mapping	Global
Select users and groups for administrative roles	6.6.18.1a.5: Administrative Roles	Global
Making LTPA-secured calls across WebSphere domains	6.6.18.1a.6: Authentication	Global
Configuring SSL support	6.6.18.1a.7: General	Default

IBM WebSphere Application Server provides security at several levels.The security characteristics of an individual application can come fromany of these levels. At the most general level are the global securitycharacteristics set up to act as application defaults. This filebriefly describes these global values.

In WebSphere, the global defaults for security apply to allapplications. Some of the values can be changed on anapplication-by-application basis, and others remain constant acrossall applications.

An example of a value that can be set on a per-application basisis the type of authentication procedure. You must establish adefault procedure, but this value is used for applicationsthat do not explicitly indicate how they will authenticate users.

An example of value that cannot be changed on a per-applicationbasis is whether to ignore security or not. In Application Server,security is either enabled or disabled. If it is enabled, allapplications are secured according to their configurations. Ifsecurity is disabled, all applications run unsecurely, regardlessof their configurations.

6.6.18.1a.1: Enabling security with the Java administrative console

IBM WebSphere Application Server security can be enabled or not enabled. If securityis not enabled, all other security settings are ignored.

Selecting how to enable security

The administrator can enable server security by selecting the **Enable Security**check box on the **General** tabbed page of the Security Center. The administrator can use the **General** tabbed page to specify additional general settings.

6.6.18.1a.2: Specifying how to authenticate users with the Java administrative console

Use the Authentication tab of the Security Centerwizard to specify how to authenticate or verify the user data receivedas a result of a challenge (such as a logon screen).

The WebSphere security server must havesome way to check the user ID and password, digital certificate, or otheruser identification for credibility. It relies on the authenticationmechanism specified by the administrator.

Selecting how to authenticate user data

Users can be authenticated by one of two authentication mechanisms, either theoperating system user registry or Lightweight Third-Party Authentication (LTPA).

The operating system user registry simply compares users to valid usersin the underlying operating system. When the administrator selects the Local Operating Systemauthentication mechanism, the Authentication tabbed page changes to allow theadministrator to set a security server ID and password under which the application will run. Theinformation is used for delegation of the application resource.

The Local Operating System authentication mechanism supports the basic challenge type. If the administrative server is running as a non-root user, then the Local OperatingSystem cannot be used. LTPA authentication in connection with LDAP or with the Custom User Registry must be used to enable security. Similarly, if the administrative server is beingused in a multi-node configuration, LTPA authentication must be used.

When the administrator selects **Lightweight Third-Party Authentication (LTPA)** as theauthentication mechanism, the **Authentication** tabbed page changes. This change enables theadministrator to specify LTPA settings and information about the Lightweight Directory Access Protocol (LDAP)-compliant directory service product to be used, or the custom user registry. LTPA causes a search to be performedagainst the selected registry (LDAP or custom user registry). LTPA supports both the basic and certificate challenge types.

The help files that describe the OS, LTPA, LDAP, and custom user registry settings provide guidance forcompleting options on the **Authentication** tabbed page.

6.6.18.1.a.3: Selecting users and groups for roles with the Java administrative console

Use the Role Mapping tab of the Security Center wizard to assign users or groups to a particular role. Different roles can have different security authorizations. Mapping users or groups to a role authorizes those users or groups to accessapplications defined by the role.

Users, groups and roles are defined when an application is installed or configured.

Mapping users or groups to roles

The administrator maps a user or group as follows:

1. In the Role Mapping tabbed pane, the administrator selects an application.
2. Click **Edit Mapping** to open the Role Mapping dialog.
3. In the Role Mapping dialog, the administrator selects a role and clicks on **Select** to open the Select Users/Groups dialog.
4. In the Select Users/Groups dialog, the administrator selects who is authorized access for the role.
5. Click **OK** when finished mapping a user or group to a role.
6. The administrator repeats the previous two steps for other roles or as needed.
7. Click **OK** to exit the **Role Mapping** dialog.
8. Click **OK** or **Apply** on the Security Center.

6.6.18.1.a.4: Assigning users to Run As roles using the Java administrative console

Use the **Run As Role Mapping** tab of the **Security Center** wizard to assign a user only to a particular Run As role. During delegation the user assigned to the Run As Role will be used when making invocation to other methods. See InfoCenter section 5.1.4, "The WebSphere delegation model," for detail description.

Before performing this subtask

Before completing the Run As Role Mapping subtask, the administrator needs to completesubtasks in the Role Mapping tabbed pane of the Security Center and map users or groupsto the roles.

Mapping users to Run As roles

To map users to Run As roles:

1. In the **Run As Role Mapping** tabbed pane, the administrator selects an application.
2. Click **Edit Mapping** to open the **Run As Role Mapping** dialog.
3. In the **Run As Role Mapping** dialog, the administrator selects a role and clicks on **Select** to open the Select User dialog.
4. In the Select User dialog, the administrator enters the User ID/Password of a user who should havebeen granted the same role or who belongs to a group that has been granted the same role(in the Role Mapping task).
5. Click **OK** when finished mapping a user to a Run As Role.
6. The administrator repeats the previous two steps for other roles or as needed.
7. Click **OK** to exit the **Run As Role Mapping** dialog.
8. Click **OK** or **Apply** on the Security Center.

6.6.18.1.a.5: Selecting users and groups for administrative roles with the Java administrative console

Use the **Administrative Roles** tabbed page of the **Security Center** wizard to assign users or groups to the administrative role.WebSphere security model has the configuration capability to assign any user or group to have the WebSphere administrator authority. This is encapsulated with the notion of an "AdminRole" which is scoped to the WebSphere administrative application. Any user who has been granted the administrative role, or is part of a group which has been granted the administrative role, will be able to administer the WebSphere administrative domain. This role will grant such a user or a group the capability to perform any WebSphere administrative function. For example, the administrator can create a new application server, stop a running server, deploy an application, and configure security settings.

Mapping users or groups to administrative roles

The administrator maps a user or group as follows:

1. In the **Administrative Roles** tabbed pane, the administrator selects an application.
2. Click **Edit Mapping** to open the **Administrative Roles** dialog.
3. In the Select Users/Groups dialog, the administrator selects who is authorized access for the role.
4. Click **OK** when finished mapping a user or group to a role.
5. Click **OK** to exit the **Administrative Roles** dialog.
6. Click **OK** or **Apply** on the Security Center.

6.6.18.1a.6: Making LTPA-secured calls across WebSphere domains with the Java administrative console

If applications in two different WebSphere Application Server domainsneed to be able to communicate, the two WebSphere application servers mustshare security information so that the servers themselves cancommunicate. Specifically, the LTPA component of the administrativeservers in both domains must use the same LTPA key. This allows the twoservers to communicate securely with each other, and it allows the called serverto decrypt security information from the calling server. Otherwise, the WebSphereapplication server in the calling domain cannot authenticateto the application server in the called domain.

See below for an example.

This article describes the procedure for making LTPA-secured calls:

1. Generate keys
2. Export the key information
3. Make the file accessible to the second domain
4. Import the key information

Generate keys

Use the **Generate Keys** button on the **Authentication** tabbedpage to generate LTPA keys.

When LTPA keys generated, you must provide a password that is used to protected the keys. This password is required when the keys are imported from a file into another WebSphere Application Server domain.

Export the key information

You must export the calling domain's LTPA keys to a fileso that the key can be made available to another domain,where the keys are imported from the file.

Before LTPA keys can be exported, they have to be created.Such keys are typically created when security is enabled for the first time using the LTPA authentication mechanism for the domain, or can be created any time by clicking the **Generate Keys** button. When the LTPA keys are created,you must provide a password that is used to protect the keys. Thispassword is required when the keys are imported from a fileinto another application, so you *must* have this password.

To export the LTPA key information, perform these steps:

1. Start the administrative server for the domain, if necessary.
2. Start the administrative console, if necessary.
3. Click on the **Console**action bar and then choose **Security Center** from the drop-downmenu.
4. Click the **Authentication** tab in the Security Center.
5. Ensure that LTPA is selected as the authentication mechanism.
6. Click the **Export Key** button.
7. When prompted, specify the name and location of the fileto contain the LTPA keys. You can use any file name and extension.Note the name and extension you specify; this file must laterbe imported by the application in the second domain.

8. Click **Save** to save the file.
9. Click **Cancel** to close the wizard. (This procedurehas not changed any global security setting, so there are nonew settings to save.)

Make the file accessible to the second domain

The file containing the exported keys must be installed in a locationwhere the importing administrative server can find it. For example, to move thefile from one machine to another, you can put it on a floppy disk andinstall it on the second machine. This file contains security keys,so treat it with care. Some sites have policies describing howsuch transfers can be done.

Import the key information

You must import the LTPA keys of calling domain from thefile. This allows the called domain to decrypt informationencrypted by the calling domain.

To import the key information from a file, perform these steps:

1. Start the administrative server for the domain, if necessary.
2. Start the administrative console, if necessary.
3. Click on the **Console**action bar and then choose **Security Center** from the drop-downmenu.
4. Click the **Authentication** tab in the Security Center.
5. Ensure that LTPA is selected as the authentication mechanism.
6. Click the **Import Key** button.
7. When prompted, select the file that was generatedduring the export step.
8. Click **Open**.
9. When prompted, type the LTPA password established wheninitially generating the keys.
10. Click **OK** to import the keys.
11. Stop and restart the administrative server.

Example of LTPA-secured calls across domains

Suppose that a servlet running in DomainA needs to call an enterprise bean running in Domain B.Before this exchange can take place, the two WebSphere applicationservers have to exchange LTPA key information. To exchange the necessaryinformation between the two domains, three things must be done:

1. The keys for the LTPA component in the calling application'sdomain must be exported to a file. In the example scenario,the calling application is the servlet.
2. The file must be made accessible to the administrative serverof the called WebSphere Application Server domain.
3. The key information from the calling domain must be importedby the LTPA component of the called domain. In the examplescenario, the called application is the enterprise bean.

6.6.18.1a.7: Configuring SSL in WebSphere Application Server

- "What is Secure Socket Layer?" and related concepts
- Overview: WebSphere Application Server's use of SSL
- Configuring SSL for browsers
- Configuring SSL for Web servers
- Configuring SSL for IBM HTTP Server, specifically
- Configuring SSL for WebSphere plug-ins for Web servers
- Configuring SSL for WebSphere Application Server

Overview: WebSphere Application Server's use of SSL

SSL (Secure Socket Layer) is used by several WebSphere Application Server components in order to provide secure communication. In particular, SSL is used by:

- HTTPS: the application server's built-in HTTPS transport.
- ORB. the application server's client and server ORB.
- LDAPS: the admin server's secure connection to the LDAP registry used for authentication. This is available only in WebSphere Application Server Advanced Edition.

The administrative model in WebSphere Application Server allows these various SSL components to be centrally managed by configuring the *default SSL Settings*. Furthermore, any of the *default settings* can be overridden by configuring the specific SSL settings for HTTPS, ORB, and LDAPS. This provides both central administration as well as individual configurability which may be required for the various uses of SSL.

Configuring SSL for the browser

Configuring SSL for the browser is browser-specific. Consult your browser documentation for instructions.

Generally speaking, when the you type "https://..." instead of "http://...", the browser creates an SSL connection instead of a simple TCP connection to the Web server. The browser then typically either prompts the user or fails to connect if it was unable to validate the Web server or to agree upon the level of security options (the strength of the encryption algorithm to use).

Configuring SSL for the Web server

Configuring SSL for the Web server depends on the type of Web server. Consult your Web server documentation forinstructions.

Generally speaking, when SSL is enabled, an SSL key file is required. This key file should contain both the CA certificates (signer certificates) as well as any personal certificates. Client authentication can also be enabled; by default, it is disabled.

In order for the client certificate (the certificate from the browser) to be forwarded by the WebSphere Web server plug-in to the WebSphere Application Server, client authentication must be enabled for the Web server. Enabling client authentication in WebSphere Application Server itself is not required unless you want to authenticate the WebSphere Web server plug-in (or any other clients connecting directly to the WebSphere Application Server over SSL).

Configuring SSL for IBM HTTP Server, specifically

This section provides a brief example of configuring SSL for IBM HTTP Server. See the IBM HTTP Server documentation for the most recent and complete instructions. Note also that the *httpd.conf.sample* file of your Web server provides examples of all directives, including the SSL-related directives.

1. Create a keyfile using the IHS key management utility.
    1. Create a directory at a location such as *"product_installation_root/*myKeys"

       This directory will be used to hold all of your SSL key files and certificates.
    2. Start the Key Management Utility from the IBM HTTP Server start menu.

       To start this utility on a Windows platform, click: **Start -> Programs -> IBM HTTP Server -> Start Key Management Utility**
    3. Click the **Key Database File** menu and select **New**.
    4. Specify settings and click **OK**:
        - Key Database Type: CMS Key Database File
        - File Name: WebServerKeys.kdb
        - Location: The path to your "myKeys" directory
    5. Enter a password for your SSL key file (twice for confirmation).
    6. Check the "Stash the password to a file?" option. Click **OK**.

       This causes a file named "WebServerKeys.sth" to be created containing an encoded form of the password. Note that this encoding prevents a casual viewing of the password but is not highly secure. Therefore, operating system permissions should be used to prevent all access to this file by unauthorized persons.

7. When you see the list of default **Signer Certificates**, click the **Signer Certificates** menu and select **Personal Certificates**.

   If you have a server certificate from a CA (for example, Verisign), you can click **Import** to import this certificate into your SSL key file. You will be prompted for the type and location of the file containing the server certificate.

   If you do not have a valid server certificate from a CA, but want to test your system, click **New Self-Signed**.

   You will be prompted minimally to enter a **Key Label** such as "Test" and **Organization**, such as "IBM". Choose to use the default values for other values.
8. Click the **Key Database File** menu and select **Close**.

2. Add the following lines to the bottom of your httpd.conf file:

```
        LoadModule ibm_ssl_module modules/IBMModuleSSL128.dll        Listen 443        SSLEnable
Keyfile "product_installation_root/myKeys/WebServerKeys.kdb"       # SSLClientAuth required
```

   This causes the Web server to listen on port 443 (the default SSL port).

   Uncomment the last line containing "SSLClientAuth required" if you want to enable client authentication. This will cause IHS to send a request for a certificate to the browser. Your browser may prompt you to choose a certificate to send to the Web server in order to perform client authentication.
3. Start your IBM HTTP Server.
4. Test your configuration from a browser by entering a URL such as:

```
https://localhost
```

   If you are using a self-signed certificate, instead of a certificate issued by a CA such as Verisign, then your browser should prompt you to see if you want to trust the unknown signer of the server's certificate. Additionally, if you enabled client authentication, then your browser may prompt you to select a certificate to send to the Web server in order to perform client authentication. The page should then be displayed.

Configuring SSL for WebSphere plug-ins for Web servers

After SSL is working between your browser and Web server, proceed to configure SSL between the Web server plug-in and the WebSphere Application Server product. This is not required if the link between the plug-in and application server is known to be secure or if your applications are not sensitive. If privacy of application data is a concern, however, this connection should be an SSL connection.

Step 1: Creating an SSL key file for the WebSphere Web server plug-in

When configuring SSL, you must first create an SSL key file.

Note that if you are using the IBM HTTP Server, you *may* use the same SSL key file which the Web server is using; however, it is recommended that separate SSL key files be used because the trust policy for the connection to the web server will likely be different than the trust policy for the connection to the application server.

For example, we may want to allow many browsers to connect to the Web server's HTTPS port, whereas we only want to allow a small, well-known number of WebSphere plug-ins to connect directly to a WebSphere application server's HTTPS port. The following is an example of how to create an SSL key file for your WebSphere plug-in which will only allow the plug-in to connect to the application server on it's SSL port.

1. Create the directory *product_installation_root*\myKeys if you have not already done so.

   This directory will contain all of the SSL key files and extracted certificates that you will create.
2. Start the key management utility of GSKit.

   GSKit is the SSL implementation used by the WebSphere plug-in, which is the same implementation used by the IBM HTTP Server.

   The default path on Windows to this utility is C:\Program Files\ibm\gsk5\bin\gsk5ikm.exe.
3. Click the **Key Database File** pulldown and select **New**.
4. Specify settings and click **OK**:
   - **Key database type**: CMS Key Database File
   - **File name**: plug-inKeys.kdb
   - **Location**: your myKeys directory
5. Enter a password for your SSL key file (twice for confirmation).
6. Check the **Stash the password to a file?** option. Click **OK**.

   This causes a file such as *"product_installation_root*\myKeys\plug-inKeys.sth to be created containing an encoded form of the password. This encoding prevents a casual viewing of the password but is not highly secure. Therefore, operating system permissions should be used to prevent all access to this file by unauthorized persons.
7. When you see the list of default **Signer Certificates**, select the first certificate and click **Delete**.
8. Repeat the previous step until all of the signer certficates have been deleted.
9. Create a self-signed certificate:
   1. Click the **Signer Certificates** menu and select **Personal Certificates**.
   2. Click **New Self-Signed**.
   3. Enter "plug-in" for the **Key Label** and "IBM" for the **Organization**.

4. Click **OK**.

10. Extract the certificate so that you can import it into the application server key file later.
    1. Click **Extract Certificate**.
    2. Specify settings:
        - **Base64-encoded ASCII data**: Data Type
        - **Certificate file name**: plug-in.arm
        - **Location**: path to your myKeys directory
    3. Click **OK**.
11. Click the **Key Database File** menu and select **Close**.

Step 2: Modifying the WebSphere Web server's plug-in configuration file

Now that you have created the SSL key file for the plug-in, edit the plug-in configuration file so that it references your key file.

The following is an example of the plug-in configuration file. This configuration causes the plug-in to forward HTTP requests to the HTTP port of the application server, and to forward HTTPS requests to the HTTPS port of the application server.

The SSL configuration information is specified for *secureServer1*, which is the only member of the *secureServers* group. All HTTPS requests are forwarded to the *secureServers* group. (A server group is a concept that issupported only in *Advanced Edition*, not in *Advanced Single Server Edition*.)

The SSL key file is specified by the **keyring** property, and the stash file (which contains an encoded password) is specified by the **stashfile** property. Make sure that the path of this file is specified in your Web server configuration (for example, in "httpd.conf" for IHS).

```
<?xml version="1.0"?> <Config>     <Log LogLevel="Error"
Name=<"product_installation_root\logs\native.log">  <VirtualHostGroup Name="standardHost">
<VirtualHost Name="*:80"/>     </VirtualHostGroup>     <VirtualHostGroup Name="secureHost">
<VirtualHost Name="*:443"/>     </VirtualHostGroup>     <UriGroup Name="WebSphereURIs">         <Uri
Name="/servlet/snoop/*"/>         <Uri Name="/servlet/snoop"/>         <Uri
Name="/servlet/snoop2/*"/>         <Uri Name="/servlet/snoop2"/>         <Uri Name="/servlet/hello"/>
<Uri Name="/ErrorReporter"/>         <Uri Name="/servlet/*"/>         <Uri Name="/servlet"/>
<Uri Name="*.jsp"/>         <Uri Name="/j_security_check"/>         <Uri Name="/webapp/examples"/>
<Uri Name="/WebSphereSamples"/>         <Uri Name="/WebSphereSamples/SingleSamples"/>         <Uri
Name="/theme"/>     </UriGroup>    <ServerGroup Name="standardServers">         <Server
Name="standardServer1">             <Transport Hostname="localhost" Port="9080" Protocol="http"/>
</Server>     </ServerGroup>    <ServerGroup Name="secureServers">         <Server
Name="secureServer1">             <Transport Hostname="localhost" Port="9443" Protocol="https">
<Property name="keyring" value="product_installation_root\myKeys\plug-inKeys.kdb">
<Property name="stashfile" value="product_installation_root\myKeys\plug-inKeys.sth">
</Transport>         </Server>     </ServerGroup>     <Route VirtualHostGroup="standardHost"
UriGroup="WebSphereURIs" ServerGroup="standardServers"/>     <Route VirtualHostGroup="secureHost"
UriGroup="WebSphereURIs" ServerGroup="secureServers"/>     </Config>
```

The XML implementation of the plug-in configuration file could changebefore this documentation is updated again. Consult the actual configuration file installed onyour system with your current product version and fix pack level as the most current and correct version of the XML syntax.

Configuring SSL for WebSphere Application Server

The administrative console provides the following access points to SSL settings.

Use the Default SSL Settings to centrally manage SSL settings for resources in the administrative domain. Any of the default settings can be overridden in the settings for an individual resource type -- the transport, ORB, or LDAPs security settings.

- Default SSL Settings

  Open the Security Center and click **Default SSL Configuration**.
- HTTPS SSL settings for the HTTP transport of a Web container

  Edit the transport properties. In particular, select the **Enable SSL** check box.
- ORB SSL settings

  The ORB currently uses the default SSL settings.
- LDAPS SSL settings

  Use the Security Center with LTPA selected as the **Authentication Mechanism**in order to display the LDAP configuration settings. Click **SSL Configuration**.

The above ettings that can be configured through any of these SSL settings is described by the:

- SSL property reference

In the SSL settings dialog, note the **Crypto Token** button for configuring settings for supported cryptographic devices.

Configuring SSL for the application server's HTTPS transport

In order to configure SSL, you must first create an SSL key file. The contents of this file depend on whom you want to allow to communicate *directly* with the application server over the HTTPS port (in other words, you are defining the HTTPS server security policy).

This article presents a restrictive security policy, in which only a well-defined set of clients (the WebSphere plug-ins for the Web server) are allowed to connect to the application server HTTPS port. The following procedure for creating an SSL key file without the default signer certificates follows that restrictive trend.

Step 1: Create an SSL key file without the default signer certificates

1. Start IKeyMan.

   On Windows, start IKeyMan from the WebSphere Application Server entry on the Windows Start menu.
2. Create a new key database file.
   1. Click **Key Database File** and select **New**.
   2. Specify settings:
      - **Key database type**: JKS
      - **File Name**: appServerKeys.jks
      - **Location**: your myKeys directory, such as *"product_installation_root*\myKeys
   3. Click **OK**.
   4. Enter a password (twice for confirmation) and click **OK**.
3. Delete all of the signer certificates.
4. Click **Signer Certificates** and select **Personal Certificates**.
5. Add a new self-signed certificate.
   1. Click **New Self-Signed** to add a self-signed certificate.
   2. Specify settings.
      - **Key Label**: appServerTest
      - **Organization**: IBM
   3. Click **OK**.
6. Extract the certificate from this self-signed certificate so that it can be imported into the plug-in's SSL key file.
   1. Click **Extract Certificate**.
   2. Specify settings:
      - **Data Type**: Base64-encoded ASCII data
      - **Certificate file name**: appServer.arm
      - **Location**: the path to your myKeys directory
   3. Click **OK**.
7. Import the plug-in's certificate.
   1. Click **Personal Certificates** and select **Signer Certificates**.
   2. Click **Add**.
   3. Specify settings:
      - **Data Type**: Base64-encoded ASCII data
      - **Certificate file name**: appServer.arm
      - **Location**: the path to your myKeys directory
   4. Click **OK**.
8. Enter "plug-in" for the label and click **OK**.
9. Click **Key Database File**.
10. Select **Exit**.

Step 2: Add the signer certificate of the application server to the plug-in's SSL key file

1. Start the key management utility.
2. Click the **Key Database File** menu and select **Open**.
3. Select the file *product_installation_root*\myKeys\plug-inKeys.kdb.
4. Enter the associated password and click **OK**.
5. Click **Personal Certificates** and select **Signer Certificates**.
6. Click **Add**.
7. Specify settings.
   - **Data Type**: Base64-encoded ASCII data
   - **Certificate File Name**: appServer.arm
   - **Location**: the path to your myKeys directory.

8. Click **OK**.
9. Click **Key Database File** and select **Exit**.

Step 3: Reference the key file in WebSphere Application Server systems administration

Reference the appropriate SSL key file in the default SSL settings configuration panel or in the HTTPS SSL settings configuration panel. Here, we will use the default SSL settings panel.

1. Start the administrative console.
2. Open the Security Center.
3. Specify settings in the default SSL configuration.
    - **Key File Name**: *product_installation_root*/myKeys/appServer.jks
    - **Key File Password**: *enter your password*
    - **Key File Format**: JKS
    - **Trust File Name**: (empty)
    - **Trust File Password**: (empty)
    - **Client Authentication**: selected
4. Save your changes.

Step 4: Stop the servers and start them again

The configuration is complete. In order to activate the configuration, stop and restart both the Web server and the application server.

6.6.18.1a.8: Selecting users and groups with the Java administrative console

1. Display the page for selecting users and groups by clicking**Console -> Security Center** on the console menu bar, thenselecting the **Role Mapping** or **Administrative Role** tabbed page.

   You might also encounter this tabbed page as part of installingan enterprise application or module.
2. Select a role from the table and click **Select**.
3. Select who should be assigned the role.

   View properties reference for:

   - Select Users/Groups window

   At runtime, the authorization checking will grant access in the following order: **Everyone**, **All authenticated users**, and then **Select users/groups**. If a user or group is in more than one of these roles, the first match will grant access.

   If you opt to select a user or group for the role, then enter a name in the search field or enter a search pattern. For important search usage notes, see:

   - Select Users/Groups window

   After a result of the search is displayed in the **Available Users/Groups** tree view, select one or more users or groups and click **Add**.
4. Click **OK** to commit the role to user or group mapping.
5. Repeat the steps for each role that needs to be mapped.

Related references

Properties for mapping security roles and "run as" roles to users and groups

6.6.18.6: Avoiding known security risks in the runtime environment

Securing the properties files

WebSphere Application Server depends on several configuration filescreated during installation. These files contain password informationand should be protected accordingly. Although the files are protectedto a limited degree during installation, this basic level of protectionis probably not sufficient for your site. You should ensure that thesefiles are protected in compliance with the policies of your site.

The files are found in the bin and properties subdirectories in theWebSphere *<product_installation_root>*.The configuration files include:

- In the bin directory: admin.config
- In the properties directory:
  - sas.server.props
  - sas.client.props
  - sas.server.props.future

Failure to adequately secure these files can lead to abreach of security in your WebSphere applications.

Securing properties files on Windows NT

To secure the properties files on Windows NT, follow this procedurefor each file:

1. Open the Windows Explorer for a view of the files and directories on the machine.
2. Locate and right-click the file to be protected.
3. On the resulting menu, click Properties.
4. On the resulting dialog, click the Security tab.
5. Click the Permissions button.
6. Remove the Everyone entry.
7. Remove any other users or groups who should *not* be granted access to the file.
8. Add the users who should be allowed to access the file. At minimum, add the identity under which the administrative server runs.

Securing properties files on UNIX systems

This procedure applies only to the ordinary UNIX filesystem. If yoursite uses access-control lists, secure the files by using that mechanism.

For example, if your site's policy dictates that the only user who shouldhave permission to read and write the properties files is the root user,to secure the properties files on a UNIX system follow this procedurefor each file:

1. Go to the directory where the properties files reside.
2. Ensure that the desired user (in this case, root) owns each file and that the owner's permissions are appropriate (for example, rw-).
3. Delete any permissions given to the "group".

4. Delete any permissions given to the "world".

Any site-specific requirements can affect the desired owner, group andcorresponding privileges.

Risks illustrated by example applications

The level of security appropriate to a resource varies with thesensitivity of the resource. Information considered confidentialor secret deserves a higher level of security than public information,and different enterprises will assess the same information differently.Therefore, a security system needs to be able to accommodate a widerange of needs. What is reasonable in one environment can be considereda breach of security in another.

The following describes some user practices and their potential risks.When applicable, it uses components of the example application to illustrate the point.

Invoker Servlet

Purpose: The invoker servlet serves servlets by class name.For example, if you invoke /servlet/com.test.HelloServlet, the invokerwill load the servlet class (if it is in its classpath) and executethe servlet.

Security consideration: By using this servlet, a user can accessany other servlet in the application, without going through the proper channels. For example, if /servlet/testHello is a URI associated with com.test.HelloServlet, and if that URI is protected, user must beauthenticated to invokes /servlet/testHello, but any user can invoke/servlet/com.test.HelloServlet, circumventing the security on the URI.This is a security exposure if you have secured servlets installed inthe system.

Solution: Avoid installing this servlet in your configuration.

An application's error page

Purpose: In case of application errors, users are redirectedto an error page associated with the Web application. This can beany type of Web resource to which customers should be redirectedin case of an error.

Security consideration: This page should be unprotected. Ifit is protected, the server cannot authenticate the user from the context and therefore cannot send the user to the error page whenan error occurs.

Solution: Do not secure these resources.

The web application "examples"

Purpose: This application is available as part of the defaultinstallation.

Security consideration: The servlets available in this application can export sensitive information, for example, theconfiguration of your server.

Solution: The "examples" Web application should not beinstalled in a production environment.

Avoiding other known security risks

This file addresses specific problem areas. As always, periodically check the product Web site Library page for the latest information. See alsothe product Release Notes.

- To avoid a security risk, ensure that the WebSphere Application Server document root and the Web server document root are different. Keep your JSP files in the WebSphere Application Server document

root or it will be possible for users to view the source code of the JSP files.

WebSphere Application Server checks browser requests against its list of virtual hosts. If the host header of the request does not match any host on the list, WebSphere Application Server lets the Web server serve the file. Suppose the requested file is a JSP file in the Web server document root -- the JSP file is served as a regular file.

This problem has been noticed in scenarios using Netscape Enterprise Server. Due to the nature of the problem, it is possible that other Web server brands are susceptible.

- **Microsoft Internet Information Server users:**
  To use the Microsoft Internet Information Server with security enabled, in combination with IBM WebSphere Application Server security, you need to:
  - Configure IIS authentication settings to Anonymous.
  - Disable NTLM (Windows NT Challenge/Response) in the Microsoft Management Console
  - Disable Basic Authentication on the Microsoft Management Console

  Look for the setting on the Directory Security tab of the WWW Services properties.

  Problems are common when Internet Information Server NTLM is enabled along with IBM WebSphere Application Server security. The above settings are recommended to avoid problems.
- **Users of Distinguish Names (DN) in LDAP:**
  Make sure you use Distinguished Names (DNs) that your directory service product supports. Although WebSphere Application Server security supports valid LDAP DNs, some directory-service products support only a subset. For example, testing revealed that some directory services do not support all valid LDAP DNs. Specifically, a valid DN of the form OID.9.2.x.y.z=foo was rejected by one or more of the supported directory services.

  Also, directory services vary in how they represent DNs, and DNs are both case- and space-sensitive. In some cases, this leads to situations in which a user enters a valid DN and is authenticated but is still refused access. This problem is often solved by using the Common Name (the short name) rather than the full Distinguished Name.
- **Users of digital certificates with European characters:**
  If you use the iKeyman GUI tool to obtain manage certificates that contain European characters in names, the GUI will not display them. For example, a digital certificate contains the name of the company that owns the certificate and the name of the company that issued the certificate. In the US, there are companies that use symbols instead of letters in their names, like @Home and $mart $hopper. European characters in certificate names will not be displayed by the GUI.

6.6.18.7: Protecting individual application components and methods

Protecting enterprise beans after redeployment

All methods in enterprise beans and Web applications are unprotectedby default.

Security is not automatically updated when changes are made to a bean. Itwill be updated after the old application is stopped, the new application is deployed into the runtime, and the new application is started.

Adding a method to a bean

If you add a method to a bean, you must use the Application AssemblyTool to associate the new method with a role.

Modifying a method on a bean

If you modify a method on a bean, you must use the ApplicationAssembly Tool to ensure that the method still has a role associatedwith it.

Unprotecting resources

All methods in enterprise beans and Web applications are unprotected by default. If you have add a single method-to-role mapping to an enterprise-bean method, the user will be given an option to assign "DenyAllRole" role to all other unprotected methods during application installation. If the unprotected methods are assigned the "DenyAllRole" role, then these methods are protected; nobody is permitted to use them. If the unprotected methods are not assigned the "DenyAllRole" role, these methods are not protected and anyone can access those methods.

Unprotecting an entire application

During application assembly, if you have assigned roles to methods withan application, you have protected those methods. To unprotectthe methods, you can do either of the following:

- Use the Application Assembly Tool to remove the method-to-role mappings for every method in the application
- Assign the Everyone subject to all of the roles in the application, either during application installation or using the **Security Center** after installation

Unprotecting a Method

The only way to unprotect a specific method is to use the ApplicationAssembly Tool to edit the method-to-role mapping. Change the role associatedwith the method to a different role, one that is associated only withthe Everyone subject.

6.6.18.8: Using Microsoft Active Directory as an LDAP Server

To use Miscrosoft Active Directory as the LDAP server for authenticationwith WebSphere Application Server, there are some specific steps you musttake. By default, Microsoft Active Directory does not allowanonymous LDAP queries. To make LDAP queries or browse thedirectory, an LDAP client must bind to the LDAP server usingthe distinguished name (DN) of an account that belongs to theAdministrator group of the Windows system.

To set up Microsoft Active Directory as your LDAP server, followthis procedure:

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 NT/2000 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 does not have to be the same account as used in the previous step.
3. Use the WebSphere Application Server administrative console to set up the information needed to use Microsoft Active Directory:
    1. Start the administrative server for the domain, if necessary.
    2. Start the administrative console, if necessary.
    3. On the administrative console, click **Console -> Security Center** on the console menu bar.
    4. Select the **Authentication** tabbed page. On it, select Lightweight Third Party Authentication (LTPA) as the authentication mechanism.
    5. 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 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`
        - **Bind 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
    6. Click **OK** button to save the changes.
    7. Stop and restart the administrative server to make the changes take effect.

6.6.18.9: Specifying authentication options in sas.client.props

You can use the sas.client.props file to direct WebSphere ApplicationServer to authenticate users by prompting or by using a user ID and password set in the properties file. The following steps describe theprocedure:

1. Locate the sas.client.props file. By default, it is located in the properties directory under the *<product_installation_root>* of your WebSphere Application Server installation.
2. Edit the file to set up the authentication procedure:
   - To authenticate by prompting, set the loginSource property to the value "prompt":

     ```
     com.ibm.CORBA.loginSource=prompt
     ```

     Note that when using prompt, a graphical panel is presented for the user for collecting the user ID and password. Pure Java clients must call the JDK API System.exit(0) at the end of the program in order to properly end the Java process. This is because the JDK starts a backward AWT thread that is not killed when the login prompt disappears. If you choose not to use a System.exit(0) call, pressing Ctrl-C ends the process.
   - To authenticate by prompting on the console (stdout), set the loginSource property to the value "stdin". The user is then prompted for user ID and password by using a non-graphical console prompt. This is currently only supported by a pure Java client.
   - To authenticate by the values configured in the file, set the loginSource property to the value "properties" and set the desired values for the loginUserid and loginPassword properties:

     ```
     com.ibm.CORBA.loginSource=properties          com.ibm.CORBA.loginUserid=<user_ID>
     com.ibm.CORBA.loginPassword=<password>
     ```
3. Save the file.

6.6.18.10: The demo keyring

Do *not* use the demo keyring in production systems. It includesa self-signed certificate for testing purposes, and the privatekey for this certificate can be obtained easily, which puts the securityof all certificates stored in the file at risk. This keyringis intended only for testing purposes.For information on obtainingproduction certificates, see Requestingcertificates; for information on creating keyring files,see Tools for managing certificates and keyrings.(The links will only work if you are reading this as part of the InfoCenter that you can obtain fromhttp://www.ibm.com/software/webservers/appserv/infocenter.html).

6.6.18.12: Crytographic token support

To understand how to make WebSphere Application Server (both the runtime and the IKeyMan key management utility) work correctly with any crypto hardware, you should become familiar with the JSSE documentation available from the Application Server product installation:

product_installation_root/java/docs/jsse/readme.jsse.ibm.html

Be sure to unzip the file:

product_installation_root/java/docs/jsse/native-support.zip

to the appropriate location; otherwise, link errors will occur.

Follow the documentation that accompanies your device in order to install your crypto hardware. Installation instructions for IBM crypto hardware devices can be found athttp://www.ibm.com/security/cryptocards/html/library.shtml

The product supports the use of the following cryptographic devices.

These can be used by an SSL client or server:

- IBM 4758-23
- nCipher nForce
- Rainbow Cryptoswift

These can be used by SSL clients:

- IBM Security Kit Smartcard
- GemPlus Smartcards
- Rainbow iKey 1000/2000 (USB "Smartcard" device)
- Eracom CSA800

IBM HTTP Server Version 1.3.19 supports the following cryptographic devices. [Thisinformation is provided for convenience. Consultthe IBM HTTP Server Web site and documentation as the ultimate authority].

Cryptographic devices	Client or server	Interface	Operating system
Rainbow Cryptoswift	Client or server	BSAFE 3.0	Windows NT, Solaris, HP-UX
nCipher nFast	Client or server	BHAPI plugin under under BSAFE 4.0	Windows NT, Solaris
nCipher nForce accelerator mode	Client or server	BHAPI/BSAFE	Windows NT, Solaris
nCipher nForce - key storage mode	Client or server	PKCS11	Windows NT, Solaris, HP-UX, AIX, Linux
IBM4758	Client or server	PKCS11	Windows NT, AIX

 Be sure to check the WebSphere Application Server prerequisites Web sitefor the currently supported version(s) of IBM HTTP Server.