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. 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 security tasks, are described in6.6.0.3: Web administrative console overview.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 5.3, 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:

- 5.5.6: 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.

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.

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.

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.

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.

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.

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. 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.

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.

  In this edition, the registry must be the local operating-system registry.

- **Digest authentication**

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

- **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.

5.1.6: 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.
- WebSphere security, which relies on and enhances all of the above.

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 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: Tools for managing keys

WebSphere Application Server Single Server Option providestools for managing keys. The same set of tools is used tomanage digital certificates, as well, although certificatesare not supported in this version.

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.

5.5.6: 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

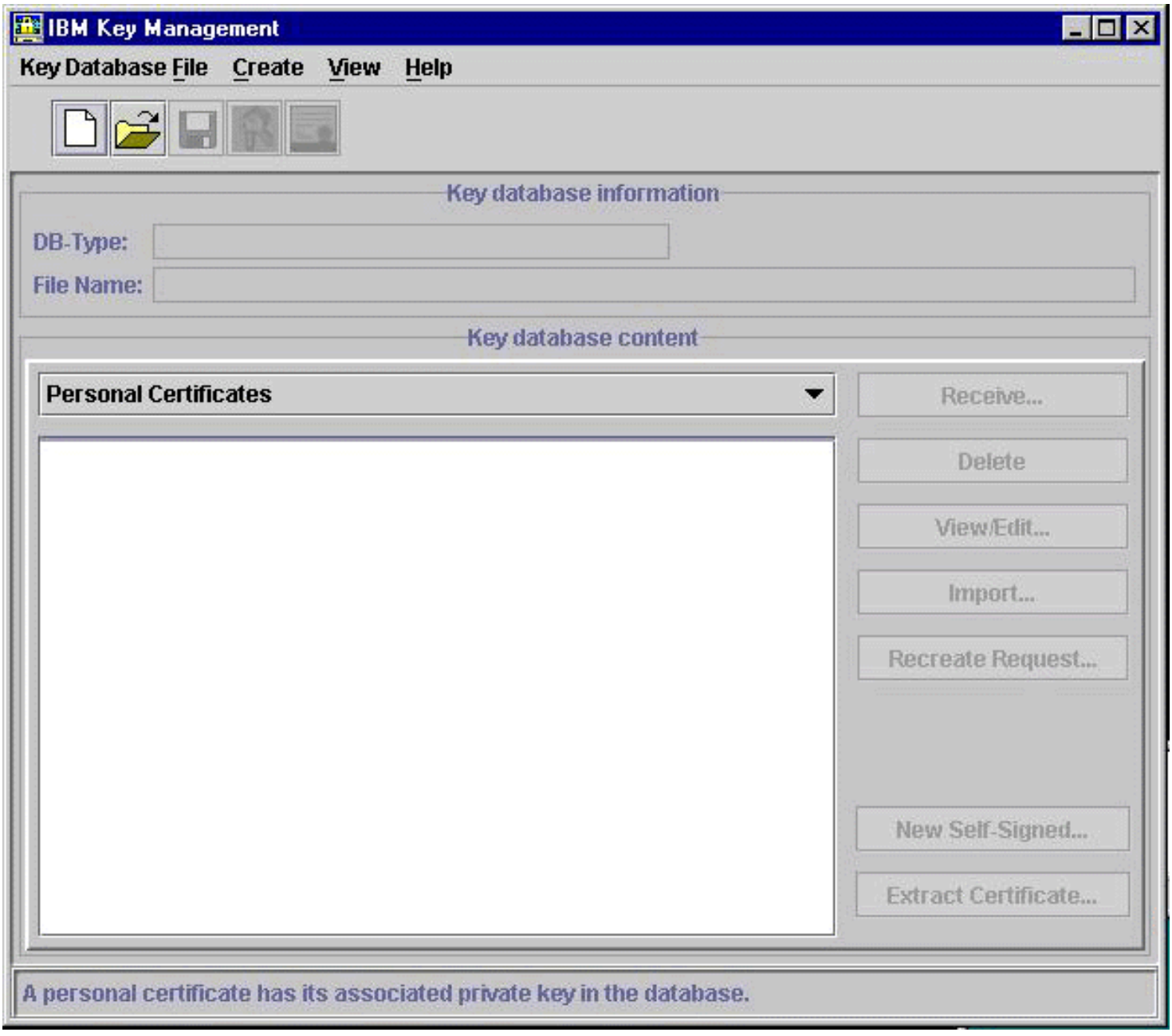
WebSphere provides a graphical tool, the IBM Key Management tool (iKeyman)for managing keys and certificates. The graphical tool is easierto use than the command-line tools, which makes it ideal for occasionalor casual use.

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 keyring file. See article 5.5.6.2.1.1, Creating a serverkeyring, for details.
2. Create a client keyring file. See article 5.5.6.2.1.2, Creating aclient keyring, for details.
3. Enable Websphere Application Server to access the client andserver keyring files. See article5.5.6.2.5, Making client and server keyrings accessible, fordetails.

5.5.6.2.1.1 Creating a server keyring

The first step in creating a self-signed test certificate is to create a serverkeyring. It contains a private key for the server for which the test certificate is beingrequested and a public key for certificate requests. To create a server keyring, completethe following steps:

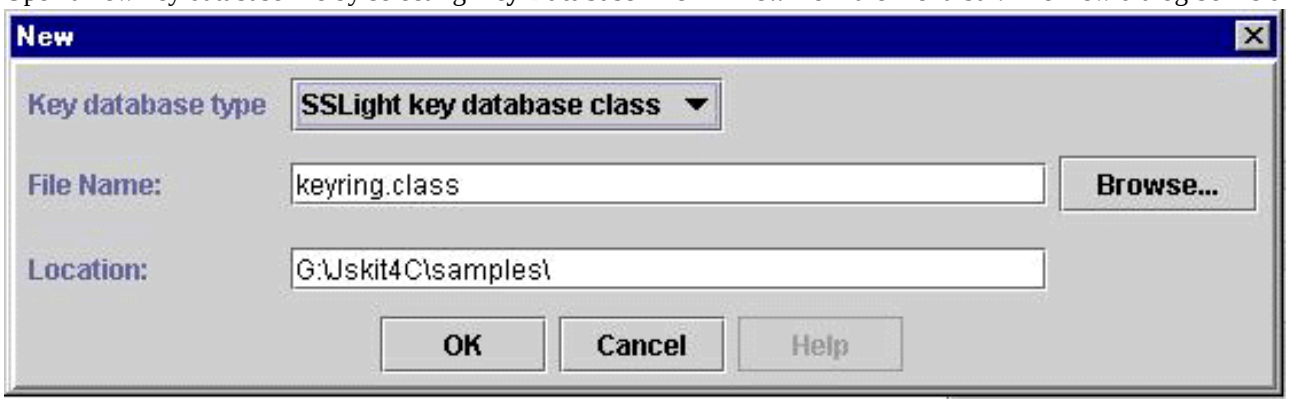
1. Start the IBM Key Management tool. See article 5.5.6.2, The IBM Key Management tool, for instructions.
2. Create a server keyring file.
3. Create a new self-signed personal certificate.
4. Export the public key from the server keyring file. This key is required by the client keyring file.

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

Create a server keyring file

To create a server 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 server keyring file. In this example, the file name is ServerKeyring.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 keyrings 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.

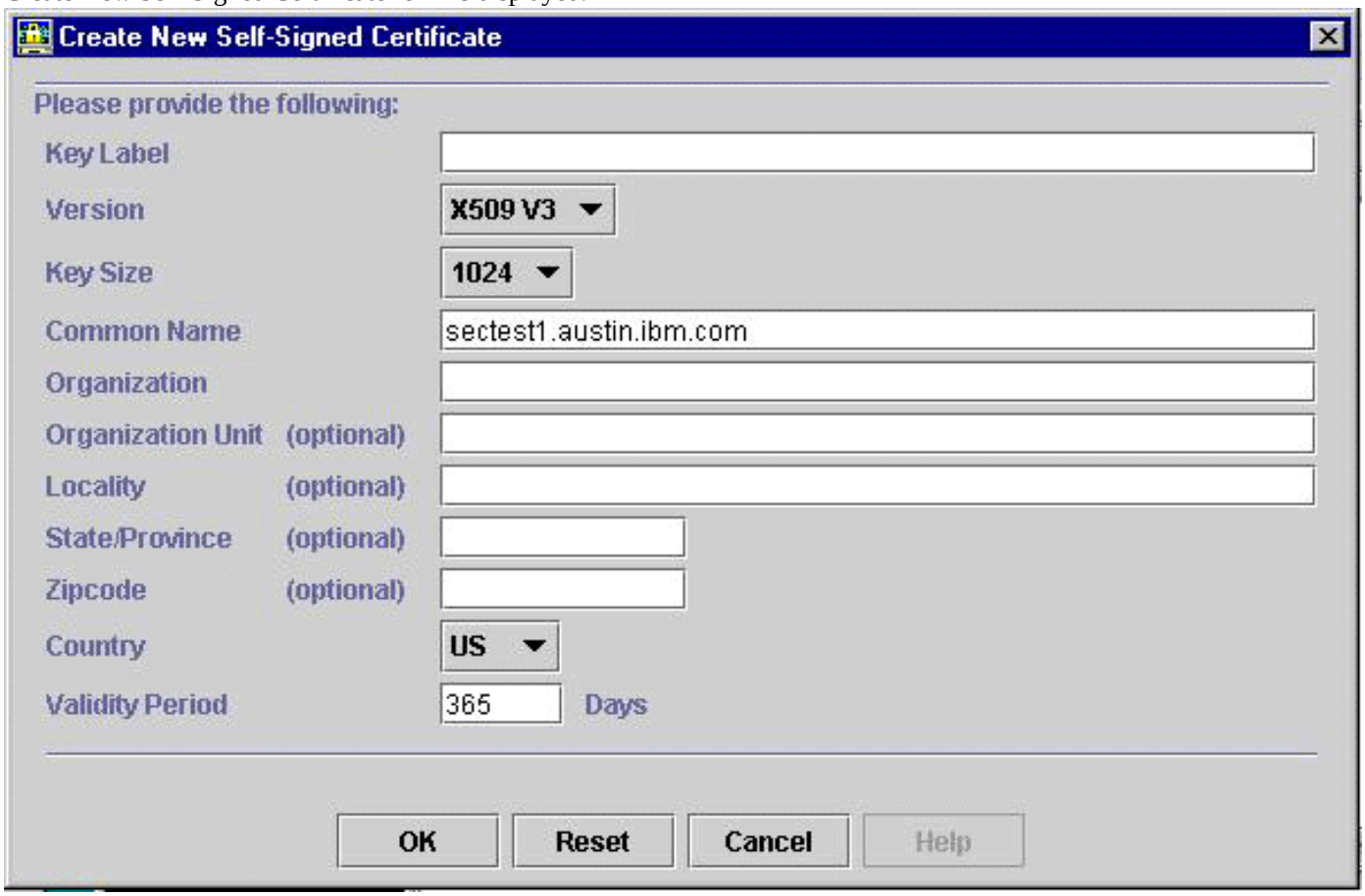
6. Click the **OK** button to continue. The tool now displays all of the available default signer certificates. These are the public keys of the most common CAs. You can add, view or delete signer certificates from this screen.

Create a new self-signed personal certificate

Creating a self-signed personal certificate creates a private key and public key withinthe server keyring file. A server keyring file contains both a private and public key. Aclient keyring file only contains the public key of the self-signed certificate, but as atrusted signer.

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-signed certificate.

Key Label

Give the certificate a key label, which is used to uniquely identify the certificate within the keyring. If you have only one certificate in each keyring, 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.
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 ServerKeyring.jks file now contains a self-signed personal certificate. You must copy the keyring file to the designated directory on the server's host.
4.

If you have only one personal certificate, it is set as the default certificate for the database. If you have more than one, you must select one as the default certificate. You can change the default certificate 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
5.

Export the public certificate

The client keyring file needs to reference the public certificate created for theself-signed personal certificate. To enable the client keyring file to use the publiccertificate, export the public certificate from the server keyring 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, the name 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 keyring

The second step in creating a self-signed test certificate is to create a clientkeyring. It is a trusted signer to the public key for the self-signed test certificate. Tocreate a client keyring, complete the following steps:

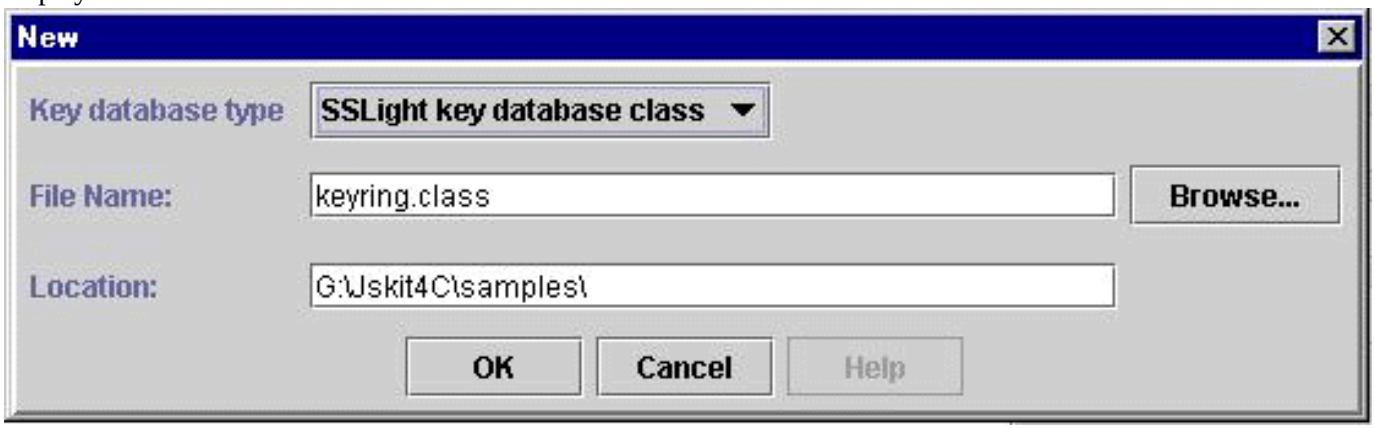
1. Start the IBM Key Management tool if you have not already done so. See article 5.5.6.2, The IBM Key Management tool, for instructions.
2. Create a client keyring file.
3. Import the public key that was exported from the server keyring file.
4. Set the certificate as a trusted root.
5. Exit the IBM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

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

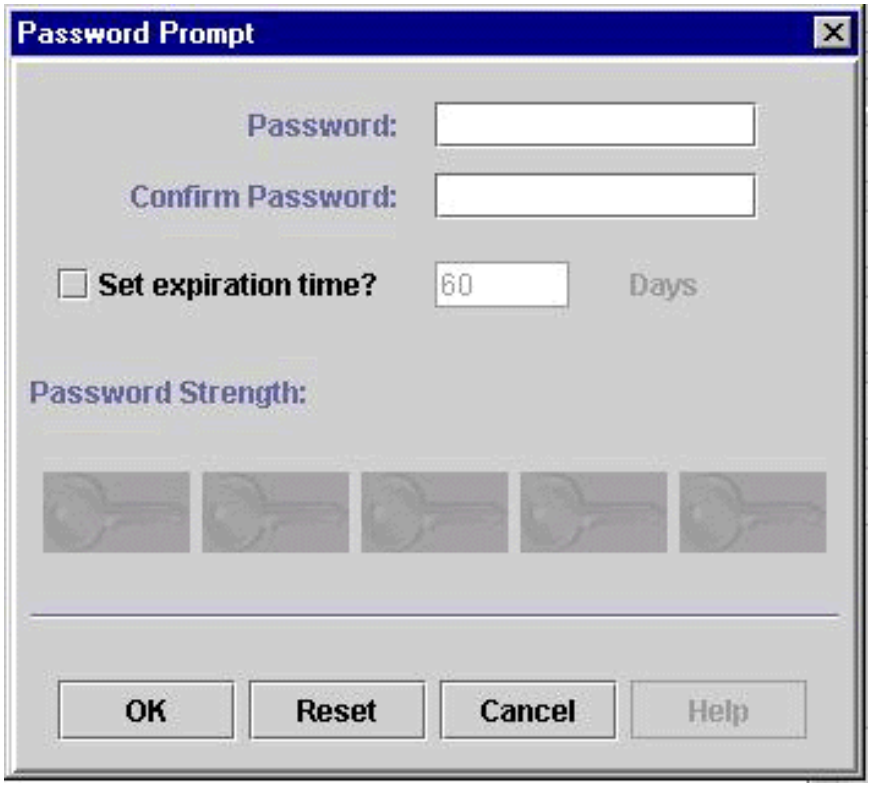
Create a client keyring file

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 ClientKeyring.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 keyrings 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 server keyring

Next, you need to import the public key certificate that was exported from the serverkeyring. (See article 5.5.6.2.1.1, Creating a server keyring.)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 trusted root

The client certificate must be a trusted root of the public key certificate that youjust 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 truster root** is checked.
4. Click **OK**.

Exit the IBM Key Management tool

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

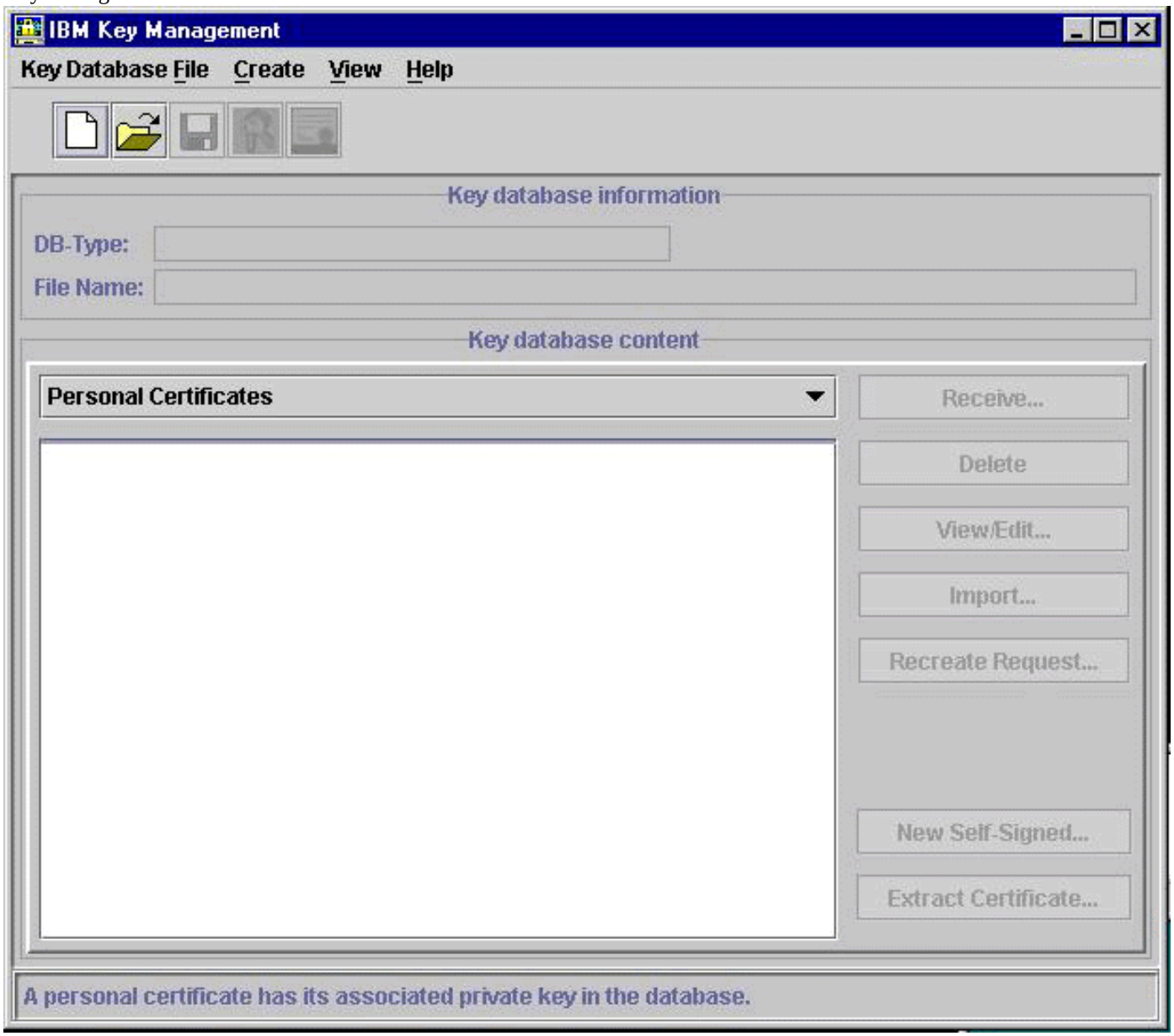
5.5.6.2.2: Creating a certification request

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.

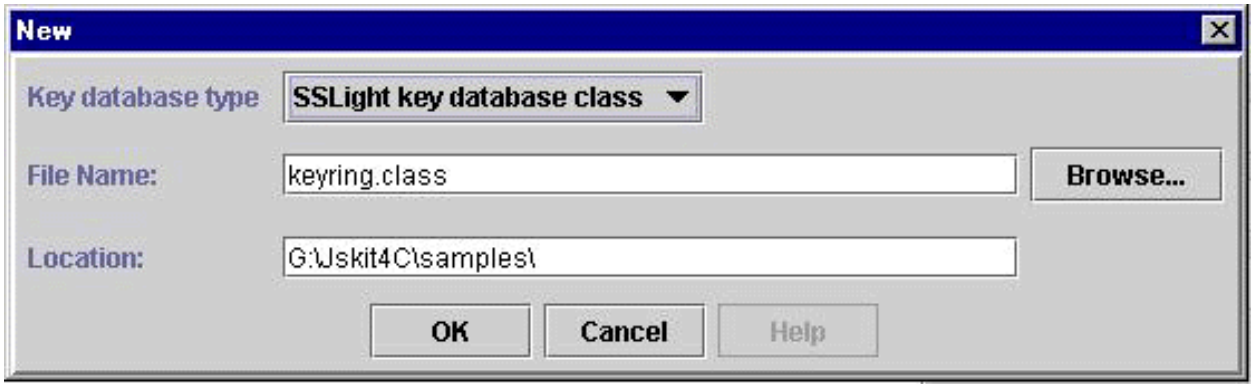
With iKeyman, generating a certificate signing request also generatesa private key for the server for which the certificate is beingrequested. The private key remains in the server's keyring class,so it stays private: the public key is included in the CSR.

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

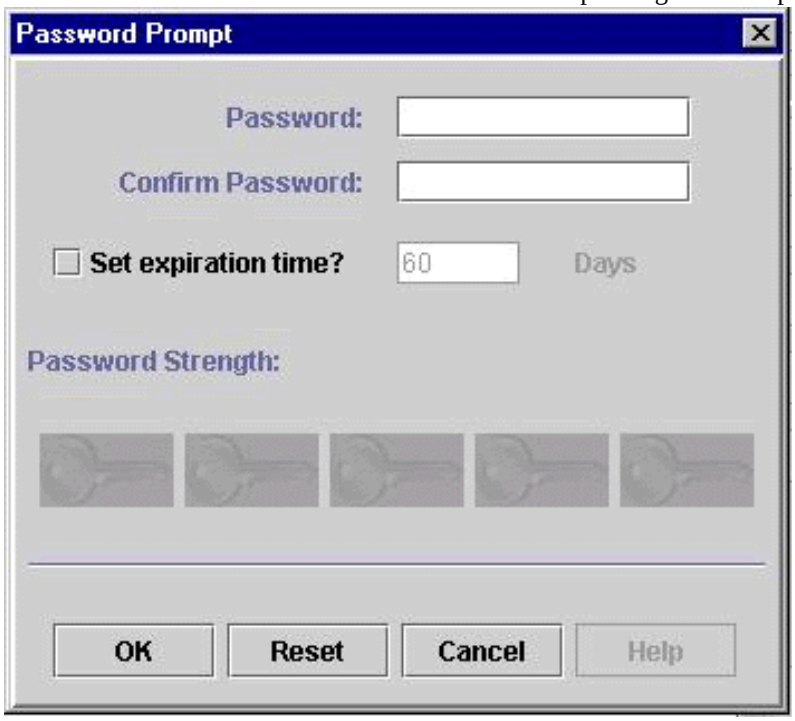
1. Start the IBM Key Management tool. See article 5.5.6.2, The IBM Key Management tool,for instructions. This displays the IBM Key Management window.



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.



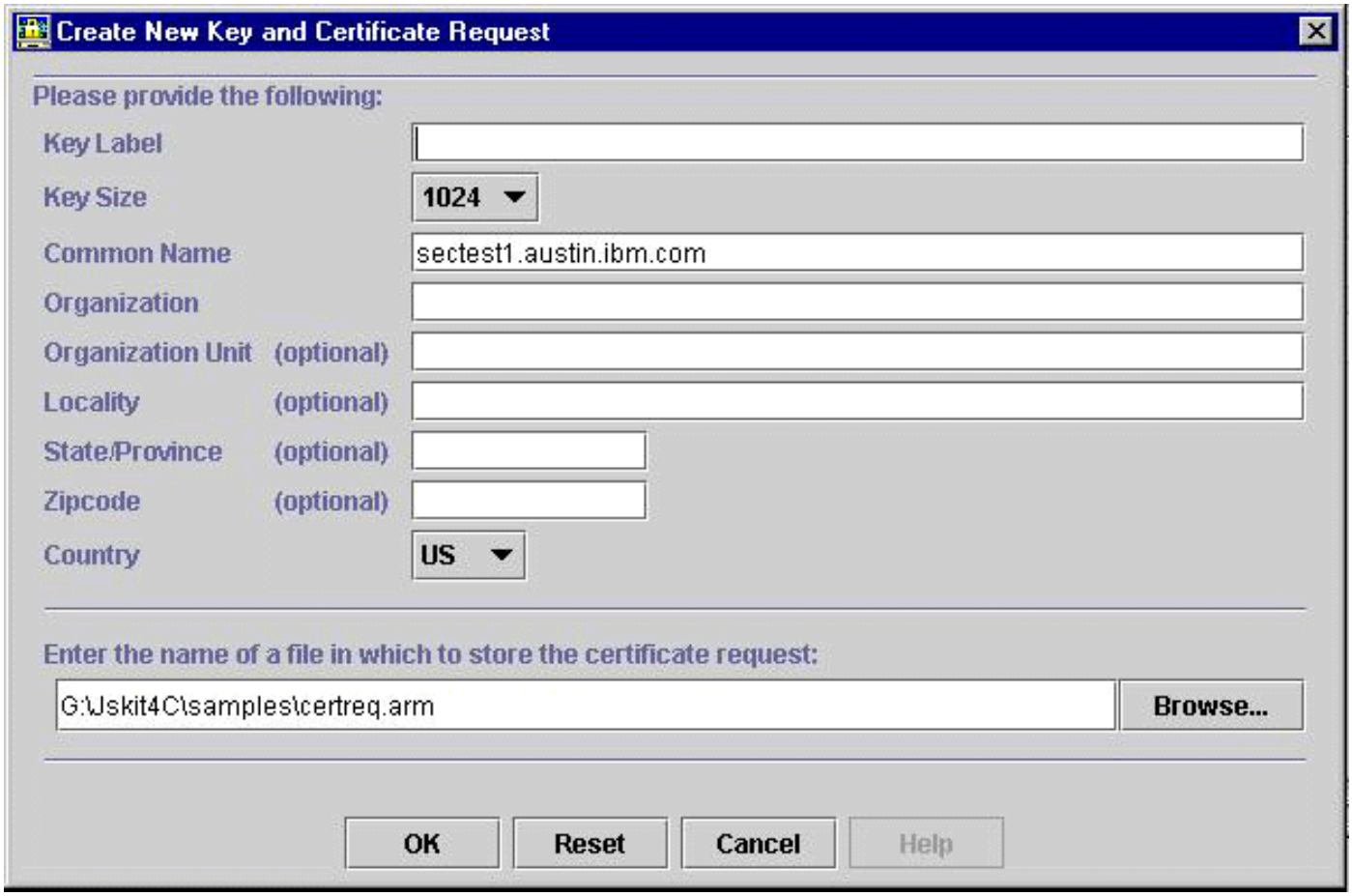
5. Click the **OK** button to continue. The Password Promptdialog box is displayed.



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.

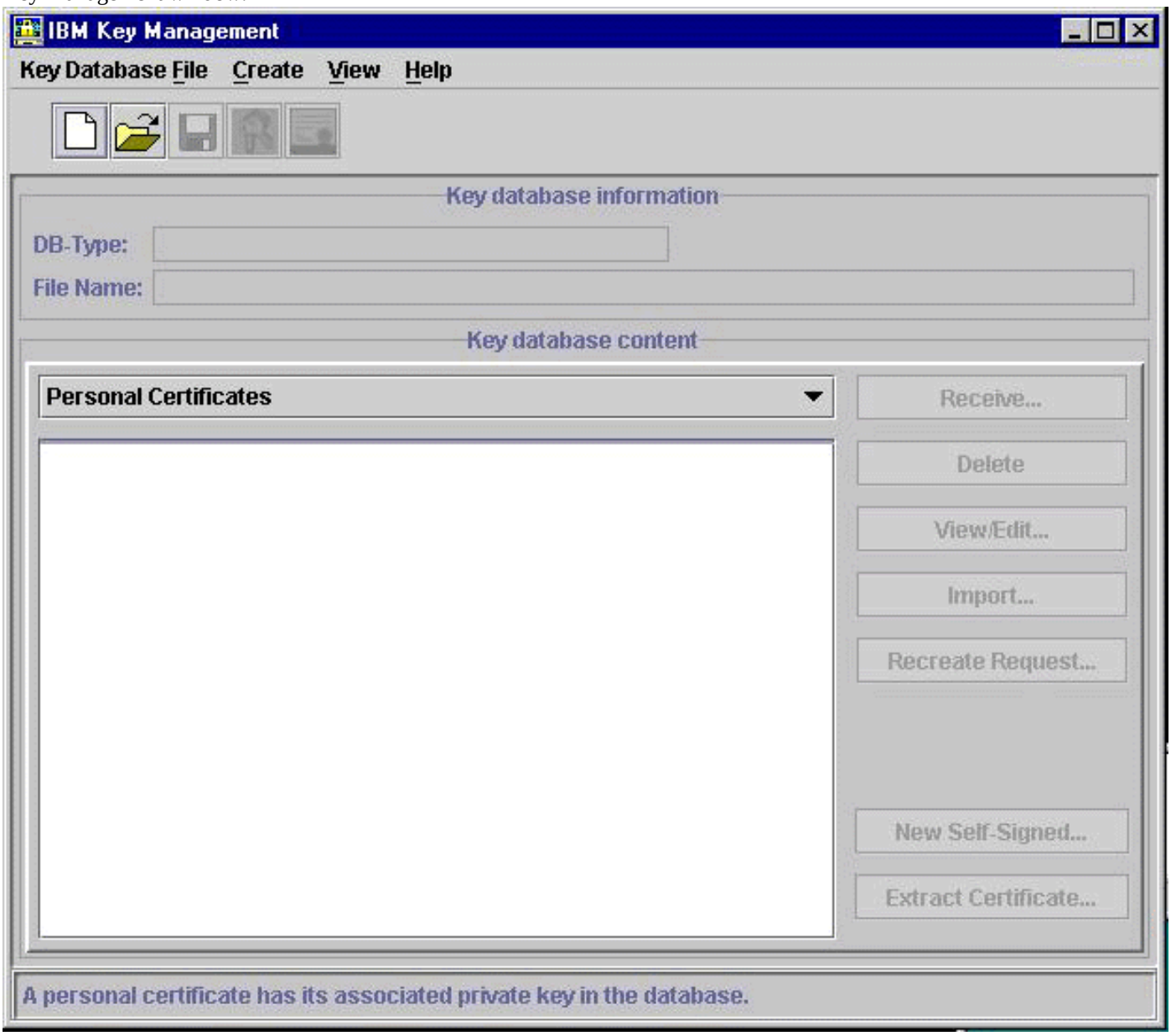
You must now submit the certificate-request file to the CA. Theprocedure will vary with the CA and with the type of certificate(test or production) being requested.

5.5.6.2.3: Placing a signed digital certificate intoa keyring

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

To place a signed certificate into a server's keyring, 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 keyring file at the prompt and click **Open**. The password prompt dialog box is displayed.
5. Enter the keyring'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 keyring. 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 keyring 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 keyrings accessible

After you have created keyring classes and inserted the necessary certificates, youneed to make the keyring classes accessible to the client and server programs.

To use created server and client keyrings in your WebSphere environment, you must firstcopy them to the client and server machines.

- Copy the client keyring file (ClientKeyring.jks) to the following location on the client machine:
  `product_installation_root/etc/ClientKeyring.jks`
- Copy the server keyring file (ServerKeyring.jks) to the following location on the server machine:
  `product_installation_root/etc/ServerKeyring.jks`

Managing the Server SSL Keyring Files

The administrative model in WebSphere Application Server allows the SSL settings foreach WebSphere component to be centrally and individually managed. SSL settings arecentrally managed in the administrative console through the default SSL Settings panel. Inaddition, any of the default settings can be overridden for an individual component byusing the HTTPS, ORB, and LDAPS SSL settings panels. See article6.6.18, Securing applications, for more detailed information about using theadministrative console to configure WebSphere security.

Always use the administrativeconsole to manage the server keyring files as changes made in the console overwrite anymanual changes to the sas.server.props file. Client keyring files are managed in thesas.client.props file because clients can be located on a remote machine.

The Default SSL Settings panel can be used to configure WebSphere Application Servercomponents using SSL. Parameters that are set through the ORB SSL Settings panel overridethe default SSL settings for the ORB. Regardless of which settings are in effect, the ORBuses these settings as follows. (Additionally, the ORB requires the SAS properties fileson the client and server to be configured as described below.)

Key file name

The path of the SSL key file used by server connections. For the server keyring file generated in this document, add the following to this field: product_installation_root/etc/ServerKeyring.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 and stored in the server-cfg.xml file.

Key file format

The only key file format currently supported by the AEs ORB is **JKS**.

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 and stored in the server-cfg.xml file. For the client keyring file generated in this document, add the following to this field:
product_installation_root/etc/ClientKeyring.jks

Trust file password

The password for the SSL trust file for client connections. On the server, the trust file password is configured in the administrative console and stored in the server-cfg.xml file.

Client Authentication

The WebSphere AEs ORB does not currently support SSL client authentication using digital certificates. Editing this value will have no effect.

Managing the Client SSL Keyring Files

You need to modify the sas.client.props file, which is located in the product installation root/properties directory. If you used"WebAS" as the password when you generated the client and server keyrings, youneed to make the following changes to the sas.client.props file:

- com.ibm.CORBA.SSLClientKeyRing=product_installation_root/etc/ClientKeyring.jks
- com.ibm.CORBA.SSLClientKeyRingPassword=WebAS
- com.ibm.CORBA.SSLServerKeyRing=product_installation_root/etc/ServerKeyring.jks
- com.ibm.CORBA.SSLServerKeyRingPassword=WebAS

You can now start your WebSphere application using the newly created keyring classes.

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)

5.7: 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/IP connectionsare automatically made. These exceptions include name-server lookups,is_a queries, and a few other special methods. SSL connectionsare always the default for business methods.

SAS gets some of the information it needs from the server's IOR. Additionalinformation is obtained from property files, one on the client sideand two on the server side (sas.server.props and server-cfg.xml). The configuration file server-cfg.xml is created from changes made in the Administrator's Console. A Java client, for example, uses the client-sideconfiguration file, but a server acting as a client uses the server-sidefiles. WebSphere Application Server provides two pre-configured propertiesfiles, sas.client.props and sas.server.props, which can be modified.Applications can also use other files. 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 by following the URL with a browser on the systemwhere the file resides. If the browser can read the file, the URL is valid. The com.ibm.CORBA.ConfigURL propertyis typically specified on the java command line of the client program by using 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 when enabling security. It can also be set manually in the $(WAS_ROOT)/config/server-cfg.xml file under the "security" tag.
- **Server SSL key file and password**: These values are set from the Administrator's Console on the server. See 5.7.3: ORB SSL Configuration. The SSL key file contains the keys used by the server to identify the server and to encrypt and decrypt messages. The strength of the encryption algorithm used is determined when the connection is established. The password protects the contents of the SSL key file.
- **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:
  - `confidentiality (high)`: 128-bit encryption and digital signing
  - `integrity (medium)`: 40-bit encryption and digital signing
  - `authenticity (low)`: no encryption and 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:
  - `confidentiality (high)`: 128-bit encryption and digital signing
  - `integrity (medium)`: 40-bit encryption and digital signing
  - `authenticity (low)`: no encryption and 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.
- `stdin`: The user is prompted for user ID and password by using a non-graphical console prompt. Currently only supported in the client properties file.
- `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 login (i.e., com.ibm.CORBA.loginSource=properties), you must specify the <*realm*> which you are trying to login to. There are two ways to do this. One is to set the `com.ibm.CORBA.principalName` in that file to `<realm>/<loginUserid>`, where the <*loginUserid*> is the same as the value in 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.

  Example: `com.ibm.CORBA.loginUserid=userid com.ibm.CORBA.principalName=REALM/userid`

  The other way to handle this is to specify the <*realm*> on the same line as <*loginUserid*>.

  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`

The only supported value for a pure client is `basicauth`. The authentication target for the server is always localos for WebSphere AE Single-Server.

- **Server associations allowed by this client**: This determines the type of association the client can establish with the server. This value is determined by setting one of the following properties to `true`. The properties are:
  ❍ `com.ibm.CORBA.SSLTypeIServerAssociationEnabled`: Type I is an SSL connection in which only server authentication is performed at the SSL transport layer. (Note: Client authentication takes place at the security-context layer with `basicauth` (user ID/password) credentials.)

  If this property is set to false, a TCP/IP connection is created, regardless of any other SSL properties specified.
- **Client keyring file and password**: These are specified using the client-side properties `com.ibm.CORBA.SSLClientKeyRing` and `com.ibm.CORBA.SSLClientKeyRingPassword`. The The keyring-file value specifies the keyring file that contains the server's public key. This key is used to encrypt outgoing messages and to decrypt incoming messages. The password protects the contents of keyring file.

This information, with the exception of the client-side keyringfile and password, is used by SAS to construct the SSL connectionto the server. During this process, the client uses the publickey in the keyring file to secure messages.

WebSphere Application Server provides a keyring file, DummyKeyring,for use in test and development environments. This keyring fileshould *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-security `confidentiality` value, the messages will be encrypted with 128-bit algorithms and digitally signed;
- If the coalesced QOP is the medium-security `integrity` value, the messages will be encyrpted with 40-bit algorithms and digitally signed.
- If the coalesced QOP is low-security `authenticity` value, only digital signing will occur.

In cases where client authentication is required but the login informationis not specified, the message is sent over an insecure TCP/IP connection.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.

Authenticating the user

When a server first receives a request, a user must be authenticated and authorized before the method can be invoked.Part of SAS's responsibility is to authenticate the user to the user registry to validate that they are who they say. TheSAS programming model has APIs for authenticating 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 is planned for a future release.

Invoking the method

Once SAS authenticates the user, a credential is created with information about the user. This credential is associatedwith the thread of execution and the method is invoked in the container after being authorized.

Sending a response back to the client

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

5.7.3: ORB SSL Configuration

The SSL implementation used by the application server is the IBM JSSE (Java Secure Sockets Extension). Configuring JSSE is very similar to configuring most other SSL implementations (e.g. GSKit); however, a couple of differences are worth noting.

JSSE allows both signer and personal certificates to be stored in a SSL key file, but it also allows a separate file, called a trust file, to be specified. A trust file can contain only signer certificates. Therefore, you could put all of your personal certificates in an SSL key file and your signer certificates in a trust file. This can be desirable, for example, if an inexpensive hardware device that is used as the key file has only enough memory to hold a single personal certificate. All of the signer certificates are 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 recognizes standard file formats such as JKS (Java Key Store). As such, SSL key files cannotbe shared between the plug-in and application server, and a different implementation of the key management utility (IKeyMan) must be used in order to manage application server key and trust files. IKeyMan can be started on Windows from the WebSphere **Start** menu.

Configuring SSL through an SSL Settings panel

The administrative model in WebSphere Application Server allows the SSL settings for each of the WebSphere components to be both centrally or individually managed. SSL settings are centrally managed through the default SSL Settings panel. Furthermore, any of the default settings can be overridden for an individual component by using the HTTPS, ORB, or LDAPS SSL settings panels.

The location of each of these SSL settings panels is as follows:

- Default SSL settingsExpand "Security", then select "Default SSL Settings"
- HTTPS SSL settingsExpand "Nodes --> the name of the host --> Application Servers --> Default Server --> Web Container --> HTTP Transports --> *.9443". Then select "SSL".
- ORB SSL SettingsExpand "Nodes --> the name of the host --> Application Servers --> Default Server --> ORB Settings". Then select "Secure Socket Layer Settings".
- LDAPS SSL Settings

As mentioned, the Default SSL Settings can be used to configure the various different components using SSL. ORB SSL Settings can be specified, in addition, to override the Default SSL Settings specifically for the ORB. Regardless of which settings are in effect, the ORB uses these settings in the following way. Additionally,the ORB requires some configuration in the SAS properties files on the client and server. These are describedin the following topics.

Key file name

The path of the SSL key file used by server connections. Any server connection (that is, listener ports) uses this key file, which should contain the server's private key. A default key file, DummyKeyring.jks is configured by default and is located in the $(WAS_ROOT)/etc directory. This file is included to simplify test and development. After a system is ready for production usage, a new keyring file should be generated by using the ikeyman.bat located in the $(WAS_ROOT)/bin directory. A self-signed cerficate can be generated, or a CA can be used to create a personal certificate trusted by most clients. The DummyKeyring contains most of the standard CA signer certficates.

On the server, the key file name is configured in the Administrative Console and stored in the server-cfg.xml file. On a pure client, the key file name is configured in the sas.client.props file under the property com.ibm.CORBA.SSLServerKeyRing.

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 and stored in the server-cfg.xml file. On a pure client, the key file password is configured in the sas.client.props file under the property com.ibm.CORBA.SSLServerKeyRingPassword.

Key file format

The only key file format currently supported by the Single-Server Orb is "JKS". The first refresh will introduce a property (mentioned in 2.4.1 above) to specify other key file formats.

Trust file name

The path of the SSL trust file used by clients. This trust file should contain signer certificates used to determine if it should trust the signer of the server's certificate. The Single-Server Orb does not support SSL client authentication. Instead, it performs client authentication using Basic Auth credentials (i.e., prompting for a userid and password). Support for mutual SSL authentication using digital certificates is planned for a future release.

On the server, the trust file name is configured in the Administrative Console and stored in the server-cfg.xml file. On a pure client, the trust file name is configured in the sas.client.props file under the property com.ibm.CORBA.SSLKeyRing.

Trust file password

The password for the SSL trust file for client connections.

On the server, the trust file password is configured in the Administrative Console and stored in the server-cfg.xml file. On a pure client, the trust file password is configured in the sas.client.props file under the property com.ibm.CORBA.SSLKeyRingPassword.

Client Authentication

The Single-Server Orb does not currently support SSL client authentication using digital certificates. It does however support basic auth client authentication where the user is prompted for a userid and password. Any protected method accessed will require Basic Auth credentials from the user before attempting to be invoked at the server. These credentials are sent over to the server where they are authenticated using the Local OS user registry.

Quality of Protection (Security Level)

In previous releases, the Orb had a different mechanism for specifying the "quality of protection" for SSL connections. There are two properties which must be configured for SSL to determine the strength of the connection. These two properties must be specified in the sas.server.props file for the server and in the sas.client.props file for the client.

For clients (including servers acting as clients), the property to specify the security level or quality of protection is:

`com.ibm.CORBA.standardPerformQOPModels=authenticity -or- integrity -or- confidentialityThe default is confidentiality.`

One of these three values can be chosen. In the first Single-Server refresh, these will be converted to low, medium, and high to be consistent with the JSSE implementation of other WebSphere components. Authenticity will be equivalent to Low, Integrity to Medium, and Confidentiality to High. The cipher suites mentioned above, associated with Low, Medium, and High will map to Authenticity, Integrity, and Confidentiality, respectively.

For servers, the property to specify the security level or quality of protection is:

`com.ibm.CORBA.standardClaimQOPModels=authenticity -or- integrity -or- confidentialityThe default is confidentiality.`

In the current implementation, regardless of the values specified on standardPerformQOPModels, clients will always make a connection at least as strong as the value specified on the server for standardClaimQOPModels. If the client standardPerformQOPModels is stronger than the server standardClaimQOPModels, that higher value will be honored.

If the security level is `confidentiality` (high) and either a server or client SSL configuration, the enabled cipher suites 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_SHA`

If the security level is `integrity` (medium) and either a server or client SSL configuration, the enabled cipher suites are:

`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 `authenticity` (low) and a server SSL configuration, the enabled cipher suites are:

`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 `authenticity` (low) and a client SSL configuration, the enabled cipher suites are:

```
SSL_RSA_WITH_NULL_MD5SSL_RSA_WITH_NULL_SHA
```

Enable Crypto Token Support

Crypto tokens are not supported by the Single-Server Orb in the initial release. The first refresh will introduce support for crypto tokens.

Dynamic Properties

Dynamic properties simply allow the configuration of some less frequently used JSSE properties. The name, possible values, default value, and a brief description of each property follows.

1. com.ibm.ssl.protocol

   In the initial release, the only supported value for this property is "SSL". The first release will allow all of these settings to be configurable.
2. com.ibm.ssl.keyStoreProvider

   In the initial release, the only supported value for this property is "IBMJCE".
3. com.ibm.ssl.keyManager

   In the initial release, the only supported value for this property is "IbmX509".
4. com.ibm.ssl.trustStoreProvider

   In the initial release, the only supported value for this property is "IBMJCE".
5. com.ibm.ssl.trustManager

   In the initial release, the only supported value for this property is "IbmX509".
6. com.ibm.ssl.trustStoreType

   In the initial release, the only supported value for this property is "JKS".
7. com.ibm.ssl.enabledCipherSuites

   In the initial release, this property is not supported. This will be configurable after the first refresh of the WebSphere Single-Server Edition.

Other Orb SSL Properties

There are a few other Orb SSL properties which determine whether SSL connections are supported incoming or outgoing on a particular server or client. These properties are com.ibm.CORBA.SSLTypeIClientAssocationEnabled and com.ibm.CORBA.SSLTypeIServerAssociationEnabled. Both of these are set to true by default indicating that both incoming and outgoing SSL connections are supported. These properties are specified in both the sas.client.props file (for pure clients) and the sas.server.props file (for servers).

```
com.ibm.CORBA.SSLTypeIClientAssociationEnabled=truecom.ibm.CORBA.SSLTypeIServerAssociationEnabled=true
```

At times, you may wish for an application server to only accept TCP connections incoming. However, any outgoing connections (client connections) may need to be SSL connections to other servers that require it. To accomplish this, set these two properties in the sas.server.props the following way:

```
com.ibm.CORBA.SSLTypeIClientAssociationEnabled=falsecom.ibm.CORBA.SSLTypeIServerAssociationEnabled=true
```

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

You determine the quantity of information collected duringtracing by setting the trace level:

- 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

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
```

To send all trace messages to the standard output stream, use thefollowing settings:

```
com.ibm.CORBA.securityDebug=consolecom.ibm.CORBA.securityTraceLevel=advanced
```

To send activity and error messages to both the standard outputstream and a file, and to send exception and trace messages tothe file only, use following settings:

```
com.ibm.CORBA.securityDebug=consolecom.ibm.CORBA.securityTraceLevel=advancedcom.ibm.CORBA.securityActivityOutputMode=bothcom.ibm.CORBA.securityErrorsOutputMode=bothcom.ibm.CORBA.securityExceptionsOutputMode=filecom.ibm.CORBA.securityTraceOutputMode=filecom.ibm.CORBA.securityTraceOutput=c:/WebSphere/AppServer/logs/sas.log
```

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

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 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 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 administrative console

com.ibm.CORBA.principalName

Specifies the principal under which the WebSphere administrative server runs.

valid values: a user name in the registry

default value: no default value

client/server usage: sas.client.props only

com.ibm.CORBA.loginSource

Indicates the source for the user IDs and passwords.

valid values: stdin, key file, prompt, properties

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

com.ibm.CORBA.keyFileName

Specifies the file containing login information.

valid values: a valid, fully qualified path and filename

default value: No default value

client/server usage: sas.server.props only.

SSL Properties

For more information on configuring SSL, see5.7.3: ORB SSL Configuration.

com.ibm.CORBA.SSLClientKeyRing

Specifies the class name for the SSL client keyring file, for example, DummyKeyring.jks. This is the keyring file used by a client for outbound SSL connections.

valid values: a class name for an SSL client keyring

default value: no default value, but a default can be set during installation

client/server usage: can be directly edited in the sas.client.props file; the server-side value must be set by using the administrative console

com.ibm.CORBA.SSLClientKeyRingPassword

Sets the password for the SSL client keyring file.

valid values: a string

default value: WebAS

client/server usage: can be directly edited in the sas.client.props file; the server-side value must be set by using the administrative console

com.ibm.CORBA.SSLServerKeyRing

Specifies the class name for the SSL server keyring file, for example, DummyKeyring.jks. This is the keyring file used by the server for inbound SSL connections.

valid values: a class name for an SSL server keyring

default value: no default value, but a default can be set during installation

client/server usage: can be directly edited in the sas.client.props file; the server-side value must be set by using the administrative console

com.ibm.CORBA.SSLServerKeyRingPassword

Sets the password for the SSL server keyring file.

valid values: a string

default value: WebAS

client/server usage: can be directly edited in the sas.client.props file; the server-side value must be set by using the administrative console

com.ibm.CORBA.SSLKeyRing

Specifies the default class name for the SSL keyring file used by both the client and the server, for example, DummyKeyring.jks.

valid values: a class name for an SSL keyring

default value: no default value, but a default can be set during installation

client/server usage: can be directly edited in the sas.client.props file; the server-side value must be set by using the administrative console

com.ibm.CORBA.SSLKeyRingPassword

Sets the password for the SSL keyring file.

valid values: a string

default value: WebAS

client/server usage: can be directly edited in the sas.client.props file; the server-side value must be set by using the administrative console

com.ibm.CORBA.SSLTypeIClientAssociationEnabled

Specifies whether SSL Type I client association is enabled or not. The value determines whether a server can accept SSL Type I connections. SSL Type I connections authenticate only the server using SSL.

valid values: false, no, true, yes

default value: true

client/server usage: sas.client.props and sas.server.props

com.ibm.CORBA.SSLTypeIServerAssociationEnabled

Specifies whether SSL Type I server association is enabled or not. The value determines whether the server permits clients to make SSL Type I server connections. SSL Type I connections authenticate only the server using SSL.

valid values: false, no, true, yes

default value: true

client/server usage: sas.client.props and sas.server.props

com.ibm.CORBA.standardClaimQOPModels

Specifies the minimum level of security protection required and supported by a server for inbound connections. The actual level of protection used on a connection is based on the server's minimum, but if the client is prepared to provide a higher level and the server supports it, the actual protection can exceed the server's stated minimum requirement.

valid values: authenticity, confidentiality, integrity

default value: confidentiality

client/server usage: sas.client.props and sas.server.props

com.ibm.CORBA.standardPerformQOPModels

Specifies the level of security protection that a client, or a server acting as a client, expects to perform on outbound connections. The actual level of protection used on a connection is based on the server's minimum, but if the client is prepared to provide a higher level and the server supports it, the actual protection can exceed the server's stated minimum requirement.

valid values: authenticity, confidentiality, integrity

default value: confidentiality

client/server usage: sas.client.props and sas.server.props

Miscellaneous properties

com.ibm.CORBA.securityEnabled

Indicates whether security is enabled or not.

valid values: false, no, true, yes

default value: false

client/server usage: can be directly edited in the sas.client.props file; the server-side value must be set by using 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

Security in the WebSphere environment offers two ways forauthentication of principals to take place:

- Basic authentication
- Authentication against the local operating system

Credential associated with each type contain different informationabout the principal.

The credentials created for basic authentication contain informationthat is not yet verified. Such credentials are typically createdon the client side of an application and sent to the server forauthentication, after which an authenticated credential is created.The basic-authorization credential contains the user ID and passwordfor the user requesting authentication.

When the server receives the basic-authorization credential duringthe establishment of a secure association, one of the other types ofcredentials is created if the information about the user canbe authenticated according to the local registry or LDAP registry.

To manipulate a credential object, an application must getaccess to a credential object. To get access to a credentialobject, an application must:

1. Acquire credentials (either by logging in or receiving them as part of an incoming request).
2. Extract the credential object:
    1. Get a reference to the security Current object.
    2. Extract the desired type of credential.

Copying a credential

Copying a credential object creates a new Credentials object that isan exact duplicate (or *deep copy*) of the original Credentials object.The method, Credentials.copy(), returns a reference to the newly createdcopy. Copying credentials is typically done when an application needs toreturn a Credentials object to a caller but does not want the caller tobe able to modify the original Credentials object.

```
   ...   // Get a reference to the security Current object.   ...   // Extract the credential
object.   creds = ...   // Make a copy of the credential object.
org.omg.SecurityLevel2.Credentials newcreds = creds.copy();   ...
```

Retrieving information from a credential

You can use the Credentials.get_attributes method. This method takesan attribute-type list as an argument, and sets the values for eachattribute type in the list. To use this method, you must firstcreate a list of attribute types. Each position in the listholds the value of a different attribute; you must construct anlist to hold the attributes you want to retrieve.

The code sample illustrates the the retrieval values for of fourattributes.This procedure demonstrates how you can acquire the security attributesof a credential. This is used to determine the security name and host identityof the principal that invoked the current method request, including thehost where the principal is logged in. This procedure is performedon a Credentials object. The security name and host name are security attributesthat have been introduced by WebSphere. Therefore, they are identifiedby the IBM_BOSS_FAMILY_DEFINER, in attributes family 2. The security runtime must be installed and the ORB must be initialized.

```
   ...   // Get a reference to the security Current object.   ...   // Extract the credential
object.   creds = ...   // Create and initialize the attribute-type list.
org.omg.Security.AttributeType[] attributeTypeList =       new org.omg.Security.AttributeType[4];
// Establish the type of attribute each index holds.   org.omg.Security.ExtensibleFamily familyOMG =
new org.omg.Security.ExtensibleFamily((short) 0, (short) 1);   org.omg.Security.ExtensibleFamily
familyIBM =       new org.omg.Security.ExtensibleFamily((short) 8, (short) 2);
attributeTypeList[0] =       new org.omg.Security.AttributeType(familyIBM,
com.ibm.IExtendedSecurity.CredAttrSecName.value);   //    new
org.omg.Security.AttributeType(familyOMG, org.omg.Security.Public.value);   attributeTypeList[1] =
new org.omg.Security.AttributeType(familyOMG, org.omg.Security.AccessId.value);
attributeTypeList[2] =       new org.omg.Security.AttributeType(familyOMG,
org.omg.Security.GroupId.value);   attributeTypeList[3] =       new
org.omg.Security.AttributeType(familyIBM, com.ibm.IExtendedSecurity.CredAttrHostName.value);   //
Make sure all values are initally null.    org.omg.Security.Attribute[] attributeList = null;try{
// Extract the attributes from the credential.   attributeList =
creds.get_attributes(attributeTypeList);   // Retrieve the securityName.   String secName = new
String(attributeList[0].value);   // Retrieve the AccessID.   String AccessID = new
String(attributeList[1].value);   // Retrieve the GroupID.   String GroupID = new
String(attributeList[2].value);   // Retrieve the HostName.   String HostName = new
String(attributeList[3].value);}catch (org.omg.Security::InvalidAttributeType e ){
```

```
e.printStackTrace();}catch (org.omg.Security::DuplicateAttributeType e ){   e.printStackTrace();}
```

WebSphere combines the CORBA.Principal and the SecurityLevel2.Credentialsinterfaces in the IExtendedSecurity.Credentials interface. TheIExtendedSecurity module contains IBM extensions to the standard interfacesdefined by the Object Management Group (OMG) and new interfaces introducedby IBM.

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.

A LoginHelper wrapper class is provided to simplify the use of the SAS programming model. For information on this LoginHelper wrapper class, see 5.4.1.2: The LoginHelper Class. SAS also has aLoginHelper class but it provides lower level login functionality. It only actually does the login and does not have any other helper methods included to manipulate the credentials like the one mentioned in section 5.4.1.2.

```
...// 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.

If authentication succeeds, the methods create the authenticated credential,which can then be set on the thread of execution, typically as theinvocation credential for further requests. A credential created by using a local registry cannot be forward to another WebSphere node.

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.

A LoginHelper wrapper class is provided to simplify the use of the SAS programming model. For information on this LoginHelper wrapper class, see 5.4.1.2: The LoginHelper Class. SAS also has aLoginHelper class but it provides lower level login functionality. It only actually does the login and does not have any other helper methods included to manipulate the credentials like the one mentioned in section 5.4.1.2.

```
...// 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.

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.

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.

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

6.6.18.1a.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 or ORB settings.

- Default SSL Settings

  In the console tree view, click **Security -> Default SSL Settings**.
- HTTPS SSL settings for the HTTP transport of a Web container

  Display the transport properties. Click **SSL**.
- ORB SSL settings

  Display the ORB settings. Click **Secure Socket Layer Settings**.

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. In the tree view, click **Security -> Default SSL Settings**.
3. Specify settings.
    - **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. Click **OK**.
5. Save your server configuration.

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.

6.6.18.3: Administering security with the Web console

Use the Web console to enable and disable global security,using the local operating system registry to authenticate users.After enabling security, access to this administrative consolewill be guarded by a login screen.

Work with security configurations by locating them in the tree on the left side of the console:

Click the **Security** entry in the tree.

6.6.18.3.1: Enabling global security with the Web console

To enable global security and configure default SSL settings:

1. In the tree on the left side of the console, click **Security**.
2. Specify the user ID and password under which the application server will run.
3. Specify the security settings. Be sure to select the check box next to **Security enabled**.
4. Click **OK**.
5. In the tree on the left side of the console, click **Security** -> **Default SSL Settings**.
6. Specify the security settings.
7. Click **OK**.
8. Save your configuration.
9. (Optional) To have the configuration take effect:
    1. Stop the server
    2. Start the server again.

6.6.18.3.3: Removing global security with the Web console

To enable global security and configure default SSL settings:

1. In the tree on the left side of the console, click **Security**.
2. Specify the security settings. Be sure that the check box next to **Security enabled** is **not** selected.
3. Click **OK**.
4. Save your configuration.
5. (Optional) To have the configuration take effect:
    1. Stop the server
    2. Start the server again.

6.6.18.3.6: Specifying user IDs for the server and administrator with the Web console

Before enabling security, specify an ID and password under which the application server will run:

1. In the tree on the left side of the console, click **Security** -> **Local OS Authentication** todisplay the authentication settings.
2. Click the **Local OS User Registry** link.
3. On the resulting panel, specify a valid **Server ID** and **Server Password**. Theuser ID-password pair must be defined already in the user registry of the local operatingsystem, and must have "Act as Operating System" privileges (on Windows NT).
4. Click **OK**.

You can use the same ID and passwordto log in to the Web administrative console, or you can set up a different ID and password.

Setting up another user ID for logging in to the Web console

To set it up so that you can log in to the administrative console using an ID and password other than the server ID and password:

1. Expand the tree on the left side of the console to display **Nodes** -> ***hostname*** -> **Enterprise Applications** -> **Server Administration Application**.
2. In the property sheet for the application, locate and click the **Mapping Roles to Users** task to display the panel for mapping roles to users.
3. For the Administrator role, specify values for the users, groups, and other settings. This is where you can specify additional IDs other than the server ID.
4. When finished changing the settings, click **Next** to display the confirmation page.
5. When finished confirming the settings, click **Finish**. The modifications will be saved to the EAR file for the application.
6. Save your configuration.
7. (Optional) To have the configuration take effect:
    1. Stop the server
    2. Start the server again.

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.

6.6.18.9: 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.

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.