# Security -- table of contents

## Overview

## Development

## Using certificates

## SSL-LDAP

## Trust association

## Administration

# 5: Securing applications -- special topics

IBM WebSphere Application Server provides security components thatcollaborate with other security elements in your WebSphere environment,as 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.

Both types of security information are supplied in the administrativeconsole for WebSphere Application Server. General administrative tasks,including standard security tasks, are described in6.6.0.3: Web administrative console overview.Information about the standard security tasks appearsin 6.6.18: Securing applications.

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

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

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

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

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

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

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

# 5.1: The WebSphere security components

Security for the WebSphere Application Server product is managed as acollaborative effort by these components distributed throughout WebSphereApplication Server:

- Security server
- Security collaborator
- Security plug-in

These components collectively make up the WebSphere securityapplication. Each is attached to a primary WebSphere Application Server component:

- The security server is attached to the WebSphere administrative server.
- The security collaborator is attached to application servers.
- The security plug-in is attached to supported Web servers.

Together, the security components provide a unified security model, allowing a single policy to govern the security of a diverse set of resources.

A realm is the domain in which a security system operates. All of the related applications reside in the same realm. The administrator sets the name for the realm, and applications and their supporting services like security operate within that single realm.

## The security server

The run-time security components (the plug-in and the collaborator)consult the security server, which controls security policies andprovides authentication and authorization services. The run-timecomponents enforce these policies.

The security server has two primary purposes:

- To centralize control over security policies such as permissions.
- To provide application-wide services like authentication and authorization.

In both of these capacities, the security server acts as atrusted third-party for security policy and control. Therun-time components consult the security server for policyinformation and for services such as authentication and authorization,including token services for Lightweight Third-Party Authentication (LTPA).

The security server provides authentication services.

The security server uses persistent storage (like a DB2database) to store its configuration information permanently.

## The security collaborator

The security collaborator is a component of the application server process, which acts as a common run-time environment for servlets, JavaServer Pages(JSP) files, and enterprise beans. When a Java client attempts toinvoke a method on a servlet or an enterprise bean, the securitycollaborator does the following:

- Performs an authorization check.
- Logs security-tracing information.
- Enforces the delegation policy.

For example, when a Web client invokes a servlet,the user is prompted for a user ID and password, and themethod invocation is passed to the application server. Thesecurity collaborator authenticates the user against theuser registry, and if the authentication succeeds, thecollaborator consults the security server to determine if theuser is authorized to invoke the servlet.

If so, the collaborator then consults the security server for anydelegation information. The security collaborator builds a securitycontext that contains the appropriate information for the user. If,for example, the delegation policy says that methods are invoked underthe user's credentials, the directly requested servlet and any methodsit calls are invoked under the user's credentials. For each indirectinvocation, the collaborator ensures that the user has permission onthe invoked method as well.

## The security plug-in

The security plug-in resides with the Web server and protectsaccess to HTML pages served by the Web server. (The securitycollaborator performs all other security checks.)When a Web resource is protected by WebSphere ApplicationServer security, the security plug-in consults the security serverfor authentication and authorization services.

For example, when a user invokes a servlet by entering itsURL in a Web browser, the target Web server receives the request.It ascertains whether the URL is protected. If the URL isprotected, it challenges the user to provide auser ID and password. The plug-in collects this informationand passes it to the security server for authentication. Ifthe authentication succeeds, the plug-in consults the securityserver to determine if the user has the permissions necessaryto access the URL. If the user is authorized, the plug-in sets upa security context for the user, including the security credentials,and passes the request to the servlet engine in the application server process. Before invoking a specific method onthe servlet, the security collaborator extracts the credentialsand verifies that the user is authorized to invoke thatmethod.

ℹ The security plug-in protects the URL name space, not thephysical files. This means that if two URLs point to the samephysical resource, security on one URL can grant accesswhile security on another URL can deny it.

IBM WebSphere Application Server Version 3.5 Security Overview is available from the IBM White Papers site.

# 5.1.1: Security features

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

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

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

**Authentication policies and services**

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

**Authorization policies and services**

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

**Delegation policies**

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

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

**A unified security administration model**

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

**Single sign-on support**

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

**Password encoding in configuration files**

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

## 5.1.2: The WebSphere authentication model

Authenticationis the process of determining if a user is who the userclaims to be. WebSphere Application Server authenticates usersby using one of several authentication mechanisms.For example, it can challenge users to provide a password, or it canrequire them to provide a digital certificate.Available authentication procedures include the following:

- **No authentication**

  If no authentication is used, users are not required to prove their identities.

- **Basic authentication**

  Basic authentication is a familiar form of authentication, in which the security service requests an identifier and password combination from a user when the user attempts to access a resource.

  After a user provides an identifier and password, the security service validates them against a database of known users , which can take the form of a simple registry or a distributed directory service. If the user-provided information is valid, the security system considers the user authenticated.

- **Digital certificates**

  Instead of requiring identifier-and-password combinations from users, an application can require users to present digital certificates, which act as electronic identification cards. The security service examines the information in the certificate to authenticate the user.

- **The form-based login challenge**

  Instead of using identifier-and-password combinations or digital certificates, application designers can write custom challenges for applications. The authentication procedure in a custom challenge can take any form the application developers can implement.

# 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, like the Java security manager, uses a capability-basedmodel for security. In WebSphere, individual resources are collectedinto applications, and methods are collected into method groups. Eachuser has a set of (application, method-group) pairs, which indicatesthe methods within an application on which the user has rights. Each(application, method-group) pair is called a permission. The WebSphereadministrator grants users access to applications by doing thefollowing:

1. Mapping sets of related resource into applications.
2. Mapping sets of related methods into method groups.
3. Granting users permissions lists.

When a user attempts to perform an operation, the security runtime determines thepermissions that will grant access. If the requesting user has at least one of thenecessary lists, the authorization check succeeds and the user is permitted to perform theoperation.

# 5.1.3.1: Securing applications and resources

Within WebSphere Application Server, you define security requirementsin terms of:

- Resources
- Applications

This file describes applications and their component resources.

## Resources

A *resource* is a specific file or program to which you wantto control access. It belongs to an application and its methods areassociated with method groups. Examples of resources include servlets,enterprise beans, Web pages, and JSP files. For example,a bank can implement accounts as enterprise beans and provide aWeb-based user interface.

Resources are divided into the following types:

**Web resources**

> Web resources represent components that can be accessed only from Web clients, like servlets, HTML files, and JSP files. The Web resource is a logical representation of a physical resource; precisely, the Uniform Resource Identifier (URI), the part of the address used to access the resource). For example, a servlet called HelloWorld and accessed through the URL http://host.com/servlet/hello is protected by protecting the URI (the logical resource) servlet/hello.
>
> A Web resource is protected only if its URI is protected, and any particular resource can have several URIs pointing to it. If a physical resource must be securely protected from Web access, then every URI that represents the physical resource must be protected. A physical Web resource can be accessed through more than one URI. To completely secure the application, all URIs must be secured.

**Enterprise-bean resources**

> An enterprise-bean resource is either a particular instance of a bean within an enterprise bean home, or the home itself (including all beans it represents). Each bean belongs to a single home, and all beans within a home are of the same class.
>
> An enterprise bean resource can belong to multiple enterprise applications. Therefore, security is expressed in terms of homes, not particular bean instances. For example, if an application has an enterprise bean called AccountBean that encapsulates account information, and if a user is given permission on the AccountBean, the user can access every account in the system. (It is not possible, at this level, to specify that a user can access only the user's account. That degree of granularity must be accomplished programmatically within the bean itself.)

## Applications

An *application* is a collection of resources that can beprotected as a unit. An application usually consists related resourcesthat encapsulate a piece of business logic. For example, a bank cancreate a Savings application to track the balances of savings accounts.This application includes the resources related to savings accounts.

Applications are divided into the following types:

**Web applications**

> A Web application is a group of servlets that share a common servlet context. URIs representing HTML and JSP files can be included indirectly in a Web application by adding the file invoker servlet (represented by "/") that services the files or the JSP file enabler ("/*.JSP file") to the Web application.

A Web application can be added to an enterprise application, but a specific Web application can belong to only one enterprise application.

### Enterprise applications

An enterprise application is a collection of Web applications and optional enterprise-bean resources. Administrators can define authentication and authorization policies for enterprise applications.

A Web application can belong to only one enterprise application, but enterprise-bean resources can belong to multiple enterprise applications.

# 5.1.4: The WebSphere delegation model

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, 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, every enterprise application (acollection of resources) can have an associated identity. Therefore, youcan use the specified-identity delegation policy to run beans under theidentity of the application to which they belong.

## Creating a delegation policy

In WebSphere Application Server, the delegation policy is determined byvalues associated with the methods of enterprise bean. The initialvalues are retrieved from the deployment descriptor of a bean,but the administrator can modify them. Delegation policiescan be applied jointly to all the methods in a bean orseparately to individual methods.

A delegation policy is created by setting the values ofone (or two) attributes:

- **RunAsMode**: determines the identity under which methods art to be run. The valid values follow:
  - `CLIENT_IDENTITY`
  - `SYSTEM_IDENTITY`
  - `SPECIFIED_IDENTITY`
- **RunAsIdentity**: specifies the principal when the RunAsMode is `SPECIFIED_IDENTITY`.

# 5.1.6: Relationship to the operating environment

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

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

The types of security involved include:

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

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

# 5.3: Changes to security since Version 2.0x

Some security features have changed with respect to the security offered by IBMWebSphere Application Server Version 2.0x. This table summarizes the differences.

| Version 3/3.5 | Version 2 |
|---|---|
| Users and groups must originate in a directory service or user registry. | Users and groups could be created directly in WebSphere Application Server, independent of a directory service product or the user registry of the operating system. |
| You protect resources individually and at the application level. The security properties at the application level differ from those you set for individual resources. | Individual resources were secured, but the product did not offer protection to applications (collections of related resources). |
| Enterprise beans are protected. The *method group* concept discussed in the Enterprise JavaBeans (EJB) specification Version 1.1 is integral to security policy for all types of resources in Version 3+. | Only servlets and other Web files, such as HTML pages and JavaServer Pages (JSP) files, were protected. |
| Method groups and application-level security define authorization policies. | Realms and access-control lists defined authorization policies. |
| There is just one realm, to which all items belong. The administrator names the realm. | Multiple realms were offered. |
| A discrete security server process provides centralized security services and policy enforcement to one or more application server runtimes. | Security features were part of a single application server runtime. |
| Sophisticated functionality, including Single Sign On (SSO), delegation, and the use of LTPA and digital certificates, is supported. | Basic security policy and services were provided. |

# 5.4: Overview: Using programmatic and custom logins

This section describes the use of programmatic login techniques and custom challenge capabilities in WebSphere Application Server.

When applications require 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 can be prompted for authentication data in many ways. The challengetype configured for the application 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 custom challenge to provide an application-specificHTML form for collecting login information.

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 work will occur unless WebSphere security is enabled.Additionally, if you want to use the custom challenge type,you must configure security as follows (click an item to linkto detailed property help for the item):

- Set the challenge type to Custom.
- Set the authentication mechanism to LTPA.
- Set the LoginURL and ReloginURL field to the URL of your HTML login form.
- Enable Single Sign-On, if used.

## 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>/hosts/default_host/examples/security/LoginHelper.java
```

and the class file is installed at:

```
<installation_root>/servlets/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`*`>/hosts/default_host/examples/security/ServerSideAuthenticator.java`

and the class file is installed at:

`<`*`installation_root`*`>/servlets/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.

   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's customchallenge type. A custom challenge works in the following manner:

1. An unauthenticated user attempts to use a resource secured with a custom challenge.
2. The user is redirected to the LoginURL, which takes the user to an HTML form that collects authentication information.
3. The user inserts his or her user ID and password into the form and submits it.
4. The submission triggers a servlet that authenticates the user.

WebSphere provides two servlets that can be used as a basis for writingcustom-challenge servlets, an AbstractLoginServlet and the CustomLoginServlet, whichextends the AbstractLoginServlet.

In web-based applications, it is often desirable to maintain login information acrossmultiple sites so each site doesn't have to require the user retype the information. Youcan use the WebSphere single sign-on framework to allow the authentication information tobe passed along automatically. WebSphere provides a helper class called SSOAuthenticatorthat simplifies the handling of single sign-on.

# 5.4.3.1: The AbstractLoginServlet class

The AbstractLoginServlet class that encapsulatesall of the functionality of a server side login.

The following list summarizes the public methods in the AbstractLoginServletclass. The source and class files are installed in the directory`<installation_root>/servlets/`.

**init()**

This method initializes the servlet.

*Syntax:*

```
void init() throws ServletException
```

**login()**

This authenticates the user and, if requested, sets up the necessary information for single sign-on.

*Syntax:*

```
Object login (HttpServletRequest req, HttpServletResponse res,                     boolean setSSO)
throws ServletException
```

**postLogin()**

This is an abstract method.

*Syntax:*

```
void postLogin(HttpServletRequest req, HttpServletResponse res)      throws ServletException
```

**logout()**

This destroys the user's credentials and, if necessary, unsets the single sign-on information.

*Syntax:*

```
void logout(HttpServletRequest req, HttpServletResponse res,                    Object retCreds)
throws ServletException
```

## 5.4.3.2: The CustomLoginServlet class

The CustomLoginServlet extracts the user ID, password, and redirect URLinformation from the HTML form by which the user logged in. The redirectURL specifies the Web site to which the user is requesting access.The CustomLoginServlet invokes the necessary methods on its parent servlet,AbstractLoginServlet.

The following list summarizes the public methods in the CustomLoginServletclass. The source and class files are installed in the directory`<`*`installation_root`*`>/servlets/`.

**init()**

This method initializes the servlet, reading the default redirect URL if it exists. The default redirect URL specifies the Web page to which the user will be forwarded if authentication is successful.

*Syntax:*

```
void init(ServletConfig conf) throws ServletException
```

**doPost()**

The primary entry point into the servlet, this method is designed to be called as the result of an HTML form post. The method reads and validates the posted parameters, then calls the login method in the base class LoginServlet.

*Syntax:*

```
void doPost(HttpServletRequest req, HttpServletResponse res)      throws ServletException,
IOException
```

**postLogin()**

This method is called after performing a successful login, so it is a good place to establish an HTTP session or perform other actions related to the logged-on user. This method runs under the identity of the user.

*Syntax:*

```
void postLogin(HttpServletRequest req, HttpServletResponse res)      throws ServletException
```

## 5.4.3.3: The SSOAuthenticator class

The SSOAuthenticator class is a WebSphere-provided utility class thatcan be used by servlet developers to write custom-login servlets.

The following list summarizes the public methods in the SSOAuthenticator class. The class files is installed at`<installation_root>/servlets/SSOAuthenticator.class`.

**SSOAuthenticator()**

The constructor creates an SSOAuthenticator object and initializes it based on the SSO configuration within WebSphere Application Server. The folliwing conditions must be met for successful construction:

- WebSphere security is enabled
- LTPA is selected as the authentication mechanism
- Single Sign-On (SSO) is enabled

*Syntax:*

```
SSOAuthenticator() throws IllegalStateException
```

**login()**

These methods create an LPTA cookie and set the cookie on the HTTP response header. The first method takes a boolean argument, force_auth, whose value determines if the user (based on the identifier and password) is authenticated. If force_auth is TRUE, authentication occurs; if not, only the identifier and password are used in the cookie.

The second login method does not take the force_auth argument. It always attempts authentication and is equivalent to calling the first with the force_auth argument set to TRUE.

Both methods return CORBA security credentials.

*Syntax:*

```
org.omg.SecurityLevel2.Credentials login(String userID, String password,
HttpServletRequest req,                                  HttpServletResponse res
boolean force_auth)     throws org.omg.SecurityLevel2.LoginFailedorg.omg.SecurityLevel2.Credentials
login(String userID, String password,                                HttpServletRequest
req,                                    HttpServletResponse res)     throws
org.omg.SecurityLevel2.LoginFailed
```

**logout()**

This method logs the user out. After this method runs, the user must be authenticated again before making any additional requests.

*Syntax:*

```
void logout(HttpServletRequest req, HttpServletResponse res)
```

# 5.5: Certificate-based authentication

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

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

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

Certificate-based authentication relies on several relatedtechnologies:

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

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

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

## Trust

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

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

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

## Protection from forgery

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

## Use of certificates by owners

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

# 5.5.1: Introduction to public-key cryptography

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

## Private-key encryption

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

- It requires a key for every pair of individuals who need to communicate privately. The necessary number of keys rises dramatically as the number of participants increases.
- The fact that keys must be shared between pairs of communicators means the keys must somehow be distributed to the participants. The need to transmit secret keys makes them vulnerable to theft.
- Participants can communicate only by prior arrangement. There is no way to send a usable encrypted message to someone spontaneously. You and the other participant must have made arrangements to communicate by sharing keys.

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

## Public-key encryption

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

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

- Public-key encryption requires only two keys per participant. The total number of keys rises much less dramatically as the number of participants increases than it does in private-key encryption.
- The need for secrecy is more easily met. The only thing that needs to be kept private is the private key, and since it does not need to be shared, it is less vulnerable to theft in transmission than the shared key in a private-key system.
- Public keys can be published. This eliminates the need for prior sharing of a secret key before communication. Anyone who knows your public key can use it to send you a message that only you can read.

Public-key encryption is also called *asymmetric* encryption, becausethe same key cannot be used to encrypt and decrypt the message. Instead,one key of a pair is used to undo the work of the other. WebSphere ApplicationServer uses the RSA public/private key-encryption algorithm.

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

# 5.5.2: Introduction to digital signatures

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

- The integrity of the message: Is the message intact? That is, has the message been modified between the time it was digitally signed and now?
- The identity of the signer of the message: Is the message authentic? That is, was the message actually signed by the user who claims to have signed it?

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

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

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

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

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

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

# 5.5.3: Introduction to digital certificates

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

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

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

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

## Contents of a digital certificate

A certificate contains several pieces of information, includinginformation about the owner of the certificate and the issuing CA.Specifically, a certificate includes:

- The *distinguished name* (DN) of the owner. A DN is a unique identifier, a fully qualified name including not only the *common name* (CN) of the owner, but the owner's organization and other distinguishing information.
- The public key of the owner.
- The date on which the certificate was issued.
- The date on which the certificate expires.
- The distinguished name of the issuing CA.
- The digital signature of the issuing CA. (The message-digest function is run over all the preceding fields.)

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

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

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

## Requesting certificates

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

- The distinguished name of the owner (the user for whom the certificate is being requested).
- The public key of the owner.
- The digital signature of the owner.

The message-digest function is run over all these fields.

The CA verifies the signature with the public key in the requestto ensure that the request is intact and authentic. The CA thenauthenticates the owner. Exactly what the authentication consists ofdepends on a prior agreement between the CA and the requestingorganization. If the owner in the request is successfully authenticated,the CA issues a certificate for that owner.

## Using certificates: Chains of trust and self-signed certificates

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

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

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

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

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

# 5.5.4: Requesting certificates

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

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

## Time requirements

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

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

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

## Requirements on the format of information

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

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

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

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

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

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

# 5.5.4.1: Getting a test certificate from acertificate authority

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

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

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

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

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

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

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

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

# 5.5.4.2: Getting a production certificate from a certificate authority

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

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

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

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

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

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

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

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

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

11. Review the Server Certificate Agreement. To accept the conditions and submit your request, click the Accept button. If reject the conditions, click the Decline button.

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

# 5.5.4.3: Using test certificates

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

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

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

# 5.5.5: Mapping certificates to users for client authentication and authorization

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

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

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

## Mapping by exact distinguished name

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

The mapping is case insensitive. For example, the following twoDNs match on a case-insensitive comparison:

`"cn=Smith, ou=NewUnit, o=NewCompany, c=us""cn=smith, ou=newunit, o=NewCompany, c=US"`

If a match is found, authentication succeeds, and if nomatch is found, authentication fails.

## Mapping by filtering certificate attributes

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

If you are matching the Subject CN field in the certificate to theuid attribute of the LDAP entry, a certificate with the Subject DN `"cn=Smith, ou=NewUnit, o=NewCompany, c=us"`matches an LDAP user entry with uid=Smith.

To use this mapping technique, you must request CertificateMapping and set up the certificate filter in the administrative console.

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

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

# 5.5.6: Tools for managing certificates and keys

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

- A graphical tool, called iKeyman, the IBM Key Management tool.
- A package of Java command-line tools, com.ibm.cfwk.tools. The package contains four tools:
    - KeyGenTool, which generates keys
    - MakeCertRequest, which generates certificate requests
    - MakeCertTool, which generates certificates
    - VaultTool, which is used to manage the certificate database

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.

# 5.5.6.1: The CFWK tools for certificate and key management

WebSphere provides command-line tools for managing certificates and keyringsin the com.ibm.cfwk.tools Java package, which contains four tools:

- KeyGenTool, which generates keys
- MakeCertRequest, which generates certificate requests
- MakeCertTool, which generates certificates
- VaultTool, which is used to manage the certificate database

You can do all the necessary work for managing certificatesand keyrings with these tools. The first three tools aresingle-purposes commands; there is only one way to run each.The last, VaultTool, supports of suite of subcommands.

## Using the tools

To use the certificate-management tools, you must put the filescfwk.zip and cfwk-tools.zip at the front of your classpath.These files are located in the AppServer/lib directory of theWebSphere installation. For example, on Windows NT, set the CLASSPATHvariable as shown:

```
set
classpath=<WS-install>\AppServer\lib\cfwk.zip;<WS-install>\AppServer\lib\cfwk-tools.zip;%CLASSPATH%
```

# 5.5.6.1.1: The KeyGenTool tool

This command creates a pair of RSA keys (private and public) andplaces them in a file.

*Syntax:*

```
java com.ibm.cfwk.tools.KeyGenTool --forge U_keyalg U_keyfile
```

## Arguments:

**U_keyalg**

The key algorithm: "RSA/512/F4" or "RSA/1024/F4"

**U_keyfile**

The file in which to place the key

## Examples:

```
java com.ibm.cfwk.tools.KeyGenTool --forge
"RSA/512/F4"D:\projects\websphere\keyrings\WebAS.TestServer.keyGenerating key. Might take a while...
```

The example above creates keys for the server called TestServerand places them in a file called WebAS.TestServer.key.

```
java com.ibm.cfwk.tools.KeyGenTool --forge
"RSA/512/F4"D:\projects\websphere\keyrings\WebAS.TestCA.keyGenerating key. Might take a while...
```

The example above creates keys for a local, non-production CA calledTestCA and places them in a file called WebAS.TestCA.key.

ℹ This is a general-purpose tool with applications beyond those discussedhere. This page discusses only the subset of options relevant tomanaging certificates for WebSphere Application Server programs.

# 5.5.6.1.2: The MakeCertRequest tool

This tool generates a certification request, which can be sent toa certificate authority (CA).

*Syntax:*

```
java com.ibm.cfwk.tools.MakeCertRequest [--from U_fromdate]
[[--to U_todate] | [--for U_for]]                                        U_X500Name
U_keyfile                                        U_signalg
U_csrfile
```

## Arguments:

**--from U_fromdate**

The start date for the certificate, in the format dd/mm/yyyy

**--to U_todate**

The expiration date for the certificate, in the format dd/mm/yyyy

**--for U_for**

The lifetime of the certificate, in the format *n*d, where *n*is an integer and d stands for "days"

**U_X500Name**

The distinguished name of the owner, in valid X.500 format

**U_keyfile**

The file holding the owner's public key

**U_signalg**

The signature algorithm, either "MD5 with RSA" or "SHA1 with RSA"

**U_csrfile**

The file in which to store the certification request

ℹ This is a general-purpose tool with applications beyond those discussedhere. This page discusses only the subset of options relevant tomanaging certificates for WebSphere Application Server programs.

# 5.5.6.1.3: The MakeCertTool tool

This command generates a certificate.

*Syntax:*

```
java com.ibm.cfwk.tools.MakeCertTool [--serial U_serial]                                     [--from
U_fromdate]                                         [[--to U_todate] | [--for U_for]]
[--issuer U_X500Name]                                         [--subject U_X500Name]
[--sign-alg  U_signalg]                                         [--sign-key U_keyfile]
[--subject-key U_keyfile]                                         [--cert-file U_certfile]
```

## Arguments:

**--serial U_serial**

A serial number to be placed in a certificate, to uniquely identify the certificate

**--from U_fromdate**

The start date for the certificate, in the format dd/mm/yyyy

**--to U_todate**

The expiration date for the certificate, in the format dd/mm/yyyy

**--for U_for**

The lifetime of the certificate, in the format *n* d, where *n*is an integer and d stands for "days"

**--issuer U_X500Name**

The distinguished name of the issuer, in valid X.500 format

**--subject U_X500Name**

The distinguished name of the owner, in valid X.500 format

**--sign-alg U_signalg**

The signature algorithm, either "MD5 with RSA" or "SHA1 with RSA"

**--sign-key U_keyfile**

The file containing the issuer's public key

**--subject-key U_keyfile**

The file containing the owner's public key

**--cert-file U_certfile**

The file to hold the generated certificate

## Examples:

```
java com.ibm.cfwk.tools.MakeCertTool --serial 0 --for 2y--issuer "cn=WebAS Test CA, OU=SWG, O=IBM,
c=US"--sign-alg "MD5 with RSA"--sign-key d:\projects\websphere\keyrings\WebAS.TestCA.key--cert-file
d:\projects\websphere\keyrings\WebAS.TestCA.certCreating certificate...
```

The example above creates a self-signed certificate for TestCA. Ituses the MD5 digest function and TestCA's public key to sign thecertificate. (Since the subject and the issuer are the same, thecertificate is "self-signed.") The resulting certificate isstored in a file called WebAS.TestCA.cert.

```
java com.ibm.cfwk.tools.MakeCertTool --serial 0 --for 2y--issuer "cn=WebAS Test CA, OU=SWG, O=IBM,
c=US"--subject "cn=WebAS Test Server, OU=SWG, O=IBM, c=US"--sign-alg "MD5 with RSA"--sign-key
d:\projects\websphere\keyrings\WebAS.TestCA.key--subject-key
d:\projects\websphere\keyrings\WebAS.TestServer.key--cert-file
d:\projects\websphere\keyrings\WebAS.TestServer.certCreating certificate...
```

The example above creates a certificate for the TestServer signedby the TestCA. The server's certificate includes the server's publickey and is signed by the issuing CA (our TestCA) using the MD5 digestfunction and the TestCA's public key. The server's certificate isplaced in a file called WebAS.TestServer.cert.

This is a general-purpose tool with applications beyond those discussedhere. This page discusses only the subset of options relevant tomanaging certificates for WebSphere Application Server programs.

# 5.5.6.1.4: The VaultTool tool

The VaultTool command supports a suite of subcommands forcreating and managing keyrings. This tool maintains a masterdatabase of certificates and keys, the vault, from whichselected contents can be exported to keyrings for individualsusers.

The tool does not directly manipulate keyring files;all manipulation is done in the vault. For example, to adda new certificate to a server's keyring, you add the certificateto the vault and export a new edition of the keyring file forthe server to use.

## Common syntax

When invoking VaultTool, you must specify a password and thefile in which the vault is stored. The password simply protectsaccess to the vault file. This syntax is common to all VaultToolcommands. Each subcommand takes a specific set of arguments.

```
java com.ibm.cfwk.tools.VaultTool --password U_vaultpasswd U_vaultfile     subcommand
subcommand-args
```

## Subcommands

The tool supports the following subcommands;

- **list**: lists the contents of the vault
- **delete**: deletes specific information from the vault
- **add public cert**: adds the certificate for a CA to the vault
- **add private key**: places a private key in the vault
- **add public chain**: adds a chain of certificates to the vault
- **container**: writes information from the vault to a keyring file

Although each VaultTool subcommand takes its own set ofarguments, they are drawn from a common set. The followinglists the arguments used by the VaultTool subcommands.

**U_label**

A string used to categorize information for a specific principal; all keys, certificates, and chains used by a principal should have the same label.

**U_info**

User information to be stored with an entry. This information can be stored in one of three encodings, hexadecimal, binary, or base 64. The U_info itself is expressed as a string appended by -hex, -bin or -b64, for example, A1269E-hex.

**U_keyfile**

The file holding a user's keys.

**U_certfile**

A file containing a certificate.

**U_signerfile**

A file containing a certificate for a principal that has signed certificates for others, used when adding a chain to the vault.

**U_sslightpassword**

The password for the keyring database.

**U_class**

The class name for the keyring file (an SSLight database). Do *not* include the .class file extension.

**list**: Lists the contents of the vault, in a short or long format.
*Syntax:*

```
list --long
```

**delete**: Removes entries under a specific label from the vault.
*Syntax:*

```
delete U_label
```

**add public cert**: Adds the certificate for a trustworthyCA to the vault. Use this to build the trust base for your application.
*Syntax:*

```
add public cert U_label                [U_info]                U_certfile
```

*Example:*

```
java com.ibm.cfwk.tools.VaultTool --password
"WebAS"d:\projects\websphere\keyrings\WebAS.Test.Vault.vltadd public certWebAStestCA 00-hex
d:\projects\websphere\keyrings\WebAS.TestCA.cert
```

The example above adds the self-signed certificate for thelocal TestCA to the vault. This entry in the vault is stored under thelabel *WebAStestCA*.

**add private key**: Copies the private key from a keyfile and adds it to the vault.
*Syntax:*

```
add private key U_label                [U_info]                U_keyfile
```

*Example:*

```
java com.ibm.cfwk.tools.VaultTool --password
"WebAS"d:\projects\websphere\keyrings\WebAS.Test.Vault.vltadd private keyWebAStestServer 00-hex
d:\projects\websphere\keyrings\WebAS.TestServer.key
```

The example above adds the private key for the TestServer, storedin the file WebAS.TestServer.key, to the vault stored in the fileWebAS.Test.Vault.vlt. This entry in the vault is stored under thelabel *WebAStestServer*.

**add public chain**: Adds a chain of certificates tothe vault. A certificate chain includes a server includes theserver's certificate and the certificate of each prior issuingCA (there may be more than one). The complete chain must be addedat one invocation of the tool, and each certificate must beread from a separate file.
*Syntax:*

```
add public chain U_label                 [U_info]                 U_certfile
U_signerfile+
```

When adding both a key and a certificate chain for the same owner tothe vault, the values specified for U_label and U_info must matchin both the "add private key" and "add public chain" invocations.

*Example:*

```
java com.ibm.cfwk.tools.VaultTool --password
"WebAS"d:\projects\websphere\keyrings\WebAS.Test.Vault.vltadd public chainWebAStestServer 00-hex
d:\projects\websphere\keyrings\WebAS.TestServer.certd:\projects\websphere\keyrings\WebAS.TestCA.cert
```

The example above a chain of certificates for the TestServer to thevault. The chain includes two certificates: the TestServer's certificateand the certificate the issuing CA. Note that the U_label ("WebAStestServer")and the U_info ("00-hex") used in this example match those used in the"add private key" example for the TestServer.

**container**: Writes the contents (or labeled components) of thevault out to a keyring file. The keyring class is written to the currentdirectory, regardless of the specified Java package name. You must thencopy it to a location matching the Java package.
*Syntax:*

```
container -p U_sslightpassword           U_class          [U_label*]
```

*Example:*

```
java com.ibm.cfwk.tools.VaultTool --password
"WebAS"d:\projects\websphere\keyrings\WebAS.Test.Vault.vltcontainer -p "WebAS"
com.ibm.websphere.DummyKeyringWebAStestServer WebAStestCA
```

The example above exports the information in the vault forthe TestServer and TestCA (requested by the labels "WebAStestServer"and "WebAStestCA") to the WebSphere provided keyring filecalled DummyKeyring. This keyring file is protected with thepassword "WebAS."

This is a general-purpose tool with applications beyond those discussedhere. This page discusses only the subset of options relevant tomanaging certificates for WebSphere Application Server programs.

# 5.5.6.1.5: Example: generating and using testcertificates

The following sequence of commands illustrates how to createa test CA, use the test CA to generate test certificates for aserver, and put the necessary information into a keyring class.This example uses the default password ("WebAS") for the vaultand the keyring class, and writes to the default keyring class(com.ibm.websphere.DummyKeyring).

## Setting the classpath

To use the certificate-management tools, you must put the filescfwk.zip and cfwk-tools.zip at the front of your classpath.These files are located in the AppServer/lib directory of theWebSphere installation. For example, on Windows NT, set the CLASSPATHvariable as shown:

```
set
classpath=<WS-install>\AppServer\lib\cfwk.zip;<WS-install>\AppServer\lib\cfwk-tools.zip;%CLASSPATH%
```

## Creating the keys, certificates, and files

Create a pair of keys for the test CA, called "TestCA."

```
% java com.ibm.cfwk.tools.KeyGenTool --forge
"RSA/512/F4"D:\projects\websphere\keyrings\WebAS.TestCA.key
```

Create a pair of keys for the test server, called "TestServer."

```
% java com.ibm.cfwk.tools.KeyGenTool --forge
"RSA/512/F4"D:\projects\websphere\keyrings\WebAS.TestServer.key
```

Create a self-signed certificate for the test CA and place itin the file called WebAS.TestCA.cert. This uses the CA's keysfrom the WebAS.TestCA.key file created above.

```
% java com.ibm.cfwk.tools.MakeCertTool --serial 0 --for 2y--issuer "cn=WebAS Test CA, OU=SWG, O=IBM,
c=US"--sign-alg "MD5 with RSA"--sign-key d:\projects\websphere\keyrings\WebAS.TestCA.key--cert-file
d:\projects\websphere\keyrings\WebAS.TestCA.cert
```

Create a certificate for the TestServer using the server's key(in the file WebAS.TestServer.key), signed by the CA's key (in the fileWebAS.TestCA.key). Put the new certificate in a file calledWebAS.TestServer.cert.

```
% java com.ibm.cfwk.tools.MakeCertTool --serial 0 --for 2y--issuer "cn=WebAS Test CA, OU=SWG, O=IBM,
c=US"--subject "cn=WebAS Test Server, OU=SWG, O=IBM, c=US"--sign-alg "MD5 with RSA"--sign-key
d:\projects\websphere\keyrings\WebAS.TestCA.key--subject-key
d:\projects\websphere\keyrings\WebAS.TestServer.key--cert-file
d:\projects\websphere\keyrings\WebAS.TestServer.cert
```

Add the private key of the server to the vault. The vault is storedin the file WebAS.Test.Vault.vlt.

```
% java com.ibm.cfwk.tools.VaultTool --password
"WebAS"d:\projects\websphere\keyrings\WebAS.Test.Vault.vltadd private keyWebAStestServer 00-hex
d:\projects\websphere\keyrings\WebAS.TestServer.key
```

Add the certificate chain for the server to the vault. This chainincludes the certificates for the server and for the CA that signedthe server's certificate (from the files WebAS.TestServer.cert andWebAS.TestCA.cert, respectively).

```
% java com.ibm.cfwk.tools.VaultTool --password
"WebAS"d:\projects\websphere\keyrings\WebAS.Test.Vault.vltadd public chainWebAStestServer 00-hex
d:\projects\websphere\keyrings\WebAS.TestServer.certd:\projects\websphere\keyrings\WebAS.TestCA.cert
```

Add the CA's self-signed certificate (WebAS.TestCA.cert) to the vault.

```
% java com.ibm.cfwk.tools.VaultTool --password
"WebAS"d:\projects\websphere\keyrings\WebAS.Test.Vault.vltadd public certWebAStestCA 00-hex
d:\projects\websphere\keyrings\WebAS.TestCA.cert
```

Export the information about the TestCA and the TestServer from thevault to a keyring class (the default DummyKeyring).

```
% java com.ibm.cfwk.tools.VaultTool --password
"WebAS"d:\projects\websphere\keyrings\WebAS.Test.Vault.vltcontainer -p
"WebAS"com.ibm.websphere.DummyKeyring WebAStestServer WebAStestCA
```

ℹ The keyring class is written to the current directory, regardless of thespecified Java package name. You must copy it to a location correspondingto the Java package. For example, keyring class name iscom.ibm.websphere.DummyKeyring, the file must be moved to thecom/ibm/websphere directory and that directory must be on the CLASSPATHvariable.

## 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

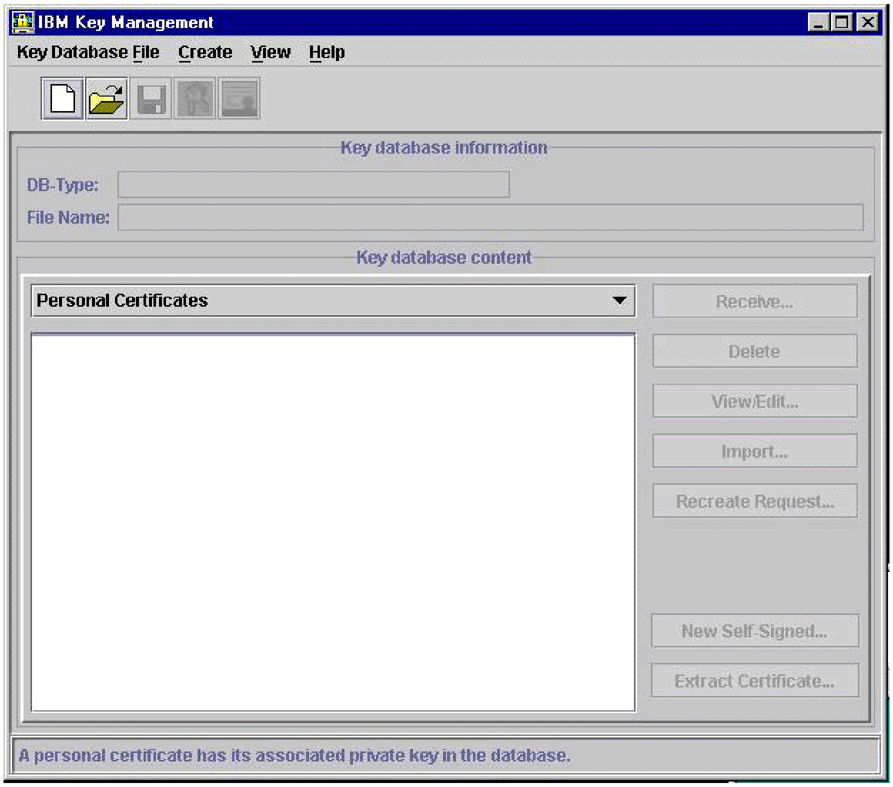
To use the iKeyman tool, you must put the necessary files at thefront of your classpath. Three of the files, gsk4cls.jar,cfwk.zip and cfwk.sec, are included as part of WebSphere (in theAppServer/lib directory). The fourth, swingall.jar, is part ofJava itself. For example, on Windows NT, set the CLASSPATH variable as shown:

```
set
classpath=<WS-install>\AppServer\lib\cfwk.zip;<WS-install>\AppServer\lib\gsk4cls.jar;<JdkDir>\lib\swingall.jar;<WS-install>\AppServer\lib;%CLASSPATH%
```

To start the iKeyman tool, use this command:

```
java -Dkeyman.javaOnly=true com.ibm.gsk.ikeyman.Ikeyman
```

The iKeyman window appears as shown below.



A personal certificate has its associated private key in the database.

# 5.5.6.2.1: Creating a self-signed test certificate

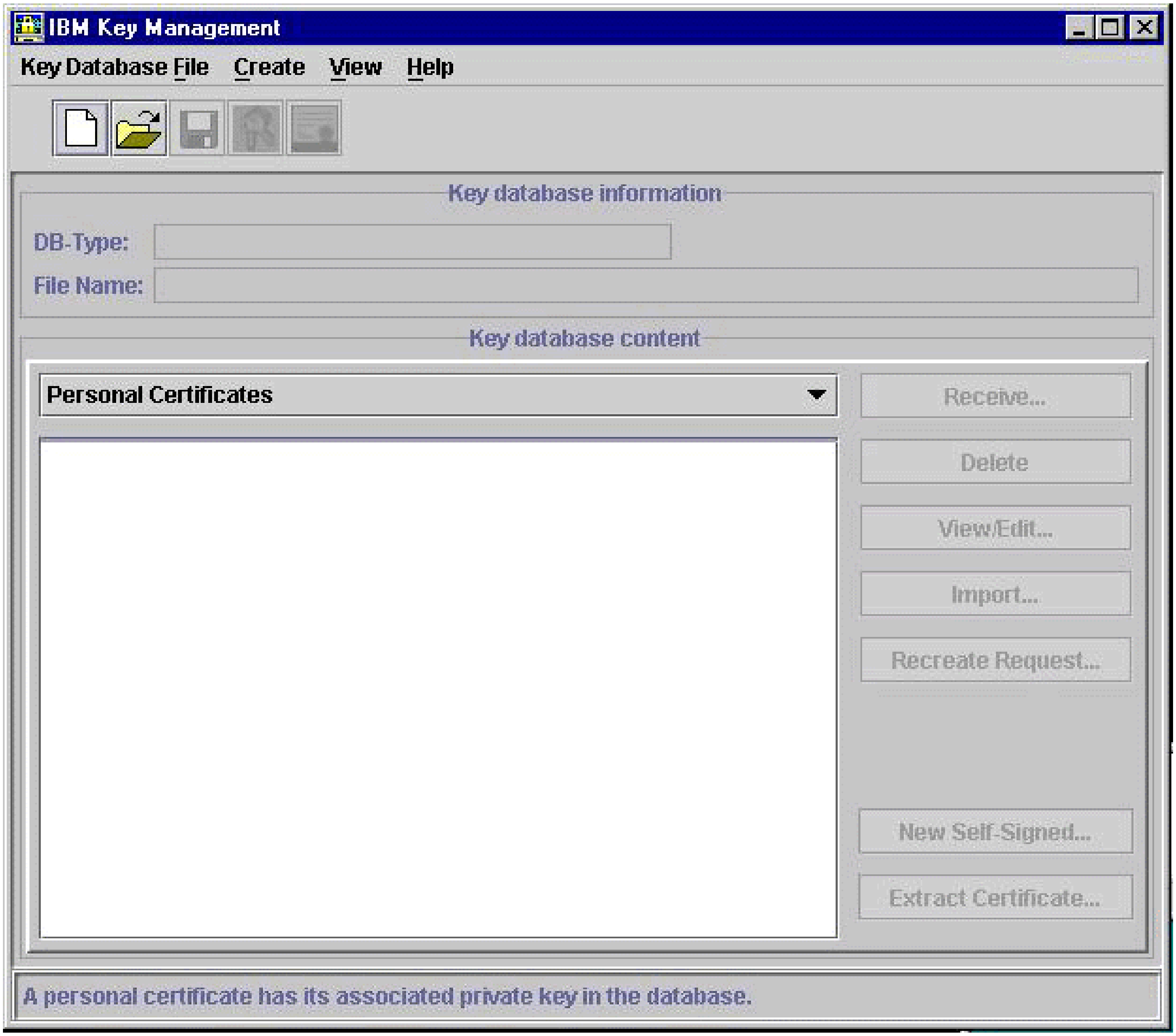
For test purposes, you can create a self-signed certificatespecifically for a server and its Secure Sockets Layer (SSL) basedJava clients. You can also set up a temporary certificateauthority by creating a self-signed certificate and using it to signother certificates.

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:

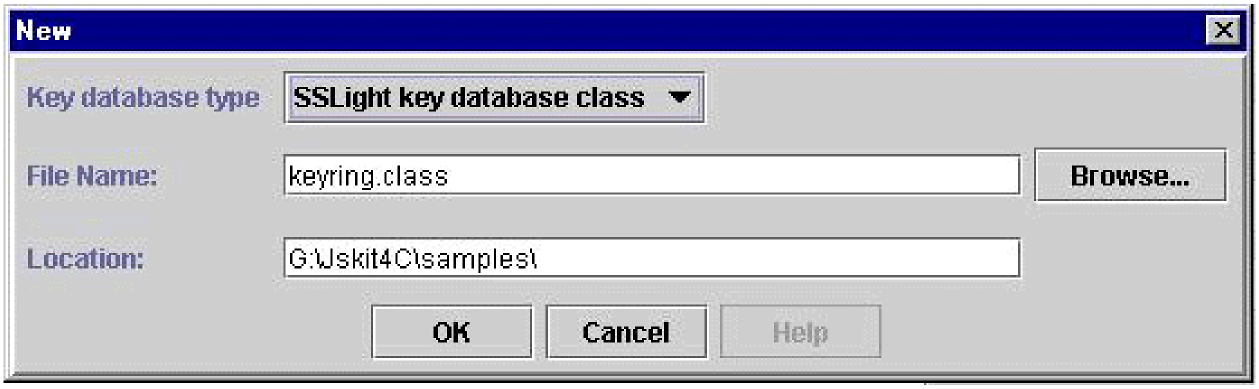
1. Start the IBM Key Management tool. This displays the IBM Key Management window.

```
java -Dkeyman.javaOnly=true com.ibm.gsk.ikeyman.Ikeyman
```



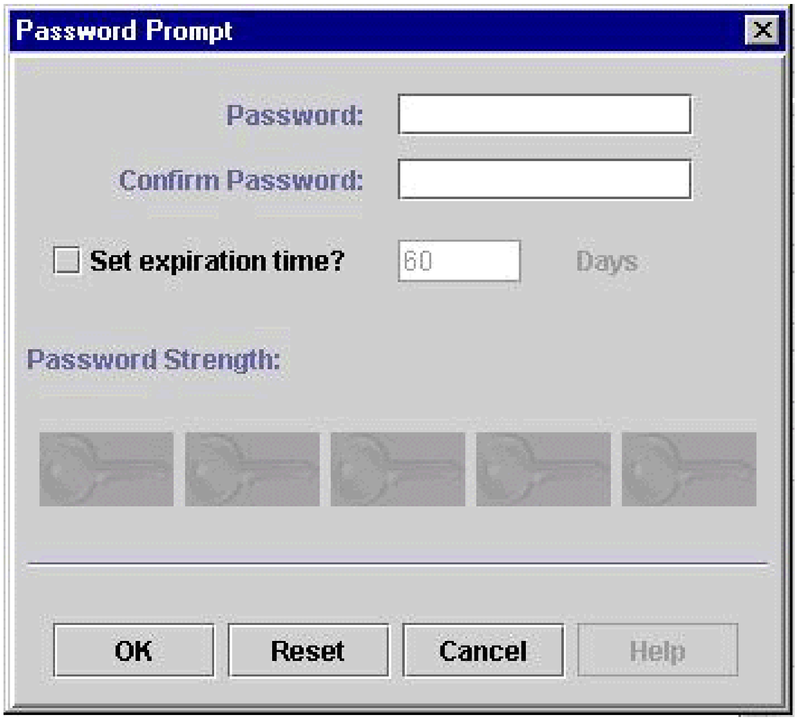
A personal certificate has its associated private key in the database.

2. Open a new key database file by selecting **Key Database File --> New** from the menu bar. The New dialog box is displayed.
3. Enter the name (including the .class extension) and location of the file for your new key database class. Files are typically named for the servers they belong to.



Key database type: SSLight key database class
File Name: keyring.class
Location: G:\Jskit4C\samples\

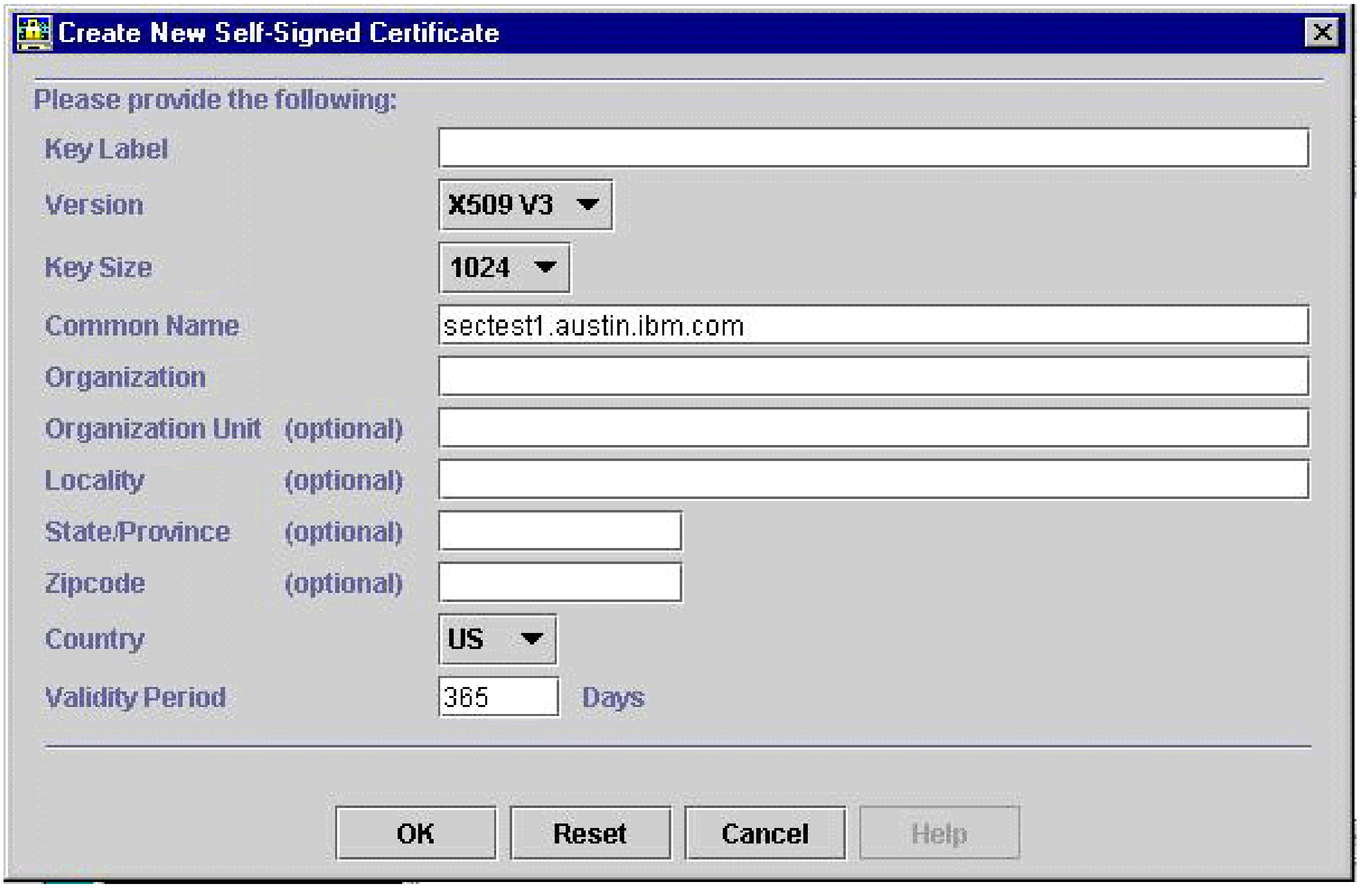
4. Click the **OK** button to continue.
5. The Password Prompt dialog box is displayed. Enter a password to restrict access to the key database. You will need to set the keyring-password properties (e.g., com.ibm.CORBA.SSLKeyRingPassword and com.ibm.CORBA.SSLClientKeyRingPassword) to this password so that the keyring class can be opened by iKeyman during runtime.



☐ Set expiration time? 60 Days

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.
7. The tool now displays all of the available default signer certificates. You can add, view or delete signer certificates from this screen. To continue creating a self-signed certificate, either click the **New Self-Signed...** button on the tool bar or select **Create --> New Self-Signed Certificate...** from the menu bar.
8. The Create New Self-Signed Certificate form is displayed. Enter the appropriate information for your self-signed certificate.


Create New Self-Signed Certificate

Please provide the following:

Key Label

Version: X509 V3

Key Size: 1024

Common Name: sectest1.austin.ibm.com

Organization

Organization Unit (optional)

Locality (optional)

State/Province (optional)

Zipcode (optional)

Country: US

Validity Period: 365 Days

**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.

9. Click the **OK** button to continue. The resulting key database class contains a self-signed certificate and its private key, and the class can be used for both a server and a client. You must copy the keyring file to the designated directory on the server's host.

   If you have only one personal certificate, it will be 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.

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

# 5.5.6.2.2: Creating a certification request

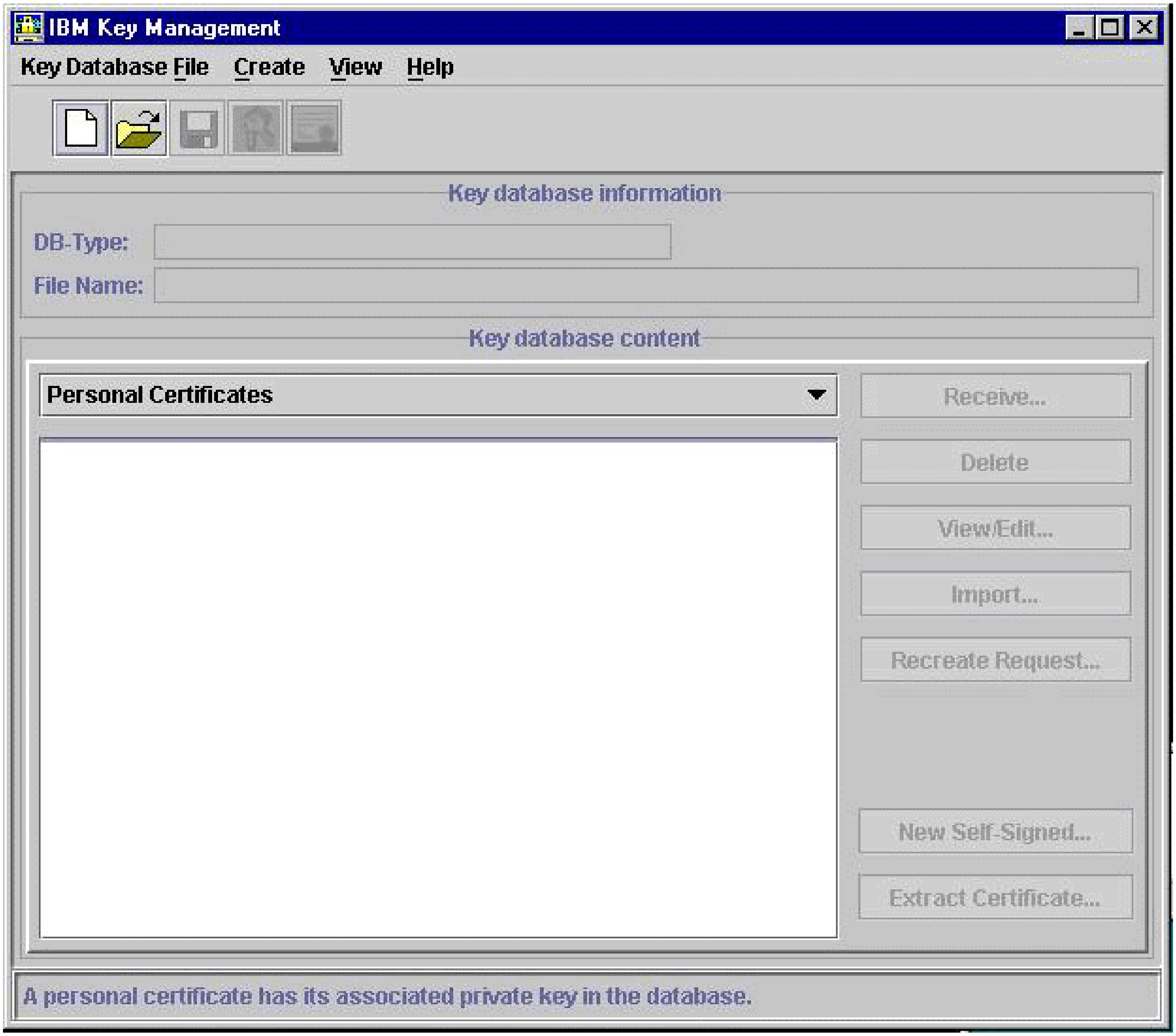
To obtain a certificate from a certificate authority, you mustsubmit a certificate signing request (CSR). You can request eitherproduction or test certificates from a CA with a CSR.

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:

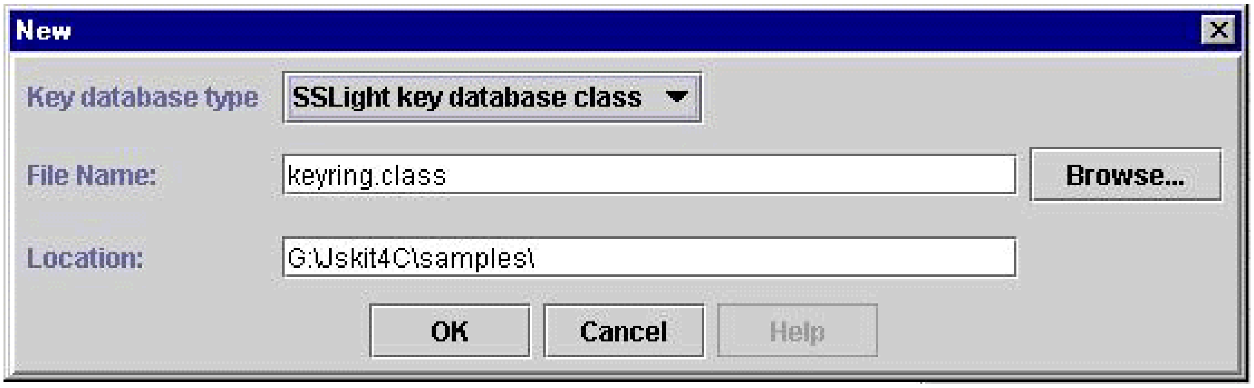
1. Start the IBM Key Management tool. This displays the IBM Key Management window.

   ```
   java -Dkeyman.javaOnly=true com.ibm.gsk.ikeyman.Ikeyman
   ```



A personal certificate has its associated private key in the database.

2. Open a new key database file by selecting **Key Database File --> New** from the menu bar.
3. The New dialog box is displayed. Enter the name (including the .class extension) and location of the file for your new key database class. Files are typically named for the servers they belong to.

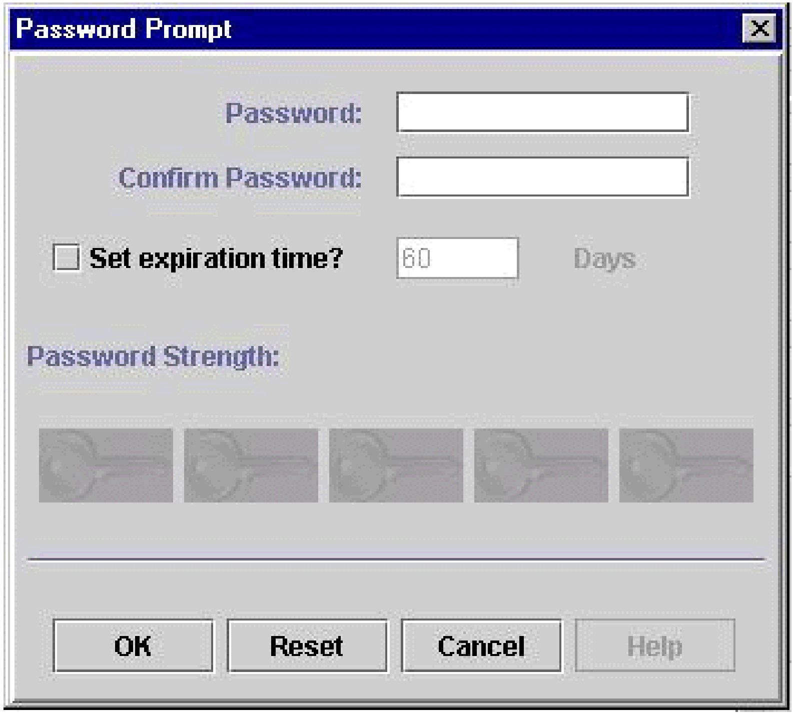


Key database type: SSLight key database class

File Name: keyring.class

Location: G:\Jskit4C\samples\

4. Click the **OK** button to continue.
5. The Password Prompt dialog box is displayed. Enter a password to restrict access to the key database. You will need to set the keyring-password properties (e.g., com.ibm.CORBA.SSLKeyRingPassword and com.ibm.CORBA.SSLClientKeyRingPassword) to this password so that the keyring class can be opened by iKeyman during runtime.



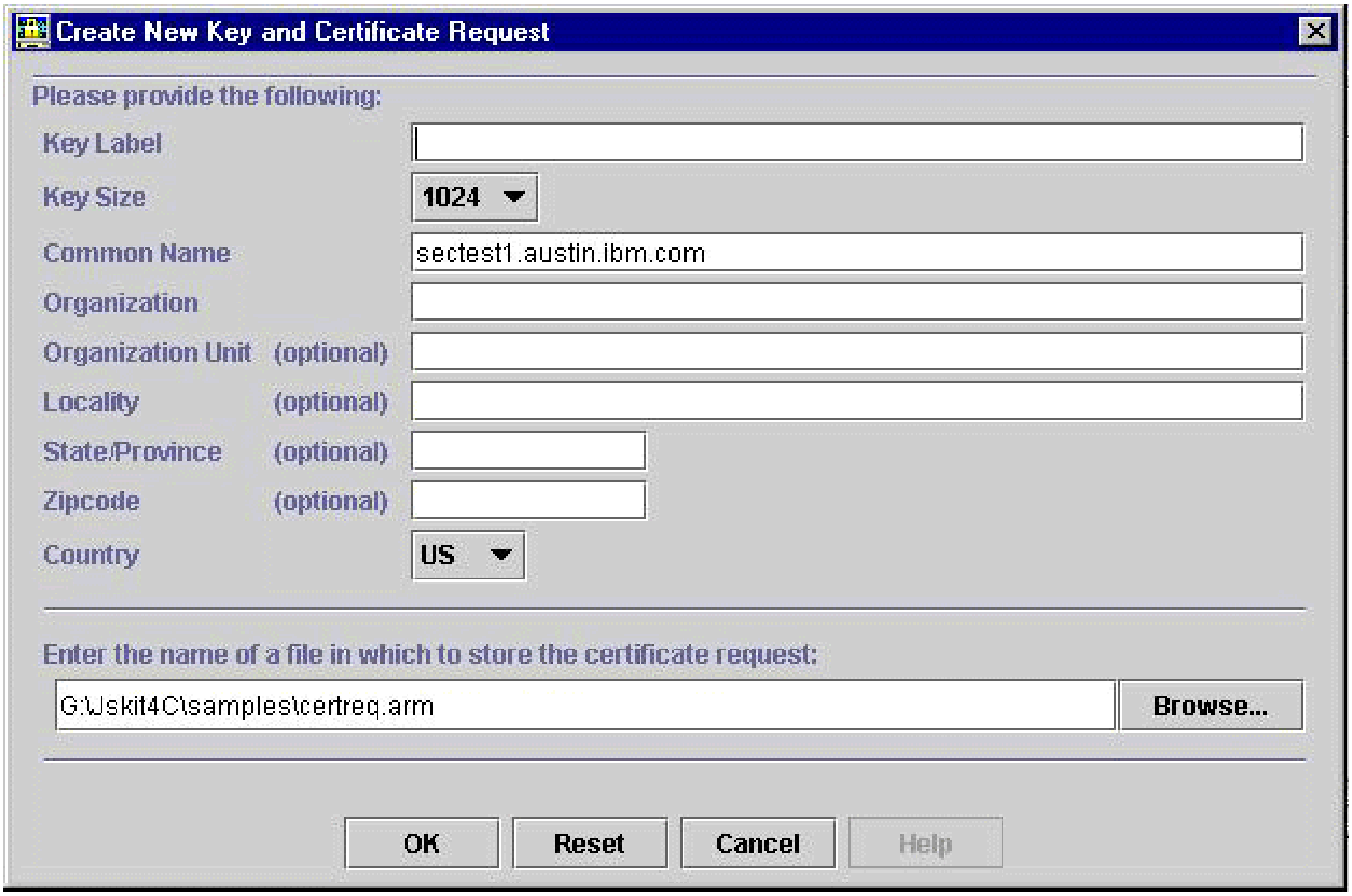
Password:

Confirm Password:

☐ Set expiration time? 60 Days

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.
7. 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.
8. Click the **New...** button.
9. 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.



Create New Key and Certificate Request

Please provide the following:

Key Label:

Key Size: 1024

Common Name: sectest1.austin.ibm.com

Organization:

Organization Unit (optional):

Locality (optional):

State/Province (optional):

Zipcode (optional):

Country: US

Enter the name of a file in which to store the certificate request:

G:\Jskit4C\samples\certreq.arm

**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.

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.

10. Click the **OK** button.
11. An Information panel is displayed to indicate that the request file has been successfully created. Click the **OK** button to dismiss the panel.
12. 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.

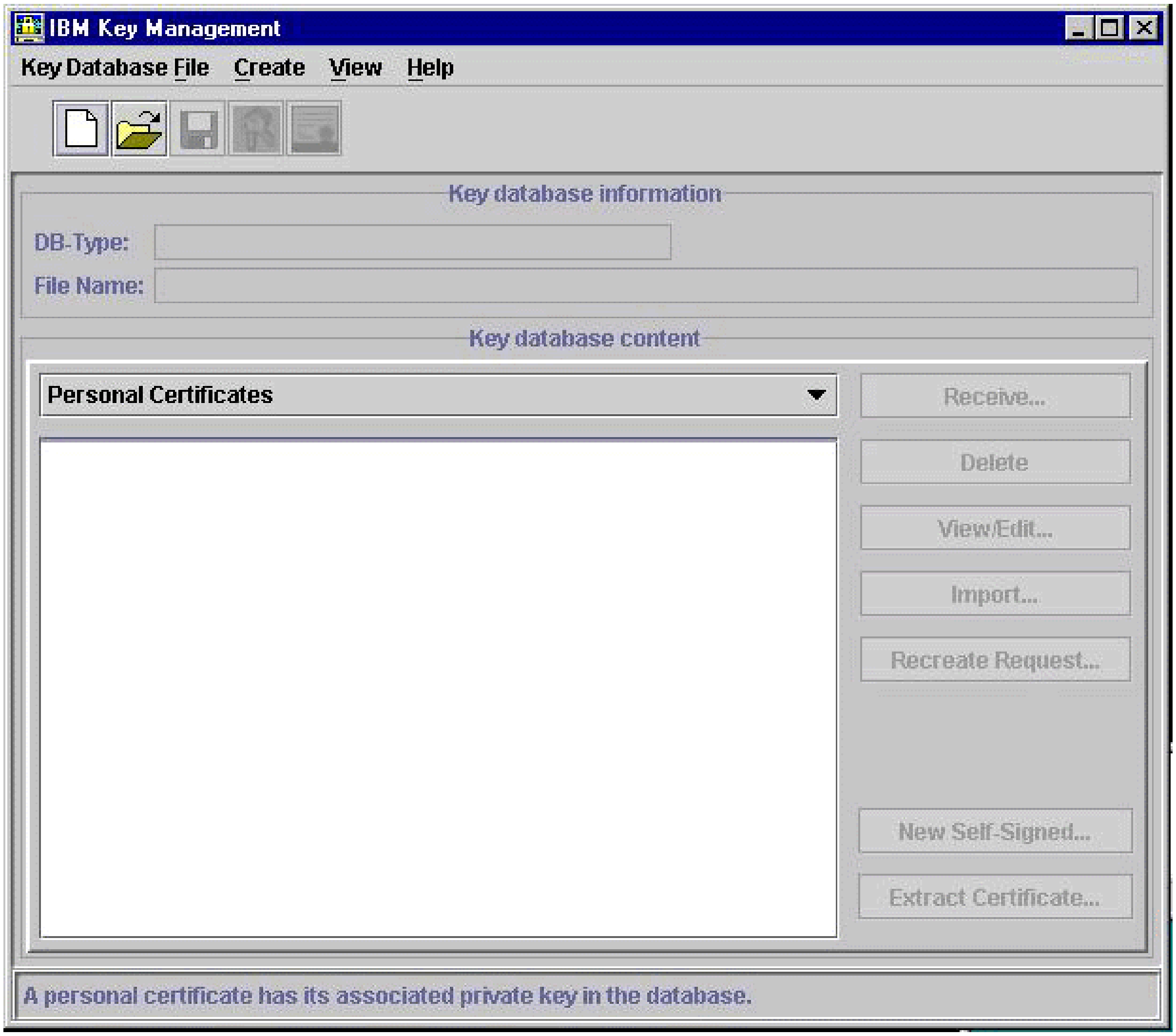
## 5.5.6.2.3: Placing a signed digital certificate intoa keyring

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 identity andto distribute its public key. This file describes how to place a newcertificate (either a test or a production certificate) into a keyringusing the iKeyman tool.

To place a signed certificate into a server's keyring, complete thefollowing steps:

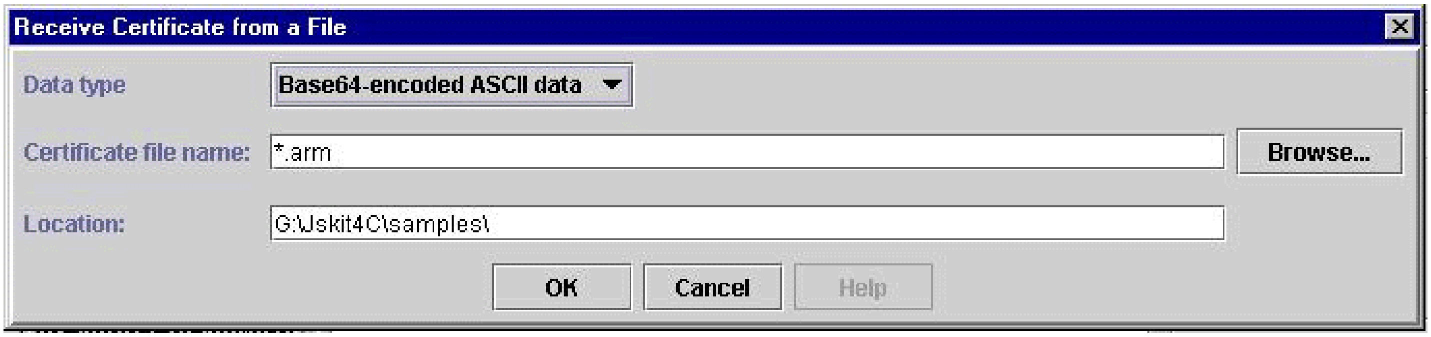
1. When you receive e-mail from the CA containing your certificate, save the message into a file. In this example, the certificate was saved to a file called PolicyServer1.responseMail.arm.
2. Start the IBM Key Management tool. This displays the IBM Key Management window.

```
java -Dkeyman.javaOnly=true com.ibm.gsk.ikeyman.Ikeyman
```



A personal certificate has its associated private key in the database.

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.
5. Click the **OK** button to continue.
6. Click on the certificate types pull-down list beneath **Key Database Context**, and select **Personal Certificates** (the default).
7. Click the **Receive...** button.
8. The Receive Certificate from a File dialog window is displayed. Enter the name of the file containing the saved e-mail. You can also use the **Browse...** button to find and select the file.



Data type: Base64-encoded ASCII data

Certificate file name: *.arm

Location: G:\Jskit4C\samples\

9. Click the **OK** button to continue to add the certificate in the file to the previously selected keyring.
10. Optionally, to verify that the certificate has been added, click the **View/Edit...** button in the main window.

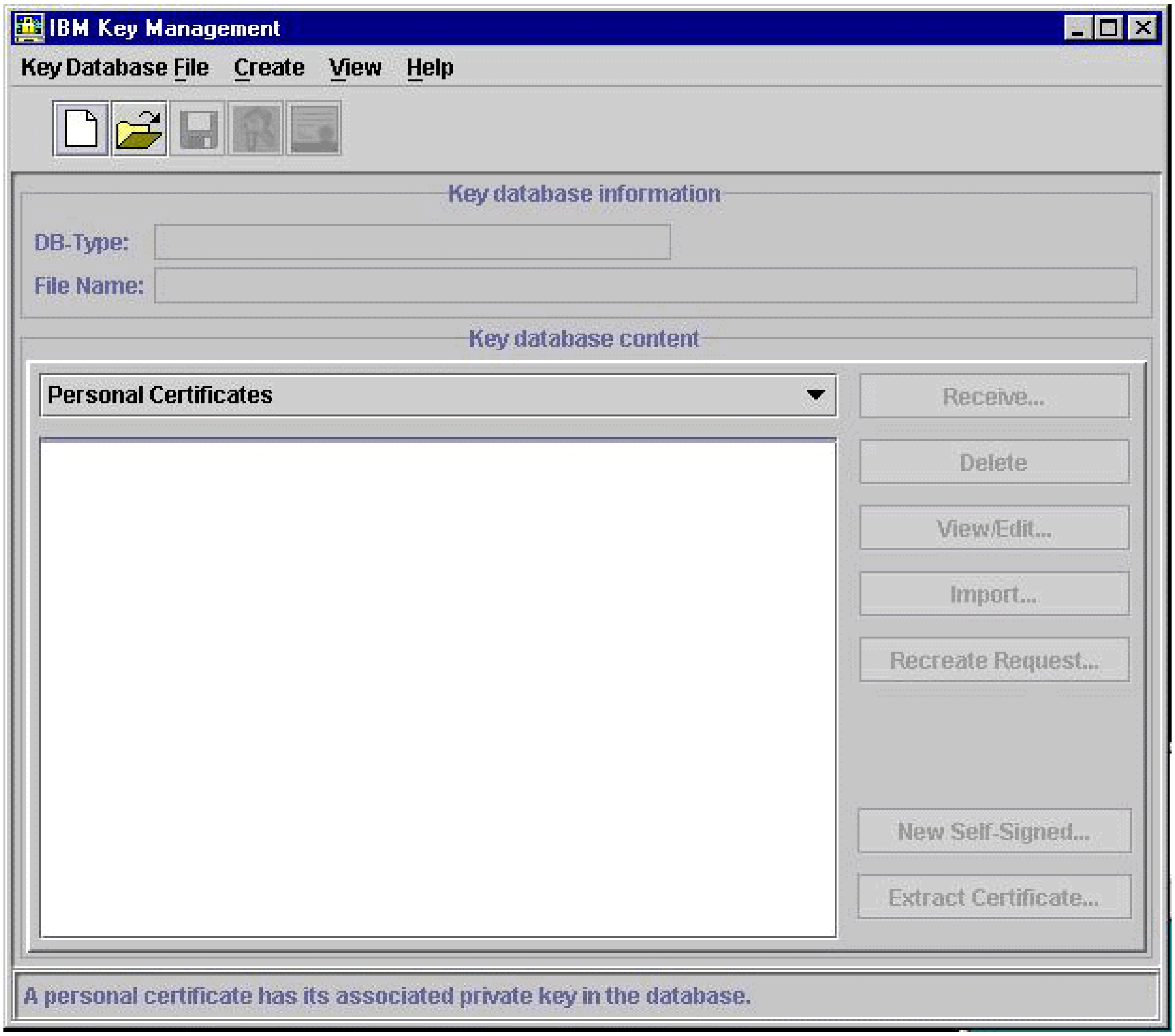
At this point, the server's keyring contains both its private key(which was generated as part of requesting the certificate) and thecertificate.

## 5.5.6.2.4: Adding a CA certificate to aclient's keyring class

To allow a client to authenticate to a server, the client needs acopy of the certificate of the server's CA. To add a CA certificateto a client's key database class, complete the following steps:

1. Start the IBM Key Management tool. This displays the IBM Key Management window.

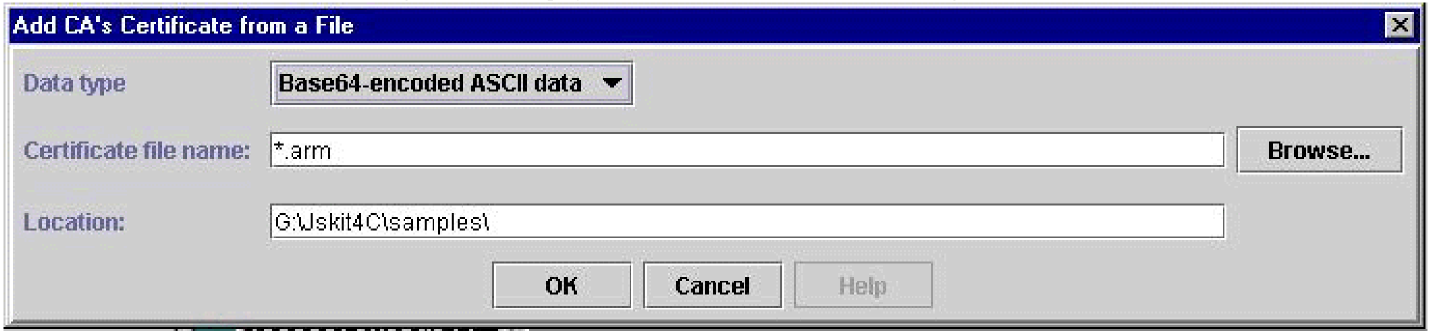
```
        java -Dkeyman.javaOnly=true com.ibm.gsk.ikeyman.Ikeyman
```



Personal Certificates

A personal certificate has its associated private key in the database.

2. Open the client's key database file by selecting **Key Database File** --> **Open** from the menu bar.
3. Enter the name and location of the keyring file at the prompt.
4. Click the **OK** button to continue.
5. Click on the certificate types pull-down list beneath **Key Database Context**, and select **Signer Certificates**.
6. 
7. Click the **Add** button.
8. The Add CA's Certificate from a File dialog window is displayed. Enter the name of the file containing the certificate.



Data type: Base64-encoded ASCII data

Certificate file name: *.arm

Location: G:\Jskit4C\samples\

9. Click the **OK** button to continue.
10. Close the IBM Key Management window to exit iKeyman.

You also need to make the keyring class available to Java clientsby setting the appproperiate properties.

# 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 mustspecify them in a number of files:

- admin.config
- adminserver.bat
- adminclient.bat
- sas.server.props
- sas.client.props

If you created client and server keyrings called **testclient.class** and **testserver.class**respectively, and you used "WebAS" as the password when you generated them, youneed to make the following changes:

- In the admin.config file: Add the directory holding the keyring classes to the front of the com.ibm.ejs.sm.adminserver.classpath variable.
- In the adminclient.bat and adminserver.bat files: Add the directory holding the keyring classes to the front of the %WAS_CP% variable.
- In the sas.client.props file, set the following properties:
  - com.ibm.CORBA.SSLKeyRing=testclient
  - com.ibm.CORBA.SSLKeyRingPassword=WebAS
  - com.ibm.CORBA.SSLServerKeyRing=testserver
  - com.ibm.CORBA.SSLServerKeyRingPassword=WebAS
- In the sas.server.props file, set the following properties:
  - com.ibm.CORBA.SSLKeyRing=testserver
  - com.ibm.CORBA.SSLKeyRingPassword=WebAS
  - com.ibm.CORBA.SSLClientKeyRing=testclient
  - com.ibm.CORBA.SSLClientKeyRingPassword=WebAS

## Managing the Server SSL Keyring Files

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

ℹ Always use theadministrative console to manage the server keyring files as changesmade in the console overwrite any manual changes to thesas.server.props file. Client keyring files are managed in thesas.client.props file because clients can be located on a remotemachine.

The Default SSL Settings panel can be used to configure WebSphereApplication Server components using SSL. Parameters that are setthrough the ORB SSL Settings panel override the default SSL settingsfor the ORB.

Regardless of which settings are in effect, the ORB usesthese settings as follows. (Additionally, the ORB requires the SASproperties files on the client and server to be configured asdescribed below.)

*Key file name*

The path of the SSL key file used by server connections. For the server 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.

# 5.5.7: Introduction: Setting up an LDAP connection over SSL

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

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

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

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

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

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

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

# 5.5.7.1: Establishing connections between application servers and LDAP servers

1. Disable WebSphere security before shutting down the administrative server and client. This is not strictly necessary, but it makes recovery easier if something goes wrong.
2. Create your own keyring. To use SSL between WebSphere Application Server and the LDAP server, you have to create your own keyring. The DummyKeyring file that comes with WebSphere is not sufficient because it does not contain the necessary information about the servers involved. See the articlesunder section 5.5.6, Tools for managingcertificates and keys, for instructions on how to create keyringswith the WebSphere Application Server key tools.
3. Place your keyring class in the appropriate directory. Because you are replacing the default keyring class with your own, make sure that WebSphere Application Server is able to find the new class file. The keyring class is a Java class, so the CLASSPATH variable is used to find it. For example, if you give the file a class name like com.mycompany.test.keyring, you need to add the search path com/mycompany/test to the CLASSPATH.

   If your chosen search path is not on WebSphere's default CLASSPATH, then you have to update the following files:

   - *admin.config*: Add the search path to the `com.ibm.ejs.sm.adminserver.classpath` property.
   - *adminserver.bat* and *adminclient.bat*: Add the search path to the WAS_CP variable. Insert a line like this:
     `set WAS_CP=%WAS_CP%;search-path`
4. Update the property files.
   - WebSphere determines the keyring file to use and its password by examining properties in the files sas.server.props and sas.client.props. The following properties determine the keyring to use:
     - com.ibm.CORBA.SSLKeyRing
     - com.ibm.CORBA.SSLClientKeyRing
     - com.ibm.CORBA.SSLServerKeyRing

     Set these properties to the name of your keyring class, for example, com.ibm.websphere.TestKeyring.
   - When you create a file for a keyring class, you specify a password for the file. This password protects the file from unrestricted use. The following properties are used to specify the password for a application that needs to use the keyring:
     - com.ibm.CORBA.SSLKeyRingPassword
     - com.ibm.CORBA.SSLClientKeyRingPassword
     - com.ibm.CORBA.SSLServerKeyRingPassword

     These properties default to the value "WebAS," so if you used a different password when you created the keyring file, change the value of these properties accordingly.

   After you modify the properties files, you need to delete the file sas.server.props.future, or your changes to the other properties files will not take effect when you restart the administrative server.
5. Restart the administrative server and client and configure LDAP.
   1. Set the Challenge Type to Basic (under **Security --> Specify Global Settings --> Application Defaults**)
   2. Set the Authentication Mechanism to LDAP (under **Security --> Specify Global Settings --> Authentication Mechanism**)

   3. Set up your LDAP registry (under **Security --> Specify Global Settings --> User Registry**)
      1. Set the port to 389.
      2. Do *not* check the box that says "Use SSL to connect to directory" yet.
   4. Click **Finish**. The application server now communicates with the LDAP server. You can verify the communication with your LDAP server by monitoring its connections.
6. Stop and restart the administrative server and client. You will now be prompted to authenticate against the LDAP registry.

At this point, you know that WebSphere Application Server can communicatewith the LDAP server successfully.

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

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

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

    To add the additional certificate to the keyring class, you must:
    1. Add the certificate to the keyring class. For example, you can use the VaultTool add public cert command:
        ```
                     % java  com.ibm.cfwk.tools.VaultTool --password  "vltpwd"             myVault.vlt
        add public cert LDAPCA 00-hex myLDAPCA.cert
        ```
    2. Create a new file for the keyring class, including the new certificate. For example, you can use the VaultTool container command:
        ```
                     % java com.ibm.cfwk.tools.VaultTool --password  "vltpwd"             myVault.vlt
        container -p  "ringpwd" com.ibm.websphere.TestKeyring             myTestServer myTestCA LDAPCA
        ```
3. Enable the SSL connection in WebSphere.
    1. Modify your LDAP configuration (under **Security** --> **Specify Global Settings** --> **User Registry**).
        1. Set the port to 636.
        2. Check the box labelled **Use SSL to connect to directory**.
    2. Click **Finish**.
4. Stop and restart the administrative server and client. After they restart, you are prompted to login to the LDAP registry.

## Tips

- If your SSL connection does not work, try the following:
    1. Verify that your LDAP server is listening to port 636.
    2. Verify that the LDAP server's certificate is still valid.
- If you need to export the certificate for the LDAP server's CA from keyring or other type of file, look for an option that lets you export the certificate in DER binary format. The tools you have can vary with the LDAP server.
- If you transfer a certificate file from a remote host by using FTP, be sure to set the transfer mode to binary.
- Make sure that your place your updated keyring class in the correct location.

# 5.5.7.3: Example: Generating and using a test keyring

The following sequence of commands illustrates the steps usedin creating a keyring for a test server and adding a certificatefor an LDAP server so that the WebSphere-to-LDAP communicationcan occur over an SSL connection.

Until the addition of the LDAP certificate to the vault,this example is structurally identical to the example forgenerating and using test certificates, with one exception:that example exports the vault to the DummyKeyring class, andthis one creates an application-specific TestKeyring class.

## Setting the classpath

To use the certificate-management tools, you must put the filescfwk.zip and cfwk-tools.zip at the front of your classpath.These files are located in the AppServer/lib directory of theWebSphere installation. For example, on Windows NT, set the CLASSPATHvariable as shown:

```
set
classpath=<WS-install>\AppServer\lib\cfwk.zip;<WS-install>\AppServer\lib\cfwk-tools.zip;%CLASSPATH%
```

## Creating the keys, certificates, and files

Create pairs of keys for TestServer and TestCA.

```
% java com.ibm.cfwk.tools.KeyGenTool --forge "RSA/512/F4" myTestServ.key % java
com.ibm.cfwk.tools.KeyGenTool --forge "RSA/512/F4" myTestCA.key
```

Create a self-signed certificate for the TestCA.

```
% java com.ibm.cfwk.tools.MakeCertTool --serial 0 --for 2y--issuer "cn=myTestCA"--sign-alg "MD5 with
RSA"--sign-key myTestCA.key--cert-file myTestCA.cert
```

Create a certificate for the TestServer.

```
% java com.ibm.cfwk.tools.MakeCertTool --serial 0 --for 2y--issuer "cn=myTestCA"--subject
"cn=myTestServer"--sign-alg "MD5 with RSA"--sign-key myTestCA.key--subject-key
myTestServ.key--cert-file myTestServ.cert
```

Add TestServer's private key to the vault.

```
% java com.ibm.cfwk.tools.VaultTool --password "vltpwd" myVault.vltadd private key myTestServer
00-hex myTestServ.key
```

Add the chain of certificates for TestServer to the vault.

```
% java com.ibm.cfwk.tools.VaultTool --password "vltpwd" myVault.vltadd public chain myTestServer
00-hex myTestServ.cert myTestCA.cert
```

Add the TestCA's self-signed certificate to the vault.

```
% java com.ibm.cfwk.tools.VaultTool --password "vltpwd" myVault.vltadd public cert myTestCA 00-hex
myTestCA.cert
```

Export the information for TestServer and TestCA to the TestKeying file.

```
% java com.ibm.cfwk.tools.VaultTool --password "vltpwd" myVault.vltcontainer -p "ringpwd"
com.ibm.websphere.TestKeyring myTestServer myTestCA
```

After establishing the WebSphere-to-LDAP connection, you need toadd the LDAP server's CA certificate to the keyring. With this in place,the WebSphere-to-LDAP communication can take place over SSL.

To add this certificate to the keyring, you must add it to thevault and then regenerate the keyring file.

First, add the LDAP CA's certificate to the vault.

```
% java com.ibm.cfwk.tools.VaultTool --password "vltpwd" myVault.vltadd public cert LDAPCA 00-hex
myLDAPCA.cert
```

Second, generate a new version of the keyring class, includingthe LDAP CA's label in the set of information to export.

```
% java com.ibm.cfwk.tools.VaultTool --password "vltpwd" myVault.vltcontainer -p "ringpwd"
com.ibm.websphere.TestKeyringmyTestServer myTestCA LDAPCA
```

ℹ The keyring class is written to the current directory, regardless of thespecified Java package name. You must copy it to a location correspondingto the Java package. This is true each time you regenerate the keyringfile.

# 5.5.7.4: Example: Generating keyring files for SSL

This procedure describes how to create keyring files that permitSSL communications between WebSphere Application Server and anLDAP server. This require the creation of two keyring files,one for the server and one for the client. The server's keyring storesthe public and private key of the server, and the certificate authority'scertificate. The client's keyring stores the server's public key and the CA'sroot certificate.

1. Download the external public certificate for the root certificate authority (root CA) and save it to a file. In this example, the file is called caroot.arm.
2. Generate the server-side keyring file.
    1. Request a certificate for the server, if it doesn't already have one.
        1. Generate a certificate request and save it to a file. In this example, the file is called certreq.arm.
        2. Submit the request to the certificate authority.
        3. Save the newly obtained certificate to a file. In this example, the file is called newcert.arm.
    2. Place the certificate into a keyring file. This can be done using either the keytool command-line tool or the graphical IBM Key Managment (Ikeyman) tool. For example, if you are using the Ikeyman tool, you must:
        1. Create a new keyring-class file. In this example, the file is called ServerKeyring.class.
        2. Specify the the certificate in the newcert.arm file as the certificate to be received into the keyring file. This is done on the Personal Certificates panel in the Ikeyman tool.
        3. 
        4. The client will also need access to the server's certificate, so extract the certificate and save it to a file. In this example, the file is called websphere.arm.
        5. Add the certificate of the signing CA, saved in the file caroot.arm, to the keyring file. This is done on the Signer Certificates panel in the Ikeyman tool.
3. Generate the client-side keyring file. This can be done using either the keytool command-line tool or the graphical IBM Key Managment (Ikeyman) tool. For example, if you are using the Ikeyman tool, you must:
    1. Create a new keyring-class file. In this example, the file is called ClientKeyring.class.
    2. Add the certificate of the signing CA, saved in the file caroot.arm, to the keyring file. This is done on the Signer Certificates panel in the Ikeyman tool.
    3. Add the certificate of the server, saved in the file websphere.arm, to the keyring file. This is also done on the Signer Certificates panel in the Ikeyman tool.
4. Install the new keyring files into the WebSphere Application Server environment:
    - Both the ServerKeyring.class and ClientKeyring.class files must be placed on the server. Modify the following lines in the sas.server.props file:

      ```
                    com.ibm.CORBA.KeyRingFile=ServerKeyring
      com.ibm.CORBA.KeyRingPassword=WebAS              com.ibm.CORBA.SSLClientKeyRingPassword=WebAS
      com.ibm.CORBA.SSLClientKeyRing=ClientKeyring
      ```
    - The client side requires only the ClientKeyring.class file. Modify the following lines in the sas.client.props file:

      ```
                     com.ibm.CORBA.SSLKeyRing=ClientKeyring
      com.ibm.CORBA.SSLKeyRingPassword=WebAS               com.ibm.CORBA.SSLServerKeyRing=ClientKeyring
      com.ibm.CORBA.SSLServerKeyRingPassword=WebAS
      ```

# 5.6: Establishing trust association with a reverse proxy server

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

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

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

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

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

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

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

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

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

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

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

## Configuring WebSphere Application Server to run in trust association

WebSphere Application Server must be configured to run intrust-assocation mode by setting up the trust-association interceptorsthat are going to receive HTTP requests from the trusted proxy server.

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

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

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

The following describes each of the property-value pairs:

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

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

## Configuring the WebSeal interceptor (optional)

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

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

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

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

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

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

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

## Configuring WebSeal

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

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

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

where

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

For example, the command:

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

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

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

```
-p <port_number>
```

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

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

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

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

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

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

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

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

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

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

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

## Sample configuration

This section describes a sample configuration.

- WebSphere Application Server is installed on the machine was_host.raleigh.ibm.com.
- The Web server is Netscape Enterprise Server, also installed on the machine was_host.raleigh.ibm.com. The Web server is listening on port 4343 for SSL requests.
- The LTPA security mechanism is used, with the LDAP server residing on the machine ldap_host.raleigh.ibm.com.
- WebSeal is installed on the machine webseal_host.raleigh.ibm.com. It listens on port 444 for SSL requests.
- A junction was created using the following command:

  ```
  junctioncp create -c -t ssl -h was_host.raleigh.ibm.com -p 4343 /myjunction
  ```
- In the WebSeal iv.conf file, the following lines are included:

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

  where the ID web_user with password testpassword is registered in the WebSphere Application Server registry.
- In the Policy Director secmgrd.conf file, the following line is included:

  ```
  junction-ca-cert-file=/opt/intraverse/lib/certs/junctioncacert.pem
  ```
- The ID testuser1 with password sherlock is a valid WebSeal user. It is also a valid WebSphere Application Server user.

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

- To test access without WebSeal, the user enters the following in the Web browser:

  ```
  https://was_host.raleigh.ibm.com/servlet/snoop
  ```
- To test access through WebSeal, the user enters the following:

  ```
  https://webseal_host.raleigh.ibm.com:444/aim/servlet/snoop
  ```

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

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

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

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

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

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

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

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

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

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

# 5.6.3: Writing a custom interceptor

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

## Using the TrustAssociationInterceptor interface

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

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

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

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

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

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

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

## Making your custom interceptor configurable

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

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

- `abstract public void cleanup();`

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

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

# 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.
3. Securing specific HTTP and EJB methods (optional).

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.

## Security task wizards in Java console

Of the current administrative clients, WebSphere AdministrativeConsole provides the most comprehensive support for securingapplications, in the form of security task wizards for:

- Enabling product security
- Defining a security realm and set of valid users
- Specifying how to authenticate users seeking access to applications
- Organizing methods (functions, operations) into groups for protection
- Granting users permissions to access applications

# 6.6.18.1: Securing applications with the Java administrative console

The table summarizes the security wizards provided for accomplishingthe tasks necessary to secure an application.

| Goal | Wizard | Conceptual overview | Instructions |
|---|---|---|---|
| 1. Enable security; set application security default and global values;specify how to authenticate users | Configure Global Settings | 6.6.18.1a | 6.6.18.1.1a |
| 2. Secure a particular application, making users authenticate their identitiesbefore using it | Configure Application Security | 6.6.18.1b | 6.6.18.1.1b |
| 3. Configure custom method groups as an optional step towardsdefining who will be allowed to access applications | Configure Method Groups | 6.6.18.1c | 6.6.18.1.1c |
| 4. Assign the methods in a resource, such as a servlet, to a custom ordefault method group for protection | Configure Resource Security | 6.6.18.1d | 6.6.18.1.1d |
| 5. Specify which users and groups can access which methods in which applications | Configure Permissions | 6.6.18.1e | 6.6.18.1.1e |

## Test the application

At this point, your resources will be secure. A user who runs aclient program that accesses secured resources will be promptedto log in. The user must log in with an account that has beengranted access to the resources, otherwise the user will be deniedaccess. The visible effect of this denial is that the clientprogram will trigger an authorization failure, for example,a java.rmi.ServerException that contains acom.ibm.ejs.EJSSecurityException.

# 6.6.18.1.1: Securing applications

The procedure for securing an application with the Java console (WebSphereAdministrative Console) is as follows:

1. Specify global and default security settingsfor all applications
2. Configure security for the particular application
3. Configure custom method groups to protect methods
4. Assign methods to custom or default method groups
5. Assign permissions allowing users access to methodgroups and applications

# 6.6.18.1.1a: Specifying global settings with the Java administrative console

1. Start the Configure Global Settings task by one of two methods:
    - By clicking Console -> Tasks -> Configure Global Settings from the console menu bar.
    - By clicking Configure Global Settings from the drop-down list on the Wizards toolbar button.
2. Complete the task, referring to the information below for assistance.
3. Stop the administrative server and start it again for the changes to takeeffect.

    The next time the administrator opens the WebSphere AdministrativeConsole, the administrator will be prompted to log in (if security is enabled) using an ID and password specified during Global Settingsconfiguration.

## General

Use the **General** tab to specify whether to enable security. If the check box is *not* selected, any other security settings you specify will be disregarded.

Additional settings are available on this page, including the security cache timeoutand default SSL configuration.

## Application Defaults

Specify a default security realm andchallenge type for applications. The administrator can later override thesevalues in the security settings for an individual application.

The challenge type is the way in which users will be challenged for their credentials,for example, using a digital certificate or user ID and password combination.

To refuse to service requests that not are transmitted over SSL (Secure Sockets Layer),click the option for using SSL to connect the client and Web server.

## Authentication Mechanism

Use the Authentication Mechanism tabbed page to specify how to authenticate theinformation presented by users trying to access an application or resources.

The administrator canhave users or groups authenticated against either the local operating system userregistry (such as Windows NT User Manager program) or an LTPA-enabled LDAP or custom directory service product.

## User Registry

Use the User Registry page to specify details about the authentication mechanism youchose.

The contents of this page vary according to the authentication mechanism. If youchose the directory service option, consult the properties help forfilling in the filters and other values.

# 6.6.18.1.1b: Configuring application security

1. Start the Configure Application Security task by one of two methods:
   - By clicking Console -> Tasks -> Configure Application Security from theconsole menu bar.
   - By clicking Configure Application Security from the drop-down list on theWizards toolbar button.
2. Specify the enterprise application to which to apply security.
3. Click Next to proceed. Modify the application security defaults if necessary.
4. Click Finish.

# 6.6.18.1.1c: Configuring custom method groups

1. Start the Configure Method Groups task by one of two methods:
    - By clicking Console -> Tasks -> Configure Method Groups from theconsole menu bar.
    - By clicking Configure Method Groups from the drop-down list on theWizards toolbar button.
2. Specify to add a new method group.
3. Click Next to proceed. Type a name for a new method group.
4. Click Finish.

When finished configuring method groups, exit the task by clickingany other resource or task in the administrative console.

# 6.6.18.1.1c.1: Viewing custom method groups

To confirm the existence of a custom method group:

1. Start the Configure Method Groups task.
2. Specify to remove an existing method group.
3. On the resultingpage, verify that the new custom method group is displayedin the set of existing method groups.
4. Cancel the task without actually removing a method group.

Of course, even if the administrator does not perform the abovetask, the administrator will detect any problems with custom method groupcreation when he or she proceeds to assign methods to the method groups.

# 6.6.18.1.1d: Configuring resource security

1. Start the Configure Resource Security task by one of two methods:
    - By clicking Console -> Tasks -> Configure Resource Security from theconsole menu bar.
    - By clicking Configure Resource Security from the drop-down list on theWizards toolbar button.
2. Specify the resource to which to apply security.

    Note that servlets, JSP files, and Web pages are not represented directly,but can be selected according to their "Web resource" configurations. Web resourcesspecify Web paths to these resources. If configuring resource security fora servlet, JSP file, or Web page, select the appropriate Web resource.
3. Click Next. A prompt asks whether to use the default method groups.

    If you specify Yes, the security system will take a first pass at groupingthe methods of the resource into the default method groups. It will use the methodnames to decide which groups to put them in.

    For example, it will protect HTTP PUT methods with the WriteMethods group.

    If you decline to use the default method groups, you will need to specifya method group for each method in the resource.

    ℹ The distribution of the methods into thedefault method groups is not finalized until you finish this task wizard. Youcan always try the default method groups, then reverse the operation if youdo not like the results. Just do not click the Finish button until youare sure about the method groups!
4. If the Finish button is available, click it. If configuring resourcesecurity for an enterprise bean, click Next to proceed to a final panel forspecifying delegation settings.
5. Delegation allows an enterprise bean method to execute under another identity.

    The top half of the task panel is for specifying the default identity underwhich bean methods will execute. If you click SPECIFIED, specify the user.

    Use the bottom half of the task panel to select a particular enterprise beanfor which you want to override the default run-as identity. Click the bean, thenspecify SYSTEM, CLIENT, or SPECIFIED in the area to the right of the enterprisebean list box. If you click SPECIFIED, specify a run-as identity.

    The delegation settings you specify here override any run-as information in the deployment descriptor of the enterprise bean.
6. Click Finish.

ℹ If your Web server is already running when you configure resourcesecurity for an HTML file, JSP file, or other Web resource, you needto stop the WebSphere administrative server and start it again forthe change to take effect.

# 6.6.18.1.1d.1: Default method groups

The product predefines these method groups:

| **Method group** | **Typical Web resource methods** | **Typical EJB methods** |
|---|---|---|
| ReadMethods | GET and POST | Methods beginning with the string "get" (such as getName()) |
| WriteMethods | PUT | Methods beginning with the string "set" (such as setName()) |
| RemoveMethods | DELETE | All remove() methods of an enterprise bean home object |
| CreateMethods | None | All create() methods of an enterprise bean home object |
| FinderMethods | None | All find*() methods of an enterprise bean home object |
| ExecuteMethods | Methods that do not fit in other groups | Methods that do not fit in other groups |

In addition, the administrator can create custom method groups.

# 6.6.18.1.1e: Configuring permissions

1. Start the Configure Permissions task by one of two methods:
    - By clicking Console -> Tasks -> Configure Permissions from theconsole menu bar.
    - By clicking Configure Permissions from the drop-down list on theWizards toolbar button.
2. Click a permission, such as *AnyApplicationName*-ReadMethods.

    The administrator can view permissions by application or bymethod group:
    - Viewing by application shows only the permissions associatedwith a particular application, such as:
        - Application_A-CreateMethods
        - Application_A-WriteMethods
        - Application_A-CustomMethodGroup
    - Viewing by method group shows only the permissions associatedwith a particular method group, such as:
        - Application_A-CreateMethods
        - Application_B-CreateMethods
        - Application_C-CreateMethods
3. Click the Add button to produce a search dialog.
4. Use the search dialog to give permission to everyone or selected users or groups.You can search for a user or group in your local operatingsystem user registry or directory service product.
5. When finished with the search dialog, click the OK button.
6. Back in the main console window, verify that the user or group is listedunder the permission you granted to the user or group.

When finished configuring method groups, exit the task by clickingany other resource or task in the administrative console.

## Securing WebSphere administrative accounts

Ability to administer WebSphere Application Server after it has beensecured is governed by a Web application. You can set up an initial accountand additional administrative accounts to access the secured product. Seethe information about administrative accounts for details and instructions.

## Setting permissions to authenticiate against local and domain registries (Windows)

WebSphere Application Server security supports authentication both against the domain registry and the local registry of a supported, Windows-based machine. The administrator can force authentication against the local registry by setting permissions appropriately.

If a machine is part of a Windows domain, when a user authenticates to WebSphere Application Server security, the user is first authenticated against the domain registry. If that fails, the user is authenticated against the local operating system registry.

If the user exists in both the local and domain registries, and authorization has been granted to the local user, it becomes necessary to qualify the user name when logging on to WebSphere security.

For an example of the implications of setting permissions, suppose a machine named "LOCAL" belongs to a domain named "DOMAIN." The users "user1" and "user2" exist in both the LOCAL and DOMAIN registries:

- LOCAL\user1
- LOCAL\user2
- DOMAIN\user1
- DOMAIN\user2

Suppose the WebSphere administrator configures permissions such that the following users can access a WebSphere resource:

- LOCAL\user1
- DOMAIN\user2

When user1 logs on to access a resource, he or she must specify LOCAL\user1 (not simply user1) as the user name for successful authentication. When user2 logs on, he or she can specify simply user2.

# 6.6.18.1.2: Securing cloned applications

In an environment containing models and clones, each model and clonemust be secured individually. Securing a model does not automaticallysecure its clones.

For example, if you clone an application server that contains secure enterprise applications, then you need to secure those same enterpriseapplications (if you want to) on the cloned application servers.

Secure a cloned application as you would secure any new application.

# 6.6.18.1.4: Properties related to security

The WebSphere Administrative Console provides security wizards foraccomplishing various goals. Property (field) help provides detailedinformation about options and data fields in the wizards.

| Goal | Wizard | Property help |
|---|---|---|
| Enable security; set application security default and global values;specify how to authenticate users | Configure Global Settings | 6.6.18.1.4a: Global Security settings |
| Secure a particular application, making users authenticate their identitiesbefore using it | Configure Application Security | 6.6.18.1.4b: Application security settings |
| Configure custom method groups as an optional step towardsdefining who will be allowed to access applications | Configure Method Groups | 6.6.18.1.4c: Method group settings |
| Assign the methods in a resource, such as a servlet, to a custom ordefault method group for protection | Configure Resource Security | 6.6.18.1.4d: Resource Security settings |
| Specify which users and groups can access which methods in which applications | Configure Permissions | 6.6.18.1.4e: Permission settings |
| Search for users defined in the operating system registry or LDAPserver | Available from multiple security tasks | 6.6.18.1.4f: Security search dialog |

# 6.6.18.1.4a: Properties for configuring global settings

Settings on the General, Authentication Mechanism, and User Registry tabbedpages specify global settings for all applications to share. These values cannotbe customized for individual applications.

The Application Defaults page specifies default settings that the administrator canaccept or override for individual applications.

| Goal | Wizard page/Property help |
|---|---|
| Enable security; specify how long to cache authentication lookup results | 6.6.18.1.4a.1: General page |
| Define a security realm; specify a default challenge type; specify aWeb page for user logon | 6.6.18.1.4a.2: Application Defaults page |
| Select either the operating system registry or an LDAP directory serviceto authenticate users | 6.6.18.1.4a.3: Authentication Mechanism page |
| Specify the identity under which the product security server will run;provide details about LDAP directory if it is the chosen authentication mechanism | 6.6.18.1.4a.4: User Registry page |

# 6.6.18.1.4a.1: General settings of the Configure Global Settings task

**Enable security**

Specifies whether to enable or turn off WebSphere Application Server security.

If the administrator deselects the check box, all other security settings will be disregarded andapplications and resources will be unprotected.

**Security Cache Timeout**

Specifies how many seconds servers should cache security information received from the user registry or directory service, improving the performance with respect to authorization lookups.

To make changes to this property take effect, stop and restart the application server or servers under which the applications using the security settings will run.

# 6.6.18.1.4a.2: Application Default settings of the Configure Global Settings task

**Realm Name**

Specify the security realm to which the application should belong. See article 0.18.7 to learn more.

**Challenge Type - None**

Specifies that clients will not be challenged for authentication information.

⚠ If the administrator has protected a resource within an application, selecting None will deny users access to that resource.

**Challenge Type - Basic**

Specifies that clients will be prompted for a user ID and password, usually acquired through a basic HTTP 401 challenge.

**Challenge Type - Certificate**

Specifies that clients must provide a digital certificate for authentication.

If the administrator additionally selects Default to Basic, clients without certificates will be permitted to use the basic authentication scheme.

**Challenge Type - Custom**

Specifies that clients will log in using servlet-generated Web pages you specify in the Login URL and Relogin URL fields.

Currently, the administrator needs to enter the same URL in each of the two fields. The URL is intended to reference a Web page containing an HTML-based login form, but the administrator can enter the URL of any Web page, whether or not it offers a login form.

For example, the field could contain the URL

```
http://host.name.com/login/deny.html
```

for a Web page created to deny access to users without allowing the users to attempt login.

**Login URL**

Specifies the fully-qualified path to the Web page to be presented for users to log on to. The administrator should complete this field if he or she specified the Custom challenge type. Currently, this field must match the Relogin URL.

The product does not validate this field or the Relogin URL.

If Single Sign-On (SSO) is enabled, the URL must be contained within thedomain specified in the Single Sign-On configuration.

**Relogin URL**

Specifies the fully-qualified path to the Web page to be presented when the connection is released and a user must log on again. Complete this field if you specified the Custom challenge type. Currently, this field must match the Login URL.

**Use SSL to connect client and Web server**

Specifies that an SSL connection is required between the client and Web server. Requests that do not arrive over SSL will be refused.

This check box applies to the Basic, Certificate, and Custom challenge types.

# 6.6.18.1.4a.3: Authentication Mechanism settings of the Configure Global Settings task

**Local Operating System**

Specifies that information will be authenticated with the underlying operating system's user registry. Usually, such registries apply basic authentication, checking a user ID and password.

This selection influences the fields displayed on subsequent tabbed pages. If the administrator enables authentication by the Local Operating System, some properties described in this file will not be displayed because they do not apply to that situation.

If using a Windows-based operating system belonging to a domain, see the note about configuring permissions.

**Lightweight Third Party Authentication (LTPA)**

Specifies that basic or certificate authentication will be used to authenticate the user with an LDAP directory service. If you select LTPA, provide additional information:

- **Token Expiration:** Specifies how many minutes can pass before a client using an LTPA token must authenticate again. LTPA uses tokens to store the authenticated status of a client.

  Legal Values:

  - A **positive** integer indicates the token life, in minutes
- **Generate Keys:** Specifies whether the LTPA mechanism should generate a new set of encrypted keys right now. When prompted for a password, supply a string that is used by the underlying key generation mechanism.

  When the administrator selects LTPA as the authentication mechanism, encryption keys are generated automatically. The administrator need not click this button unless he or she would like those initial keys to be replaced by new keys.
- **Import from File:** Specifies whether to import a file containing the encryption keys. This allows IBM WebSphere Application Server to share keys from other IBM products that support this functionality.

  ℹ If the administrator specified a password when he or she created the key file, the administrator will be prompted for that password when he or she tries to import the key file.
- **Export to File:** Specifies whether to export a file containing the encryption keys. This allows IBM WebSphere Application Server to share the keys with other IBM products that support this functionality.
- **Enable Single Sign On:** Enabling Single Sign On (SSO) tells LTPA to store extra information in the tokens so that other applications can accept clients as already authenticated by WebSphere Application Server. When clients try to access the other applications, they will not be interrupted and asked to log in.

  - **Domain:** Restrict SSO to servers in the domain you specify in this field.
  - **Limit to SSL connections only** Specifies to use a connection with SSL for Single Sign On, to prevent the SSO token from flowing over non-secure connections.

  WebSphere Application Server Version 3.5 introduces support for Single Sign On with Domino Server. WebSphere Application Server can import and export keys and provide a Single Sign On between the WebSphere Application Server and Domino environments.

# 6.6.18.1.4a.4: User Registry settings of the Configure Global Settings task

The content of the User Registrytabbed page changes depending on the selections on the AuthenticationMechanism tabbed page:

- If the administrator selected Local Operating System on the Authentication Mechanism tabbed page, only the Security Server ID and Security Server Password properties will be displayed on the User Registry page.
- If the administrator selected LTPA on the Authentication Mechanism tabbed page, several additional properties described in this file will be available on the User Registry tabbed page.

**Security Server ID**

Specifies the user ID the Application Server Version 3 security server component will run under.

The ID corresponds either to an operating system ID or an LDAP directory ID, depending on the selection on the Authentication Mechanism tabbed page.

**Security Server Password**

Specifies the password the Application Server Version 3 security server will run under.

**Directory Type**

Specifies the directory service product to use to locate information against which to authenticate users and groups.

View supported directory services

All of the supported directory service choices have predefined filters and ID maps the administrator can view by clicking the Advanced button. If the administrator changes the filters or ID maps, the Directory Type will automatically change to Custom.

"Custom" can refer to any of the supported directory types, with customized filters and ID maps, or to an unsupported directory service for which the administrator has defined filters and ID maps using the Advanced options.

**Advanced**

Specifies optional properties the administrator can use to define search filters and ID maps for the selected directory service. The administrator can also specify how certificates will be used to locate entries in the LDAP directory service.

- **Initial JNDI Context Factory:** Specifies the JNDI Context Factory to use. If the field is blank, Application Server uses the Context Factory provided by IBM.
- **Directory Type:** Specifies the brand of the directory service. Only directory services compatible with Application Server Version 3 are listed.

  If the administrator changes the filter and ID map values on the Advanced dialog box, the Directory Type will change to Custom, even if the filters and ID maps the administrator is defining apply to a supported directory service.
- **User Filter:** Specifies the property by which to look up users in the directory service. For example, to look up users based on their user IDs, specify `(ampersand(uid=%v)(objectclass=inetOrgPerson) where` *`ampersand`* `is the ampersand symbol.`

  For more information about this syntax, see the LDAP directory service documentation.
- **Group Filter:** Specifies the property by which to look up groups in the directory service.

- **User ID Map:** Specifies the piece of information that should represent users when users are displayed. For example, to display entries of the type object class = inetOrgPerson by their IDs, specify `inetOrgPerson:uid`.

  This field takes multiple `objectclass:property` pairs delimited by a semicolon (";").
- **Group ID Map:** Specifies the piece of information that should represent groups when groups are displayed. For example, to display groups by their names, specify `*:cn`.

  The * is a wildcard character that searches on any object class in this case. This field takes multiple `objectclass:property` pairs delimited by a semicolon (";").
- **Group Member ID Map:** Specifies which property of an objectclass stores the list of members belonging to the group represented by the objectclass.

  This field takes multiple `objectclass:property` pairs delimited by a semicolon (";"). For more information about this syntax, see the LDAP directory service documentation.
- **Certificate Mapping:** Specifies the certificate field(s) against which to check certificate validity.
  - **Exact Distinguished Name:** Checks certificate validity against the exact distinguished name held by the LDAP directory service. It locates the subject DN of the certificate in the directory.
  - **Unique Key:** Checks certificate validity using a hash function on two predetermined attributes.
  - **Certificate Filter:** Enables the Filter field for specifying an property of your choice.

    Create an LDAP search filter with the contents of the certificate that will attempt to match a single Directory entry. An example of a search filter is:

    (*ampersand*)`(cn=${Subject:cn})(version=${Version}))`

    where *ampersand* is the ampersand symbol.

    The list of possible variable substitutions referring to portions of the certificate is given here (in the format "variable = meaning"):
    - **PublicKey** = Public Key of the certificate
    - **Issuer:attribute** = The issuer distinguished name of the certificate. An attribute value must be specified that allows the administrator to select a specific attribute of the Distinguished Name. To retrieve the entire Distinguished Name, use the "DN" attribute.
    - **NotAfter** = The date at which the certificate is no longer valid
    - **NotBefore** = The date before which the certificate is not valid
    - **SerialNumber** = The serial number of the certificate
    - **SigAlgName** = The signature algorithm name
    - **SigAlgOID** = The OID of the signature algorithm
    - **SigAlgParams** = The DER encoded signature algorithm parameters
    - **Subject:attribute** = The subject distinguished name of the certificate. An attribute value must be specified that allows the administrator to select a specific attribute of the Distinguished Name. To retrieve the entire Distinguished Name, use the attribute DN.
    - **Version** = The version number
- **Certificate Filter:** If you specified the Filter Certificate Mapping, this property specifies the certificate property against which to check certificate validity.

**Host**

Specifies the host name of the machine on which the directory service resides.

**Port**

Specifies a port number for the directory service. Port 389 is the LDAP default.

**Base Distinguished Name**

Specifies the base distinguished name of the directory service, indicating the starting point for LDAP searches of the directory service. (See RFC 1779 for a discussion of this technique).

Some examples include:

- uid=anyusername
- ou=people
- o=ibm

This field is required unless the product will be using a Domino directory service, in which case the administrator can leave the field blank to bind anonymously.

The host name, port, and base DN you specify in the Host, Port, and Base Distinguished Name fields are combined to form an LDAP URL, such as

```
ldap://myserver:1234/o=ibm
```

where myserver:1234 is the host name and optional port number for the directory service, and o=ibm is the base distinguished name.

**Bind Distinguished Name**

Specifies the distinguished name for Application Server to use to bind to the directory service. If left blank, the Application Server binds anonymously.

See the previous Base Distinguished Name field description for examples of distinguished names.

**Bind Password**

Specifies the password for the Application Server to use to bind to the directory service.

**Use SSL to connect to directory**

Specifies whether to use an SSL connection between the security server and your LDAP directory service.

If the administrator selects this option, the SSL connection will use the same SSL keyring as the one defined for SSL connections between application servers.

# 6.6.18.1.4a.4.1: Supported directory services

For a list of supported directory services, see the prerequisites Web site discussed in the article about the site. An additional Custom option is available for tailoring any of the default filters to fit a *supported* LDAP directory service.

# 6.6.18.1.4b: Properties for configuring application security

**Application Identity - User ID**

Specifies the user ID under which the application will run. The ID is usedfor delegation of the application's resources.

**Application Identity - Password**

Specifies a password for the Application Identity.

# 6.6.18.1.4: Properties for configuring method groups

**New Method Group**

Specifies a new method group. Type the group name and click Add.

To see the new method group, expand the Method Groups folder.

**Remove Method Group**

Removes the selected method group. Note, the default (predefined) methodgroups cannot be removed.

# 6.6.18.1.4d: Properties for configuring resource security

**Run-As Mode**

Indicates the Run As mode specified by the deployment descriptor of the enterprise bean.

- SYSTEM - The bean method will run with the security identity of a privileged system account
- CLIENT - The bean method will run with the security identity of the client
- SPECIFIED - The bean method will run with the Run As Identity you specify

If the value is SPECIFIED, the administrator must map the delegation ID(Run As Identity) to an application identity established using the ConfigureApplication Security wizard.

The Run As Identity is specified in the bean deployment descriptor.For more information, see the Sun Microsystems Enterprise JavaBeans 1.0 specification.

Note, the product maps enterprise bean methods to application identities, rather than to Run As Identities,for methods whose Run As Modes are SPECIFIED. The deployment tooldoes not allow roles to be specified by the Run As Identity of a beandeployment descriptor.

**Run-As Identity**

Specifies the security identity under which the enterprise bean method will be executed.

This value is ignored if the Run As Mode is CLIENT or SYSTEM.

The Run As Identity is specified in the bean deployment descriptor. For more information,see the Sun Microsystems Enterprise JavaBeans 1.0 specification.

**Mapped Application Identity**

Specifies the application identity to which to map a bean or bean method. You need onlydo this if the bean method's Run As Mode is "SPECIFIED."

The application identity is a user ID and password the administrator specifiesusing the Configure Application Security task. It is the user ID and passwordunder which the application and its methods will run.

**Specified Methods**

Specifies the enterprise bean methods for which delegation properties (Run As Mode, and so on) arebeing established.

**Resource**

Indicates the resource with which the administrator is working. To changethe selected resource, return to a previous wizard page, on which you canselect a different resource.

# 6.6.18.1.4e: Properties for configuring permissions

**WebSphere Permissions**

Specifies the cross product of all available Applications and MethodGroups. Eachpair (permission) is represented by a `Application Name-MethodGroup Name` notation. Expand a permission to see the users granted that permission.

# 6.6.18.1.4f: Properties for the security search dialog

The search dialog lets the administrator locate users and groups inthe underlying operating system user registry or in the directoryservice, whichever is the current authentication mechanism.

Use this reference to look up a particular property as you configurethis object type. Please note, not all properties are available from every administrative interface. To learn how to access these propertiesfrom a given administrative interface, refer to the task help forconfiguring this object type.

**Everyone, All Authenticated Users, or Selection**

Specifies for who to search for, with respect to the userregistry or directory service.

**Search For**

Specify whether to search for users, groups, or roles. The administratorcan only search for roles if the directory service supports them.

**Search Filter**

Specify a pattern against which to search for principals. Awildcard character ("*") can be used.

**Search Results**

Lists matches to the specified Search Filter. The SearchResults area remains empty until the administrator clicks the Search button.

# 6.6.18.1a: Summary of security settings with the Java administrative console

Use the Configure Global Settings task wizard to specify global and default security settings for all applications:

- Global settings apply to existing and future applications and cannot be customized.
- Default settings apply only to future applications and can be customized.

The default settings are used as a template or starting point for configuring individual applications. The administrator should still explicitly configure security settings foreach application.

| Goal | Wizard page description | Global or default? |
|---|---|---|
| Enable security; specify how long to cache authentication lookup results | 6.6.18.1a.1: General | Global |
| Specify default security realm and challenge type for applications | 6.6.18.1a.2: Application Defaults | Default |
| Specify how to authenticate users | 6.6.18.1a.3: Authentication Mechanism | Global |
| Provide detail about the selected authentication mechanism | 6.6.18.1a.4: User Registry | Global |

IBM WebSphere Application Server provides security at several levels.The security characteristics of an individual application can come fromany of these levels. At the most general level are the global securitycharacteristics set up to act as application defaults. This filebriefly describes these global values.

In WebSphere, the global defaults for security apply to allapplications. Some of the values can be changed on anapplication-by-application basis, and others remain constant acrossall applications.

An example of a value that can be set on a per-application basisis the type of authentication procedure. You must establish adefault procedure, but this value is used for applicationsthat do not explicitly indicate how they will authenticate users.

An example of value that cannot be changed on a per-applicationbasis is whether to ignore security or not. In Application Server,security is either enabled or disabled. If it is enabled, allapplications are secured according to their configurations. Ifsecurity is disabled, all applications run unsecurely, regardlessof their configurations.

# 6.6.18.1a.1: About enabling security with the Java administrative console

Configure Global Settings task:

▶ 1. Enable security
2. Set application security defaults
3. Specify how to authenticate users
4. Provide details about the authentication mechanism

IBM WebSphere Application Server security can be enabled or not enabled. If securityis not enabled, all other security settings are ignored.

# 6.6.18.1a.2: About setting application security defaults with the Java administrative console

Configure Global Settings task:

1. Enable security
▶ 2. Set application security defaults
3. Specify how to authenticate users
4. Provide details about the authentication mechanism

Use the Application Defaults tab of the Configure Global Settingswizard to specify the default security realm and challenge typefor applications. These values provide a starting configuration for applications, but some can be refined on an application-by-application basis. For example, you can set a default challenge type for authentication but allow some applications to use different challenge types.

## Selecting a default security realm

All applications need to belong to a security realm. In Version 3,all applications must belong to the same security realm.

The administrator can specify a default realm. When a particular applicationis configured, the administrator can override thedefault realm by specifying a different realm for the application.

Single Sign-On (SSO) support is applied to realms. If a user who haslogged on to one realm tries to access an application in anotherrealm, the user will be prompted to log into the second realm.

## Selecting a default challenge type

The challenge type specifies how users will be challenged for authentication credentials whenthey try to access a resource or application.

Challenge types range from no challenge, a user ID and password, or a digital certificate to a custom challenge using Web pages.

# 6.6.18.1.a.3: About specifying how to authenticate users with the Java administrative console

Configure Global Settings task:

1. Enable security
2. Set application security defaults
3. ▶ Specify how to authenticate users
4. Provide details about the authentication mechanism

Use the Authentication Mechanism tab of the Configure Global Settingswizard to specify how to authenticate or verify the user data receivedas a result of a challenge (such as a logon screen).

The WebSphere security server must havesome way to check the user ID and password, digital certificate, or otheruser identification for credibility. It relies on the authenticationmechanism specified by the administrator.

## Before performing this subtask

Before completing the Authentication Mechanism subtask, theadministrator needs to use other Configure Global Settings subtasks tospecify how to challenge users for identification when theytry to access applications.

## Selecting how to authenticate user data

Users can be authenticated by one of two authentication mechanisms, either theoperating system user registry, or a supported directory service.

Whichever authentication mechanism the administrator selects, theadministrator can use the General tabbed pageto specify a Security Cache Timeout value. The timeout specifies how long the securitysystem should keep authentication data received from the directory service or user registry.

The timeout value specifies the number of seconds after which authenticationinformation will be considered unreliable. The next time the information is needed,it will be sought again using the authentication mechanism.

# 6.6.18.1a.4: About providing authentication mechanism details with the Java administrative console

Configure Global Settings task:

1. Enable security
2. Set application security defaults
3. Specify how to authenticate users

▶ 4. Provide details about the authentication mechanism

Use the User Registry tab of the Configure Global Settings wizard tospecify details about the chosen authentication mechanism, such as anoperating system user registry or LDAP directory service.

## Before performing this subtask...

Before completing the User Registry information, the administrator needsto use other Configure Global Settings subtasks to specify (1) how to collect identity informationfrom users trying to access applications and other resources and (2) how toauthenticate the information received.

## Selecting either the OS user registry or LDAP directory service

The user registry or directory service keeps records of users with permission to access resources in the systems administration domain. The security systemlooks to the user registry or directory service to provide information for determiningwhether to authenticate a user or group successfully.

The operating system user registry simply compares users to valid usersin the underlying operating system. When the administrator selects the Local Operating Systemchallenge type, the User Registry tabbed page dynamically changes to allow theadministrator to set a security ID and password under which the application will run. Theinformation is used for delegation of the application resource.

When the administrator selects Lightweight Third-Party Authentication (LTPA) as theauthentication mechanism, the User Registry tabbed page changes. This change enables theadministrator to specify information about the Lightweight Directory Access Protocol (LDAP)-compliant directory service product to be used.

# 6.6.18.1b: About configuring application security with the Java console

IBM WebSphere Application Server provides security at several levels.The security characteristics of an individual application can come fromany of these levels. At the most general level are the global securitycharacteristics set up to act as application defaults. However,the administrator can and should set application-specific values that either comply with or override the global defaults.

The Configure Application Security task lets the administrator override the challenge typeand realm defaults specified as the global settings. Usingthe task wizard, the administrator specifies the realm, challenge type and security identity(user ID and password) for a particular application.

For other properties, such as the authentication mechanism and userregistry details, the application will share the global settings. These settingscannot be customized for a particular application.

Enabling security inthe administrative console is also a global setting, applied as "all or none"to applications in the administrative domain.

# 6.6.18.1c: About assigning method groups with the Java console

The Work with Method Groups task wizard provides an easy way to create custommethod groups.

Assuming you have configured one or more applications already, defining custom methods groups can be considered the optioal first step of a three-part task:

1. Create custom method groups, if desired. WebSphere provides a set of default method groups, which you can use instead of creating your own.
2. Assign the methods of each resource to method groups. WebSphere provides a default assignment of methods to the predefined method groups, if you choose to use it. If you do, you should check the assignment and modify it appropriately.
3. Assign permissions (access to method groups of applications) to users.

## Example

Suppose you have defined an application namedMonthlySalesChart. The application contains a servlet. Youcan configure application-specific security as follows:

| | |
|---|---|
| Create custom methods | You can protect all methods that allow writing to a database with a method group called AllowedToWrite and put read-only methods in a method group called AllowedToRead. |
| Assign methods to method groups | You can put the HTTP_PUT method of the servlet into the AllowedToWrite method group, and the HTTP_POST method into the AllowedToRead method group. |
| Assign permissions | Give users the following permissions as appropriate:<br>• MonthlySalesChart-AllowedToRead<br>• MonthlySalesChart-AllowedToWrite |

# 6.6.18.1d: About assigning methods to method groups with the Java console

After defining optional custom method groups, the administrator can specify whichmethods in application resources belong to each method group.

Because IBM WebSphere Application Server provides default method groups, definingcustom groups is optional.

If the administrator specifies to use the default groups when working with methodgroups, WebSphere ApplicationServer will take a first pass at categorizing the bean methods into the various groups, basedon the method names.

Afterwards, the administrator can create additional groups and move methods there, ormove methods among the default groups if needed. Use the Work with Method Groups task to establish new method groups.

To sort resource methods into method groups for protection, perform theConfigure Resource Security task once for each resource in the application.

⚠ If the Web server is already runningwhen the administrator configures resource security for an HTML file, JSP file, or otherWeb resource, the administrator must stop the WebSphere administrative server andstart it again for the protection to take effect.

# 6.6.18.1e: About assigning permissions

Use the Assign Permissions task to assign permissions to users and groups,giving them access to method groups in enterprise applications.

# 6.6.18.5: Managing security IDs for the application server and administrative accounts

## Choosing the process identity

During installation, you must identify an existing user ID and password under which the WebSphere administrative server and application serverswill run. It is the operating system identity associated with theprocess. The operating system uses the identity to determine accessto resources such as files and sockets. It is not an ID that is typicallyused by a human user.

If you are using the operating system registry as the authenticationmechanism for checking the identity, then the identify must meetthe following requirements:

- On UNIX platforms, you must use the **root** account.
- On Windows NT, the account must be a member of the Administrators group and must have the rights to "Log on as a service" and to "Act as part of the operating system."

  ℹ Do not use an account whose name matches the name of your machine or Windows Domain. The WebSphere administrative server will not work in such a case.

  ℹ WebSphere requires the NT Browser Service to be active because WebSphere uses this service to contact the NT Primary Domain Controller (PDC). Also, be aware that, although WebSphere uses the NT PDC, it does not make use of the NT Backup Domain Controller (BDC). If the PDC is not available, WebSphere does not default to the BDC.

If you are using an LDAP directory service for authentication,then the process identity does not need any special privileges. See the information about running as non-root on UNIX-based systems.

## Establishing the administrative identity

When you enable WebSphere security by using the Configure Global Settings security administration task,you configure an initial administrative identity for WebSphere.This identity needs to be a valid user for theauthentication mechanism you have chosen (an operating system userregistry or LDAP directory service), but it does not need "root" orother special privileges.

After configuring the administrative identity, when you restart theadministrative server and try to administer the product, you must login with the administrative identity when you are prompted for a userID and password.

You can also configure the product security to allow administrativeaccess by other IDs, in addition to the initial ID you established.

## Setting up additional administrative accounts

During the installation of WebSphere Application Server, you mustidentify an existing account that will act as the first administrativeaccount for WebSphere. After enabling security, this account willbe the only one authorized to administer WebSphere. You can, however,use the account to authorize other administrative users.

To authorize other valid accounts defined in the operating systemuser registry or in your directory service product, use the AssignPermissions task on the Tasks tab of the WebSphere administrativeconsole (in the Security task group). With this task, you can grantusers access to the protected functions, which are listed in the formatAdminApplication-*function_name*in the task.

Access to the administrative functions of the IBM WebSphereApplication Server product is controlled by the *admin*application, to which the functions belong.

## Steps

1. Click the Tasks tab to display the Tasks tree.
2. Click Security --> Assign Permissions.
3. Click an AdminApplication-*function_name*function.
4. Click the Add button to produce a search dialog.
5. Use the search dialog to give permission to everyone or selected users or groups. You can search for a user or group in your local operating system user registry or directory service product.
6. Click the OK button when you are finished with the search dialog.
7. Back in the main console window, verify that the user or group is listed under the permission you granted to the user or group.
8. Exit this task by choosing another task on the Tasks tab.

## Giving NT users administrative privileges

During the installation of WebSphere Application Server, you mustidentify an existing account that will act as the first administrativeaccount for WebSphere. On Windows NT, the account must be a member ofthe Administrators group and must have the rights to "Log on as a service"and to "Act as part of the operating system."

To give an account these rights, follow this procedure:

1. Start the user manager for Windows NT or Domains and click Start --> Programs --> Administrative Tools (Common) --> User Manager.
2. Select Policies --> User Rights from the menu bar on the dialog box.
3. Check the Show Advanced User Rights check box in the dialog box.
4. From the list labeled Right:, select Log on as a service.
5. If the administrative account is not listed in the Grant To: list:
    - Click Add.
    - Click the Show Users button in the resulting dialog box.
    - Select the individual User or Group.
    - Click Add to include the account in the Add Names list.
    - Click OK to exit the dialog box.
6. Click OK in the User Rights Policy dialog box.
7. Return to the second step and repeat the procedure, specifying the "Act as part of the operating system" right instead of the "Log on as a service" right.
8. Close the User Manager window.

If you then open the Services menu and modify the Log On As accountfor the service, the account you specify here will automatically begranted the "Log on as a service" right.

ℹ Do not use an account whose name matches the name of your machineor Windows Domain as the administrative account. The WebSphereadministrative server will not work in such a case.

## Changing passwords for administrative accounts

Good security requires the periodic changing of passwords, and thisincludes those for your WebSphere administrative accounts. These passwordshave to be changed in two places, in a particular order. If this is doneincorrectly, it can create a situation in which the WebSphere administrativeserver cannot restart. This file describes the best way to change anadministrative password.

## Steps

1. Make sure the WebSphere administrative server is running. This is crucial. Do *not* change an administrative password unless the server is running.
2. Change the password in the user registry by using the utility for your operating system or LDAP service.
3. Login to the WebSphere administrative console using the new password. Attempts to use the old password will fail.
4. Click Security --> Specify Global Settings --> User Registry in the administrative console.
5. Change the password for the administrative user to the new password.
6. Stop and restart the administrative server.

# 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

ℹ 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.

In WebSphere, Web resources are not protected by default. Additionally,the WebSphere example applications install some demonstrationservlets that perform administrative tasks. Because Web resourcesare not secured by default, and because these servlets performwork usually reserved for administrators, they represent a possiblesecurity problem, particularly for production applications.

The following describes some user practices and their potential risks.When applicable, it uses components of the example application to illustrate the point.

## Serving static HTML files

*Purpose:* This servlet serves static HTML pages. For example, the "file" servlet in the default configuration under Web applicationcalled "examples" serves static pages from the application's documentdirectory (e.g., <*WAS_HOME*>/hosts/default_host/examples/web).If you access http://<*host*>/webapp/examples/index.html,this servlet is invoked to serve the page.

*Security consideration:* If you place a file containing confidential information within these directories, someone who knows the filename butdoes not have access to it through the operating system can invoke thisservlet to obtain the file.

*Solution:* Protect all the files by protecting the URI associated with the "file" servlet. Such a URI typically ends with a "/" (e.g.,/webapp/examples/). Alternatively, if you want to protect only certainfiles (e.g., /webapp/examples/test.html) served by this servlet, thenprotect the files individually. To do this, you must:

1. Create a URI for each file.
2. Associate the URI with the file servlet by adding the URI to the web-path list of the servlet.
3. Secure access to the URI.

## 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.

## The Web application "admin"

*Purpose:* To administer WebSphere configuration.

*Security consideration:* Currently, security is *not*supported for this feature, even if you have enabled security. If youinstall this application, anyone can change the configuration, evenif WebSphere security is enabled.

*Solution:* The "admin" 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:**
  The "unique key certificate" filter is offered as a security option in the WebSphere administrative console, but is not supported for Application Server Version 3.x.

  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

Security is not automatically updated when changes are made to a bean.You must redeploy the resource security in order for the method groups topick up the changes to the bean.

## Adding a method to a bean

If you add a method to a bean, you must go back into resource securityand associate the new method with a method group.

## Modifying a method on a bean

If you modify a method on a bean, you must resecure the bean as follows:

1. Delete the method group for the bean.
2. Click **Finish**.
3. Re-associate the method group with the modified method.

## Unprotecting resources

Resources protected under WebSphere can be unprotected, if necessary.Depending on the resources and how they are configured into applications,the techniques for removing security differ. This file describes howto remove security in the following situations:

- All resources associated with an enterprise application
- A particular bean associated with an enterprise application
- All URIs associated with a web application
- A particular URI associated with a web application

## Unprotecting all resources associated with an enterprise application

If you want to remove protection from all the resources associated withan enterprise application, the most efficient approach is to unprotectthe application itself. For example, if you have granted the permissions associated with the application ("*application-methodgroup*"pairs) to a specific user, group or to all authenticated users, the resourcesare considered protected. To unprotect these resource, you can grant thosepermissions to "Everyone". By granting the permissions to everyone, a user need not be authenticated to access the resources under that application.

## Unprotecting an enterprise bean associated with an enterpriseapplication

If you want to remove protection from a specific bean (or set of beans)associated with an application while maintaining the security on the otherresources in the application, remove the bean (or beans) from theapplication and create a new application that is explicitly unprotected.

When you remove beans from the application, the security configuration associated with the application no longer applies to them. However, enterprisebeans are protected unless security policies to the contrary are specified.To completely unsecure them, you need to create a new application consistingof the beans to be unsecured. After performing security configurationsteps, grant the permissions associated with the new application to"Everyone." This is equivalent to unprotecting all the resourcesassociated with the new application.

To remove resources from a secured enterprise application, use the "EditEnterprise Application" task. On the last panel, you can remove resourcesassociated with the application. Use it to remove the desired beans.

## Unprotecting all URIs associated with a web application

If you want to remove protection from a web application (including allassociated URIs) while maintaining the security on the other resourcesin the enterprise application, remove the web application (or applications)from the enterprise application.

To remove resources from a secured enterprise application, use the "EditEnterprise Application" task. On the last panel, you can remove resourcesassociated with the application. Use it to remove the desired webapplications.

## Unprotecting specific URIs

If you want to remove protection from specific URIs in a web application,remove the method-group configuration for the URIs. Use the "ConfigureSecurity Method Groups" task and select the URI you want to unprotect.After the URI is selected, proceed to the next screen, where you viewthe classification of methods into method groups. For example, theHTTP_GET method may belong to the ReadMethods method group. Selectthe method groups associated with the methods you want to unprotectand remove them. This eliminates the associate between a method groupand a URI, leaving the URI unprotected. Because web resources areunprotected by default, no authentication is required to access them.

## Protecting individual JSP files

This file describes the steps necessary for selectively protecting JSP files, that is, how to protect individual JSP filesbased on their Web paths (URIs) when you do not want to applythe same protection to *all* the JSP files in the system.

Note, the instructions for adding a JSP Web path to a webapplication advise you to use the "Add a JSP or a web resource" taskwizard in the administrative console. This action adds theJSP Web path, not the actual JSP file, to the Web application.But when you follow the configuration steps to protect a JSP Web path, the Web path is treated separately from theWeb application; instead, it is treated as a Web server resource.Therefore, security does not work as intended.

The following procedure will be needed until product defect number88065 is addressed. Check the "fixed defects" list accompanying IBMWebSphere Application Server fix packs to ascertain whether a givenfix pack has addressed the defect.

To protect individual JSP files using WebSphere security, follow these steps:

1. If you used the "Add a JSP or web resource task" to introduce a new JSP Web path and associate with Web applications, remove all of the Web paths.
2. Start the WebSphere administrative console.
3. Select the Topology view.
4. Expand the Topology tree to show the node, application server, and servlet engine containing the Web application to which you want to add the JSP.

5. Select the JSP processor servlet in that Web application.
6. In the list of Web paths, locate:
   `/default_host/<webapp-path>/*.jsp`
   where default_host is the default virtual host or one that you have created, and <*webapp-path*> is the path to the Web application.
7. Click "Add" to add to the Web path list.
8. Enter the JSP Web path (URI) that you want to protect, such as:
   `/default_host/<webapp-path>/toBeProtected.jsp`.
   If you have multiple files to protect, enter the URI for each one.
9. Apply your changes.
10. Follow the resource security configuration steps to protect these newly added JSP files.
11. Restart the application server hosting the Web application and JSP files.

# 6.6.18.8: Using Microsoft Active Directory as an LDAP Server

To use Miscrosoft Active Directory as the LDAP server for authenticationwith WebSphere Application Server, there are some specific steps you musttake. By default, Microsoft Active Directory does not allowanonymous LDAP queries. To make LDAP queries or browse thedirectory, an LDAP client must bind to the LDAP server usingthe distinguished name (DN) of an account that belongs to theAdministrator group of the Windows system.

To set up Microsoft Active Directory as your LDAP server, followthis procedure:

1. Determine the full DN and password of an account in the Administrators group. For example, if the Active Directory administrator creates an account in the Users folder of the Active Directory Users and Computers Windows NT/2000 control panel and the DNS domain is ibm.com, the resulting DN has the following structure:
   `cn=<adminUsername>, cn=users, dc=ibm, dc=com`
2. Determine the short name and password of any account in the Microsoft Active Directory. This does not have to be the same account as used in the previous step.
3. Use the WebSphere Application Server administrative console to set up the information needed to use Microsoft Active Directory:
   1. Start the administrative server for the domain, if necessary.
   2. Start the administrative console, if necessary.
   3. On the administrative console, click **Wizards**.
   4. Select the **Configure Global Security Settings** task.
   5. Click the **User Registry** tab and set the following fields as described:
      - **Security Server ID**: The short name of the account chosen in 2
      - **Security Server Password**: the password of the account chosen in step 2
      - **Directory Type**: Active Directory
      - **Host**: The DNS name of the machine running Microsoft Active Directory
      - **Base Distinguished Name**: the domain components of the DN of the account chosen in step 1. For example:
        `dc=ibm, dc=com`
      - **Bind Distinguished Name**: the full DN of the account chosen in step 1. For example:
        `cn=<adminUsername>, cn=users, dc=ibm, dc=com`
      - **Bind Password**: the password of the account chosen in step 1
   6. Click **OK** button to save the changes.
   7. Stop and restart the administrative server to make the changes take effect.

# 6.6.18.9: Specifying authentication options in sas.client.props

You can use the sas.client.props file to direct WebSphere ApplicationServer to authenticate users by prompting or by using a user ID and password set in the properties file. The following steps describe theprocedure:

1. Locate the sas.client.props file. By default, it is located in the properties directory under the *<product_installation_root>* of your WebSphere Application Server installation.
2. Edit the file to set up the authentication procedure:
    - To authenticate by prompting, set the loginSource property to the value "prompt":
      ```
      com.ibm.CORBA.loginSource=prompt
      ```
    - 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

During product installation, you must decide whether or not to check a boxlabeled "Use demo keyring file." Keyrings are used for certain typesof authentication.

If you plan to produce your own keyrings, you do not need tocheck this check box. If you are not sure, check the box. Thisway, if you later need the functionality for testing, you'll have it.Your decision for this check box will *not* affect theoverall success of the security installation.

ℹ Do *not* use the demo keyring in production systems. It includesa self-signed certificate for testing purposes, and the privatekey for this certificate can be obtained easily, which puts the securityof all certificates stored in the file at risk. This keyringis intended only for testing purposes.For information on obtainingproduction certificates, see Requestingcertificates; for information on creating keyring files,see Tools for managing certificates and keyrings.(The links will only work if you are reading this as part of the InfoCenter that you can obtain fromhttp://www.ibm.com/software/webservers/appserv/infocenter.html).

# 6.6.18.11: SecureWay Directory Version 2.1

- Overview
- Software requirements

## Overview

Version 2.1 of the SecureWay Directory provides many newenhancements over its predecessor, eNetwork LDAP Directory Version1.1.1, which was originally only available on AIX4.3.1. The major enhancements include:

- DB2 Version 5.0 as the directory data storage facility
- Alias support
- Improved search and ACL support
- Support for popular Web servers
- Significant scalability
- Improvements to replication and performance

The IBM SecureWay Directory V2.1 includes an LDAP Version 2 server(RFC 1777,1778, 1779). It is enhanced to support aliases and IETF LDAPVersion 3 extensions for SSL, referrals, replication and accesscontrol.

SSL provides encryption of data and authentication using X.509v3public-key certificates. The server may be configured to run with orwithout SSL support. The server supports LDAP referrals, allowingdirectories to be distributed across multiple LDAP servers. Replicationis supported which makes additional read-only copies of the directoryavailable, improving performance and reliability of access to the directoryinformation. A powerful, easy-to-manage access control model issupported. Configuration and administration of the LDAP Directory isaccomplished through an improved web-based interface.

This product is available on AIX, Windows NT/Intel and Solarisplatforms. It currently supports ten languages including English,French, German, Italian, Spanish and Brazilian Portuguese on AIX andNT. Catalan is also supported on AIX. It does not support DBCSlanguages on Solaris.

The SecureWay Directory, Version 2.1 supports up to fifteen millionentries with peak sub-second response time for searches.

Performance of the Directory is improved with statement caching andoptimization. Multi-threading improvements allow Directory clients toperform true multi-threaded connections and make concurrent operations withthe DB2 server. The DB2 program provided with the Directory may only beused by the SecureWay Directory function.

ACL support provides role-based authorization, assigns multiple usersownership of an entry, and removes the requirement to set ACLs on every nodeto give users access to their own information.

The Web servers Apache, Lotus Domino Go, Netscape FastTrack, and NetscapeEnterprise Web servers are supported for LDAP administration.

Directory client access is supported using LDAP or HTTP protocols.Client applications can be developed using the enhanced elements provided forsupporting LDAP Version 3 protocols and APIs. Also included is the JavaNaming and Directory Interface (JNDI) client API that provides Javaapplications with access to LDAP-enabled directories. Both clientssupport access to SecureWay Directories using LDAP Version 2 or Version3. Directory client applications can be built for Windows NT, Windows95, Solaris, and HP-UX using the IBM LDAP Client Pack, which can be orderedseparately (PRPQ 5799-GAN). Also shipped with the SecureWay Directoryis Directory Sample 1, a client application that creates a directory fortesting LDAP.

Directory Sample 1 is provided without support.

Also shipped with the SecureWay Directory is Directory Sample 1, a clientapplication that creates a directory for testing LDAP. Directory Sample1 is provided without support.

Standards:

- RFC 1777 Lightweight Directory Access Protocol
- RFC 1778 String Representation of Standard Attribute Syntaxes
- RFC 1779 String Representation of Distinguished Names
- RFC 1823 LDAP Application Program Interface
- RFC 1960 A String Representation of LDAP Search Filters

Interoperability/Compatibility: The SecureWay Directory replicationinteroperates with the OS/390 LDAP Server.

## Software requirements

The product supports three operating systems:

- Microsoft Windows NT (3) Workstation/Server Version 4.0 withservice pack 3 or later (NTFS file system is required for security support)
- Sun Solaris Version 2.5.1 (SunOS 5.5.1) orVersion 2.6 (SunOS 5.6) (4)
- IBM AIX Version 4.2.1 with APAR IX72127; or Versions4.3.0 and 4.3.1 with APAR IX72439, IX74821,IX75022 and PTF U457544; or Version 4.3.2 (2)

The required IBM Universal Databases are:

- For NT, one of the following:
  - UDB V5.0 for Windows NT with fixpak US9044f
  - UDB V5.2 for Windows NT
- For Solaris, one of the following:
  - UDB V5.0 for Solaris with fixpak U457228f
  - UDB V5.2 for Solaris
- For AIX, one of the following:
  - UDB V5.0 for AIX with fixpak U457227f
  - UDB V5.2 for AIX

For Directory server, the requirements are:

- One of the following installed and configured Web servers:
  - Apache 1.2.5 or later
  - Lotus Domino Go Webserver 4.6.2 or later
  - Netscape FastTrack Server, Version 2.0.1 or later
  - Netscape Enterprise Server, Version 3.5.1 or later
  - Microsoft IIS 2.0
- Java Runtime 1.1.6
- A minimum of 64 MB RAM
- DB2 for AIX, Version 5.0.0.39, Workgroup Edition

**Note:** Lotus Domino Go Webserver 4.6.2.5 and Netscape FastTrackVersion 3.0.1 are available in the AIX Version 4.2 and4.3 Bonus Packs. DB2 for AIX, Version5.0.0.39 is included with the IBM SecureWayDirectory. DB2 fixes to upgrade from Version5.0.0.0 can be found on the Web at URL:

http://www.software.ibm.com/data/db2/library

The Directory client requires:

- AIX Version 4.2.1, Version 4.3, Version4.3.1, or Version 4.3.2
- A frame-enabled browser that supports HTML Version 3.0, or later,and a browser that supports Java Runtime 1.1.6.