IBM Systems - iSeries

e-business and Web serving WebSphere Application Server - Express Version 5 Security

Version 5 Release 4

IBM Systems - iSeries

e-business and Web serving WebSphere Application Server - Express Version 5 Security

Version 5 Release 4

Note

Before using this information and the product it supports, be sure to read the information in "Notices," on page 187.

Third Edition (February 2006)

This edition applies to version 5 of IBM WebSphere Application Server - Express for iSeries (product number 5722-IWE) and to all subsequent releases and modifications until otherwise indicated in new editions. This version does not run on all reduced instruction set computer (RISC) models nor does it run on CISC models.

Contents

Security

WebSphere$^{(R)}$ Application Server - Express is an Internet technology, and it is very crucial that you have a good Internet security policy in place before implementing WebSphere Application Server - Express. Even if your application runs only on your company's intranet, dangers still exist and your system needs to be protected.

While no system can ever be completely secured, you can implement certain security measures to discourage attacks. Before you deploy your WebSphere Application Server - Express solution, make sure you have studied and understand how your implementation affects your system security policy, and adjust your plan accordingly. See "iSeries security resources" on page 2 for links to information about creating a system-wide security plan.

This topic covers two areas of security concerns for WebSphere Application Server - Express: protecting your WebSphere resources (such as servlets, JSP files, and HTML files) and protecting the WebSphere product itself (its files, directories, and user profiles).

Protect your WebSphere resources

To secure WebSphere Application Server - Express and your WebSphere resources, you have these options:

"Securing Web resources with IBM HTTP Server for i5/OS" on page 4
You can use Web server directives to limit access to your servlets and JSP files. Web server directive-based security is typically easier to configure than WebSphere security and may provide better performance.

"Securing resources with WebSphere security" on page 5
WebSphere Application Server - Express provides a layered, role-based security architecture. WebSphere security supports Java$^{(TM)}$ 2 security, J2EE, and CORBA security. See this topic for information about developing, assembling, and deploying secured applications, as well as how to configure WebSphere security with the administrative console.

Protect the WebSphere product

See these topics for additional security information:

"Run application servers under specific user profiles" on page 179
By default, application servers run under the QEJBSVR user profile. If you want to use a different user profile, see this topic for instructions.

"Securing iSeries objects and files" on page 180
This topic describes iSeries objects and files that need to be secured with i5/OS$^{(R)}$ security.

"Password encoding" on page 181
This topic provides information about encoding passwords that are in configuration and properties files.

iSeries security resources

These resources will help you set up a strong security policy for your iSeries(TM) server.

Manuals

AS/400 Tips and Tools for Securing your AS/400, SC41-5300

This book provides a set of practical suggestions for using the security features of iSeries and for establishing operating procedures that are security-conscious.

i5/OS Security Reference, SC41-5302

This book provides information about planning, setting up, managing, and auditing security on your iSeries system. It describes all the features of security on the system and discusses how security features relate to other aspects of the system, such as work management, backup and recovery, and application design.

Redbooks and Redpapers

WebSphere Application Server - Express V5.0 for iSeries

This IBM Redpaper covers the basics of installing and configuring WebSphere Application Server - Express.

WebSphere Application Server Version 5.0 Security

This IBM Redbook provides an overview of WebSphere Application Server Version 5.0 Security, including J2EE security and programmatic security techniques. It also provides information about end-to-end security solutions that include WebSphere Application Server Version 5.0 as part of an e-business solution.

AS/400 Internet Security: Protecting Your AS/400 from HARM in the Internet, SG24-4929-00

This document describes all you need to know about iSeries security and how the different security elements fit together. This will help you understand the comprehensive iSeries security options available to secure your system and data.

HTTP Server (powered by Apache): An Integrated Solution for IBM iSeries Servers, SG24-6716-00

This IBM Redbook is designed to help you plan, install, configure, troubleshoot, and understand the HTTP Server (powered by Apache) running on the IBM iSeries server. It includes configuring the HTTP server for basic authentication, access control and SSL. It also walks you through the steps to implement Web application serving with Java featuring WebSphere Application Server.

Web sites

AS/400 Technical Studio: AS/400 Security Workshop

The AS/400 Security Workshop covers topics such as Internet Security. Use the iSeries server Security Advisor to help determine your optimal security settings.

Common Security Interoperability Version 2 (CSIv2) Specification

WebSphere security supports the use of CSIv2 as an authentication protocol. For more information about CSIv2, see the specification.

Java 2 Platform Security for Java 2 SDK, Standard Edition, 1.3

See this document for information about Java 2 Security architecture, policy permissions, and certificate support. WebSphere Application Server - Express supports the use of Java 2 security.

Programming model

WebSphere Application Server - Express ships some security documentation, which is installed as part of the product. By default, these documents are located in the /QIBM/ProdData/WebAS5/Base/web/docs directory.

-
- **iKeyman Documentation**
  See web/docs/ikeyman/ikmuserguide.pdf for the SSL Introduction and iKeyman.

Product documentation

iSeries Information Center (V5R4)

Digital certificate management
Digital Certificate Manager (DCM) is a free iSeries feature to centrally manage certificates for your applications

IBM HTTP Server for i5/OS
Information for this topic applies to HTTP Server (powered by Apache)

Security and Directory Services
Read this information to understand iSeries e-business security and Directory Services offerings

iSeries Information Center (V5R3)

Digital certificate management
Digital Certificate Manager (DCM) is a free iSeries feature to centrally manage certificates for your applications

IBM HTTP Server for i5/OS
Information for this topic applies to HTTP Server (powered by Apache)

Security and Directory Services
Read this information to understand iSeries e-business security and Directory Services offerings

iSeries Information Center (V5R2)

Digital certificate management
Digital Certificate Manager (DCM) is a free iSeries feature to centrally manage certificates for your applications

IBM HTTP Server for i5/OS
Information for this topic applies to HTTP Server (powered by Apache)

Security and Directory Services
Read this information to understand iSeries e-business security and Directory Services offerings

Securing Web resources with IBM HTTP Server for i5/OS

You can use the protection directives of IBM HTTP Server for i5/OS to secure Web resources. Performance may be better when using this mechanism, but you lose the ability to administer all of your security information in the WebSphere administrative application.

If there are static resources such as images that need no security check to be applied, they may be served up directly by the HTTP server without the performance impact of checking the WebSphere security.

For example, if WebSphere has resources within the URI `/webapp/SecureWebApplication/servlet/*`, a directive could be specified to allow the serving of images without a WebSphere security check. For example, with the IBM HTTP Server (powered by Apache), you can add this directive to your Web server instance configuration:

```
Alias /images/ /nonsecure/images/
```

Because WebSphere security does not apply to these resources, WebSphere Application Server - Express does not either authenticate or deny the request.

Note: The WebSphere administrative console can only be protected using WebSphere security, not Web server protection. However, when WebSphere security is enabled, Web server protection of servlets is not supported. If you are currently using Web server protection and want to enable WebSphere security, first remove the protection directives from the Web server configuration and then configure WebSphere security to protect your Web resources.

Additionally, servlets that are protected by a Web server obtain null when they call the getRemoteUser() or getAuthType() method of the request object if WebSphere security is enabled for the application server and WebSphere protection is not configured for the servlets. For more information, see "Using getRemoteUser() and getAuthType() methods" on page 5.

The WebSphere Application Server - Express product includes an internal HTTP server. The internal HTTP server cannot be configured to protect Web resources. In a production-level environment, you should ensure that the internal HTTP port number is **not** configured on the virtual host that is associated with the Web module.

To configure an IBM HTTP Server instance (powered by Apache), use the Location directive. The following example shows how to use the Location directive to protect the servlet /webapp/SecureServerWebApp/BasicServlet:

```
Location /webapp/SecureServerWebApp/BasicServlet
   AuthName happywas
   ProfileToken off
   AuthType Basic
   order deny,allow
   require valid-user
   allow from all
   deny from all
   PasswdFile %%SYSTEM%%
   UserID %%SERVER%%
/Location>
```

For more information about configuring an IBM HTTP Server instance, see the IBM HTTP Server for i5/OS documentation in the iSeries Information Center:

- For V5R4
- For V5R3
- For V5R2

Note: The WebSphere Application Server - Express product contains an internal HTTP server that is used for testing applications and to serve the administration application without the use of an external HTTP server. If you decide to protect your WebSphere resources with IBM HTTP Server (powered by Apache), which is the external HTTP server, you must disable access to the internal HTTP server.

To disable access to the internal HTTP server, perform these steps in the WebSphere administrative console:

1. In the navigation menu, click **Environment —> Virtual Hosts**.
2. In the Virtual Hosts page, click the name of the virtual host (for example, default_host).
3. Under **Additional Properties**, click **Host Aliases**.
4. Select the entry with a Port number that does not correspond to the external HTTP server port. (By default, the external port number is 80. In this case, select the host name with a port other than 80.)
5. Click **Delete**.
6. Click **Save** to save your configuration.

Using getRemoteUser() and getAuthType() methods

The getRemoteUser() and getAuthType() methods are methods of the HttpServletRequest interface (javax.servlet.http.HttpServletRequest). If the user has been authenticated, the getRemoteUser() method returns the login of the user making the request. If the user has not been authenticated, the getRemoteUser() method returns null. The getAuthType() method returns the name of the authentication scheme used to protect the servlet (for example, BASIC or SSL) or, if the servlet is not protected, the getAuthType() method returns null.

For both methods, the data returned depends on whether security is enabled in the application server where the servlet is deployed:

- If security is not enabled, a servlet is requested, and the servlet is configured with Web server protection, then the getRemoteUser() method returns the login, and getAuthType() returns the authentication scheme.
- If security is enabled and a servlet is requested, both methods return null when WebSphere protection is not configured for the servlet.
- If security is enabled, a servlet is requested, and the servlet is configured with WebSphere protection, then the getRemoteUser() method returns the login, and the getAuthType() method returns the configured authentication scheme.

Securing resources with WebSphere security

The WebSphere security system allows you to control access to WebSphere resources (such as servlets, JSP files, and HTML files). The security system also provides the infrastructure that ensures that the security choices you have made are enforced.

For more information about WebSphere security, see these topics:

"Overview of WebSphere security"
This topic provides an overview of WebSphere security architecture and its features.

"Developing secured applications" on page 11
See this topic for information about writing secured applications for WebSphere Application Server - Express.

"Assemble secured applications" on page 79
When you assemble your application for deployment, you can specify security settings. See this topic for more information.

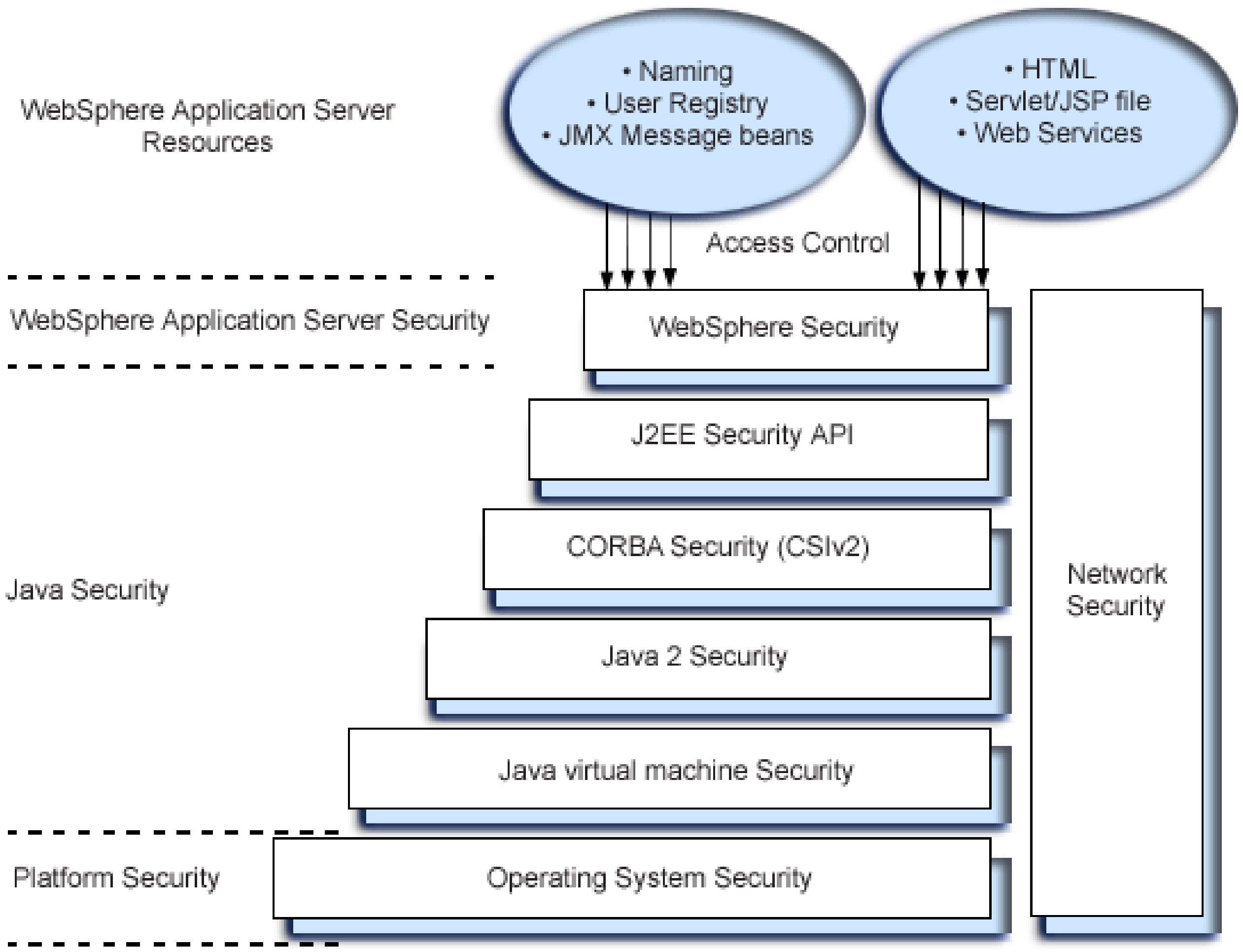
"Deploy secured applications" on page 82
This topic describes security considerations when you deploy or install your secured applications in the WebSphere Application Server - Express run time.

"Configure WebSphere security" on page 84
This topic describes how to enable and configure the various levels of WebSphere security.

"Tune your security configuration" on page 173
See this topic for information about improving the performance of WebSphere security.

Overview of WebSphere security

WebSphere Application Server -Express security is built on a layered security architecture, as shown in the figure below. This topic discusses the security protection offered by each security layer. The following figure illustrates the building blocks that comprise the operating environment of WebSphere Security:



The building blocks of WebSphere security are these:

- **Operating System Security**
  The security infrastructure of the underlying operating system provides certain security services to the WebSphere Security Application. This includes the file system security support to secure sensitive files in WebSphere product installation. You can configure the product to obtain authentication information directly from the operating system user registry.
- **Network Security**
  The Network Security layers provides transport level authentication as well as message integrity and encryption. WebSphere Application Server - Express inter-server communications can be configured to use Secure Socket Layer (SSL) and HTTPS. Additionally, IP Security and Virtual Private Network (VPN) may be used for added message protection.
- **JVM 1.3.1**
  The Java virtual machine security model provides a layer of security above the operating system layer.
- **Java 2 Security**
  The Java 2 Security model offers fine-grained access control to system resources including file system, system property, socket connection, threading, class loading, and so on. Application code must explicitly grant the required permission in order to access a protected resource.
- **CORBA Security**
  Any calls made among secure ORBs are invoked over the Common Security Interoperability Version 2 (CSIv2) security protocol, which sets up the security context and the necessary quality of protection. WebSphere Application Server - Express also supports the Secure Association Service (SAS) security protocol.

- **J2EE Security**
  The security collaborator enforces J2EE-based security policies and supports J2EE security APIs.
- **WebSphere Security**
  WebSphere security enforces security policies and services in a unified manner on access to Web resources and Java Management Extensions (JMX) administrative resources. It consists of WebSphere security technologies and features to support the needs of a secure environment.

WebSphere Application Server - Express configuration data is stored in XML descriptor files. Those XML configuration files should be protected by operating system security. Passwords and other sensitive configuration data can be modified through the WebSphere administrative console. Hence, the administrative console Web application is setup with data constraints that require the administrative console servlets and JSP files to be accessed only through SSL connections when global security is enabled.

After installation, the administrative console HTTPS port is configured to use the DummyServerKeyFile.jks and DummyServerTrustFile.jks with the default self-signed certificate. Using the Dummy key and trust file certificate is not safe. You should generate your own certificate immediately. To ensure maximum security, enable global security first and then complete other configuration tasks.

WebSphere Application Server - Express servers interact with each other through CSIv2 and SAS security protocols as well as HTTP and or HTTPS protocols. CSIv2 and SAS protocols can be configured to use SSL when WebSphere Application Server global security is enabled. The WebSphere Application Server - Express administrative subsystem uses SOAP JMX connectors or RMI/IIOP JMX connectors to pass administrative commands and configuration data.

When global security is disabled, the SOAP JMX connector uses HTTP protocol and the RMI/IIOP connector uses TCP/IP protocol. When global security is enabled, the SOAP JMX connector always uses HTTPS protocol. When global security is enabled, the RMI/IIOP JMX connector may be configured to either use SSL or to use TCP/IP. You should enable global security and enable SSL to protect the sensitive configuration data.

Disabling application server security does not affect the administrative subsystem in that application server. The administrative subsystem is controlled only by the global security configuration. Both administrative subsystem and application code in an application server share the optional per server security protocol configuration.

Security for J2EE resources is provided by the Web container. The container provides two kinds of security: **declarative security** and **programmatic security**.

In declarative security, an application security structure includes data integrity and confidentiality, authentication requirements, security roles, and access control in a form external to the application. In particular, the deployment descriptor is the primary vehicle for declarative security in the J2EE platform. WebSphere maintains J2EE security policy including information derived from the deployment descriptor and specified by deployers and administrators in a set of XML descriptor files. At runtime, the container uses the security policy defined in the XML descriptor files to enforce data constraints and access control.

When declarative security alone is not sufficient to the security model of an application, programmatic security may be used by application code to make access decisions.

When global security is enabled and application server security was not disabled at the server level, J2EE applications security will be enforced. If security policy is specified for a Web resource, access control is performed by the Web container when the resource is requested by a Web client. The Web container challenges the Web client for authentication data if there is none present according to the specified authentication method. The challenge ensures that the data constraints are met, and determines if the authenticated user has the required security role.

The Web security collaborator enforces role-based access control by using an access manager implementation. An access manager makes authorization decisions based on security policy derived from the deployment descriptor. An authenticated user principal is allowed to access the requested Servlet or JSP file if it has one of the required security roles.

Servlets and JSP pages may use the HttpServletRequest methods isUserInRole() and getUserPrincipal().

IBM has applied security role based access control concepts to the administrative resources of WebSphere Application Server - Express. These resources include the JMX system management subsystem, the user registry, and the JNDI name space. WebSphere administrative subsystem defines four administrative security roles:

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

You can perform most administrative work including installing new applications and application servers if you have a role as a configurator. There are certain configuration tasks a configurator does not have sufficient authority to do when global security is enabled. These include modifying WebSphere Application Server - Express server identity and password, LTPA password and keys, and assigning users to administrative security roles. Those sensitive configuration tasks require the administrative role.

WebSphere Application Server - Express administrative security is enforced when global security is enabled. It is recommended that WebSphere Application Server - Express global security be enabled to protect administrative subsystem integrity. Application server security can be selectively disabled if there is no sensitive information to protect.

WebSphere Application Server - Express uses the Java 2 Security Model to create a secure environment to run application code. Java 2 Security provides a fine-grained and policy based access control to protect system resources such as files, system properties, opening socket connections, loading libraries, and so on. J2EE 1.3 Specification defines the typical set of Java 2 security permissions that Web components should have. These security permissions are shown in the tables below.

J2EE Security Permissions set for Web components

Security Permission	Target	Action
java.lang.RuntimePermission	loadLibrary	
java.lang.RuntimePermission	queuePrintJob	
java.net.SocketPermission	*	connect
java.io.FilePermission	*	read, write
java.util.PropertyPermission	*	read

WebSphere Application Server - Express Java 2 Security implementation was based on the J2EE 1.3 Specification. The specification granted Web components read and write file access permission to any file in the file system, which may be too broad. WebSphere Application Server - Express default policy gives Web components read and write permission to the subdirectory and the subtree where the Web module was installed. The default Java 2 security policy for all Java virtual machines and WebSphere Application Server - Express server processes are contained in the java.policy file (the default policy for the Java virtual machine) and the server.policy file (the default policy for all product server processes).

To simplify policy management, WebSphere Application Server - Express policy is based on resource type rather than code base (location). There are default policy files for embedded resources, for libraries shared by multiple applications, and for J2EE applications:

- spi.policy, which is for embedded resources defined in resources.xml, such as JavaMail and JDBC drivers.
- library.policy, which is for shared libraries that are defined by the administrative console.
- app.policy, which is the default policy for J2EE applications

In general, applications should not require more permissions to run than those recommended by the J2EE Specification, so they can be portable among application servers. Some applications, however, may require more permissions. WebSphere Application Server - Express allows a policy file (was.policy) for each application, which can be packaged with the application and grants extra permissions to that application. Note that granting extra permissions to an application should be handled with great care because of the potential of compromising system integrity.

WebSphere Application Server - Express uses a permission filtering policy file to alert users when an application requires permissions that are on the filter list during application installation, and could cause the offending application installation to fail.

For example, the java.lang.RuntimePermission exitVM permission should not be given to an application so that no application code is allowed to terminate the WebSphere Application Server - Express application server. The filtering policy is defined by the filterMask element in the filter.policy file.

WebSphere Application Server - Express also performs runtime permission filtering based on the runtime filtering policy to ensure no application code has been granted any permission that is considered harmful to system integrity.

WebSphere Application Server - Express supports Java 2 Security as part of its security implementation. However, when trusted applications are being deployed, Java 2 Security can be disabled for the application server.

The global security configuration and Java 2 Security configuration are stored in a set of XML configuration files. Both role-based access control and Java 2 Security permission-based access control are employed to protect the integrity of the configuration data. System integrity is maintained in these ways:

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

Other WebSphere Application Server - Express runtime resources are protected by similar mechanisms. It is very important to enable WebSphere Application Server - Express global security and to enforce Java 2 Security. The J2EE specification defines several authentication methods for Web components. WebSphere Application Server - Express supports HTTP basic authentication, form-based authentication, and HTTPS client certificate authentication. **HTTPS** means that the HTTP protocol is performed using an SSL connection. In an SSL connection, the server always presents a certificate to the client during an authentication handshake. With HTTPS client certificate authentication, the authentication handshake requires an additional step where the client also presents a certificate to authenticate itself to the server. This is also known as mutual authentication.

When using client certificate login, it is more convenient for the browser client if the Web resources are configured with an integral or confidential data constraint. If a browser uses HTTP to access the Web resource, the Web container automatically redirects it to the HTTPS port. The CSIv2 security protocol also supports client certificate authentication. You can establish a trust relationship between an HTTP server and a WebSphere Application Server - Express server by configuring an SSL connection with mutual authentication.

WebSphere Application Server - Express supplies a plug-in module that can be used by the IBM HTTP Server (powered by Apache), or by the Domino HTTP Server. The plug-in forwards client HTTP and HTTPS requests on to the WebSphere Application Server - Express server. The transport on which these requests and replies are exchanged can be configured to use SSL for additional protection. The configuration can specify that the plug-in also needs to provide a certificate to authenticate itself to the WebSpehere Application Server - Express server.

Developing secured applications

IBM WebSphere Application Server - Express provides security components that provide or collaborate with other services to provide authentication, authorization, delegation, and data protection. WebSphere Application Server - Express also supports the security features described in the Java 2 Enterprise Edition (J2EE) specification.

An application goes through three stages before it is ready to run:

1. Development
2. Assembly
3. Deployment

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

See these topics for more information about programmatic security:

“Develop secure Web applications” on page 12

“Develop servlet filters for form login processing” on page 14

“Develop form login pages” on page 18

“Develop with JAAS to log in programmatically” on page 21

“Develop your own J2C principal mapping module” on page 26

“Develop custom user registries” on page 28

"Develop a custom interceptor for trust associations" on page 75

Develop secure Web applications

Programmatic security is used by security aware applications when declarative security alone is not sufficient to express the security model of the application. Programmatic security consists of these steps:

1. Add the required security methods in the servlet or JSP code.
2. Create security-role-ref element with role-name field. This element is not strictly required because the security API isUserInRole() method can use the actual role name as a parameter. However, it is good practice to use role references so the software component is reusable.

For a full code example, see "Example: Secure Web applications code" on page 13.

Add required security methods in the servlet code

Programmatic security consists of these methods of the HttpServletRequest interface:

- **getRemoteUser()**
  This method returns user name the client used for authentication. returns null if no user has been authenticated.
- **isUserInRole (String rolename)**
  This method returns true if the remote user is granted the specified security role. If remote user is not granted the specified role or if no user is authenticated, it returns false.
- **getUserPrincipal()**
  This method returns java.security.Principal object containing the remote user name. If no user is authenticated, it returns null.

Programmatic servlet security methods can be added inside any of the servlet's doGet(), doPost(), doPut(), doDelete() service methods. Here is an example of usage of programmatic security APIs:

```
public void doGet(HttpServletRequest request, HttpServletResponse response) {
  ...
  // to get remote user using getUserPrincipal()
  java.security.Principal principal = request.getUserPrincipal();
  String remoteUser = principal.getName();

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

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

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

Create security-role-ref element with role-name field

This step is required to programmatically secure an application. If security-role-ref is not created during development, make sure it is created during assembly stage.

When the isUserInRole() method is used, a security-role-ref element should be declared in the deployment descriptor with a role-name element that contains the role name that is passed to this method. Because actual roles are created during the assembly stage of the application, a developer can use the logical role as role name and provide enough hints to the assembler in the description of the security-role-ref element to link that role to the actual role. During assembly, the assembler creates a role-link sub element to link the role-name to the actual role.

You can create the security-role-ref element during development with the WebSphere Development Studio Client:

1. Open the web.xml file for your application. It is located in the WEB-INF directory.
2. Click the **Security** tab.
3. Next to the Security roles window, click **Add**. Type a name and a decription for the security role. Repeat this step until you have added all the necessary roles.
4. Click the **Servlets** tab.
5. In the Servlets window, select the servlet for which you want to define the security-role-ref element.
6. Next to the Authorized roles window, click **Edit**.
7. Select the appropriate roles. Click **OK**.
8. Save the web.xml file.

An example where it is useful to define logical roles is when you want a Web application to access external resources and to control the access to external resources by using its own authorization table (external-resource to remote-user mapping). In this case, the getUserPrincipal() or getRemoteUser() method can be used to get the remote user, and then the application can consult its own Authorization Table to check authorization. The remote user information can also be used to retrieve the corresponding users information from an external source such as database. isUserInRole() can also be used similarly.

Here is an example:

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

During assembly, the assembler creates role-link as shown below:

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

Example: Secure Web applications code: This example illustrates a Web application (a servlet) that uses the programmatic security model. The example is one way to use the programmatic security model but not necessarily the only way. The application can use the information that is returned by the getUserPrincipal(), isUserInRole() and getRemoteUser() methods in any way that is meaningful to that application. However, it is strongly recommended that you use the declarative security model whenever possible.

```
import javax.servlet.*;

public class HelloServlet extends HttpServlet {
  public void doPost(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, java.io.IOException {
  }

  public void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, java.io.IOException {
    String s = "Hello";

    // get remote user using getUserPrincipal()
    java.security.Principal principal = request.getUserPrincipal();
    String remoteUserName = "";

    if (principal != null) {
      remoteUserName = principal.getName();
    }

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

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

      // display Hello username for managers and bob.
      if (isMgr || remoteUserName.equals("bob")) {
        s = "Hello " + remoteUserName;
      }
      String message = "<html> \n" +
        "<head><title>Hello Servlet</title></head>\n" +
        "<body> \n" +
        "<h1> " + s + "</h1>\n ";
      byte[] bytes = message.getBytes();

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

After you develop the servlet, you can create a security role reference for the HelloServlet as show below:

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

Develop servlet filters for form login processing

You can control the look and feel of a login screen with the form-based login mechanism. This mechanism allows you to define a form-based login page to retrieve a user ID and password. You can also specify an error page to display when authentication fails.

Servlet filters can be used to dynamically intercept requests and responses to transform or use the information contained in the requests or responses. One or more servlet filters can be attached to a servlet or a group of servlets. Servlet filters can also be attached to JSP and HTML pages. All the attached servlet filters are called before invoking the servlet.

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

To perform pre-login and post-login actions with servlet filters, you can configure servlet filters either for a form-login page or for the `/j_security_check` URL. j_security_check is posted by the form login page with the `j_username` parameter (which contains the username) and the `j_password` parameter (which contains the password). A servlet filter can then use username and password information to perform more authentication or other processing. For an example, see “Example: Servlet filters” on page 16.

A servlet filter should implement the javax.servlet.Filter class. Here are the methods in the Filter class that need to be implemented:

- **init(javax.servlet.FilterConfig cfg)**
  This method is called by the container exactly once when the servlet filter is placed into service. The FilterConfig parameter that is passed to this method contains the initialization parameters of the servlet filter.
- **destroy()**
  This method is called by the container when the servlet filter is taken out of service.
- **doFilter(ServletRequest req, ServletResponse res, FilterChain chain)**
  This method is called by the container for every servlet request that is mapped to this filter before the container invokes the servlet itself. The FilterChain parameter that is passed to this method can be used to invoke the next filter in the chain of filters. The original requested servlet runs when the last filter in the chain calls the chain.doFilter() method. Therefore, all filters should call the chain.doFilter()

method. If additional authentication checking in the filter code results in failure, the original servlet need not be instantiated. In this case, the chain.doFilter() method does not need to be called, instead it can be redirected to some other error page.

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

An example of Servlet Filter is shown below: This Login Filter can be mapped to `/j_security_check` to perform pre-login and post-login actions.

```
import javax.servlet.*;

public class LoginFilter implements Filter {

  protected FilterConfig filterConfig;

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

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

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

    // perform pre-login action here

    // calls the next filter in chain.  j_security_check if
    // this filter is mapped to j_security_check.
    chain.doFilter(request, response);

    // perform post-login action here.
    }
  }
```

The servlet filter class file should be placed in the WEB-INF/classes directory of the application.

Configuring servlet filters

WebSphere Development Studio Client can be used to configure the servlet filters. Follow these steps:

1. Open the web.xml file (which is located in the WEB-INF directory) for your Web project.
2. A name can be assigned to servlet filter, and the corresponding implementation class can be assigned to the servlet filter. Optionally, initialization parameters that get passed to the init() method of the servlet filter can also be assigned.

   To configure the servlet filter, click the **Servlets** tab in the web.xml graphical interface. Click **Initialization** to specify parameters.

   After you configure the servlet filter, the application deployment descriptor (web.xml) contains a servlet filter configuration that is similar to this one:

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

```
        <param-name>ParamName</param-name>
        <param-value>ParamValue</param-value>
      </init-param>
    </filter>
```

Note: You can display the source of the web.xml file by clicking the **Source** tab.

3. A servlet or a URL pattern can be mapped to the servlet filter. Under **URL mappings**, click **Add** and specify the servlet or URL pattern.

   After you map the servlet filter to a servlet or a URL, the application deployment descriptor(web.xml) contains a servlet mapping that is similar to this one:

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

Servlet filters can be used to replace CustomLoginServlet. Servlet filters can also be used to perform additional authentication, auditing and logging.

Example: Servlet filters: This example illustrates one usage of the servlet filters to perform pre-login and post-login processing during form login. See the “Code license and disclaimer information” on page 190 for legal information about this code example.

```
// Servlet Filter source code: LoginFilter.java
/**
 * A Servlet filter example: This example filters j_security_check and
 * performs pre-login action to determine if the user trying to log in
 * is in the revoked list. If the user is in revoked list, an error is
 * sent back to the browser.
 *
 * This filter reads the revoked list file name from the FilterConfig
 * passed in the init() method. Reads the revoked user list file and
 * creates a revokedUsers list.
 *
 * When doFilter method is called, the user being logged in is checked
 * to make sure that the user is not in the revoked User list.
 *
 */

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

public class LoginFilter implements Filter {

  protected FilterConfig filterConfig;
  java.util.List revokeList;


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

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


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

 /**
  * doFilter() : doFilter() method called before the servlet that this filter
  * is mapped is invoked. Since this filter is mapped to j_security_check,
  * this method is called before j_security_check action is posted.
  */
  public void doFilter(ServletRequest request,
                       ServletResponse response,
                       FilterChain chain)
   throws java.io.IOException, ServletException {

     HttpServletRequest req = (HttpServletRequest)request;
     HttpServletResponse res = (HttpServletResponse)response;

     // pre login action

     // get username
     String username = req.getParameter("j_username");

     // if user is in revoked list send error
     if ( revokeList.contains(username) ) {
       res.sendError(javax.servlet.http.HttpServletResponse.SC_UNAUTHORIZED);
       return;
     }

     // call next filter in the chain : let j_security_check authenticate user
     chain.doFilter(request, response);

     // post login action
   }

 /**
  * readConfig() : Reads revoked user list file and creates a revoked user list.
  */
  private void readConfig() {
    if ( filterConfig != null ) {

      // get the revoked user list file and open it.
      BufferedReader in;

      try {
        String filename = filterConfig.getInitParameter("RevokedUsers");
        in = new BufferedReader( new FileReader(filename));
      }
      catch (FileNotFoundException fnfe) {
        return;
      }

      // read all the revoked users and add to revokeList.
      String userName;
      try {
        while ( (userName = in.readLine()) != null ) {
          revokeList.add(userName);
        }
      }
      catch (IOException ioe) {
      }
    }
  }
}
```

This example shows a portion of the application deployment descriptor (web.xml) that lists the LoginFilter configuration and mapping to j_security_check:

```
<filter id="Filter_1">
  <filter-name>LoginFilter</filter-name>
  <filter-class>LoginFilter</filter-class>
  <description>Performs pre-login and post-login operation</description>
  <init-param>
    <param-name>RevokedUsers</param-name>
    <param-value>
     /QIBM/UserData/WebASE/ASE5/instance/installedApps/application/revokedUsers.lst
    </param-value>
  </init-param>
  </filter-id>
<filter-mapping>
  <filter-name>LoginFilter</filter-name>
  <url-pattern>/j_security_check</url-pattern>
</filter-mapping>
```

Here is an example of the revoked user list file:

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

Develop form login pages

A Web client (browser) can authenticate a user to a Web server by using one of these mechanisms:

- **HTTP basic authentication**
  A Web server requests the Web client to authenticate, and the Web client passes the user and password information in the HTTP header.
- **HTTPS client authentication**
  This mechanism requires a user (Web client) to possess a public key certificate. The Web client sends this certificate to a Web server that requests a client certificate. This is a strong authentication mechanism that uses HTTPS protocol.
- **Form-based authentication**
  With this authentication mechanism, you can control the look and feel of the login screens.

The HTTP Basic authentication transmits user password from the Web client to the Web server in simple Base64 encoding. Form based authentication transmits the user password from browser to Web server in plain text. Therefore, both HTTP Basic authentication and Form Based authentication are not very secure unless HTTPS is used.

The Web application deployment descriptor contains information about what authentication mechanism to use. When form-based authentication is used, the deployment descriptor also contains entries for login and error pages. A login page can be either a HTML page or a JSP page. This login page is displayed on the Web client when a secured resource (such as a servlet, JSP, or HTML page) is accessed from the application. If the authentication fails, the error page is displayed. You can write your own login and error pages to fit the needs of your application. During assembly of the application, an assembler can set the authentication mechanism for the application and set the login and error pages in the deployment descriptor.

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

- The sendRedirect() method initially displays the form login page in the Web browser. It later redirects the Web browser back to the originally requested protected page. The sendRedirect(String URL) method tells the Web browser to use the HTTP GET (not the HTTP POST) request to get the page specified in the URL. If HTTP POST is the first request to a protected servlet or JSP file, and no previous authentication or login occurred, then HTTP POST is not delivered to the requested page.

However, HTTP GET is delivered because form login uses the sendRedirect() method, which behaves as a HTTP GET request that tries to display a requested page after a login occurs.

- Using HTTP POST, you may experience a scenario where an unprotected HTML form collects data from users and then posts this data to protected servlets or JSP files for processing, but the users are not logged in for the resource. To avoid this scenario, structure your Web application or permissions so that users are forced to use a form login page before the application performs any HTTP POST actions to protected servlets or JSP files.

For more information and code examples, see "Example: Form login."

1. Create a form login page and the components to perform the form-based authentication.
2. Create an error page. The error page can be programmed to retry authentication or display an error message that is appropriate.
3. (Optional) Create a form logout page.
4. Assemble the login, error, and logout pages in a WAR file. The pages should be placed relative to the root directory of the WAR file. For example, if the login page is configured as `/login.html` in the deployment descriptor, it is placed in the root directory of the WAR file.

Example: Form login: The following is an example of how the form should be coded into the HTML page:

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

The action of the login form must always be `j_security_check`. The j_username input field should be used to get the user name, and the j_password input field should be used to get the user's password.

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

Here is an example of an HTML login page:

```
<!-- login.html -->
<!DOCTYPE HTML PUBLIC "-//W3C/DTD HTML 4.0 Transitional//EN">
<html>
<head>
  <meta http-equiv="Pragma" content="no-cache">
  <title>Security FVT Login Page </title>
</head>

<body>

  <h2>Form Login</h2>

  <form method="post" action="j_security_check">
    <p>
      <strong>You may have entered an invalid user ID
        or password. To correct the problem, please enter
        your correct user ID and password. If you have
        forgotten your user ID or password, please contact
        the server administrator.</strong>
    </p>
    <p>
      <strong>Please enter user ID and password:</strong>
      <br>
      <strong>User ID</strong>
      <input type="text" size="20" name="j_username">
      <strong>Password</strong>
      <input type="password" size="20" name="j_password">
    </p>

    <p>
      <strong>And then click this button:</strong>
      <input type="submit" name="login" value="Login">
    </p>
  </form>

</body>
</html>
```

Here is an example error page in a JSP file:

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

</body>
</html>
```

After you configure the Web application to use form-based authentication, the deployment descriptor contain the login configuration as shown below:

```
<login-config id="LoginConfig_1">
  <auth-method>FORM</auth-method>
  <realm-name>Example Form-Based Authentication Area</realm-name>
  <form-login-config id="FormLoginConfig_1">
    <form-login-page>/login.html</form-login-page>
    <form-error-page>/error.jsp</form-error-page>
  </form-login-config>
</login-config>
```

Here is the directory structure for a sample Web application archive (WAR) file that shows the login and error pages for these examples:

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

Form logout

Form logout is a mechanism that allows uses to logout from an application without having to close all Web browser sessions. After the user logs out, access to a protected Web resource requires reauthentication. This feature is not required by J2EE specifications but is provided as an additional feature in WebSphere security.

A form logout works in the following manner:

1. The URI for the logout-form URI is specified in the Web browser.
2. The browser loads the form.

3. The user clicks the submit button of the form to logout.
4. The WebSphere security code logs out the user.
5. Upon logout, the user is redirected to a logout exit page.

Form logout does not require any attributes in any deployment descriptor. It is simply an HTML or JSP file that is included with the Web application. The form logout page is like most HTML forms. However, the form logout page, like the form login page, it has a special POST action that is recognized by the Web container. The Web container then dispatches it to a special internal WebSphere form logout servlet. The POST action in the form logout page must have a value of `ibm_security_logout`.

A logout exit page can be specified in the logout form. The exit page can be a HTML or JSP file to which users a redirected after they log out. The logout exit page must reside within the same Web application as the logout form. The logout exit page is simply specified as a parameter in the form logout page. If no logout exit page is specified, a default logout HTML message is returned to the user.

Here is a sample HTML logout form. This form configures the logout exit page to redirect the user back to the login page after logout.

```
<!DOCTYPE HTML PUBLIC "-//W3C/DTD HTML 4.0 Transitional//EN">
<html>
<head>
  <meta http-equiv="Pragma" content="no-cache">
  <title>Logout Page </title>
<head>

<body>
  <h2>Sample Form Logout</h2>
  <form method="post" action="ibm_security_logout" name="logout">
    <p>
      <strong> Click this button to logout:</strong>
      <input type="submit" name="logout" value="Logout">
      <input type="hidden" name="logoutExitPage" value="/login.html">
    </p>
  </form>

</body>
</html>
```

Develop with JAAS to log in programmatically

Java Authentication and Authorization Service (JAAS) is a new feature in WebSphere Application Server - Express Version 5. JAAS represents the strategic APIs for authentication and replaces the CORBA programmatic login APIs of previous releases. Additionally, WebSphere Application Server - Express provides some extensions to JAAS.

If the application is using custom JAAS login configuration, please make sure that the custom JAAS login configuration is properly defined. See “Configure Java Authentication and Authorization Service login” on page 170 for details.

Some of the JAAS APIs are protected by Java 2 Security permissions, if these APIs are used by application code, please make sure that these permissions are added to the application was.policy file. See “Configure the was.policy file” on page 164 for more information. For more details of which APIs are protected by Java 2 Security permissions, please check the J2SDK, JAAS, and WebSphere Application Server - Express APIs javadoc. The following lists some of the APIs used in the sample code in this documentation and the Java 2 Security permissions required by these APIs:

- javax.security.auth.login.LoginContext constructors are protected by javax.security.auth.AuthPermission createLoginContext.
- javax.security.auth.Subject.doAs() and com.ibm.websphere.security.auth.WSSubject.doAs() are protected by javax.security.auth.AuthPermission doAs.

- javax.security.auth.Subject.doAsPrivileged() and com.ibm.websphere.security.auth.WSSubject.doAsPrivileged() are protected by javax.security.auth.AuthPermission doAsPrivileged.

WebSphere Application Server - Express provides these extensions to JAAS:

- **com.ibm.websphere.security.auth.WSSubject**
  Due to a design oversight in the JAAS 1.0 specification, javax.security.auth.Subject.getSubject() does not return the Subject associated with the thread of execution inside a java.security.AccessController.doPrivileged() code block. This can present a inconsistent behavior that is problematic and causes undesireable effort. com.ibm.websphere.security.auth.WSSubject provides a workaround to associate Subject to thread of execution.

  com.ibm.websphere.security.auth.WSSubject extends the JAAS model to J2EE resources for authorization checks. If the Subject associates with the thread of execution within com.ibm.websphere.security.auth.WSSubject.doAs() or if the com.ibm.websphere.security.auth.WSSubject.doAsPrivileged() code block contains product credentials, the Subject will be used for J2EE resources authorization checks. For more information, see com.ibm.websphere.security.auth.WSSubject.

- **WebSphere JAAS login configurations**
  WebSphere provides a JAAS login configuration for applications to perform programmatic authentication to the WebSphere security runtime. This WebSphere JAAS login configuration performs authentication to the WebSphere configured authentication mechanism (SWAM or LTPA) and user registry (LocalOS, LDAP or Custom) based on the authentication data supplied. The authenticated Subject from this JAAS login configuration contains the required Principal and Credentials that can be used by WebSphere security runtime to perform authorization checks on J2EE role-based protected resources.

  Here is the JAAS login configuration provided by WebSphere:

  - **WSLogin JAAS login configuration**
    This is a generic JAAS login configuration that can be used by servlets and JSP components to perform authentication based on a user ID and password or on a token that is passed to the WebSphere security runtime.

    Note: A subject that is authenticated with the above JAAS login configuration contains a com.ibm.websphere.security.auth.WSPrincipal and a com.ibm.websphere.security.auth.WSCredential. If the authenticated Subject is passed in com.ibm.websphere.security.auth.WSSubject.doAs() (or the other doAs() methods), the WebSphere security runtime can perform authorization checks on J2EE resources based on the Subject com.ibm.websphere.security.auth.WSCredential.

- **User defined JAAS login configurations**
  You can can define other JAAS login configurations. See "Configure Java Authentication and Authorization Service login" on page 170 for details. Use these login configurations to perform programmatic authentication to your user defined authentication mechanism. However, the Subjects from these user defined JAAS login configurations might not be usable by WebSphere security runtime to perform authorization checks if they do not contain the required principal and credentials.

Version 5.0.1 and later:

- The subject object generated by the WSLoginModuleImpl instance contains a principal that implements the WSPrincipal interface. Using the getCredential() method for a WSPrincipal object returns an object that implements the WSCredential interface. You can also find the WSCredential object instance in the PublicCredentials list of the subject instance. You should retrieve the WSCredential object from the PublicCredentials list instead of using the getCredential() method.
- The getCallerPrincipal() method for the WSSubject class returns a string that represents the caller security identity.

- The Subject object generated by the J2C DefaultPrincipalMapping module contains a resource principal and a PasswordCredentials list. The resource principal represents the caller.

For programmatic login with JAAS, the product provides an implementation of the javax.security.auth.callback.CallbackHandler interface, which is called com.ibm.websphere.security.auth.callback.WSCallbackHandlerImpl.

This com.ibm.websphere.security.auth.callback.WSCallbackHandlerImpl allows application to "push" authentication data to the WebSphere LoginModule to perform authentication. This can be useful for server side application code to authenticate an identity and use the identity to invoke downstream J2EE resources. Here is an example:

```
javax.security.auth.login.LoginContext lc = null;

try {
  lc = new javax.security.auth.login.LoginContext("WSLogin",
        new com.ibm.websphere.security.auth.callback.WSCallbackHandlerImpl
        ("user", "securedpassword"));

  // create a LoginContext and specify a CallbackHandler implementation
  // CallbackHandler implementation determine how authentication data is collected
  // in this case, the authentication data is "pushed" to the authentication
  // mechanism implemented by the LoginModule.
} catch(javax.security.auth.login.LoginException e) {
  System.err.println("ERROR: failed to instantiate a LoginContext "
                     + "and the exception: " + e.getMessage());
  e.printStackTrace();

  // may be javax.security.auth.AuthPermission "createLoginContext" is not granted
  // to the application, or the JAAS login configuration is not defined.
}

if (lc != null) {
  try {
    lc.login();  // perform login

    // get the authenticated subject
    javax.security.auth.Subject s = lc.getSubject();

    // Invoke a J2EE resources using the authenticated subject
    com.ibm.websphere.security.auth.WSSubject.doAs(s,
                 new java.security.PrivilegedAction() {
      public Object run() {
        try {
          bankAccount.deposit(100.00);
          // where bankAccount is an protected resource
        } catch (Exception e) {
          System.out.println("ERROR: error while accessing resource, exception: " +
                             e.getMessage());
          e.printStackTrace();
        }
        return null;
      }
    }
  ) catch (javax.security.auth.login.LoginException e) {
    System.err.println("ERROR: login failed with exception: " + e.getMessage());
    e.printStackTrace();

    // login failed, might want to provide relogin logic
  }
}
```

See "Example: JAAS programmatic login" on page 25 for more information.

Find the root cause login exception from a JAAS login (Version 5.0.2 and later)

If you get a LoginException after issuing the LoginContext.login() API, you can find the root cause exception from the configured user registry. In the login modules, the registry exceptions are wrapped by a com.ibm.websphere.security.auth.WSLoginFailedException. This exception has a getCause() method that allows you to pull out the exception that was wrapped.

Note: You are not always guaranteed to get an exception of type WSLoginFailedException, but you should note that most of the exceptions generated from the user registry show up here.

The following is a LoginContext.login() API example with associated catch block. WSLoginFailedException has to be casted to com.ibm.websphere.security.auth.WSLoginFailedException if you want to issue the getCause() API.

Note: The determineCause() example below can be used for processing CustomUserRegistry exception types.

```
try {
  lc.login();
} catch (LoginException le) {
  // Drill down through the exceptions as they might cascade through the runtime.
  Throwable root_exception = determineCause(le);

  // Now you can use "root_exception" to compare to a particular exception type.
  // For example, if you have implemented a CustomUserRegistry type, you would know
  // what to look for here.
}

/* Method used to drill down into the WSLoginFailedException to find the
 * "root cause" exception */
public Throwable determineCause(Throwable e) {
  Throwable root_exception = e, temp_exception = null;

  // Keep looping until there are no more embedded WSLoginFailedException
  // or WSSecurityException exceptions.
  while (true) {
    if (e instanceof com.ibm.websphere.security.auth.WSLoginFailedException) {
      temp_exception =
         ((com.ibm.websphere.security.auth.WSLoginFailedException) e).getCause();
    }
    else if (e instanceof com.ibm.websphere.security.WSSecurityException) {
      temp_exception =
         ((com.ibm.websphere.security.WSSecurityException) e).getCause();
    }
    else if (e instanceof com.ibm.ws.security.registry.os400.OS400RegistryException) {
      // get OS400 specific error code, if configured
      System.out.println ("Error code from OS400 exception: "
         + ((com.ibm.ws.security.registry.os400.OS400RegistryException)e).getErrorCode());
      return e;
    }
    else if (e instanceof javax.naming.NamingException) {
      // check for Ldap embedded exception
      temp_exception = ((javax.naming.NamingException)e).getRootCause();
    }
    else if (e instanceof your_custom_exception_here) {
      // your custom processing here, if necessary
    }
    else {
      // This exception is not one of the types we are looking for, return now.
      // This is the root from the WebSphere perspective
      return root_exception;
    }

    if (temp_exception != null) {
      // We have an exception, let's go back and see if this has another
      // one embedded within it.
      root_exception = temp_exception;
      e = temp_exception;
      continue;
    }
    else  {
      // We finally have the root exception from this call path,
      // this has to occur at some point.
      return root_exception;
    }
  }
}
```

Example: JAAS programmatic login: The following example illustrates how application programs may perform a programmatic login using JAAS authentication:

```
LoginContext lc = null;

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

try {
  lc.login();
}
catch(LoginExcpetion le) {
  System.out.printlin("Fails to create Subject. " + le.getMessage());
  // Insert error processing code
}
```

Shown in the example, the new LoginContext is initialized with the WSLogin login configuration and the WSCallbackHandlerImpl CallbackHandler. The WSCallbackHandlerImpl is suitable for use on server side application where prompting is not desirable. A WSCallbackHandlerImpl instance is initialized by the specified user id, password, and the realm information. The present WSLoginModuleImpl class implementation that is specified by WSLogin can only retrieve authentication information from the specified CallbackHandler. A LoginContext may be constructed with a Subject object but the Subject will be disregarded by the present WSLoginModuleImpl implementation.

In addition, you can also develop your own LoginModule if the default WSLoginModuleImpl implementation cannot address all your requirements. WebSphere provides utility functions that can be used by custom LoginModule which are described in the next section.

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

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

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

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

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

    try {
      lc.login();
    } catch(LoginException le) {
      System.out.printlin("Fails to create Subject." + le.getMessage());
      // Insert error processing code
    }
```

Develop your own J2C principal mapping module

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

The J2C Connection Factory configuration contains a mapping module which defines a principal mapping module alias (mappingConfigAlias attribute) and an authentication data alias (authDataAlias attribute). At runtime the J2C managed connection factory code passes a reference of the ManagedConnectionFactory and an authDataAlias object to the configured principal mapping module via the WSPrincipalMappingCallbackHandler object. WebSphere Application Server - Express allows users to plug-in a custom principal mapping module for a connection factory if the any-authenticated-to-one mapping provided by the default principal mapping module is insufficient. A custom mapping module is a special purpose JAAS LoginModule that performs principal or credential mapping in the login method. The WSSubject.getCallerPrincipal() method can be used to retrieve the application client identity. To plug in a custom mapping module change the value of the mappingConfigAlias to the custom mapping module. However, the configuration cannot be done via the administrative console and must be done through the wsadmin scripting tool.

Follow these steps to configure a custom mapping module. Use the WebSphere administrative console to perform the first several steps. For more information, see The WebSphere administrative console in the *Administration* topic. Use the wsadmin administrative tool to perform the remaining configuration. For more information about wsadmin, see The wsadmin administrative tool in the *Administration* topic.

1. Start the WebSphere administrative console.
2. Click **Security —> JAAS Configuration**.
3. Select **Application Logins**. Click **New**.
4. Enter a unique alias for the new mapping module, and click **Apply**.
5. Click **JAAS Login Modules** to define the custom mapping module class.
6. Click **New**, and complete mapping LoginModule class name.
7. Click **Apply**. Click **Save** to save the new configuration.
8. Use wsadmin to configure a J2C Connection Factory to use the new mapping module:
   a. Start wsadmin.
   b. At the wsadmin prompt, run the `list` command to show a list of J2CConnectionFactory objects:
      ```
      wsadmin>$AdminConfig list J2CConnectionFactory
      ```
   c. To select the J2C Connection Factory, run the `show` command to show all the attributes. For example:
      ```
      wsadmin>$AdminConfig show PetStore_CF
      (cells/hillsideNetwork/nodes/hillside/servers/server1:
      resources.xml#CMPConnectorFactory_4)
      ```
   d. Examine the current mapping module configuration. Run the `show` command:
      ```
      wsadmin>$AdminConfig show {mapping
       (cells/hillsideNetwork/nodes/hillside/servers/server1:
       resources.xml#MappingModule_7)}
      ```
      The following shows sample results of the command:
      ```
      {authDataAlias {}} {mappingConfigAlias DefaultPrincipalMapping}
      ```
      As shown in the previous example, the J2C Connection factory is configured to use the DefaultPrincipalMapping login configuration.
   e. Modify the mapping module configuration to use the new mapping module. Run the `modify` command:
      ```
      wsadmin>$AdminConfig modify {mapping
       (cells/hillsideNetwork/nodes/hillside/servers/server1:
       resources.xml#MappingModule_7)} {{mappingConfigAlias myMappingModule}}
      ```

   You may check the result with the `show` command:
   ```
   wsadmin>$AdminConfig show {mapping
    (cells/hillsideNetwork/nodes/hillside/servers/server1:
    resources.xml#MappingModule_7)} {authDataAlias {}}
    {mappingConfigAlias myMappingModule}
   ```
   Note: The authDataAlias property is left undefined. In practice, the authDataAlias is passed at runtime to the custom mapping module. Using the authDataAlias property to look up user IDs and passwords requires the WebSphere Common Configuration Model (WCCM) programming interface, which is not available at this time.
9. Save your changes. Enter the `save` command:
   ```
   wsadmin>save
   ```

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

To develop your own principal and credential mapping LoginModule, see JAAS LoginModule Developer's Guide

(http://java.sun.com/security/jaas/doc/module.html).

In particular, a mapping module needs to obtain the security identity of the caller. The WSSubject.getCallerPrincipal() static method returns a java.lang.String object that represents the caller's security identity. Note that the return type is different from that of the getCallerPrincipal() method of the EJBContext interface, which is java.security.Principal object.

Develop custom user registries

WebSphere Application Server - Express security supports the use of custom registries in addition to LocalOS and LDAP registries for authentication and authorization purposes. A custom user registry is a user registry that you implement. You must implement the UserRegistry

interface that is provided by WebSphere Application Server - Express. A custom implemented user registry can support virtually any type of user registry, such as a relational database or a flat file. The custom user registry provides considerable flexibility in adapting WebSphere Application Server - Express security to various environments where a user registry other than LDAP or LocalOS already exists.

Implementing a custom user registry is a software development effort. Use the methods defined in the UserRegistry interface to make calls to the desired registry to obtain user and group information. The interface defines a very general set of methods, so it can be used to encapsulate a wide variety of registries. For more information, see "UserRegistry interface methods" on page 30. A custom user registry can be configured as the active user registry when configuring WebSphere Application Server - Express global security.

Note: Make sure that your implementation of the custom registry does not depend on any WebSphere Application Server - Express components such as data sources. Do not have this dependency because security is initialized and enabled prior to most of the other WebSphere Application Server - Express components during startup. For example, if your implementation uses a data source to connect to a database, use JDBC to connect to the database instead.

See these code examples for a simple implementation of a custom user registry:

- "Example: UserRegistry.java file" on page 35
- "Example: FileRegistrySample.java file (Versions 5.0 and 5.0.1)" on page 41 (Versions 5.0 and 5.0.1)
- "Example: FileRegistrySample.java file (Version 5.0.2 and later)" on page 57 (Version 5.0.2 and later)
- "Example: Groups.props file" on page 73
- "Example: Users.props file" on page 73
- "Example: Results.java file" on page 74

Perform these steps to develop a custom user registry:

1. If you are not familiar with the concept of custom user registries, see "Custom user registries" on page 29. This topic explains each of the methods in the interface in detail.
2. Implement all of the methods in the interface except for the createCredential() method, which is implemented by WebSphere Application Server - Express.
3. Build your implementation.
4. To compile your code, you need the sas.jar and wssec.jar files in your classpath. For example:

```
javac -extdirs /QIBM/ProdData/WebASE/ASE5/java/ext:/QIBM/UserData/Java400/ext:
  /QIBM/ProdData/Java400/jdk13/lib/ext:/QIBM/ProdData/WebASE/ASE5/lib
  -classpath /QIBM/ProdData/WebASE/ASE5/lib/sas.jar:
  /QIBM/ProdData/WebASE/ASE5/lib/wssec.jar
  com/ibm/websphere/security/FileRegistrySample.java
```

5. Follow the steps in "Configure the custom user registry" on page 96 to configure your implementation using the WebSphere administrative console.

Note: As of Version 5.0.1, you can use JDBC connections to connect to the custom user registry database.

Custom user registries: A custom user registry is a user registry that you implement with the UserRegistry Java interface that is provided by the product. A custom implemented user registry can support virtually any type of user registry, such as a relational database or a flat file. The custom user registry provides considerable flexibility in adapting product security to various environments where some notion of a user registry, other than Lightweight Directory Access Protocol (LDAP) or Local Operating System (LocalOS), already exists in the operational environment.

WebSphere Application Server - Express security provides an implementation that uses various local operating system based registries (Windows, AIX, Solaris, Linux, i5/OS) and various Lightweight Directory Access Protocol (LDAP) based registries. However, there might be situations where your user and group data resides in other repositories (a database, for example) and moving this information to either the LocalOS or LDAP might not be feasible. For these situations the WebSphere Application Server - Express security provides an SPI that you can implement to interact with your current registry. The SPI is the UserRegistry interface. This interface has a set of methods that need implementing in order for the product security to interact with your registries for all security-related tasks. The LocalOS and LDAP registry implementations that are provided also implement this interface. Custom user registries are sometimes called the pluggable user registries or custom registries for short.

The UserRegistry interface is a collection of methods required to authenticate individual users (using either password or certificates) and collect information about the user (privilege attributes) for authorization purposes. It also includes methods that obtain user and group information so that they can be given access to resources. The UserRegistry interface operates on the basis of the several pieces of information. When implementing the methods in the interface, you must decide how to map the information manipulated by the UserRegistry interface to the information in your registry. The methods in the UserRegistry interface operate on the following information for users:

- **User Security Name**
  This refers to the user name, which is similar to the user profile name in the i5/OS Local OS registry. This name is used to login when prompted by a secured application. By default, the servlet methods getRemoteUser() and getUserPrincipal() return this name. The user security name is also referred to as userSecurityName, userName, or user name.
- **Unique ID**
  This ID represents a unique identifier for the user. The UserRegistry interface requires this identifier to be unique. The unique ID similar to the User ID number in i5/OS systems, system ID (SID) in Windows systems, Unique ID (UID) in UNIX systems, and distinguished name (DN) in Lightweight Directory Authentication Protocol (LDAP). This is also referred to as uniqueUserId. The unique ID is used to make the authorization decisions for protected resources.
- **Display name**
  The display name is a registry-specific string that represents a descriptive (but not necessarily unique) name for a user. If the user does not have a display name, an empty string is returned. For i5/OS, the display name is the text description for the user profile.
- **Group Security name**
  This name, which represents the security group, is also referred to as groupSecurityName, groupName, and group name.
- **Unique ID**
  The unique ID is the identifier for a group. It is also referred to as uniqueGroupId.

- **Display name**
  The display name is an optional string that describes a group.

For a description of the methods in the UserRegistry interface that need implementing, see "UserRegistry interface methods."

A simple file-based registry sample is provided. The sample is intended to familiarize you with the custom user registry feature, and should not be used in an actual production environment.
- "Example: UserRegistry.java file" on page 35
- "Example: FileRegistrySample.java file (Versions 5.0 and 5.0.1)" on page 41
- "Example: Groups.props file" on page 73
- "Example: Users.props file" on page 73
- "Example: Results.java file" on page 74

See the "Code license and disclaimer information" on page 190 for legal information about this code example.

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

To implement this interface, you must provide these methods:
- initialize(java.util.Properties)
- checkPassword(String,String)
- mapCertificate(X509Certificate[])
- getRealm
- getUsers(String,int)
- getUserDisplayName(String)
- getUniqueUserId(String)
- getUserSecurityName(String)
- isValidUser(String)
- getGroups(String,int)
- getGroupDisplayName(String)
- getUniqueGroupId(String)
- getUniqueGroupIds(String)
- getGroupSecurityName(String)
- isValidGroup(String)
- getGroupsForUser(String)
- getUsersForGroup(String,int)
- createCredential(String)

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

This method is called to initialize the UserRegistry. All the properties defined in the custom user registry panel propagate to this method.

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

This method is called during server bringup to initialize the registry. This method is also called when validation is performed by the administrative console when security is on.

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

The checkPassword method is called to authenticate users when they log in using a name (or ID) and a password. This method returns a string which, in most cases is the user being authenticated. A credential is then created for the user for authorization purposes. This user name is also returned for the servlet calls getUserPrincipal() and getRemoteUser(). See also the getUserDisplayName method for more information if you have display names in your registry. In some situations if you return a user other than the one who has logged in, ensure that the user is valid in the registry.

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

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

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

The mapCertificate method is called to obtain a user name from a X509 certificate chain supplied by the browser. The complete certificate chain is passed to this method and the implementation can validate the chain if needed and get the user information. A credential is created for this user for authorization purposes. If browser certificates are not supported in your configuration you can throw the exception, CertificateMapNotSupportedException. The consequence of not supporting certificates is that the authentication will fail if the challenge type is certificates, even if they have valid certificates in the browser.

This method is called when certificates are provided for authentication. For Web applications, when the auth-constraints are set to CLIENT-CERT in the web.xml of the application, this method is called to map a certificate to a valid user in the registry. Also, when the Identity Assertion Token (when using the CSIv2 authentication protocol) is set to contain certificates, this method is called to map the certificates to a valid user.

The parameter accepts an array of X509Certificate certificates (for example, certificate chain).

public String getRealm() throws CustomRegistryException, RemoteException;

The getRealm method is called to get the name of the security realm. The name of the realm identifies the security domain for which the registry authenticates users. If this method returns a null value, a default name of customRealm is used.

For the sample, the getRealm method returns the string customRealm. One of the calls to this method is when the registry information is validated.

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

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

The return parameter is an object of type com.ibm.websphere.security.Result.

This object contains two attributes, a java.util.List and a java.lang.boolean. The list should contain the list of users returned and the boolean flag indicates if there are more users available in the registry for the searche pattern. This boolean flag is used to indicate to the client whether more users are available in the registry.

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

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

This method accepts as parameters the pattern and the limit. A Result object, which consists of the list and a flag indicating if more entries exist, is returned. When the list is returned, use the Result.setList(List) to set the List in the Result object. If there are more entries than requested in the Limit parameter, set the boolean attribute to true in the Result object using Result.setHasMore(). The default for the boolean attribute in the Result object is false.

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

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

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

This method can be called by the product to present the display names in the administrative console or through the command line using the wsadmin tool. Use this method only for displaying.

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

This method returns the uniqueId of the user given the security name.

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

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

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

This method is called to make sure a valid user exists for a given uniqueUserId. One of the situations this method is called is to get the security name of the user is when the uniqueUserId is obtained from a token.

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

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

In the sample, this method returns true if the user is found in the registry, otherwise this method returns false. This method is primarily called in situations where knowing if the user exists in the directory or not would prevent problems later. For example, in the mapCertificate call, once the name is obtained from the certificate if the user is found to be an invalid user in the registry, you can avoid trying to create the credential for the user.

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

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

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

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

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

This method accepts as parameters the pattern and the limit. A Result object, which consists of the list and a flag indicating if more entries exist, is returned. When the list is returned, use the Result.setList(List) to set the List in the Result object. If there are more entries than requested in the Limit parameter, set the boolean attribute to true in the Result object using Result.setHasMore(). The default for the boolean attribute in the Result object is false.

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

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

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

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

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

This method returns the uniqueId of the group given the security name.

In the sample, this method returns the uniqueId of the group whose name matches the supplied name. This method is called when creating the authorization table for the application.

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

This method returns the uniqueIds of all the groups to which a user whose uniqueId matches the supplied Id belongs.

In the sample, this method returns the uniqueId of all the groups that contain this uniqueUserId. This method is called when creating the credential for the user. As part of creating the credential, all the groupUniqueIds that this user is apart of are collected and put in the credential for authorization purposes when groups are given access to a resource.

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

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

In the sample, this method returns the security name of the group whose uniqueId matches the supplied Id. This method is called to make sure a valid group exists for a given uniqueGroupId.

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

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

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

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

This method returns all the groups to which a user whose name matches the supplied name belongs. This method is similar to the getUniqueGroupIds method with the except that the securityNames are used, instead of the uniqueIds.

In the sample, this method returns all the group securityNames that contain the userSecurityName.

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

This method retrieves users from the specified group. The number of users returned is limited by the limit parameter. A limit of 0 indicates to return all the users in that group. This method is not directly called by the WebSphere Security component. However, this can be called by other components, such as some product clients like Workflow. In rare situations, if you are working with a registry where getting all the users from any of your groups is not practical (for example if there are a large number of users), you can throw the NotImplementedException for the particular group or groups. If there is no concern about returning the users from groups in the registry, it is recommended that you do not throw the NotImplemented exception when implementing this method.

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

In the example, this method gets one more user than requested if the limit parameter is not set to 0. If it succeeds in getting one more user it sets the boolean flag to true.

This method is optional and can throw the NotImplementedException if you choose not to implement it. This method accepts two parameters, the pattern and the limit. A Result object, which consists of the list and a flag indicating whether more entries exist, is returned. When the list is returned use the Result.setList(List) to set the list in the Result object. If there are more entries than requested in the limit parameter, set the boolean attribute to true in the Result object using Result.setHasMore(). The default for the boolean attribute in the Result object is false.

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

In this release of the WebSphere Application Server - Express, this method is not called. You should, instead, return null. In the example, a null is returned. This method did not exist in Version 4.0.

Example: UserRegistry.java file:

```
// 5722-IWE
// (C) COPYRIGHT International Business Machines Corp. 1997, 2003
// All Rights Reserved * Licensed Materials - Property of IBM
//
// DESCRIPTION:
//
//    This is the UserRegistry interface that Custom Registries in WebSphere
//    should implement to enable WebSphere Security to use the Custom Registry.
//

package com.ibm.websphere.security;

import java.util.*;
import java.rmi.*;
import java.security.cert.X509Certificate;
import com.ibm.websphere.security.cred.WSCredential;

/**
 * Implementing this interface enables WebSphere Security to use Custom
 * Registries. This should extend java.rmi.Remote as the registry can be in
 * a remote process.
 *
 * To implement this interface, you must provide implementations for:
 *
 * -- initialize(java.util.Properties)
 * -- checkPassword(String,String)
 * -- mapCertificate(X509Certificate[])
 * -- getRealm
 * -- getUsers(String,int)
 * -- getUserDisplayName(String)
 * -- getUniqueUserId(String)
 * -- getUserSecurityName(String)
 * -- isValidUser(String)
 * -- getGroups(String,int)
 * -- getGroupDisplayName(String)
 * -- getUniqueGroupId(String)
 * -- getUniqueGroupIds(String)
 * -- getGroupSecurityName(String)
 * -- isValidGroup(String)
 * -- getGroupsForUser(String)
 * -- getUsersForGroup(String,int)
 * -- createCredential(String)
**/

public interface UserRegistry extends java.rmi.Remote
{

  /**
   * Initializes the registry. This method is called when creating the
   * registry.
   *
   * @param props the registry-specific properties with which to
   *              initialize the  custom registry
   * @exception CustomRegistryException
   *                   if there is any registry specific problem
   * @exception RemoteException
   *    as this extends java.rmi.Remote
   **/
   public void initialize(java.util.Properties props)
      throws CustomRegistryException,
             RemoteException;

  /**
   * Checks the password of the user. This method is called to authenticate a
   * user when the user's name and password are given.
   *
   * @param userSecurityName the name of user
   * @param password the password of the user
   * @return    a valid userSecurityName. Normally this is
   *   the name of same user whose password was checked but if the
   *  implementation wants to return any other valid
   *  userSecurityName in the registry it can do so
   * @exception CheckPasswordFailedException if userSecurityName/
   *  password combination does not exist in the registry
   * @exception CustomRegistryException if there is any registry specific
   *           problem
   * @exception RemoteException as this extends java.rmi.Remote
   **/
   public String checkPassword(String userSecurityName, String password)
      throws PasswordCheckFailedException,
             CustomRegistryException,
             RemoteException;

  /**
   * Maps a Certificate (of X509 format) to a valid user in the Registry.
   * This is used to map the name in the certificate supplied by a browser
   * to a valid userSecurityName in the registry
   *
   * @param cert the X509 certificate chain
   * @return the mapped name of the user userSecurityName
   * @exception CertificateMapNotSupportedException if the particular
   *           certificate is not supported.
   * @exception CertificateMapFailedException if the mapping of the
   *           certificate fails.
   * @exception CustomRegistryException if there is any registry specific
   *           problem
   * @exception RemoteException as this extends java.rmi.Remote
   **/
   public String mapCertificate(X509Certificate[] cert)
      throws CertificateMapNotSupportedException,
             CertificateMapFailedException,
             CustomRegistryException,
             RemoteException;

  /**
   * Returns the realm of the registry.
   *
   * @return the realm. The realm is a registry-specific string indicating
   *           the realm or domain for which this registry
   *           applies.  For example, for OS400 or AIX this would be the
   *           host name of the system whose user registry this object
   *           represents.
   *           If null is returned by this method realm defaults to the
   *           value of "customRealm".
   * @exception CustomRegistryException if there is any registry specific
   *           problem
   * @exception RemoteException as this extends java.rmi.Remote
   **/
   public String getRealm()
      throws CustomRegistryException,
             RemoteException;

  /**
   * Gets a list of users that match a pattern in the registy.
   * The maximum number of users returned is defined by the limit
   * argument.
   * This method is called by GUI(adminConsole) and Scripting(Command Line) to
   * make available the users in the registry for adding them (users) to roles.
   *
   * @param pattern the pattern to match. (For e.g., a* will match all
   *   userSecurityNames starting with a)
   * @param limit the maximum number of users that should be returned.
   *  This is very useful in situations where there are thousands of
   *           users in the registry and getting all of them at once is not
   *           practical. A value of 0 implies get all the users and hence
   *           must be used with care.
   * @return a Result object that contains the list of users
   *   requested and a flag to indicate if more users exist.
   * @exception CustomRegistryException if there is any registry specific
   *           problem
   * @exception RemoteException as this extends java.rmi.Remote
   **/
   public Result getUsers(String pattern, int limit)
      throws CustomRegistryException,
             RemoteException;

  /**
   * Returns the display name for the user specified by userSecurityName.
   *
   * This method may be called only when the user information is displayed
   * (i.e information purposes only, for example, in GUI) and hence not used
   * in the actual authentication or authorization purposes. If there are no
   * display names in the registry return null or empty string.
   *
   * In WAS 4.0 custom registry, if you had a display name for the user and
   * if it was different from the security name, the display name was
   * returned for the servlet methods getUserPrincipal() and  getRemoteUser().
   * In WAS 5.0 for the same methods the security name will be returned by
   * default. This is the recommended way as the display name is not unique
   * and might create security holes.
   * However, for backward compatability if one needs the display name to
   * be returned set the property WAS_UseDisplayName to true.
   *
   * See the Infocenter documentation for more information.
   *
   * @param userSecurityName the name of the user.
   * @return the display name for the user. The display name
   *   is a registry-specific string that represents a descriptive, not
   *  necessarily unique, name for a user. If a display name does
   *           not exist return null or empty string.
   * @exception EntryNotFoundException if userSecurityName does not exist.
   * @exception CustomRegistryException if there is any registry specific
   *           problem
   * @exception RemoteException as this extends java.rmi.Remote
   **/
   public String getUserDisplayName(String userSecurityName)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;

  /**
   * Returns the UniqueId for a userSecurityName. This method is called when
   * creating a credential for a user.
   *
   * @param userSecurityName the name of the user.
   * @return the UniqueId of the user. The UniqueId for an user is
   *   the stringified form of some unique, registry-specific, data
   *  that serves to represent the user.  For example, for the UNIX
   *  user registry, the UniqueId for a user can be the UID.
   * @exception EntryNotFoundException if userSecurityName does not exist.
   * @exception CustomRegistryException if there is any registry specific
   *           problem
   * @exception RemoteException as this extends java.rmi.Remote
   **/
   public String getUniqueUserId(String userSecurityName)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;

  /**
   * Returns the name for a user given its uniqueId.
   *
   * @param uniqueUserId the UniqueId of the user.
   * @return the userSecurityName of the user.
   * @exception EntryNotFoundException if the uniqueUserId does not exist.
   * @exception CustomRegistryException if there is any registry specific
   *           problem
   * @exception RemoteException as this extends java.rmi.Remote
   **/
   public String getUserSecurityName(String uniqueUserId)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;

  /**
   * Determines if the userSecurityName exists in the registry
   *
   * @param userSecurityName the name of the user
   * @return true if the user is valid. false otherwise
   * @exception CustomRegistryException if there is any registry specific
   *           problem
   * @exception RemoteException as this extends java.rmi.Remote
   **/
   public boolean isValidUser(String userSecurityName)
      throws CustomRegistryException,
             RemoteException;

  /**
   * Gets a list of groups that match a pattern in the registy.
   * The maximum number of groups returned is defined by the limit
   * argument.
   * This method is called by GUI(adminConsole) and Scripting(Command Line) to
   * make available the groups in the registry for adding them (groups) to
   * roles.
   *
   * @param pattern the pattern to match. (For e.g., a* will match all
   *   groupSecurityNames starting with a)
   * @param limit the maximum number of groups that should be returned.
   *  This is very useful in situations where there are thousands of
   *            groups in the registry and getting all of them at once is not
   *            practical. A value of 0 implies get all the groups and hence
   *            must be used with care.
   * @return a Result object that contains the list of groups
   *   requested and a flag to indicate if more groups exist.
   * @exception CustomRegistryException if there is any registry specific
   *            problem
   * @exception RemoteException as this extends java.rmi.Remote
   **/
   public Result getGroups(String pattern, int limit)
      throws CustomRegistryException,
             RemoteException;

  /**
   * Returns the display name for the group specified by groupSecurityName.
   *
   * This method may be called only when the group information is displayed
   * (for example, GUI) and hence not used in the actual authentication or
   * authorization purposes. If there are no display names in the registry
   * return null or empty string.
   *
   * @param groupSecurityName the name of the group.
   * @return the display name for the group. The display name
   *   is a registry-specific string that represents a descriptive, not
   *  necessarily unique, name for a group. If a display name does
   *            not exist return null or empty string.
   * @exception EntryNotFoundException if groupSecurityName does not exist.
   * @exception CustomRegistryException if there is any registry specific
   *            problem
   * @exception RemoteException as this extends java.rmi.Remote
   **/
   public String getGroupDisplayName(String groupSecurityName)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;

  /**
   * Returns the Unique id for a group.

   * @param groupSecurityName the name of the group.
   * @return the Unique id of the group. The Unique id for
   *   a group is the stringified form of some unique,
   *   registry-specific, data that serves to represent the group.
   *            For example, for the Unix user registry, the Unique id could
   *  be the GID.
   * @exception EntryNotFoundException if groupSecurityName does not exist.
   * @exception CustomRegistryException if there is any registry specific
   *            problem
   * @exception RemoteException as this extends java.rmi.Remote
   **/
   public String getUniqueGroupId(String groupSecurityName)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;


  /**
   * Returns the Unique ids for all the groups that contain the UniqueId of
   * a user.
   * Called during creation of a user's credential.
   *
   * @param uniqueUserId the uniqueId of the user.
   * @return a List of all the group UniqueIds that the uniqueUserId
   *   belongs to. The Unique id for an entry is the stringified
   *  form of some unique, registry-specific, data that serves
   *  to represent the entry.  For example, for the
   *   Unix user registry, the Unique id for a group could be the GID
   *  and the Unique Id for the user could be the UID.
   * @exception EntryNotFoundException if uniqueUserId does not exist.
   * @exception CustomRegistryException if there is any registry specific
   *            problem
   * @exception RemoteException as this extends java.rmi.Remote
   **/
   public List getUniqueGroupIds(String uniqueUserId)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;

  /**
   * Returns the name for a group given its uniqueId.
   *
   * @param uniqueGroupId the UniqueId of the group.
   * @return the name of the group.
   * @exception EntryNotFoundException if the uniqueGroupId does not exist.
   * @exception CustomRegistryException if there is any registry specific
   *            problem
   * @exception RemoteException as this extends java.rmi.Remote
   **/
   public String getGroupSecurityName(String uniqueGroupId)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;

  /**
   * Determines if the groupSecurityName exists in the registry
   *
   * @param groupSecurityName the name of the group
   * @return true if the groups exists, false otherwise
   * @exception CustomRegistryException if there is any registry specific
   *            problem
   * @exception RemoteException as this extends java.rmi.Remote
   **/
   public boolean isValidGroup(String groupSecurityName)
      throws CustomRegistryException,
             RemoteException;

  /**
   * Returns the securityNames of all the groups that contain the user
   *
   * This method is called by GUI(adminConsole) and Scripting(Command Line)
   * to verify the user entered for role mapping belongs to that role
   * in the roles to user mapping. Initially, the check is done to see if
   * the role contains the user. If the role does not contain the user
   * explicitly, this method is called to get the groups that this user
   * belongs to so that check can be made on the groups that the role contains.
   *
   * @param userSecurityName the name of the user
   * @return a List of all the group securityNames that the user
   *   belongs to.
   * @exception EntryNotFoundException if user does not exist.
   * @exception CustomRegistryException if there is any registry specific
   *            problem
   * @exception RemoteException as this extends java.rmi.Remote
   **/
   public List getGroupsForUser(String userSecurityName)
      throws EntryNotFoundException,
             CustomRegistryException,
             RemoteException;

  /**
   * Gets a list of users in a group.
   *
   * The maximum number of users returned is defined by the limit
   * argument.
   *
   * This method is not used by WebSphere Application Server - Express for
   * authenticating or authorization purposes. This is, however, used by some
   * of the WebSphere clients like Workflow.
   *
   * If you are working with a registry where getting all the users from
   * any of your groups is not practical (for example if there are a large
   * number of users) you can through the NotImplementedException. Also,
   * if you implement this method, you can still throw this exception if
   * the limit exceeds some practical value.
   * When the NotImplementedException is thrown the client program should fall
   * back to some default implementation which should be documented by the
   * client.
   *
   * @param groupSecurityName the name of the group
   * @param limit the maximum number of users that should be returned.
   *  This is very useful in situations where there are lot of
   *           users in the registry and getting all of them at once is not
   *           practical. A value of 0 implies get all the users and hence
   *           must be used with care.
   * @return a Result object that contains the list of users
   *   requested and a flag to indicate if more users exist.
   * @deprecated This method will be deprecated in future.
   * @exception NotImplementedException throw this exception if it is not
   *           pratical to get this information from your registry.
   * @exception EntryNotFoundException if the group does not exist in
   *           the registry
   * @exception CustomRegistryException if there is any registry specific
   *           problem
   * @exception RemoteException as this extends java.rmi.Remote
   *
   **/
   public Result getUsersForGroup(String groupSecurityName, int limit)
      throws NotImplementedException,
             EntryNotFoundException,
             CustomRegistryException,
             RemoteException;

  /**
   * Throw the NotImplementedException for this method.
   *
   * Create Credential for a user.
   *
   * This will be implemented internally by WebSphere code and should NOT be
   * implemented by the Custom Registry implementations.
   *
   * @exception NotImplementedException Always throw this.
   **/
   public WSCredential createCredential(String userSecurityName)
      throws NotImplementedException,
      EntryNotFoundException,
             CustomRegistryException,
             RemoteException;
}
```

Example: FileRegistrySample.java file (Versions 5.0 and 5.0.1):

```
package com.ibm.websphere.security;
//
// 5722-IWE
// (C) COPYRIGHT International Business Machines Corp. 1997, 2003
// All Rights Reserved * Licensed Materials - Property of IBM
//

//----------------------------------------------------------------------
// This program may be used, executed, copied, modified and distributed
// without royalty for the purpose of developing, using, marketing, or
// distributing.
//----------------------------------------------------------------------
//

// This sample is for the Custom User Registry feature in WebSphere

//----------------------------------------------------------------------
// The main purpose of this sample is to demonstrate the use of the
// Custom Registry feature available in WebSphere. This sample is a very
// simple File based registry sample where the users and the groups information
// is listed in files (users.props and groups.props). As such simplicity and
// not the performance was a major factor behind this. This sample should be
// used only to get familiarized with this feature. An actual implementation
// of a realistic registry should consider various factors like performance,
// scalability etc.
//----------------------------------------------------------------------
import java.util.*;
import java.io.*;
import java.security.cert.X509Certificate;
import com.ibm.websphere.security.*;

public class FileRegistrySample implements UserRegistry {

   private static String USERFILENAME = null;
   private static String GROUPFILENAME = null;

   // Default Constructor
   public FileRegistrySample() throws java.rmi.RemoteException {
   }

  /**
   * Initializes the registry. This method is called when creating the
   * registry.
   *
   * @param     props   the registry-specific properties with which to
   *                    initialize the  custom registry
   * @exception CustomRegistryException
   *                    if there is any registry specific problem
   **/
   public void initialize(java.util.Properties props)
         throws CustomRegistryException {
      try {
          /* try getting the USERFILENAME and the GROUPFILENAME from
          * properties that are passed in (i.e from GUI).
          * These values should be set in the security center GUI in the
          * Special Custom Settings in the Custom User Registry section of
          * the Authentication panel.
          * For example:
          * usersFile   c:/temp/users.props
          * groupsFile  c:/temp/groups.props
          */
          if (props != null) {
             USERFILENAME = props.getProperty("usersFile");
             GROUPFILENAME = props.getProperty("groupsFile");
          }

      } catch(Exception ex) {
         throw new CustomRegistryException(ex.getMessage());
      }
      if (USERFILENAME == null || GROUPFILENAME == null) {
         throw new CustomRegistryException("users/groups information missing");
      }

   }

  /**
   * Checks the password of the user. This method is called to authenticate a
   * user when the user's name and password are given.
   *
   * @param     userSecurityName the name of user
   * @param     password the password of the user
   * @return    a valid <code>userSecurityName. Normally this is
   *            the name of same user whose password was checked but if the
   *            implementation wants to return any other valid
   *            <code>userSecurityName in the registry it can do so
   * @exception CheckPasswordFailedException if <code>userSecurityName/
   *            <code>password combination does not exist in the registry
   * @exception CustomRegistryException if there is any registry specific
   *            problem
   **/
   public String checkPassword(String userSecurityName, String passwd)
      throws PasswordCheckFailedException,
             CustomRegistryException {
      String s,userName = null;
      BufferedReader in = null;

      try {
         in = fileOpen(USERFILENAME);
         while ((s=in.readLine())!=null)
         {
            if (!(s.startsWith("#") || s.trim().length() <=0 )) {
               int index = s.indexOf(":");
               int index1 = s.indexOf(":",index+1);
               // check if the userSecurityName:passwd combination exists
               if ((s.substring(0,index)).equals(userSecurityName) &&
                       s.substring(index+1,index1).equals(passwd)) {
                  // Authentication successful, return the userId.
                  userName = userSecurityName;
                  break;
               }
            }
         }
      } catch(Exception ex) {
         throw new CustomRegistryException(ex.getMessage());
      } finally {
         fileClose(in);
      }


      if (userName == null) {
         throw new PasswordCheckFailedException(userSecurityName);
      }

      return userName;
   }

  /**
   * Maps a Certificate (of X509 format) to a valid user in the Registry.
   * This is used to map the name in the certificate supplied by a browser
   * to a valid <code>passworduserSecurityName in the registry
   *
   * @param     cert the X509 certificate chain
   * @return    the mapped name of the user <code>userSecurityName
   * @exception CertificateMapNotSupportedException if the particular
   *            certificate is not supported.
   * @exception CertificateMapFailedException if the mapping of the
```

```
 *            certificate fails.
 * @exception CustomRegistryException if there is any registry specific
 *            problem
 **/
 public String mapCertificate(X509Certificate[] cert)
    throws CertificateMapNotSupportedException,
           CertificateMapFailedException,
           CustomRegistryException {
    String name=null;
    X509Certificate cert1 = cert[0];
    try {
       // map the SubjectDN in the certificate to a userID.
       name = cert1.getSubjectDN().getName();
    } catch(Exception ex) {
       throw new CertificateMapNotSupportedException(ex.getMessage());
    }

    if(!isValidUser(name)) {
       throw new CertificateMapFailedException(name);
    }
    return name;
 }

/**
 * Returns the realm of the registry.
 *
 * @return    the realm. The realm is a registry-specific string indicating
 *            the realm or domain for which this registry
 *            applies.  For example, for OS400 or AIX this would be the
 *            host name of the system whose user registry this object
 *            represents.
 *            If null is returned by this method realm defaults to the
 *            value of "customRealm".
 * @exception CustomRegistryException if there is any registry specific
 *            problem
 **/
 public String getRealm()
    throws CustomRegistryException {
    String name = "customRealm";
    return name;
 }

/**
 * Gets a list of users that match a <code>pattern in the registy.
 * The maximum number of users returned is defined by the <code>limit
 * argument.
 * This method is called by GUI(adminConsole) and Scripting(Command Line) to
 * make available the users in the registry for adding them (users) to roles.
 *
 * @param     pattern the pattern to match. (For e.g., a* will match all
 *            userSecurityNames starting with a)
 * @param     limit the maximum number of users that should be returned.
 *            This is very useful in situations where there are thousands of
 *            users in the registry and getting all of them at once is not
 *            practical. The default is 100. A value of 0 implies get all the
 *            users and hence must be used with care.
 * @return    a <code>Result object that contains the list of users
 *            requested and a flag to indicate if more users exist.
 * @exception CustomRegistryException if there is any registry specific
 *            problem
 **/
 public Result getUsers(String pattern, int limit)
    throws CustomRegistryException {
    String s;
    BufferedReader in = null;
    List allUsers = new ArrayList();
    Result result = new Result();
```

```
      int count = 0;
      int newLimit = limit+1;
      try {
         in = fileOpen(USERFILENAME);
         while ((s=in.readLine())!=null)
         {
            if (!(s.startsWith("#") || s.trim().length() <=0 )) {
               int index = s.indexOf(":");
               String user = s.substring(0,index);
               if (match(user,pattern)) {
                  allUsers.add(user);
                  if (limit !=0 && ++count == newLimit) {
                     allUsers.remove(user);
                     result.setHasMore();
                     break;
                  }
               }
            }
         }
      } catch (Exception ex) {
         throw new CustomRegistryException(ex.getMessage());
      } finally {
         fileClose(in);
      }

      result.setList(allUsers);
      return result;
   }

  /**
   * Returns the display name for the user specified by userSecurityName.
   *
   * This method may be called only when the user information is displayed
   * (i.e information purposes only, for example, in GUI) and hence not used
   * in the actual authentication or authorization purposes. If there are no
   * display names in the registry return null or empty string.
   *
   * In WAS 4.0 custom registry, if you had a display name for the user and
   * if it was different from the security name, the display name was
   * returned for the servlet methods
   * getUserPrincipal() and  getRemoteUser().
   * In WAS 5.0 for the same methods the security name will be returned by
   * default. This is the recommended way as the display name is not unique
   * and might create security holes.
   * However, if you need the display name to be returned for backward compatability,
   * set the property WAS_UseDisplayName to true.
   *
   * @param     userSecurityName the name of the user.
   * @return    the display name for the user. The display name
   *            is a registry-specific string that represents a descriptive, not
   *            necessarily unique, name for a user. If a display name does
   *            not exist return null or empty string.
   * @exception EntryNotFoundException if userSecurityName does not exist.
   * @exception CustomRegistryException if there is any registry specific
   *            problem
   **/
   public String getUserDisplayName(String userSecurityName)
      throws CustomRegistryException,
             EntryNotFoundException {

      String s,displayName = null;
      BufferedReader in = null;

      if(!isValidUser(userSecurityName)) {
         EntryNotFoundException nsee = new EntryNotFoundException(userSecurityName);
         throw nsee;
      }
      try {
         in = fileOpen(USERFILENAME);
         while ((s=in.readLine())!=null)
         {
            if (!(s.startsWith("#") || s.trim().length() <=0 )) {
               int index = s.indexOf(":");
               int index1 = s.lastIndexOf(":");
               if ((s.substring(0,index)).equals(userSecurityName)) {
                  displayName = s.substring(index1+1);
                  break;
               }
            }
         }
      } catch(Exception ex) {
         throw new CustomRegistryException(ex.getMessage());
      } finally {
         fileClose(in);
      }

      return displayName;
   }

  /**
   * Returns the UniqueId for a userSecurityName. This method is called when
   * creating a credential for a user.
   *
   * @param     userSecurityName the name of the user.
   * @return    the UniqueId of the user. The UniqueId for an user is
   *            the stringified form of some unique, registry-specific, data
   *            that serves to represent the user.  For example, for the UNIX
   *            user registry, the UniqueId for a user can be the UID.
   * @exception EntryNotFoundException if userSecurityName does not exist.
   * @exception CustomRegistryException if there is any registry specific
   *            problem
   **/
   public String getUniqueUserId(String userSecurityName)
      throws CustomRegistryException,
             EntryNotFoundException {

      String s,uniqueUsrId = null;
      BufferedReader in = null;
      try {
         in = fileOpen(USERFILENAME);
         while ((s=in.readLine())!=null)
         {
            if (!(s.startsWith("#") || s.trim().length() <=0 )) {
               int index = s.indexOf(":");
               int index1 = s.indexOf(":", index+1);
               if ((s.substring(0,index)).equals(userSecurityName)) {
                  int index2 = s.indexOf(":", index1+1);
                  uniqueUsrId = s.substring(index1+1,index2);
                  break;
               }
            }
         }
      } catch(Exception ex) {
         throw new CustomRegistryException(ex.getMessage());
      } finally {
         fileClose(in);
      }

      if (uniqueUsrId == null) {
         EntryNotFoundException nsee = new EntryNotFoundException(userSecurityName);
         throw nsee;
      }
      return uniqueUsrId;
   }

  /**
   * Returns the name for a user given its uniqueId.
   *
   * @param     uniqueUserId the UniqueId of the user.
   * @return    the userSecurityName of the user.
   * @exception EntryNotFoundException if the uniqueUserId does not exist.
   * @exception CustomRegistryException if there is any registry specific
   *            problem
   **/
   public String getUserSecurityName(String uniqueUserId)
      throws CustomRegistryException,
             EntryNotFoundException {
      String s,usrSecName = null;
      BufferedReader in = null;
      try {
         in = fileOpen(USERFILENAME);
         while ((s=in.readLine())!=null)
         {
            if (!(s.startsWith("#") || s.trim().length() <=0 )) {
               int index = s.indexOf(":");
               int index1 = s.indexOf(":", index+1);
               int index2 = s.indexOf(":", index1+1);
               if ((s.substring(index1+1,index2)).equals(uniqueUserId)) {
                  usrSecName = s.substring(0,index);
                  break;
               }
            }
         }
      } catch (Exception ex) {
         throw new CustomRegistryException(ex.getMessage());
      } finally {
         fileClose(in);
      }

      if (usrSecName == null) {
         EntryNotFoundException ex =
            new EntryNotFoundException(uniqueUserId);
         throw ex;
      }

      return usrSecName;

   }

  /**
   * Determines if the <code>userSecurityName exists in the registry
   *
   * @param     userSecurityName the name of the user
   * @return    true if the user is valid. false otherwise
   * @exception CustomRegistryException if there is any registry specific
   *            problem
   * @exception RemoteException as this extends java.rmi.Remote
   **/
   public boolean isValidUser(String userSecurityName)
      throws CustomRegistryException {
      String s;
      boolean isValid = false;
      BufferedReader in = null;
      try {
         in = fileOpen(USERFILENAME);
         while ((s=in.readLine())!=null)
         {
            if (!(s.startsWith("#") || s.trim().length() <=0 )) {
               int index = s.indexOf(":");
               if ((s.substring(0,index)).equals(userSecurityName)) {
                  isValid=true;
                  break;
               }
            }
         }
      } catch (Exception ex) {
         throw new CustomRegistryException(ex.getMessage());
      } finally {
         fileClose(in);
      }

      return isValid;
   }


  /**
   * Gets a list of groups that match a <code>pattern in the registy.
   * The maximum number of groups returned is defined by the <code>limit
   * argument.
   * This method is called by GUI(adminConsole) and Scripting(Command Line) to
   * make available the groups in the registry for adding them (groups) to
   * roles.
   *
   * @param     pattern the pattern to match. (For e.g., a* will match all
   *           groupSecurityNames starting with a)
   * @param     limit the maximum number of groups that should be returned.
   *           This is very useful in situations where there are thousands of
   *           groups in the registry and getting all of them at once is not
   *           practical. The default is 100. A value of 0 implies get all the
   *           groups and hence must be used with care.
   * @return    a <code>Result object that contains the list of groups
   *           requested and a flag to indicate if more groups exist.
   * @exception CustomRegistryException if there is any registry specific
   *           problem
   **/
   public Result getGroups(String pattern, int limit)
      throws CustomRegistryException {
      String s;
      BufferedReader in = null;
      List allGroups = new ArrayList();
      Result result = new Result();
      int count = 0;
      int newLimit = limit+1;
      try {
         in = fileOpen(GROUPFILENAME);
         while ((s=in.readLine())!=null)
         {
            if (!(s.startsWith("#") || s.trim().length() <=0 )) {
               int index = s.indexOf(":");
               String group = s.substring(0,index);
               if (match(group,pattern)) {
                  allGroups.add(group);
                  if (limit !=0 && ++count == newLimit) {
                     allGroups.remove(group);
                     result.setHasMore();
                     break;
                  }
               }
            }
         }
      } catch (Exception ex) {
         throw new CustomRegistryException(ex.getMessage());
      } finally {
         fileClose(in);
      }
      result.setList(allGroups);
      return result;
   }

  /**
   * Returns the display name for the group specified by groupSecurityName.
   * For this version of WebSphere the only usage of this method is by the
   * clients (GUI and Scripting) to present a descriptive name of the user
   * if it exists.
   *
   * @param     groupSecurityName the name of the group.
   * @return    the display name for the group. The display name
   *            is a registry-specific string that represents a descriptive, not
   *            necessarily unique, name for a group. If a display name does
   *            not exist return null or empty string.
   * @exception EntryNotFoundException if groupSecurityName does not exist.
   * @exception CustomRegistryException if there is any registry specific
   *            problem
   **/
   public String getGroupDisplayName(String groupSecurityName)
      throws CustomRegistryException,
             EntryNotFoundException {
      String s,displayName = null;
      BufferedReader in = null;

      if(!isValidGroup(groupSecurityName)) {
         EntryNotFoundException nsee = new EntryNotFoundException(groupSecurityName);
         throw nsee;
      }

      try {
         in = fileOpen(GROUPFILENAME);
         while ((s=in.readLine())!=null)
         {
            if (!(s.startsWith("#") || s.trim().length() <=0 )) {
               int index = s.indexOf(":");
               int index1 = s.lastIndexOf(":");
               if ((s.substring(0,index)).equals(groupSecurityName)) {
                  displayName = s.substring(index1+1);
                  break;
               }
            }
         }
      } catch(Exception ex) {
         throw new CustomRegistryException(ex.getMessage());
      } finally {
         fileClose(in);
      }

      return displayName;
   }

  /**
   * Returns the Unique id for a group.

   * @param     groupSecurityName the name of the group.
   * @return    the Unique id of the group. The Unique id for
   *            a group is the stringified form of some unique,
   *            registry-specific, data that serves to represent the group.
   *            For example, for the Unix user registry, the Unique id could
   *            be the GID.
   * @exception EntryNotFoundException if groupSecurityName does not exist.
   * @exception CustomRegistryException if there is any registry specific
   *            problem
   * @exception RemoteException as this extends java.rmi.Remote
   **/
   public String getUniqueGroupId(String groupSecurityName)
```

```
    throws CustomRegistryException,
           EntryNotFoundException {
    String s,uniqueGrpId = null;
    BufferedReader in = null;
    try {
       in = fileOpen(GROUPFILENAME);
       while ((s=in.readLine())!=null)
       {
          if (!(s.startsWith("#") || s.trim().length() <=0 )) {
             int index = s.indexOf(":");
             int index1 = s.indexOf(":", index+1);
             if ((s.substring(0,index)).equals(groupSecurityName)) {
                uniqueGrpId = s.substring(index+1,index1);
                break;
             }
          }
       }
    } catch(Exception ex) {
       throw new CustomRegistryException(ex.getMessage());
    } finally {
       fileClose(in);
    }

    if (uniqueGrpId == null) {
       EntryNotFoundException nsee = new EntryNotFoundException(groupSecurityName);
       throw nsee;
    }

    return uniqueGrpId;
 }

/**
 * Returns the Unique ids for all the groups that contain the UniqueId of
 * a user. Called during creation of a user's credential.
 *
 * @param     uniqueUserId the uniqueId of the user.
 * @return    a List of all the group UniqueIds that the uniqueUserId
 *            belongs to. The Unique id for an entry is the stringified
 *            form of some unique, registry-specific, data that serves
 *            to represent the entry.  For example, for the
 *            Unix user registry, the Unique id for a group could be the GID
 *            and the Unique Id for the user could be the UID.
 * @exception EntryNotFoundException if uniqueUserId does not exist.
 * @exception CustomRegistryException if there is any registry specific
 *            problem
 **/
 public List getUniqueGroupIds(String uniqueUserId)
    throws CustomRegistryException,
           EntryNotFoundException {
    String s,uniqueGrpId = null;
    BufferedReader in = null;
    List uniqueGrpIds=new ArrayList();
    try {
       in = fileOpen(USERFILENAME);
       while ((s=in.readLine())!=null)
       {
          if (!(s.startsWith("#") || s.trim().length() <=0 )) {
             int index = s.indexOf(":");
             int index1 = s.indexOf(":", index+1);
             int index2 = s.indexOf(":", index1+1);
             if ((s.substring(index1+1,index2)).equals(uniqueUserId)) {
                int lastIndex = s.lastIndexOf(":");
                String subs = s.substring(index2+1,lastIndex);
                StringTokenizer st1 = new StringTokenizer(subs, ",");
                while (st1.hasMoreTokens())
                   uniqueGrpIds.add(st1.nextToken());
                break;
```

```
            }
         }
      }
   } catch(Exception ex) {
      throw new CustomRegistryException(ex.getMessage());
   } finally {
      fileClose(in);
   }

   return uniqueGrpIds;
}

/**
 * Returns the name for a group given its uniqueId.
 *
 * @param     uniqueGroupId the UniqueId of the group.
 * @return    the name of the group.
 * @exception EntryNotFoundException if the uniqueGroupId does not exist.
 * @exception CustomRegistryException if there is any registry specific
 *            problem
 **/
public String getGroupSecurityName(String uniqueGroupId)
   throws CustomRegistryException,
          EntryNotFoundException {
   String s,grpSecName = null;
   BufferedReader in = null;
   try {
      in = fileOpen(GROUPFILENAME);
      while ((s=in.readLine())!=null)
      {

         if (!(s.startsWith("#") || s.trim().length() <=0 )) {
            int index = s.indexOf(":");
            int index1 = s.indexOf(":", index+1);
            if ((s.substring(index+1,index1)).equals(uniqueGroupId)) {
               grpSecName = s.substring(0,index);
               break;
            }
         }
      }
   } catch (Exception ex) {
      throw new CustomRegistryException(ex.getMessage());
   } finally {
      fileClose(in);
   }

   if (grpSecName == null) {
      EntryNotFoundException ex =
         new EntryNotFoundException(uniqueGroupId);
      throw ex;
   }

   return grpSecName;

}

/**
 * Determines if the <code>groupSecurityName exists in the registry
 *
 * @param     groupSecurityName the name of the group
 * @return    true if the groups exists, false otherwise
 * @exception CustomRegistryException if there is any registry specific
 *            problem
 **/
public boolean isValidGroup(String groupSecurityName)
   throws CustomRegistryException {
   String s;
```

```
      boolean isValid = false;
      BufferedReader in = null;
      try {
         in = fileOpen(GROUPFILENAME);
         while ((s=in.readLine())!=null)
         {
            if (!(s.startsWith("#") || s.trim().length() <=0 )) {
               int index = s.indexOf(":");
               if ((s.substring(0,index)).equals(groupSecurityName)) {
                  isValid=true;
                  break;
               }
            }
         }
      } catch (Exception ex) {
         throw new CustomRegistryException(ex.getMessage());
      } finally {
         fileClose(in);
      }

      return isValid;
   }

  /**
   * Returns the securityNames of all the groups that contain the user
   *
   * This method is called by GUI(adminConsole) and Scripting(Command Line)
   * to verify the user entered for role mapping belongs to that role
   * in the roles to user mapping. Initially, the check is done to see if
   * the role contains the user. If the role does not contain the user
   * explicitly, this method is called to get the groups that this user
   * belongs to so that check can be made on the groups that the role contains.
   *
   * @param     userSecurityName the name of the user
   * @return    a List of all the group securityNames that the user
   *            belongs to.
   * @exception EntryNotFoundException if user does not exist.
   * @exception CustomRegistryException if there is any registry specific
   *            problem
   * @exception RemoteException as this extends java.rmi.Remote
   **/
   public List getGroupsForUser(String userName)
      throws CustomRegistryException,
             EntryNotFoundException {
      String s;
      List grpsForUser = new ArrayList();
      BufferedReader in = null;
      try {
         in = fileOpen(GROUPFILENAME);
         while ((s=in.readLine())!=null)
         {
            if (!(s.startsWith("#") || s.trim().length() <=0 )) {
               StringTokenizer st = new StringTokenizer(s, ":");
               for (int i=0; i<2; i++)
                  st.nextToken();
               String subs = st.nextToken();
               StringTokenizer st1 = new StringTokenizer(subs, ",");
               while (st1.hasMoreTokens()) {
                  if((st1.nextToken()).equals(userName)) {
                     int index = s.indexOf(":");
                     grpsForUser.add(s.substring(0,index));
                  }
               }
            }
         }
      } catch (Exception ex) {
         if (!isValidUser(userName)) {
```

```
         throw new EntryNotFoundException(userName);
      }
      throw new CustomRegistryException(ex.getMessage());
   } finally {
      fileClose(in);
   }

   return grpsForUser;
}

/**
 * Gets a list of users in a group.
 *
 * The maximum number of users returned is defined by the limit
 * argument.
 *
 * This method is not used by WebSphere Application Server- Express for
 * authenticating or authorization purposes. This is, however, used by some
 * of the WebSphere Application Server - Express clients like Workflow.
 *
 * If you are working with a registry where getting all the users from
 * any of your groups is not practical (for example if there are a large
 * number of users) you can through the NotImplementedException. Also,
 * if you implement this method, you can still throw this exception if
 * the limit exceeds some practical value.
 * When the NotImplementedException is thrown the client program should fall
 * back to some default implementation which should be documented by the
 * client.
 *
 * @param groupSecurityName the name of the group
 * @param limit the maximum number of users that should be returned.
 *  This is very useful in situations where there are lot of
 *           users in the registry and getting all of them at once is not
 *           practical. A value of 0 implies get all the users and hence
 *           must be used with care.
 * @return a Result object that contains the list of users
 *   requested and a flag to indicate if more users exist.
 * @deprecated This method will be deprecated in future.
 * @exception NotImplementedException throw this exception if it is not
 *           pratical to get this information from your registry.
 * @exception EntryNotFoundException if the group does not exist in
 *           the registry
 * @exception CustomRegistryException if there is any registry specific
 *           problem
 **/
public Result getUsersForGroup(String groupSecurityName, int limit)
   throws NotImplementedException,
          EntryNotFoundException,
          CustomRegistryException {
   String s, user;
   BufferedReader in = null;
   List usrsForGroup = new ArrayList();
   int count = 0;
   int newLimit = limit+1;
   Result result = new Result();

   // As mentioned in the javadoc if the registry cannot handle a
   // large limit value it can throw the NotImplementedException.
   // For eg.
   if (limit > 50)
      throw new NotImplementedException("Limit exceeds 50");

   try {
      in = fileOpen(GROUPFILENAME);
      while ((s=in.readLine())!=null)
      {
         if (!(s.startsWith("#") || s.trim().length() <=0 )) {
             int index = s.indexOf(":");
             if ((s.substring(0,index)).equals(groupSecurityName))
             {
                StringTokenizer st = new StringTokenizer(s, ":");
                for (int i=0; i<2; i++)
                   st.nextToken();
                String subs = st.nextToken();
                StringTokenizer st1 = new StringTokenizer(subs, ",");
                while (st1.hasMoreTokens()) {
                   user = st1.nextToken();
                   usrsForGroup.add(user);
                   if (limit !=0 && ++count == newLimit) {
                      usrsForGroup.remove(user);
                      result.setHasMore();
                      break;
                   }
                }
             }
          }
       }
    } catch (Exception ex) {
       if (!isValidGroup(groupSecurityName)) {
          throw new EntryNotFoundException(groupSecurityName);
       }
       throw new CustomRegistryException(ex.getMessage());
    } finally {
       fileClose(in);
    }

    result.setList(usrsForGroup);
    return result;
 }

/**
 * Create Credential for a user. For this version of WebSphere one should
 * throw an NotImplementedException. This will be implemented internally
 * by WebSphere code and should not be implemented by the Custom Registry.
 *
 * @param     userSecurityName the name of the user.
 * @return    com.ibm.websphere.security.cred.WSCredential
 * @exception CustomRegistryException if there is any problem.
 * @exception EntryNotFoundException if the uniqueGroupId does not exist.
 * @exception CustomRegistryException if there is any registry specific
 *            problem
 **/
 public com.ibm.websphere.security.cred.WSCredential createCredential(String userSecurityName)
       throws CustomRegistryException,
              NotImplementedException,
              EntryNotFoundException {
    NotImplementedException ex =
                  new NotImplementedException("createCredential not implemented");
    throw ex;
 }

 // private methods
 private BufferedReader fileOpen(String fileName)
    throws FileNotFoundException {
    try {
       return new BufferedReader(new FileReader(fileName));
    } catch(FileNotFoundException e) {
       throw e;
    }
 }

 private void fileClose(BufferedReader in) {
    try {
       if (in != null) in.close();
```

```
      } catch(Exception e) {
         System.out.println("Error closing file" + e);
      }
   }

   private boolean match(String name, String pattern) {
      RegExpSample regexp = new RegExpSample(pattern);
      boolean matches = false;
      if(regexp.match(name))
          matches = true;
      return matches;
   }
}


//----------------------------------------------------------------------
// The program provides the Regular Expression implementation used in the
// Sample for the Custom User Registry (FileRegistrySample). The pattern
// matching in the sample uses this program to search for the pattern (for
// users and groups).
//----------------------------------------------------------------------

class RegExpSample
{

    private boolean match(String s, int i, int j, int k)
    {
        for(; k < expr.length; k++)
label0:
            {
                Object obj = expr[k];
                if(obj == STAR)
                {
                    if(++k >= expr.length)
                        return true;
                    if(expr[k] instanceof String)
                    {
                        String s1 = (String)expr[k++];
                        int l = s1.length();
                        for(; (i = s.indexOf(s1, i)) >= 0; i++)
                            if(match(s, i + l, j, k))
                                return true;

                        return false;
                    }
                    for(; i < j; i++)
                        if(match(s, i, j, k))
                            return true;

                    return false;
                }
                if(obj == ANY)
                {
                    if(++i > j)
                        return false;
                    break label0;
                }
                if(obj instanceof char[][])
                {
                    if(i >= j)
                        return false;
                    char c = s.charAt(i++);
                    char ac[][] = (char[][])obj;
                    if(ac[0] == NOT)
                    {
                        for(int j1 = 1; j1 < ac.length; j1++)
                            if(ac[j1][0] <= c && c <= ac[j1][1])
                                return false;

                        break label0;
                    }
                    for(int k1 = 0; k1 < ac.length; k1++)
                        if(ac[k1][0] <= c && c <= ac[k1][1])
                            break label0;

                    return false;
                }
                if(obj instanceof String)
                {
                    String s2 = (String)obj;
                    int i1 = s2.length();
                    if(!s.regionMatches(i, s2, 0, i1))
                        return false;
                    i += i1;
                }
            }

        return i == j;
    }

    public boolean match(String s)
    {
        return match(s, 0, s.length(), 0);
    }

    public boolean match(String s, int i, int j)
    {
        return match(s, i, j, 0);
    }

    public RegExpSample(String s)
    {
        Vector vector = new Vector();
        int i = s.length();
        StringBuffer stringbuffer = null;
        Object obj = null;
        for(int j = 0; j < i; j++)
        {
            char c = s.charAt(j);
            switch(c)
            {
            case 63: /* '?' */
                obj = ANY;
                break;

            case 42: /* '*' */
                obj = STAR;
                break;

            case 91: /* '[' */
                int k = ++j;
                Vector vector1 = new Vector();
                for(; j < i; j++)
                {
                    c = s.charAt(j);
                    if(j == k && c == '^')
                    {
                        vector1.addElement(NOT);
                        continue;
                    }
                    if(c == '\\')
                    {
                        if(j + 1 < i)
                            c = s.charAt(++j);
                    }
                    else
                    if(c == ']')
                        break;
                    char c1 = c;
                    if(j + 2 < i && s.charAt(j + 1) == '-')
                        c1 = s.charAt(j += 2);
                    char ac1[] = {
                        c, c1
                    };
                    vector1.addElement(ac1);
                }

                char ac[][] = new char[vector1.size()][];
                vector1.copyInto(ac);
                obj = ac;
                break;

            case 92: /* '\\' */
                if(j + 1 < i)
                    c = s.charAt(++j);
                break;

            }
            if(obj != null)
            {
                if(stringbuffer != null)
                {
                    vector.addElement(stringbuffer.toString());
                    stringbuffer = null;
                }
                vector.addElement(obj);
                obj = null;
            }
            else
            {
                if(stringbuffer == null)
                    stringbuffer = new StringBuffer();
                stringbuffer.append(c);
            }
        }

        if(stringbuffer != null)
            vector.addElement(stringbuffer.toString());
        expr = new Object[vector.size()];
        vector.copyInto(expr);
    }

    static final char NOT[] = new char[2];
    static final Integer ANY = new Integer(0);
    static final Integer STAR = new Integer(1);
    Object expr[];

}
```

Example: FileRegistrySample.java file (Version 5.0.2 and later):

```
import com.ibm.websphere.security.cred.*;
//
// 5639-D57, 5630-A36, 5630-A37, 5724-D18
// (C) COPYRIGHT International Business Machines Corp. 1997, 2003
// All Rights Reserved * Licensed Materials - Property of IBM
//

//----------------------------------------------------------------------
// This program may be used, executed, copied, modified and distributed
// without royalty for the purpose of developing, using, marketing, or
// distributing.
//----------------------------------------------------------------------
//

// This sample is for the Custom User Registry feature
// in WebSphere Application Server.

import java.util.*;
import java.io.*;
import java.security.cert.X509Certificate;
import com.ibm.websphere.security.*;

/**
 *  The main purpose of this sample is to demonstrate the use of the
 *  Custom Registry feature available in WebSphere Application Server.
 *  This sample is a file-based registry sample where the users and the
 *  groups information is listed in files (users.props and groups.props).
 *  As such simplicity and not the performance was a major factor behind
 *  this. This sample should be used only to get familiarized with this
 *  feature. An actual implementation of a realistic registry should
 *  consider various factors like performance, scalability, thread safety,
 *  and so on.
 **/
public class FileRegistrySample implements UserRegistry {

   private static String USERFILENAME = null;
   private static String GROUPFILENAME = null;

   /** Default Constructor **/
   public FileRegistrySample() throws java.rmi.RemoteException {
   }

  /**
   * Initializes the registry. This method is called when creating the
   * registry.
   *
   * @param     props   the registry-specific properties with which to
   *                    initialize the  custom registry
   * @exception CustomRegistryException
   *                    if there is any registry specific problem
   **/
  public void initialize(java.util.Properties props)
         throws CustomRegistryException {
      try {
          /* try getting the USERFILENAME and the GROUPFILENAME from
          * properties that are passed in (i.e from GUI).
          * These values should be set in the security center GUI in the
          * Special Custom Settings in the Custom User Registry section of
          * the Authentication panel.
          * For example:
          * usersFile   c:/temp/users.props
          * groupsFile  c:/temp/groups.props
          */
          if (props != null) {
             USERFILENAME = props.getProperty("usersFile");
             GROUPFILENAME = props.getProperty("groupsFile");
          }

      } catch(Exception ex) {
         throw new CustomRegistryException(ex.getMessage(),ex);
      }

      if (USERFILENAME == null || GROUPFILENAME == null) {
         throw new CustomRegistryException("users/groups information missing");
      }

   }
```

```
/**
 * Checks the password of the user. This method is called to authenticate a
 * user when the user's name and password are given.
 *
 * @param      userSecurityName the name of user
 * @param      password the password of the user
 * @return       a valid userSecurityName. Normally this is
 *               the name of same user whose password was checked but
 *               if the implementation wants to return any other valid
 *               userSecurityName in the registry it can do so
 * @exception CheckPasswordFailedException if userSecurityName/
 *                   password combination does not exist in the registry
 * @exception CustomRegistryException if there is any registry specific
 *                    problem
 **/
 public String checkPassword(String userSecurityName, String passwd)
    throws PasswordCheckFailedException,
           CustomRegistryException {
    String s,userName = null;
    BufferedReader in = null;

    try {
       in = fileOpen(USERFILENAME);
       while ((s=in.readLine())!=null)
       {
          if (!(s.startsWith("#") || s.trim().length() <=0 )) {
             int index = s.indexOf(":");
             int index1 = s.indexOf(":",index+1);
             // check if the userSecurityName:passwd combination exists
             if ((s.substring(0,index)).equals(userSecurityName) &&
                     s.substring(index+1,index1).equals(passwd)) {
                // Authentication successful, return the userId.
                userName = userSecurityName;
                break;
             }
          }
       }
    } catch(Exception ex) {
       throw new CustomRegistryException(ex.getMessage(),ex);
    } finally {
       fileClose(in);
    }


    if (userName == null) {
       throw new PasswordCheckFailedException("Password check failed for user: "
                                       + userSecurityName);
    }

    return userName;
 }

/**
 * Maps a X.509 format certificate to a valid user in the registry.
 * This is used to map the name in the certificate supplied by a browser
 * to a valid userSecurityName in the registry
 *
 * @param      cert the X509 certificate chain
 * @return      The mapped name of the user userSecurityName
 * @exception CertificateMapNotSupportedException if the particular
 *                    certificate is not supported.
 * @exception CertificateMapFailedException if the mapping of the
 *                    certificate fails.
 * @exception CustomRegistryException if there is any registry-specific
 *                    problem
 **/
 public String mapCertificate(X509Certificate[] cert)
    throws CertificateMapNotSupportedException,
           CertificateMapFailedException,
           CustomRegistryException {
    String name=null;
    X509Certificate cert1 = cert[0];
    try {
       // map the SubjectDN in the certificate to a userID.
       name = cert1.getSubjectDN().getName();
    } catch(Exception ex) {
       throw new CertificateMapNotSupportedException(ex.getMessage(),ex);
    }

    if(!isValidUser(name)) {
       throw new CertificateMapFailedException("user: " + name + " is not valid");
    }
    return name;
 }

/**
 * Returns the realm of the registry.
 *
 * @return       the realm. The realm is a registry-specific string indicating
 *               the realm or domain
 *               for which this registry applies.  For example, for i5/OS or AIX
 *               this would be the host name of the system whose user registry
 *               this object represents.
 *               If null is returned by this method, realm defaults to the
 *               value of "customRealm".
 * @exception CustomRegistryException if there is any registry-specific
 *                    problem
 **/
 public String getRealm()
    throws CustomRegistryException {
    String name = "customRealm";
    return name;
 }

/**
 * Gets a list of users that match a pattern in the registry.
 * The maximum number of users returned is defined by the limit
 * argument.
 * This method is called by GUI (administrative console) and scripting (command line)
 * to make the users in the registry available for adding them (users) to roles.
 *
 * @param      pattern the pattern to match. (For e.g., a* will match all
 *                    userSecurityNames starting with a)
 * @param      limit the maximum number of users that should be returned.
 *                    This is very useful in situations where there are thousands of
 *                    users in the registry and getting all of them at once is not
 *                    practical. The default is 100. A value of 0 implies get all the
 *                    users and hence must be used with care.
 * @return       a Result object that contains the list of users
 *                    requested and a flag to indicate if more users exist.
 * @exception CustomRegistryException if there is any registry-specific
 *                    problem
 **/
 public Result getUsers(String pattern, int limit)
    throws CustomRegistryException {
    String s;
    BufferedReader in = null;
    List allUsers = new ArrayList();
    Result result = new Result();
    int count = 0;
    int newLimit = limit+1;
    try {
       in = fileOpen(USERFILENAME);
       while ((s=in.readLine())!=null)
       {
          if (!(s.startsWith("#") || s.trim().length() <=0 )) {
             int index = s.indexOf(":");
             String user = s.substring(0,index);
             if (match(user,pattern)) {
                allUsers.add(user);
                if (limit !=0 && ++count == newLimit) {
                   allUsers.remove(user);
                   result.setHasMore();
                   break;
                }
             }
          }
       }
    } catch (Exception ex) {
       throw new CustomRegistryException(ex.getMessage(),ex);
    } finally {
       fileClose(in);
    }

    result.setList(allUsers);
    return result;
 }

/**
 * Returns the display name for the user specified by userSecurityName.
 *
 * This method may be called only when the user information is displayed
 * ( information purposes only, for example, in GUI) and hence not used
 * in the actual authentication or authorization purposes. If there are no
 * display names in the registry return null or empty string.
 *
 * In WebSphere Application Server 4.0 custom registry, if you had a display
 * name for the user and if it was different from the security name, the
 * display name was returned for the EJB methods getCallerPrincipal() and
 * the servlet methods getUserPrincipal() and  getRemoteUser(). In WebSphere
 * Application Server Version 5, for the same methods, the security name is
 * returned by default. This is the recommended way as the display name is
 * not unique and may create security holes.
 * However, for backward compatability if one needs the display name to
 * be returned set the property WAS_UseDisplayName to true.
 *
 *See the Infocenter documentation for more information.
 *
 * @param     userSecurityName the name of the user.
 * @return      the display name for the user. The display name
 *              is a registry-specific string that represents a descriptive, not
 *              necessarily unique, name for a user. If a display name does
 *              not exist return null or empty string.
 * @exception EntryNotFoundException if userSecurityName does not exist.
 * @exception CustomRegistryException if there is any registry specific
 *                    problem
 **/
 public String getUserDisplayName(String userSecurityName)
    throws CustomRegistryException,
           EntryNotFoundException {

    String s,displayName = null;
    BufferedReader in = null;

    if(!isValidUser(userSecurityName)) {
       EntryNotFoundException nsee = new EntryNotFoundException("user: "
           + userSecurityName
           + " is not valid");
       throw nsee;
    }
```

```
      try {
         in = fileOpen(USERFILENAME);
         while ((s=in.readLine())!=null)
         {
            if (!(s.startsWith("#") || s.trim().length() <=0 )) {
               int index = s.indexOf(":");
               int index1 = s.lastIndexOf(":");
               if ((s.substring(0,index)).equals(userSecurityName)) {
                  displayName = s.substring(index1+1);
                  break;
               }
            }
         }
      } catch(Exception ex) {
         throw new CustomRegistryException(ex.getMessage(), ex);
      } finally {
         fileClose(in);
      }

      return displayName;
   }

  /**
   * Returns the unique ID for a userSecurityName. This method is called when
   * creating a credential for a user.
   *
   * @param      userSecurityName the name of the user.
   * @return       the unique ID of the user. The unique ID for an user is
   *                  the stringified form of some unique, registry-specific, data
   *                  that serves to represent the user.  For example, for the UNIX
   *                  user registry, the unique ID for a user can be the UID.
   * @exception EntryNotFoundException if userSecurityName does not exist.
   * @exception CustomRegistryException if there is any registry-specific
   *                  problem
   **/
   public String getUniqueUserId(String userSecurityName)
      throws CustomRegistryException,
             EntryNotFoundException {

      String s,uniqueUsrId = null;
      BufferedReader in = null;
      try {
         in = fileOpen(USERFILENAME);
         while ((s=in.readLine())!=null)
         {
            if (!(s.startsWith("#") || s.trim().length() <=0 )) {
               int index = s.indexOf(":");
               int index1 = s.indexOf(":", index+1);
               if ((s.substring(0,index)).equals(userSecurityName)) {
                  int index2 = s.indexOf(":", index1+1);
                  uniqueUsrId = s.substring(index1+1,index2);
                  break;
               }
            }
         }
      } catch(Exception ex) {
         throw new CustomRegistryException(ex.getMessage(),ex);
      } finally {
         fileClose(in);
      }

      if (uniqueUsrId == null) {
         EntryNotFoundException nsee =
             new EntryNotFoundException("Cannot obtain uniqueId for user: "
                 + userSecurityName);
         throw nsee;
      }

      return uniqueUsrId;
   }

  /**
   * Returns the name for a user given its uniqueId.
   *
   * @param      uniqueUserId  the unique ID of the user.
   * @return       The userSecurityName of the user.
   * @exception  EntryNotFoundException if the unique user ID does not exist.
   * @exception  CustomRegistryException if there is any registry-specific
   *                     problem
   **/
   public String getUserSecurityName(String uniqueUserId)
      throws CustomRegistryException,
             EntryNotFoundException {
      String s,usrSecName = null;
      BufferedReader in = null;
      try {
         in = fileOpen(USERFILENAME);
         while ((s=in.readLine())!=null)
         {
            if (!(s.startsWith("#") || s.trim().length() <=0 )) {
               int index = s.indexOf(":");
               int index1 = s.indexOf(":", index+1);
               int index2 = s.indexOf(":", index1+1);
               if ((s.substring(index1+1,index2)).equals(uniqueUserId)) {
                  usrSecName = s.substring(0,index);
                  break;
               }
            }
         }
      } catch (Exception ex) {
         throw new CustomRegistryException(ex.getMessage(), ex);
      } finally {
         fileClose(in);
      }

      if (usrSecName == null) {
         EntryNotFoundException ex =
            new EntryNotFoundException("Cannot obtain the user securityName for "
             + uniqueUserId);
         throw ex;
      }

      return usrSecName;

   }

  /**
   * Determines if the userSecurityName exists in the registry
   *
   * @param      userSecurityName the name of the user
   * @return       true if the user is valid; otherwise false
   * @exception CustomRegistryException if there is any registry-specific
   *                    problem
   * @exception RemoteException as this extends java.rmi.Remote
   **/
   public boolean isValidUser(String userSecurityName)
      throws CustomRegistryException {
      String s;
      boolean isValid = false;
      BufferedReader in = null;
      try {
         in = fileOpen(USERFILENAME);
         while ((s=in.readLine())!=null)
```

```
        {
           if (!(s.startsWith("#") || s.trim().length() <=0 )) {
              int index = s.indexOf(":");
              if ((s.substring(0,index)).equals(userSecurityName)) {
                 isValid=true;
                 break;
              }
           }
        }
     } catch (Exception ex) {
        throw new CustomRegistryException(ex.getMessage(), ex);
     } finally {
        fileClose(in);
     }

     return isValid;
  }


 /**
  * Gets a list of groups that match a pattern in the registy.
  * The maximum number of groups returned is defined by the limit
  * argument.
  * This method is called by GUI (administrative console) and scripting (command line)
  * to make available the groups in the registry for adding them (groups) to roles.
  *
  * @param       pattern the pattern to match. (For example, a * will match all
  *                     groupSecurityNames starting with a)
  * @param       limit the maximum number of groups that should be returned.
  *                     This is very useful in situations where there are thousands of
  *                     groups in the registry and getting all of them at once is not
  *                     practical. The default is 100. A value of 0 implies get all the
  *                     groups and hence must be used with care.
  * @return       a Result object that contains the list of groups
  *                     requested and a flag to indicate if more groups exist.
  * @exception CustomRegistryException if there is any registry-specific
  *                     problem
  **/
  public Result getGroups(String pattern, int limit)
     throws CustomRegistryException {
     String s;
     BufferedReader in = null;
     List allGroups = new ArrayList();
     Result result = new Result();
     int count = 0;
     int newLimit = limit+1;
     try {
        in = fileOpen(GROUPFILENAME);
        while ((s=in.readLine())!=null)
        {
           if (!(s.startsWith("#") || s.trim().length() <=0 )) {
              int index = s.indexOf(":");
              String group = s.substring(0,index);
              if (match(group,pattern)) {
                 allGroups.add(group);
                 if (limit !=0 && ++count == newLimit) {
                    allGroups.remove(group);
                    result.setHasMore();
                    break;
                 }
              }
           }
        }
     } catch (Exception ex) {
        throw new CustomRegistryException(ex.getMessage(),ex);
     } finally {
        fileClose(in);
```

```
      }

      result.setList(allGroups);
      return result;
   }

  /**
   * Returns the display name for the group specified by groupSecurityName.
   * For this version of WebSphereApplication Server,  the only usage of this
   * method is by the clients (GUI and Scripting) to present a descriptive name
   * of the user if it exists.
   *
   * @param      groupSecurityName the name of the group.
   * @return     the display name for the group. The display name
   *             is a registry-specific string that represents a descriptive, not
   *             necessarily unique, name for a group. If a display name does
   *             not exist return null or empty string.
   * @exception EntryNotFoundException if groupSecurityName does not exist.
   * @exception CustomRegistryException if there is any registry-specific
   *                    problem
   **/
   public String getGroupDisplayName(String groupSecurityName)
      throws CustomRegistryException,
             EntryNotFoundException {
      String s,displayName = null;
      BufferedReader in = null;

      if(!isValidGroup(groupSecurityName)) {
         EntryNotFoundException nsee = new EntryNotFoundException("group: "
             + groupSecurityName + " is not valid");
         throw nsee;
      }

      try {
         in = fileOpen(GROUPFILENAME);
         while ((s=in.readLine())!=null)
         {
            if (!(s.startsWith("#") || s.trim().length() <=0 )) {
               int index = s.indexOf(":");
               int index1 = s.lastIndexOf(":");
               if ((s.substring(0,index)).equals(groupSecurityName)) {
                  displayName = s.substring(index1+1);
                  break;
               }
            }
         }
      } catch(Exception ex) {
         throw new CustomRegistryException(ex.getMessage(),ex);
      } finally {
         fileClose(in);
      }

      return displayName;
   }

  /**
   * Returns the Unique id for a group.

   * @param      groupSecurityName the name of the group.
   * @return       the unique ID of the group. The unique ID for
   *                    a group is the stringified form of some unique,
   *                    registry-specific, data that serves to represent the group.
   *                    For example, for the UNIX user registry, the unique ID might
   *                    be the GID.
   * @exception EntryNotFoundException if groupSecurityName does not exist.
   * @exception CustomRegistryException if there is any registry specific
   *                    problem
```

```
 * @exception RemoteException as this extends java.rmi.Remote
 **/
 public String getUniqueGroupId(String groupSecurityName)
    throws CustomRegistryException,
           EntryNotFoundException {
    String s,uniqueGrpId = null;
    BufferedReader in = null;
    try {
       in = fileOpen(GROUPFILENAME);
       while ((s=in.readLine())!=null)
       {
          if (!(s.startsWith("#") || s.trim().length() <=0 )) {
             int index = s.indexOf(":");
             int index1 = s.indexOf(":", index+1);
             if ((s.substring(0,index)).equals(groupSecurityName)) {
                uniqueGrpId = s.substring(index+1,index1);
                break;
             }
          }
       }
    } catch(Exception ex) {
       throw new CustomRegistryException(ex.getMessage(),ex);
    } finally {
       fileClose(in);
    }

    if (uniqueGrpId == null) {
       EntryNotFoundException nsee =
          new EntryNotFoundException("Cannot obtain the uniqueId for group: "
           + groupSecurityName);
       throw nsee;
    }

    return uniqueGrpId;
 }

/**
 * Returns the Unique ids for all the groups that contain the UniqueId of
 * a user. Called during creation of a user's credential.
 *
 * @param      uniqueUserId the unique ID of the user.
 * @return       a list of all the group unique IDs that the unique user ID
 *                    belongs to. The unique ID for an entry is the stringified
 *                    form of some unique, registry-specific, data that serves
 *                    to represent the entry.  For example, for the
 *                    UNIX user registry, the unique ID for a group might be the GID
 *                    and the Unique ID for the user might be the UID.
 * @exception EntryNotFoundException if uniqueUserId does not exist.
 * @exception CustomRegistryException if there is any registry-specific
 *                    problem
 **/
 public List getUniqueGroupIds(String uniqueUserId)
    throws CustomRegistryException,
           EntryNotFoundException {
    String s,uniqueGrpId = null;
    BufferedReader in = null;
    List uniqueGrpIds=new ArrayList();
    try {
       in = fileOpen(USERFILENAME);
       while ((s=in.readLine())!=null)
       {
          if (!(s.startsWith("#") || s.trim().length() <=0 )) {
             int index = s.indexOf(":");
             int index1 = s.indexOf(":", index+1);
             int index2 = s.indexOf(":", index1+1);
             if ((s.substring(index1+1,index2)).equals(uniqueUserId)) {
                int lastIndex = s.lastIndexOf(":");
                String subs = s.substring(index2+1,lastIndex);
                StringTokenizer st1 = new StringTokenizer(subs, ",");
                while (st1.hasMoreTokens())
                   uniqueGrpIds.add(st1.nextToken());
                break;
             }
          }
       }
    } catch(Exception ex) {
       throw new CustomRegistryException(ex.getMessage(),ex);
    } finally {
       fileClose(in);
    }

    return uniqueGrpIds;
 }

/**
 * Returns the name for a group given its uniqueId.
 *
 * @param     uniqueGroupId the unique ID of the group.
 * @return      the name of the group.
 * @exception EntryNotFoundException if the uniqueGroupId does not exist.
 * @exception CustomRegistryException if there is any registry specific
 *                   problem
 **/
 public String getGroupSecurityName(String uniqueGroupId)
    throws CustomRegistryException,
           EntryNotFoundException {
    String s,grpSecName = null;
    BufferedReader in = null;
    try {
       in = fileOpen(GROUPFILENAME);
       while ((s=in.readLine())!=null)
       {
          if (!(s.startsWith("#") || s.trim().length() <=0 )) {
             int index = s.indexOf(":");
             int index1 = s.indexOf(":", index+1);
             if ((s.substring(index+1,index1)).equals(uniqueGroupId)) {
                grpSecName = s.substring(0,index);
                break;
             }
          }
       }
    } catch (Exception ex) {
       throw new CustomRegistryException(ex.getMessage(),ex);
    } finally {
       fileClose(in);
    }

    if (grpSecName == null) {
       EntryNotFoundException ex =
          new EntryNotFoundException("Cannot obtain the group security name for: "
           + uniqueGroupId);
       throw ex;
    }

    return grpSecName;

 }

/**
 * Determines if the groupSecurityName exists in the registry
 *
 * @param      groupSecurityName the name of the group
 * @return       true if the groups exists; otherwise false
 * @exception CustomRegistryException if there is any registry-specific
 *                    problem
 **/
 public boolean isValidGroup(String groupSecurityName)
    throws CustomRegistryException {
    String s;
    boolean isValid = false;
    BufferedReader in = null;
    try {
       in = fileOpen(GROUPFILENAME);
       while ((s=in.readLine())!=null)
       {
          if (!(s.startsWith("#") || s.trim().length() <=0 )) {
             int index = s.indexOf(":");
             if ((s.substring(0,index)).equals(groupSecurityName)) {
                isValid=true;
                break;
             }
          }
       }
    } catch (Exception ex) {
       throw new CustomRegistryException(ex.getMessage(),ex);
    } finally {
       fileClose(in);
    }

    return isValid;
 }

/**
 * Returns the securityNames of all the groups that contain the user
 *
 * This method is called by GUI (administrative console) and scripting (command line)
 * to verify the user entered for RunAsRole mapping belongs to that role
 * in the roles to user mapping. Initially, the check is done to see if
 * the role contains the user. If the role does not contain the user
 * explicitly, this method is called to get the groups that this user
 * belongs to so that check can be made on the groups that the role contains.
 *
 * @param      userSecurityName the name of the user
 * @return       a list of all the group securityNames that the user
 *                    belongs to.
 * @exception EntryNotFoundException if user does not exist.
 * @exception CustomRegistryException if there is any registry-specific
 *                    problem
 * @exception RemoteException as this extends java.rmi.Remote
 **/
 public List getGroupsForUser(String userName)
    throws CustomRegistryException,
           EntryNotFoundException {
    String s;
    List grpsForUser = new ArrayList();
    BufferedReader in = null;
    try {
       in = fileOpen(GROUPFILENAME);
       while ((s=in.readLine())!=null)
       {
          if (!(s.startsWith("#") || s.trim().length() <=0 )) {
             StringTokenizer st = new StringTokenizer(s, ":");
             for (int i=0; i<2; i++)
                st.nextToken();
             String subs = st.nextToken();
             StringTokenizer st1 = new StringTokenizer(subs, ",");
             while (st1.hasMoreTokens()) {
                if((st1.nextToken()).equals(userName)) {
                   int index = s.indexOf(":");
                   grpsForUser.add(s.substring(0,index));
                }
```

```
            }
         }
      }
   } catch (Exception ex) {
      if (!isValidUser(userName)) {
         throw new EntryNotFoundException("user: " + userName  + " is not valid");
      }
      throw new CustomRegistryException(ex.getMessage(),ex);
   } finally {
      fileClose(in);
   }

   return grpsForUser;
}

/**
 * Gets a list of users in a group.
 *
 * The maximum number of users returned is defined by the limit argument.
 *
 * In rare situations if you are working with a registry where getting all
 * the users from any of your groups is not practical (for example if there
 * are a large number of users) you can throw the NotImplementedException
 * for that particualar group(s). If there is no concern about
 * returning the users from groups in the registry it is recommended that
 * this method be implemented without throwing the NotImplemented exception.
 *
 * @param         groupSecurityName the name of the group
 * @param         limit the maximum number of users that should be returned.
 *                This is very useful in situations where there are lot of
 *                users in the registry and getting all of them at once is not
 *                practical. A value of 0 implies get all the users and hence
 *                must be used with care.
 * @return          a Result object that contains the list of users
 *                  requested and a flag to indicate if more users exist.
 * @deprecated  This method will be deprecated in future.
 * @exception     NotImplementedException throw this exception in rare situations
 *                if it is not pratical to get this information for any of the
 *                group or groups from the registry.
 * @exception     EntryNotFoundException if the group does not exist in
 *                        the registry
 * @exception     CustomRegistryException if there is any registry-specific
 *                        problem
 **/
public Result getUsersForGroup(String groupSecurityName, int limit)
   throws NotImplementedException,
          EntryNotFoundException,
          CustomRegistryException {
   String s, user;
   BufferedReader in = null;
   List usrsForGroup = new ArrayList();
   int count = 0;
   int newLimit = limit+1;
   Result result = new Result();

   try {
      in = fileOpen(GROUPFILENAME);
      while ((s=in.readLine())!=null)
      {
         if (!(s.startsWith("#") || s.trim().length() <=0 )) {
            int index = s.indexOf(":");
            if ((s.substring(0,index)).equals(groupSecurityName))
            {
               StringTokenizer st = new StringTokenizer(s, ":");
               for (int i=0; i<2; i++)
                  st.nextToken();
               String subs = st.nextToken();
```

```
                StringTokenizer st1 = new StringTokenizer(subs, ",");
                while (st1.hasMoreTokens()) {
                   user = st1.nextToken();
                   usrsForGroup.add(user);
                   if (limit !=0 && ++count == newLimit) {
                      usrsForGroup.remove(user);
                      result.setHasMore();
                      break;
                   }
                }
             }
          }
       }
    } catch (Exception ex) {
       if (!isValidGroup(groupSecurityName)) {
          throw new EntryNotFoundException("group: "
           + groupSecurityName
           + " is not valid");
       }
       throw new CustomRegistryException(ex.getMessage(),ex);
    } finally {
       fileClose(in);
    }

    result.setList(usrsForGroup);
    return result;
 }

/**
 * This method is implemented internally by the WebSphere Application Server code
 * in this release. This method is not called for the Custom Registry
 * implementations for this release. Return null in the implementation.
 *
 **/
 public WSCredential createCredential(String userSecurityName)
       throws CustomRegistryException,
              NotImplementedException,
              EntryNotFoundException {

    // This method is not called.
    return null;
 }

 // private methods
 private BufferedReader fileOpen(String fileName)
    throws FileNotFoundException {
    try {
       return new BufferedReader(new FileReader(fileName));
    } catch(FileNotFoundException e) {
       throw e;
    }
 }

 private void fileClose(BufferedReader in) {
    try {
       if (in != null) in.close();
    } catch(Exception e) {
       System.out.println("Error closing file" + e);
    }
 }

 private boolean match(String name, String pattern) {
    RegExpSample regexp = new RegExpSample(pattern);
    boolean matches = false;
    if(regexp.match(name))
        matches = true;
    return matches;
```

```
   }
}


//----------------------------------------------------------------------
// The program provides the Regular Expression implementation used in the
// Sample for the Custom User Registry (FileRegistrySample). The pattern
// matching in the sample uses this program to search for the pattern (for
// users and groups).
//----------------------------------------------------------------------

class RegExpSample
{

    private boolean match(String s, int i, int j, int k)
    {
        for(; k < expr.length; k++)
label0:
            {
                Object obj = expr[k];
                if(obj == STAR)
                {
                    if(++k >= expr.length)
                        return true;
                    if(expr[k] instanceof String)
                    {
                        String s1 = (String)expr[k++];
                        int l = s1.length();
                        for(; (i = s.indexOf(s1, i)) >= 0; i++)
                            if(match(s, i + l, j, k))
                                return true;

                        return false;
                    }
                    for(; i < j; i++)
                        if(match(s, i, j, k))
                            return true;

                    return false;
                }
                if(obj == ANY)
                {
                    if(++i > j)
                        return false;
                    break label0;
                }
                if(obj instanceof char[][])
                {
                    if(i >= j)
                        return false;
                    char c = s.charAt(i++);
                    char ac[][] = (char[][])obj;
                    if(ac[0] == NOT)
                    {
                        for(int j1 = 1; j1 < ac.length; j1++)
                            if(ac[j1][0] <= c && c <= ac[j1][1])
                                return false;

                        break label0;
                    }
                    for(int k1 = 0; k1 < ac.length; k1++)
                        if(ac[k1][0] <= c && c <= ac[k1][1])
                            break label0;

                    return false;
                }
                if(obj instanceof String)
                {
                    String s2 = (String)obj;
                    int i1 = s2.length();
                    if(!s.regionMatches(i, s2, 0, i1))
                        return false;
                    i += i1;
                }
            }

        return i == j;
    }

    public boolean match(String s)
    {
        return match(s, 0, s.length(), 0);
    }

    public boolean match(String s, int i, int j)
    {
        return match(s, i, j, 0);
    }

    public RegExpSample(String s)
    {
        Vector vector = new Vector();
        int i = s.length();
        StringBuffer stringbuffer = null;
        Object obj = null;
        for(int j = 0; j < i; j++)
        {
            char c = s.charAt(j);
            switch(c)
            {
            case 63: /* '?' */
                obj = ANY;
                break;

            case 42: /* '*' */
                obj = STAR;
                break;

            case 91: /* '[' */
                int k = ++j;
                Vector vector1 = new Vector();
                for(; j < i; j++)
                {
                    c = s.charAt(j);
                    if(j == k && c == '^')
                    {
                        vector1.addElement(NOT);
                        continue;
                    }
                    if(c == '\\')
                    {
                        if(j + 1 < i)
                            c = s.charAt(++j);
                    }
                    else
                    if(c == ']')
                        break;
                    char c1 = c;
                    if(j + 2 < i && s.charAt(j + 1) == '-')
                        c1 = s.charAt(j += 2);
                    char ac1[] = {
                        c, c1
                    };
                    vector1.addElement(ac1);
                }

                char ac[][] = new char[vector1.size()][];
                vector1.copyInto(ac);
                obj = ac;
                break;

            case 92: /* '\\' */
                if(j + 1 < i)
                    c = s.charAt(++j);
                break;

            }
            if(obj != null)
            {
                if(stringbuffer != null)
                {
                    vector.addElement(stringbuffer.toString());
                    stringbuffer = null;
                }
                vector.addElement(obj);
                obj = null;
            }
            else
            {
                if(stringbuffer == null)
                    stringbuffer = new StringBuffer();
                stringbuffer.append(c);
            }
        }

        if(stringbuffer != null)
            vector.addElement(stringbuffer.toString());
        expr = new Object[vector.size()];
        vector.copyInto(expr);
    }

    static final char NOT[] = new char[2];
    static final Integer ANY = new Integer(0);
    static final Integer STAR = new Integer(1);
    Object expr[];

}
```

Example: Groups.props file:

```
# 5722-IWE
# (C) COPYRIGHT International Business Machines Corp. 1997, 2003
# All Rights Reserved * Licensed Materials - Property of IBM
#
# Format:
# name:gid:users:display name
# where name   = groupId of the group
#       gid    = uniqueId of the group
#       users  = list of all the userIds that the group contains
#       display name = a (optional) display name for the group.
admins:567:bob:Administrative group
operators:678:jay,ted,dave:Operators group
users:789:jay,jeff,vikas,bobby:
```

Example: Users.props file:

```
# 5722-IWE
# (C) COPYRIGHT International Business Machines Corp. 1997, 2003
# All Rights Reserved * Licensed Materials - Property of IBM
#
# Format:
# name:passwd:uid:gids:display name
# where name   = userId/userName of the user
#       passwd = password of the user
#       uid    = uniqueId of the user
#       gid    = groupIds of the groups that the user belongs to
#       display name = a (optional) display name for the user.
bob:bob1:123:567:bob
dave:dave1:234:678:
jay:jay1:345:678,789:Jay-Jay
ted:ted1:456:678:Teddy G
jeff:jeff1:222:789:Jeff
vikas:vikas1:333:789:vikas
bobby:bobby1:444:789:
```

Example: Results.java file:

```
//  5722-IWE
// (C) COPYRIGHT International Business Machines Corp. 1997, 2003
//  All Rights Reserved * Licensed Materials - Property of IBM
//
//  DESCRIPTION:
//
//    This module is used by User Registries in WebSphere when calling the
//    getUsers and getGroups method. The user registries should use this
//    to set the list of users/groups and to indicate if there are more
//    users/groups in the registry than requested
//
package com.ibm.websphere.security;

import java.util.List;

/**
    This module is used by User Registries in WebSphere when calling the
    getUsers and getGroups method. The user registries should use this
    to set the list of users/groups and to indicate if there are more
    users/groups in the registry than requested
 */

public class Result implements java.io.Serializable {
    /**
      Default constructor
    */
    public Result() {
    }

    /**
       Returns the list of users/groups
       @return the list of users/groups
    */
    public List getList() {
      return list;
    }

    /**
       indicates if there are more users/groups in the registry
    */
    public boolean hasMore() {
      return more;
    }
    /**
       Set the flag to indicate that there are more users/groups
       in the registry to true
    */
    public void setHasMore() {
      more = true;
    }

    /*
```

```
        Set the list of user/groups
        @param list   list of users/groups
      */
      public void setList(List list) {
        this.list = list;
      }

      private boolean more = false;
      private List list;
}
```

Develop a custom interceptor for trust associations

If you are using a third-party reverse proxy server other than Tivoli WebSeal Version 3.6, you must provide an implementation class for the WebSphere Application Server - Express interceptor interface for your proxy server. This topic describes the interface you must implement.

1. Define the interceptor class method.
   WebSphere Application Server - Express provides the interceptor Java interface, com.ibm.websphere.security.TrustAssociationInterceptor

   which defines these methods:

   - **isTargetInterceptor(HttpServletRequest req)**
     The isTargetInterceptor() method determines if the request originated with the proxy server that is associated with the interceptor. The implementation code examines the incoming request object and determines if the proxy server that is forwarding the request is a valid proxy server for this interceptor. The result of this method determines if the interceptor processes the request or not.
   - **validateEstablishedTrust(HttpServletRequest req)**
     The validateEstablishedTrust() method determines if the proxy server from which the request originated is trusted or not. This method is called after the isTargetInterceptor() method. The implementation code must authenticate the proxy server. The authentication mechanism is specific to the proxy server. For example, in the WebSphere implementation that is provided for the WebSeal server, this method retrieves the basic authentication information from the HTTP header and validates the information against the user registry that is used by WebSphere Application Server - Express. If the credentials are invalid, the method throws the WebTrustAssociationException exception, which indicates that the proxy server is not trusted and the request is to be denied.
   - **getAuthenticatedUsername(HttpServletRequest req)**
     The getAuthenticatedUsername method is called after trust has been established between the proxy server and WebSphere Application Server - Express. WebSphere Application Server - Express has accepted the proxy server's authentication of the request and must now authorize the request. To authorize the request, the name of the original requester must be subjected to an authorization policy to determine if the requester has the necessary authorities. The implementation code for this method must extract the user name from the HTTP request header and determine if that user is entitled to the requested resource. For example, in the WebSphere-provided implementation for the WebSeal server, the method looks for an iv-user attribute in the HTTP request header and extracts the user ID associated with it for authorization.
2. Set properties in the trustedservers.properties file.
   To be configurable, the interceptor must extend com.ibm.websphere.security.WebSphereBaseTrustAssociationInterceptor. Implement these methods:

   - **init(java.util.Properties props)**
     The init(Properties) method accepts a java.util.Properties object, which contains the set of properties required to initialize the interceptor. All the properties set for an interceptor (by using the **Custom Properties** link for that interceptor or using scripting) is sent to this method. The interceptor then can use these properties to initialize itself. For example, in the product implementation for the WebSEAL server, this method reads the hosts and ports so that a request coming in can be verified

to originate from trusted hosts and ports. A return value of `0` implies that the interceptor initialization is successful. Any other value implies that the initialization is not successful and the interceptor is ignored.

Version 5.0.2 or later: If a previous implementation of the trust association interceptor returns a different error status you can either change your implementation to match the expectations or make one of the following changes:

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

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

- **cleanup()**
  The cleanup method is called when the application server is stopped. It can be used to prepare the interceptor for termination.
- **setVersion (String s)**
  The setVersion method is optional. It sets the version and is for informational purposes only. The default value is `Unspecified`.

To make your custom interceptor configurable, perform these steps:

a. Configure these methods implemented by the custom interceptor implementation.

Note: This list only shows the methods and does not include any implementation.

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

public class myTAIImpl extends WebSphereBaseTrustAssociationInterceptor
   implements TrustAssociationInterceptor {

  public myTAIImpl () {
  }

  public boolean isTargetInterceptor (HttpServletRequest req)
     throws WebTrustAssociationException {
    // Return true if this is the target interceptor, else return false.
  }

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

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

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

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

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

   Note: If your custom interceptor implements the init(Properties) method as described previously, you do not need to call the init(String). You can continue to the next step.

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

   Change your implementation to implement the init(Properties) method instead of the init(String propsfile) method. As shown in the previous example, this default implementation reads the properties to load the property file. To determine the properties file that it must read, the com.ibm.websphere.security.trustassociation.types property appends `.config` to its value.

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

   - The same property names used in the previous release are used to obtain the file name. The file name is obtained by concatenating the `.config` extension to the com.ibm.websphere.security.trustassociation.types property value. If the file name is called myTAI.properties and is located in the /QIBM/UserData/WebASE/ASE5/myInstance/properties directory, set the following properties:

```
com.ibm.websphere.security.trustassociation.types = myTAItype
com.ibm.websphere.security.trustassociation.myTAItype.config =
   /QIBM/UserData/WebASE/ASE5/myInstance/properties/myTAI.properties
```

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

```
com.ibm.websphere.security.trustassociation.initPropsFile=
  /QIBM/UserData/WebASE/ASE5/myInstance/properties/myTAI.properties
```

   Note: These properties have been wrapped for display purposes. Type them as one continuous line.

c. Compile the implementation. Make sure that wssec.jar and j2ee.jar are in the compiler class path. These JAR files are located in the /QIBM/ProdData/WebASE/ASE5/lib directory.

d. Create a `classes` subdirectory in the /QIBM/UserData/WebASE/ASE5 directory. For example, /QIBM/UserData/WebASE/Base/classes.

e. Grant read, write, and execute authority (*RWX) to the QEJBSVR user profile. You can use the i5/OS Change Authority (CHGAUT) command, for example:

```
CHGAUT OBJ('/QIBM/UserData/WebASE/ASE5/classes') USER(QEJBSVR) DTAAUT(*RWX)
```

f. Restart the server.
g. Use the administrative console to delete the default WebSEAL interceptor and add your custom interceptor. Verify that the class name is dot separated and appears in the class path. For instructions on configuring trust association interceptors, see "Configure a trust association interceptor" on page 113.
h. Click **Custom Properties** to add additional properties that are required to initialize the custom interceptor.
i. Save the configuration.
j. Stop and restart the application server.

Example: Trust association interceptor: If you are using a third party reverse proxy server other than Tivoli WebSEAL product, you must provide an implementation class for the trust association interceptor interface for your proxy server.

Using the TrustAssociation Interceptor interface

WebSphere Application Server - Express provides the interceptor Java interface, com.ibm.websphere.security.TrustAssociationInterceptor

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

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

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

After the interceptor class has been created, WebSphere Application Server - Express must be configured to use it by providing the following properties.

Perform these steps in the WebSphere administrative console:

1. Start the WebSphere administrative console.
2. Expand **Security —> Authentication Mechanisms**, and click **LTPA**.
3. Click **Trust Association**.
4. Check **Enable Trust Association**, and then click **Interceptors**.
5. If you are using WebSEAL Interceptor, check **com.ibm.ws.security.web.WebSEALTrustAssociationInterceptor**. Otherwise, click **New** to add your interceptors. WebSphere Application Server - Express uses the classpath on the server to look for the implementation class. For each interceptor that you add, click **Additional Properties** to enter the property name and value pairs. Check **Required** for each property.

The properties and values for intercepters are these (where *proxy* is the name of your proxy server):

- **com.ibm.websphere.security.trustassociation.types**
  Establish a name for your proxy. For example, if you call your proxy server `myProxy`, then set the property as follows: `com.ibm.websphere.security.trustassociation.types=myproxy`.
- **com.ibm.websphere.security.*proxy*.hostnames**
  The value for this property is the hostname of the machine where the proxy server is running.
- **com.ibm.websphere.security.*proxy*.loginId**
  The value of this property is the proxy server ID.
- **com.ibm.websphere.security.*proxy*.Id**
  The value of this property is a special header field that is sent by the proxy server with the request to WebSphere Application Server - Express.
- **com.ibm.websphere.security.*proxy*.ports**
  This is the port where your proxy server receives the user requests. You can have a different port number depending on your proxy server's configuration.

Specific name and value pairs for WebSEAL are as follows:

- com.ibm.websphere.security.trustassociation.types=webseal
- com.ibm.websphere.security.webseal.id=iv-user
- com.ibm.websphere.security.webseal.ports=443

Note: You may have a different port number depending on your WebSEAL configuration.

Assemble secured applications

Assembly is the process of packaging an application for installation (deployment) into the application server run time. As part of preparing a secured application for deployment, you specify various security settings in the deployment descriptors for your WAR or EAR file.

To assemble your secured application, perform these steps:

1. "Edit the web.xml file to add security settings" on page 80.
2. (Optional) "Add the was.policy file to applications" on page 81.
3. Export your application as a WAR or EAR file.

After you secure an application, the resulting WAR or EAR file contains security information in its deployment descriptors.

Here are the WAR file deployment descriptors that contain security settings:

- **web.xml**
  Deployment descriptor for the Web module, including security information.
- **was.policy**
  Contains the Java 2 Security permissions that are granted for the application to access secured system resources.

An application EAR file contains the following deployment descriptors, in addition to the deployment descriptors for the WAR file:

- **application.xml**
  Contains all of the roles that are used in the application.
- **ibm-application-ext.xmi**
  Contains roles mapping for users and groups.

Edit the web.xml file to add security settings

There are three types Web login (authentication) mechanisms that can be configured on a Web application: basic authentication, form-based authentication, and client certificate-based authentication. Web resources in a Web application can be protected by assigning security roles to those resources. So, you need to know in advance which Web resources need protecting and how to protect them.

Perform these steps to secure your Web application with WebSphere Development Studio Client:

1. **Open the web.xml file.**
   a. Open the project that contains your Web application.
   b. In the Navigator window, expand the WEB-INF directory. Double-click the web.xml file. The web.xml file is the deployment descriptor for your Web application. The deployment descriptor contains runtime security settings.
2. **Create security roles.**
   a. Click the **Security** tab.
   b. Next to the Security Roles table, click **Add**.
   c. Type the role name and a description of the role.
   d. Repeat steps b and c to add all required security roles.
3. **Create security constraints.**
   Security constraints are a mapping of one or more Web resources to a set of roles. To create security constraints, follow these steps:
   a. Under the Security Constraints window, click **Add**. A new security constraint is added to the window, and a new Web resource collection is added to the Web resource collection window.
   b. Select **(New Web Resource Collection)**, and click **Edit**.
   c. In the Web Resource Collections dialog, enter a name and description for the Web resource collection.
   d. Select the appropriate HTTP methods.
   e. Next to **URL Patterns**, click **Add**. Type the URL pattern (for example: `/*`, `*.jsp`, `/hello`). Consult the Servlet 2.3 Specification for more information about mapping URL patterns to servlets. The security run time uses the first exact match to map the incoming URL with URL patterns. If the exact match is not present, the security run time uses the longest match. The wild card URL pattern (such as `*.*` or `*.jsp`) is used last. Click **OK**.
   f. Next to the Authorized roles window, click **Edit**. Type a description, and select the required Role Names. Note that if you do not select any role names, no user can have access to the Web resources that are specified under these security constraints. Click **OK**.
   g. Select the appropriate user data constraint from the drop-down list. A user data constraint of **None** indicates that the communication between the Web client (browser) and the server (Web server) is transported over HTTP. A user data constraint of **Confidential** or **Integral** guarantees that the communication between the Web client and the Web server is secured and is transported over HTTPS.
   h. Repeat the previous steps to create multiple security constraints.
4. **Map security-role-ref and role-name to the role-link.**
   During development of the Web application, you can create the security-role-ref element using development tools such as WebSphere Development Studio Client. The security-role-ref element contains only the role-name field at this stage. The role-name field contains the name of the role that

is referenced in the servlet or JSP code to determine if the caller is in a specified role (if the isUserInRole() method returns true). Because security roles are created during the assembly stage, the developer uses a logical role name in the role-name field and provides enough description in the description field for the assembler to map the role actual (role-link). The Security-role-ref element is at the servlet level. A servlet or JSP file can have zero or more security-role-ref elements.

To map the elements to role-link, perform these steps:

a. Click the **Source** tab.
b. In the Outline window, expand the servlet to which you want to update the security-role-ref element.
c. Double-click the security-role-ref element.
d. In the source editor, type the appropriate values for the role-link element.
e. Repeat these steps until all servlets have the required role-link element defined.

5. **Configure the login mechanism.**
   The configured login mechanism applies to all the servlets, JSP files, and HTML resources in the Web module.

   Click the **Pages** tab, and in the Login section, specify these settings:
   - **Realm name:** Specify the realm name to use for HTTP Basic authorization. The realm is defined on the server. It is a way to divide and limit the authorization of users on a HTTP server.
   - **Authentication method:** Specifies the authentication mechanism for the Web application. As a prerequisite for gaining access to any Web resources that are protected by an authorization constraint, a user must be authenticated using the configured mechanism. Available authentication mechanisms include Basic, Digest, Form, and client certificate (Client-Cert). If you select Client-Cert, install the client certificate in the Web client (browser) and place the client certificate in the server trust keyring file.
   - (Optional) **Login page:** Browse to select a login page within the WAR file. This value is only specified for the form-based authentication method.
   - (Optional) **Error page:** Browse to select an error page within the WAR. This value is only specified for the form-based authentication method.
6. **Save the web.xml file.**

Add the was.policy file to applications

When Java 2 security is enabled for a WebSphere Application Server, all the applications that run on that WebSphere Application Server undergo a security check before accessing any system resource. An application might need a was.policy if it accesses resources that require more permissions than have been granted in the default app.policy file. By default, the product security reads an app.policy file that is located in each instance and grants the permissions in the app.policy file to all the applications in the instance. Any additional required permissions should be added in the was.policy file. The was.policy file is only required if an application requires additional permissions.

In the was.policy file, specify `codeBase` of `${application}` and add required permissions to grant additional permissions to the entire application. Similarly, use `codeBase` of `${webComponent}` to grant additional permissions to all the Web modules in the application. You can assign additional permissions to each module (WAR file) as shown in the following example.

Here is an example of the was.policy file that add extra permissions for an application:

```
// grant additional permissions to a WebModule
grant codeBase " file:aWebModule.war" {
  permission java.security.SecurityPermission "printIdentity";
};
```

To create a was.policy file for your application, perform these steps:

1. Create a was.policy file using the Java policy tool. The policytool is located in the `bin` subdirectory of your workstation's Java development kit or runtime environment installation. For example, on a Windows workstation, run C:\jdk1.3.1_02\jre\bin\policytool.exe.
2. Add the required permissions in the was.policy file using the policy tool.
3. Start WebSphere Development Studio Client.
4. Open your Web project.
5. In the Navigator window, select the META-INF directory.
6. Select **File —> Import**.
7. In the Import wizard, select **File system**, and click **Next**.
8. Click **Browse** and select the directory which contains the was.policy file that you want to add to the application. Click **OK**.
9. In the left pane, select the directory. In the right pane, select the was.policy file.
10. Click **Finish**.

When you export your project to a WAR or EAR file and install (deploy) it into the application server run time, your application is ready to run when Java 2 security is enabled.

Troubleshooting

Adding the was.policy file to your assembled application is required for applications to run properly when Java 2 security is enabled. If the was.policy file is not created and it does not contain required permissions, the application might not be able to access system resources.

The symptom of the missing permission problem is the application program getting the exception, java.security.AccessControlException. The missing permission is listed in the exception data, for example:

```
java.security.AccessControlException: access denied
(java.io.FilePermission /QIBM/ProdData/WebASE/ASE5/java/ext/mail.jar read)
```

When an application program receives this exception and adding this permission is justified, add a permission to the was.policy file, for example:

```
grant codeBase "file:${application}" {
  permission java.io.FilePermission
    "/QIBM/ProdData/WebASE/ASE5/java/ext/mail.jar", "read";
};
```

Deploy secured applications

Deploying applications that have security constraints (secured applications) is not much different than deploying applications without any security constraints. The only difference is that you may need to assign users and groups to roles for a secured application, which requires that you have the correct active registry.

If you are installing a secured application, roles are defined in the application. You then assign users and groups to these defined roles.

These steps are common for both installing an application and modifying an existing application. If the application contains roles, the application installation wizard in the WebSphere administrative console prompts you to map security roles to users and groups. (You can also perform this step when you manage installed applications.)

To install (or deploy) your secured application, perform these steps in the WebSphere administrative console:

1. Click **Applications —> Install New Application**.

2. Complete the non-security related steps prior to the step entitled **Map security roles to users and groups**.
3. Map security roles to users and groups. See “Assign users and groups to roles” for more information.
4. Complete the remaining steps to finish installing and deploying the application. If you are updating a previously installed application, stop the application and start it.

After a secured application is deployed, make sure you can access the resources in the application with the correct credentials. For example, if your application has a protected Web module, make sure you only use the users listed in the roles for that Web resource to access.

Updating and redeploying secured applications

After an application is deployed, you can use the administrative console to modify the existing users and groups mapping to roles.

After you complete the changes, make sure to save the changes. Stop and start the application for the changes to become effective.

If you need to update any other security related information, use your development tool (such as WebSphere Studio) to modify roles, method permissions, auth-constraints, data-constraints, and other security-related information. After the changes are done, save the EAR file, uninstall the old application, deploy the modified application, and start the application to make the changes effective. If information about roles is modified make sure you update the user and group information using the administrative console.

Assign users and groups to roles

This topic assumes that all the roles are already created in your application. Also, you need to make sure that the user registry used is the current or active user registry. It is preferable to have the security turned on with the user registry of your choice before you begin this process. Make sure that if you have changed anything in the security configuration (for example, enabled security or changed user registry) save the configuration and restart the server before the changes become effective.

Because the default active registry is LocalOS it is not necessary (though it is recommended) to enable security if you want to use the LocalOS registry as your registry to assign users and groups to roles. You can enable security after the users and groups are assigned in this case. The advantage of enabling security with the appropriate registry before proceeding with this task is that you can make sure you have a valid security setup (which includes checking the user registry configuration), and you can avoid problems with using the registry.

These steps are common for both installing an application and modifying an existing application. If the application contains roles, you see the **Map security roles to users/groups** link during installation application (as one of the steps) and also during application management.

To assign users and groups to security roles, perform these steps in the administrative console:

1. During the application installation process, click **Map security roles to users/groups**. All roles that belong to the application are listed. If the roles are already assigned to users or special subjects (such as All Authenticated and Everyone), they are listed here.
2. To assign the special subjects, select **Everyone** or **All Authenticated** for the appropriate roles.
3. To assign users or groups, select the role (multiple roles can be selected at the same time if the same users or groups are assigned to all the roles), and click **Lookup Users** or **Lookup groups**.
4. Get the appropriate users and groups from the registry by filling in the **limit (number of items)** and the **Search String** fields and then clicking **Search**.

The limit field limits the number of users that are obtained and displayed from the registry. The pattern is a searchable pattern that matches one or more users or groups. For example, `user*` lists users such as user1 and user2. A pattern of `*` indicates all users or groups. Use the limit and the search strings cautiously so as not to overwhelm the registry. When you use large registries (such as LDAP) where thousands of user and group information resides, a search for a large number of users or groups can make the system very slow, and the system may even fail.

A message appears at the top of the panel when a search results in more entries than you requested. You can refine your search (limit or the search pattern) until you have the required list.

5. From the **Available** box, select the users and groups that should be assigned to the role, and click **>>** to add them to the role.
6. To remove existing users or groups, select them from the **Selected** box, and click **<<** to remove them.
7. Click **OK**.

The users and groups information is added to the binding file in the application. The binding file is later used for authorization purposes.

Configure WebSphere security

See these topics for information about configuring security in WebSphere Application Server - Express:

"Configure global security"
See this topic for information about configuring the WebSphere global security.

"Assign users to administrative roles" on page 120
WebSphere global security is role-based, which means that users and groups must be assigned roles that allow them to access certain resources. See this topic for information about administrative roles.

"Assign users to naming roles" on page 122
See this topic for information about assigning users and groups to roles for the naming server.

"Configure SSL in WebSphere Application Server - Express" on page 123
If you want to use secure sockets layer (SSL) with WebSphere Application Server - Express, see this topic.

"Configure Java 2 security" on page 154
WebSphere Application Server - Express now fully supports the Java 2 security model. See this topic for information about configuring the permissions-based Java 2 security.

"Configure Java Authentication and Authorization Service login" on page 170
Java Authentication and Authorization Service (JAAS) login in new in WebSphere Application Server - Express. See this topic for information about configuring the login authentication.

"Configure J2C authentication data entries" on page 172
A Java 2 Connector (J2C) authentication data entry defines authentication data that you can share among resource adapters and JDBC data sources. See this topic for more information.

Configure global security

It is helpful to understand security from an infrastructure standpoint so that you know the advantages of different authentication mechanisms, user registries, authentication protocols, and so on. Picking the right security components to meet your needs is a part of configuring global security. For more information, see "Global security" on page 85.

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

Perform these steps:

1. **Start the WebSphere administrative console.**
   If security is currently disabled, log in with any user ID. If security is currently enabled, log in with a predefined administrative ID and password (this is typically the server user ID specified when you configured the user registry).

   In left navigation menu of the administrative console, click **Security**.
2. **"Configure a user registry"**
   WebSphere security requires the a user registry, which is used to authenticate users to protected resources.
3. **"Configure the authentication mechanism" on page 97**
   Configure the mechanism the WebSphere Application Server - Express uses to authenticate users.
4. (Optional) **"Configure single signon" on page 101**
   If you configured LTPA as your authentication mechanism and your applications contain form-based login, you may want to configure single signon.
5. (Optional) **"Configure a trust association interceptor" on page 113**
   If you are using a third-party reverse-proxy server (such as WebSEAL) in your topology, see this topic for more information.
6. (Optional) **"Change the default SSL keystore and truststore files" on page 118**
   WebSphere Application Server - Express ships with default SSL keystore and truststore files that should not be used in a production environment, although they can be used in test environments. See this topic for more information.
7. **"Enable global security" on page 119**
   After you have completed configuring your security settings, enable global security

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

Configuration of global security for a security domain consists of configuring the common user registry, the authentication mechanism, and other security information, which defines the behavior of a security domain. The other security information which you can configure includes Java 2 Security Manager, Java Authentication and Authorization Service (JAAS), Java 2 Connector authentication data entries, CSIv2 or SAS authentication protocol (RMI over IIOP security), and other miscellaneous attributes. The global security configuration usually applies to every server within the security domain.

Configure a user registry: Though different types of user registries are supported, only one active registry can be used by all of the processes in WebSphere Application Server - Express. Configuring the correct registry is a prerequiste to assigning users and groups to roles for applications. LocalOS is the default registry. However, you still need to configure the registry as the first step in enabling global security, after which you restart the servers, and then assign users and groups to roles for all your applications. For more information, see "Assign users and groups to roles" on page 83.

If you select a different user registry after users and groups are assigned to roles for your applications, it is recommended that you delete all the users and groups from the applications and reassign them after you change the registry. Deleting all the users and groups can be done through the administrative console or through wsadmin scripting.

This wsadmin command removes all the users and groups from any application:

```
$AdminApp deleteUserAndGroupEntries yourAppName
```

where *yourAppName* is the name of the application. Backing up the old application is advised before performing this operation.

If all of the user and group names are the same in both the registries and if the application bindings file does not contain the accessIDs (which are unique for each registry, even for the same user or group name), you may be able to change the registries without having to delete the users and groups information. By default, an application does not contain accessIDs in the bindings file (they are generated on the fly when the applications are started). However, if you have migrated an existing application from an earlier release or if you used the wsadmin script to add accessIDs for the applications (to improve performance), you must remove the existing user and group information and add them after you configure the new registry.

The administrative user ID is common to all user registries. The administrative ID is a member of the chosen user registry, and it has special privileges in WebSphere Application Server - Express. However, it has no special privileges in the user registry that it represents. In other words, you can choose any user ID in the registry to use as the administrative user ID. However, for LDAP user registries, ensure that the administrative user ID is a member of the registry and is not the LDAP administrative ID. Also, for LDAP registries, the member you use must be searchable.

See these topics for instructions about configuring particular types of user registries:

"Configure local operating system user registry"
If you want WebSphere security to use i5/OS user profiles to perform authentication, see this topic for instructions.

"Configure LDAP user registries" on page 87
If you want to use a supported third-party user registry, see this topic.

"Configure the custom user registry" on page 96
If your user registry product is not one of the officially supported registries or if you want to create your own registry, see this topic for more information.

Configure local operating system user registry: If you want to use the i5/OS user registry to represent the principals who access your WebSphere resources, no special user registry setup is necessary.

The i5/OS user registry is used for authentication of WebSphere users and for authorization of WebSphere users who access WebSphere resources, but not for WebSphere users who access i5/OS resources. A WebSphere application server does not run under the i5/OS user profile of the WebSphere users. Instead, the WebSphere application server runs under the i5/OS profile that is configured by the WebSphere administrator.

If you want to authorize a user for any WebSphere resource, a user profile must exist on the iSeries system for that user. Use the Create User Profile (CRTUSRPRF) command on your iSeries server to create new user IDs that can be used by WebSphere.

As installed, security is disabled for WebSphere Application Server - Express. It is necessary to take these steps to enable security. These steps will set up security based on the local operating system user registry on the iSeries system on which WebSphere Application Server - Express is installed.

Perform these steps in the WebSphere administrative console:

1. In the navigation menu, click **Security —> User Registries —> LocalOS**.
2. Enter a valid iSeries user profile name in the **Server User ID** field. The Server User ID specifies the iSeries user profile to use when the server authenticates to the underlying operating system. This is also the user that has initial authority to access the administrative application through the administrative console. The administrative user ID is common to all user registries.

   The administrative user ID is common to all user registries. The administrative ID is a member of the chosen user registry, and it has special privileges in WebSphere Application Server - Express. However, it has no special privileges in the user registry that it represents. In other words, you can select any valid user ID in the registry to use as the administrative user ID (Server User ID).

   For the Server User ID field, you can specify any iSeries user profile that meets this criteria:

- It has a status of *ENABLED.
- It has a valid password.
- It is not used as a group profile.

  Note: A group profile is assigned a unique group ID number, which is not assigned to a regular user profile. Run the Display User Profile (DSPUSRPRF) command to determine if the user profile you want to use as the Server User ID has a defined group ID number. If the **Group ID** field is set to *NONE, the user profile can be used as the administrative user ID.

3. In the **Server User Password** field, enter the valid password for the user profile you specified as the Server User ID.
4. Click **OK**.

   Validation of the user and password does not happen in this panel. Validation is only done when you click **OK** or **Apply** in the Global Security panel. If you are in the process of enabling security for the first time, complete the other steps and then go to the Global Security panel, make sure that `Local OS` is selected as the **Active User Registry**. If your changes are not validated the server may not be able to start.

Note: Until you authorize other users to perform administrative functions, you can only access the administrative console with the Server User ID and Password you specified. For more information, see "Assign users to administrative roles" on page 120.

Configure LDAP user registries: Before you configure an LDAP user registry for WebSphere security, see these topics:
- "Lightweight Directory Access Protocol" on page 88
- "Supported directory services" on page 89
- "Using specific directory servers as the LDAP server" on page 90
- "Adding users to the LDAP user registry" on page 93

To configure WebSphere security to use an LDAP user registry, perform these steps in the administrative console:
1. Click **Security —> User Registries —> LDAP**.
2. Enter a valid user name in the **Server User ID** field. You can either enter the complete distinguished name (DN) of the user or the shortname of the user as defined by the **User Filter** in the **Advanced LDAP settings** panel.
3. Enter the password of the user in the **Server User Password** field.
4. Select the type of LDAP server that is used from the **Type** drop-down list.

   Note: If you are using the i5/OS Directory Services product, select `SecureWay` from the **Type** list.

   The type of LDAP server determines the default filters that are used by the WebSphere Application Server - Express. When these default filters are changed the **Type** field changes to `Custom`, which indicates that custom filters are used. This action occurs after **OK** or **Apply** is clicked in the LDAP advanced settings panel. For more information about LDAP filters, see "Configure LDAP search filters" on page 94.

   Choose the `Custom` type from the list and modify the user and group filters to use other LDAP servers, if required. However, it is the customer's responsibility to configure and validate the filters for other LDAP servers. Also, if either IBM_Directory_Server or iPlanet is selected, the **Ignore Case** field should also be selected.
5. Enter the fully qualified host name of the LDAP server in the **Host** field.
6. Enter the LDAP server port number in the **Port** field. The host name along with the port number represent the realm for this LDAP server in the WebSphere Application Server - Express cell. So, if servers in different cells are communicating with each other (using LTPA tokens), these realms should match exactly in all the cells.

Note: If you are using single signon between a WebSphere Application Server - Express Version 5 server (either 5.0 or 5.1) and a WebSphere Application Server Version 4 application server, you must specify an LDAP server port number in the administrative console. By default, the default LDAP port number for WebSphere Application Server - Express Version 5 is 0, but for WebSphere Application Server Version 4, it is not 0. Set the LDAP port numbers for both servers to the same value. In the Version 5 administrative console, set the LDAP port number on the LDAP settings page: click **Security —> User registries —> LDAP**.

7. Enter the Base distinguished name (DN) in the **Base Distinguished Name** field. The Base DN indicates the starting point for searches in this LDAP directory server. For example, for a user with a DN of `cn=John Doe, ou=Rochester, o=IBM, c=US`, the Base DN can be specified as any of (assuming a suffix of `c=us`): `ou=Rochester, o=IBM, c=us` or `o=IBM c=us` or `c=us`. This field can be case sensitive, and it is recommended that they match the case in your directory server. This field is required for all LDAP directories except the Domino Directory. The Base DN field is optional for the Domino Server.
8. Enter the Bind DN name in the **Bind Distinguished Name** field, if necessary. The Bind DN is required if anonymous binds cannot be performed on the LDAP server to obtain user and group information. If the LDAP server is set up to use anonymous binds, leave this field blank.
9. Enter the password that corresponds to the Bind DN in the **Bind password** field, if necessary.
10. Modify the **Search Time Out** value if required. This time out value is the maximum amount of time the LDAP server waits to send a response to the product client before aborting the request. The default is 120 seconds.
11. Disable the **Reuse Connection** field only if you are using routers to spray requests to multiple LDAP servers, and if the routers do not support affinity. Leave this field enabled for all other situations.
12. Enable the **Ignore Case** flag, if required. When this is enabled, the authorization check is case insensitive. Normally, an authorization check involves checking the complete DN of a user (which is unique in the LDAP server) and is case sensitive. However, when using either IBM Directory Server, i5/OS Directory Services, or the iPlanet LDAP servers, this flag needs to be enabled because the group information obtained from the LDAP servers is not consistent in terms of case. This inconsistency only affects the authorization check.
13. Enable secure sockets layer (SSL) if the communication to the LDAP server is through SSL. Check **SSL Enabled** to enable SSL. For more information on setting up LDAP for SSL, refer to "Configure SSL connections between WebSphere Application Server - Express and an LDAP server" on page 152.
14. If SSL is enabled, select the appropriate SSL alias configuration from the drop-down list in the **SSL configuration** field.
15. Click **OK**.

    The validation of the user and password and the setup does not take place in this panel. Validation is only done when you click **OK** or **Apply** in the Global Security panel. If you are enabling security for the first time, complete the remaining steps and then go to the Global Security panel. Select `LDAP` as the **Active User Registry**. If security is already enabled but information on this panel is changed, make sure to go to the Global Security panel and click **OK** or **Apply** to validate your changes. If your changes are not validated, the server may not be able to start.

Lightweight Directory Access Protocol: Lightweight Directory Access Protocol (LDAP) is a user registry in which authentication is performed using an LDAP binding.

WebSphere Application Server - Express security provides and supports implementation for most major LDAP directory servers (LDAP servers) which can be used as the repository for user and group information. These LDAP servers are called by the product processes (servers) for authenticating a user and other security related tasks (for example, getting user or group information).

This support is provided by using different user and group filters to obtain the user and group information. These filters have default values which can be modified to fit your needs. Also, the Custom LDAP feature enables one to use any other LDAP server (which is not in the product supported list of LDAP servers) for its user registry by using the appropriate filters. Supporting new LDAP servers with the Custom LDAP feature is left to the customer. To accomplish this, you need to understand how the filters are used by the product to obtain the user and group information. See "Configure LDAP search filters" on page 94 for more information. It is expected that the customer is responsible for the validity of the filters and testing the configuration and not depend on IBM for support.

To use LDAP as the user registry, you need to know this information about your LDAP server:
- A valid user name (ID)
- The user password
- The server host and port
- The base distinguished name (DN)
- The bind DN and password (if necessary)

The user can be any valid user in the registry and should be searchable. In some LDAP servers some of the administrative users are not searchable and hence cannot be used (for example, `cn=root` in SecureWay). This user is referred to as WebSphere Application Server - Express security server ID or server ID or server user ID in the documentation. Being a server ID entails a user to have special privileges when calling some protected internal methods. Normally, this ID and password is used to log into the administrative console when security is enabled (you can use other users to login if those users are part of the administrative roles).

When security is enabled in the product, this server ID and password is authenticated with the registry during the product startup. If authentication fails, the server does not start. It is important to choose an ID and password that would not expire or change often. If the product server user ID or password need to be changed in the registry, make sure the changes are performed when all the product servers are up and running. After the changes to the registery are completed, use the steps that are described in "Configure LDAP user registries" on page 87 to change the ID, password, and other configuration information, if any. Save the configuration, stop all of the servers, and restart the servers so that the new ID or password is used by the product.

If you have problems starting the product when security is enabled, security should be disabled before the server can start up (to avoid this in the first place, make sure any changes in this panel are validated in the Global Security panel). After the server starts, you can change the ID, password, and other configuration information, and then enable security.

Supported directory services: WebSphere Application Server - Express security supports these LDAP servers:
- SecureWay
- IBM_Directory_Server
- iPlanet
- Netscape
- Domino
- Active_Directory

Although it is expected that any LDAP server that follows the LDAP specification would function, the support is limited to these specified directory servers only. You can use any other directory server by using the custom directory type and by filling in the filters required for that directory. For more information, see "Configure LDAP search filters" on page 94.

To improve performance for LDAP searches, the default filters for IBM_Directory_Server, iPlanet and Active_Directory have been defined such that when you search for a user, the result contains all the relevant information about the user (user ID, groups, and so on). This prevents the product from going to the LDAP server multiple times and improves performance. This is possible only in these directory types because these support searches where the complete information about a user can be obtained.

Also, if you use the IBM_Directory_Server, enable the IgnoreCase flag in the administrative console. This is required because when the groups information is obtained from the user object attributes, the case is not the same as the one that is obtained when you get the groups information directly. For the authorization to work in this case, perform a case-insensitive check and also check the requirement for the IgnoreCase flag.

Using specific directory servers as the LDAP server: This topic describes considerations for using particular directory server products for WebSphere security.

Using i5/OS Directory Services as the LDAP server

i5/OS Directory Services is included in the base operating system beginning in V5R1, and option 32 is no longer available, beginning with V5R2. Directory Services is part of the IBM Directory Server family of products and services and is sometimes referred to as Directory Server (formerly SecureWay Directory) for iSeries.

For V5R2, select either **SecureWay** or **IBM_Directory_Server** as the directory type. For V5R3, select **IBM_Directory_Server** as the directory type. For V5R4, select **IBM_Directory_Server** as the directory type.

Note: If you select **IBM_Directory_Server** as the LDAP directory type, you must also upgrade Directory Services to LDAP 4.1. With LDAP 4.1, Directory Services is programmed to use the new group membership attributes to improve the performance of group membership searches. For information regarding the PTFs that are required for LDAP 4.1, see iSeries Directory Services (LDAP): New V5R2 Enhancements.

(http://www.ibm.com/servers/eserver/iseries/ldap/whatsnew41.htm)

Note: Support for groups that contain other groups (nested groups) depends on specific versions of WebSphere Application Server and LDAP. For more information, see “Using nested groups in user registries” on page 91.

Using IBM Directory Server as the LDAP server

You can choose the directory type of either **IBM Directory Server** or **SecureWay** for the IBM Directory Server product. The difference between these two types is how group membership is looked up. It is recommended that you choose **IBM Directory Server** for optimum performance during run time. In the IBM Directory Server product, the group membership is an operational attribute. An entry can be a member directly by the member (`uniqueMember`). With this attribute, a group membership lookup is accomplished by enumerating the `ibm-allGroups` attribute for the entry, rather than selecting a group and browsing through the members list. To utilize this attribute in a security authorization application, use case-insensitive match so that attribute values returned by `ibm-allGroups` are all in upper case. Lower-case values are stored in the directory server.

Note: Support for groups that contain other groups (nested groups) depends on specific versions of WebSphere Application Server - Express and LDAP. For more information, see “Using nested groups in user registries” on page 91.

Using iPlanet Directory Server as the LDAP server

You can choose the directory type of either iPlanet or NetScape for your iPlanet Directory Server. The difference between these two types is group membership lookup. The iPlanet directory type is selected to use the iPlanet new grouping mechanism only. The new grouping mechanism is called roles in iPlanet, and the attribute is nsRole. Roles are a new entry grouping mechanism in iPlanet that unify entries. Roles are designed to be more efficient and easier to use for applications. For example, an application can locate the role of an entry by enumerating all the roles possessed by a given entry, rather than selecting a group and browsing through the members list. With the iPlanet directory, the WebSphere Application Server - Express security only supports groups defined by nsRole. If you are planning to use traditional grouping methods to group entries in the iPlanet server, select NetScape as the directory type.

Using MS Active Directory server as the LDAP server

To use Microsoft Active Directory as the LDAP server for authentication with WebSphere Application Server - Express, there are specific steps you must take. By default, Microsoft Active Directory does not permit anonymous LDAP queries. To create LDAP queries or browse the directory, an LDAP client must bind to the LDAP server using the distinguished name (DN) of an account that belongs to the administrator group of the Windows system. Group membership search in the Active Directory is done by enumerating the memberof attribute possessed by a given user entry, rather than browsing through the member list in each group. If you change this default behavior to browse each group, you can change the Group Member ID Map field from memberof:member to group:member.

To set up Microsoft Active Directory as your LDAP server, complete the following steps.

1. Determine the full distinguished name 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 or Windows 2000 systems control panel and the DNS domain is ibm.com, the resulting DN has the following structure:

   `cn=`*`adminUsername`*`, cn=users, dc=ibm, dc=com`
2. Determine the short name and password of any account in the Microsoft Active Directory.

   This password does not have to be the same account as used in the previous step.
3. Use the WebSphere administrative console to set up the information needed to use Microsoft Active Directory:
   a. Start the administrative server for the domain, if necessary.
   b. In the administrative console, click **Security —> User Registries —> LDAP**.
   c. Enter the following information in the LDAP settings fields:
      - **Security Server ID:** The short name of the account.
      - **Security Server Password:** The password of the account.
      - **Directory Type:** Active Directory.
      - **Host:** The DNS name of the machine running Microsoft Active Directory.
      - **Base Distinguished Name:** The domain components of the distinguished name of the account. For example: `dc=ibm, dc=com Bind`
      - **Distinguished Name:** The full distinguished name of the account. For example: `cn=`*`adminUsername`*`, cn=users, dc=ibm, dc=com`
      - **Bind Password:** the password of the account.
   d. Click **OK** to save the changes.
   e. Stop and restart the administrative server so that changes take effect.

Using a Lotus Domino Server as the LDAP server

If you choose the Lotus Domino LDAP server Version 6, and the attribute shortname is not defined in the schema, you can do either of the following:

- Change the schema to add the `shortname` attribute.
- Change the user ID map filter to replace the shortname with any other defined attribute (preferably to uid). For example, change `person:shortname` to `person:uid`.

Using nested groups in user registries: Using groups can vastly reduce the cost of administering authorization. However, for WebSphere Application Server - Express security, it is not possible to use nested groups (that is, groups that contain other groups) with the LocalOS user registry, and nested groups are not supported for LDAP user registries prior to WebSphere Application Server - Express Version 5.0.1.

The following reasons prohibit the use of nested groups:

- It is not possible for an i5/OS group profile to be a member of another group profile.
- Prior to Version 5.0.1, the mechanism that WebSphere Application Server - Express uses for determining user membership in a group does not recognize nested groups. The mechanism only searches all groups for a user's member attributes. It cannot determine if a group is itself a member of another group.

WebSphere Application Server - Express Version 5.0.1 (and later) and Version 5.1 uses the nested group feature that is new in LDAP 4.1. These versions of WebSphere Application Server - Express support the nested group feature in any IBM Directory Server product that supports LDAP 4.1. The IBM Directory Server product that runs on iSeries is called i5/OS Directory Services, and ships with i5/OS V5R2 or later. Note that fixes are required to provide full LDAP 4.1 support. For more information about i5/OS Directory Services and the necessary fixes, see iSeries Directory Services (LDAP): New V5R2 Enhancements.

(http://www.ibm.com/servers/eserver/iseries/ldap/whatsnew41.htm)

When WebSphere security is configured to use **IBM_Directory_Server** as the LDAP server type and the IBM Directory Server LDAP directory server supports LDAP 4.1, group membership is determined with the `ibm-allGroups` attribute.

For example, suppose `group92` is bound to the Administrator role of the WebSphere administrative console. The `group2` group is a member of `group92`. Because `group2` contains `user2` and `user4`, these users are authorized to log into and use the administrative console.

In this example, the LDAP directory contains the following entries:

```
dn: c=US
objectclass: top
objectclass: country
c: US
description=United States

dn: o=IBM, c=US
objectclass: top
objectclass: organization
o: IBM
description=International Business Machines

dn: cn=user2, o=IBM, c=US
objectclass: person
objectclass: inetOrgPerson
objectclass: top
objectclass: organizationalPerson
objectclass: ePerson
cn: user2
sn: Sec Two
uid: user2
userpassword: security

dn: cn=user4, o=IBM, c=US
objectclass: person
objectclass: inetOrgPerson
objectclass: top
objectclass: organizationalPerson
objectclass: ePerson
```

```
cn: user4
sn: Sec Four
uid: user4
userpassword: security

dn: cn=group2, o=IBM, c=US
objectclass: top
objectclass: groupOfNames
cn: group2
member: cn=user2, o=IBM, c=US
member: cn=user4, o=IBM, c=US
description: WSA Group Two

dn: cn=group92,o=IBM,c=US
objectclass: top
objectclass: groupOfNames
objectclass: ibm-nestedGroup
cn: group92
description: WSA Group Ninety Two
ibm-memberGroup: cn=group2, o=IBM, c=US
```

Note that, in the last entry, `group2` is a member of `group92`. Its membership is specified in the `ibm-memberGroup` attribute for `group92`.

Adding users to the LDAP user registry: You can use the Lightweight Directory Access Protocol (LDAP) directory services on the iSeries server to store user registry information. Therefore, it is necessary to add into the LDAP directory the users that you want to authorize to WebSphere resources.

You can use a variety of ways to add users, but the easiest is to create an LDAP Data Interchange Format (LDIF) file. The file contains the set of users to be added into the directory. The file is used by LDAP utilities, such as ldapModify. These utilities can be run from i5/OS or from a workstation. If you run these LDAP utilities from the i5/OS, your LDIF must reside in the iSeries integrated file system.

Note: This information is specific to the iSeries Directory Services product.

Perform the following steps:

1. Create an LDIF file. Use the iSeries Edit File (EDTF) utility, or you can use your workstation text editor to create the file and save it in the iSeries integrated file system either through a mapped (mounted) drive or by using file transfer protocol (FTP).

   For WebSphere Application Server - Express and iSeries LDAP directory services, create entries in the directory that correspond to the ePerson schema definition.

   A simple ePerson LDIF entry resembles the following example:

   ```
   dn: cn=John Doe, ou=Rochester, o=IBM, c=US
   objectclass: person
   objectclass: inetOrgPerson
   objectclass: top
   objectclass: organizationalPerson
   objectclass: ePerson
   cn: John Doe
   sn: Doe
   uid: jdoe
   userpassword: secretpass
   ```

   This LDIF entry defines an ePerson for user John Doe. John's user identification (uid) has been set to `jdoe` and his password to `secretpass`. This entry resides within the Rochester organizational unit, in the IBM organization in the United States. Each of the containing entries (`ou`, `o` and `c`) were previously defined before this ePerson entry was defined. You may define a series of LDIF entries in the same file to define LTPA users for WebSphere Application Server - Express.

   If you do not specify a value for the **userpassword** attribute, the i5/OS LDAP server attempts to authenticate LTPA users with the local i5/OS user profile that is identified by the the **uid** attribute value. This action may be desirable if users have i5/OS user profiles and do not want to manage passwords in both the i5/OS user registry and the LDAP directory.

When you create an ePerson entry, make sure that the **cn** and **uid** attributes each have a unique value. That is, you should not create two entries that have the same value for the **cn** and **uid** attributes.

Note: If you have a large user registry, login performance may be severely impacted if the Group Member ID Map property is left at its default value, which is both `groupOfNames:member` and `groupOfUniqueNames:uniqueMember`.

To address this performance problem, specify one of these object classes—not both. You must then exclusively use the selected object class to implement groups in the user registry.

2. Import the LDIF file entries into your directory on the iSeries server. Use then LDAP `ldapadd` utility in Qshell Interpreter (QSH) or from a workstation.

   For more information on importing LDIF entries, see the Directory Services documentation in the iSeries Information Center:

   - For V5R4
   - For V5R3
   - For V5R2

Configure LDAP search filters: Lightweight Directory Access Protocol (LDAP) filters are used by the WebSphere Application Server - Express to search and obtain information about users and groups from a LDAP directory server. A default set of filters are provided for each LDAP server that the product supports. These filters can be modified to fit your LDAP configuration. Once the filters are modified (and OK or Apply is clicked) the directory type in the LDAP registry panel changes to custom, which indicates that custom filters are being used. Also, you can develop filters to support any additional type of LDAP server. The effort to support additional LDAP directories is optional, and IBM does not provide support for other LDAP directory types.

To configure search filters for LDAP, perform these steps in the administrative console:

1. Click **Security —> User Registries —> LDAP**. Under **Additional Properties**, click **Advanced LDAP Settings**.
2. Modify the **user filter**, if necessary.

   The user filter is used for searching the registry for users. It is typically used for Security Role to User assignment. It is also used to authenticate a user using the attribute specified in the filter. It specifies the property for which to look up users in the directory service. For example, to look up users based on their user IDs (uid) and using the object class inetOrgPerson, specify this property:

   ```
   (&(uid=%v)(objectclass=inetOrgPerson)
   ```

   At run time, `%v` is replaced with the uid attribute of the user. The user's uid attribute must be a unique key. This means that two LDAP entries with the same object class cannot have the same uid.

   For more information about this syntax, see your LDAP directory service documentation.
3. Modify the **group filter** if necessary.

   The group filter is used for searching the registry for groups. It is typically used for Security Role to Group assignment. It specifies the property by which to look up groups in the directory service. For example, to look up groups based on their common names (CN) and using the object class of either groupOfNames or groupOfUniqueNames, specify this property:

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

   For more information about this syntax, see your LDAP directory service documentation.
4. Modify the **User ID Map filter** if necessary.

   This filter maps the short name of a user to an LDAP entry. This specifies the piece of information that should represent users when users are displayed using their short names. For example, to display entries of the type object class `inetOrgPerson` by their IDs, specify `inetOrgPerson:uid`.

   This field takes multiple objectclass:property pairs delimited by a semicolon (;). To provide a consistent value for methods like getCallerPrincipal() and getUserPrincipal(), the short name that is obtained by using this filter is used. For example the user `CN=Bob Smith, ou=austin.ibm.com, o=IBM, c=US` can log in using any attributes that were defined for him (for example, e-mail address, social security number, and so on) but when the above methods are called, the user ID `bob` is returned no matter how he logs in.

5. Modify the **Group ID Map filter**, if necessary.

   This filter maps the short name of a group to an LDAP entry. This specifies the piece of information that should represent groups when groups are displayed. For example, to display groups by their names, specify `*:cn`. The asterisk (*) is a wildcard character that searches on any object class in this case. This field takes multiple `objectclass:property` pairs delimited by a semicolon (;).

6. Modify the **Group Member ID Map** if necessary.

   This filter identifies User to Group memberships. For SecureWay, Netscape, and Domino directory types, this field is used to query all the groups that match the specified object class or classes to find if the user is contained in the attribute specified. For example, to get all the users belonging to groups whose object class is `groupOfNames` and the users are contained in the member attributes, specify `groupOfNames:member`. This 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 your LDAP directory service documentation. For the IBM Directory Server, iPlanet, and Active Directory, this is used to query all users in a group by using the information stored in the user object (instead of querying all the groups individually to find if the user exists in that group). For example, the filter `memberof:member` (for Active Directory) is used to get the `memberof` attribute of the user object to get all the groups that the user belongs to. The member attribute is used to get all the users in a group using the group object. Using the user object to obtain the group information is expected to improve performance.

7. Modify the **certificate Map Mode**, if necessary.

   The X.590 certificates can be used for user authentication when LDAP is selected as the user registry. This field is used to indicate whether to map the X.509 certificates into an LDAP directory user by EXACT_DN or CERTIFICATE_FILTER. If EXACT_DN is selected, the distinguished name in the certificate should exactly match the user entry in the LDAP server (including case and spaces). You can use the **Ignore Case** field in the LDAP settings to make the authorization case insensitive. If CERTIFICATE_FILTER is selected, fill in the appropriate certificate filter (in the next field) that should be used for mapping the certificate to a user in the LDAP.

8. If you specified the filter certificate mapping option, use this property to specify the LDAP filter to use to map attributes in the client certificate to entries in LDAP.

   If more than one LDAP entry matches the filter specification at run time, then authentication fails because it results in an ambiguous match. The syntax or structure of this filter is:

   ```
   LDAP attribute=${Client certificate attribute}
   ```

   where *attribute* an LDAP attribute that depends on the schema that your LDAP server is configured to use, and *Client certificate attribute* is one of the public attributes in your client certificate. For example, `uid=${SubjectCN})`. Note that the client certificate attribute side must start with `${` and end with `}`.

   Here is a list of client certificate attribute values. The case of the strings is important.

   - ${UniqueKey}
   - ${PublicKey}
   - ${Issuer}
   - ${NotAfter}
   - ${NotBefore}
   - ${SerialNumber}
   - ${SigAlgName}
   - ${SigAlgOID}
   - ${SigAlgParams}
   - ${SubjectDN}

- ${Version}

  To enable this field, select CERTIFICATE_FILTER for the certificate mapping.

9. Click **OK**.

   The validation of the changes (if any) does not take place in this panel. Validation is only done when the **OK** or **Apply** buttons are pressed in the Global Security panel. If you are in the process of enabling security for the first time, complete the remaining steps and go to the Global Security panel, select `LDAP` as the **Active User Registry**. If security was already enabled and any information on this panel is changed, make sure to go to the Global Security panel and click **OK** or **Apply** to validate your changes. If your changes are not validated the server may not be able to start.

Configure the custom user registry: Before you begin this task, implement and build the UserRegistry interface. For more information on the developing custom user registries, see and "Develop custom user registries" on page 28. For a sample custom user registry code example, see "Custom user registries" on page 29.

The following steps are required to configure custom user registries through the administrative console:

1. In the administrative console, click **Security —> User Registries —> Custom** in the left navigation panel.
2. Enter a valid user name in the **Server User ID** field.
3. Enter the password of the user in the **Server User Password** field.
4. Enter the full name of the location of the implementation class file in the **Custom Registry Classname** field. This should be a dot (.) separated file name. For the sample, this is `com.ibm.websphere.security.FileRegistrySample`. The file can be located in any directory in the integrated file system as long as these conditions are true:
   - It is recommended that the directory is not located in a product directory. That is, the path name of the directory should not begin with /QIBM/ProdData.
   - The directory is specified in the ws.ext.dir property.
   - The directory is specified in the server.policy file.
   - The QEJBSVR user profile has Execute (*X) authority to the directory and Read and Execute (*RX) authority to the class file and its supporting classes. For the sample, this includes the FileRegistrySample.class and RegExpSample.class files.

   To specify in the ws.ext.dir property the directory that contains your custom registry implementation class file, perform these steps in the administrative console:

   a. Expand **Servers** in the navigation menu, and click **Application Servers**.
   b. In the Application Servers page, click the name of your server.
   c. Under Additional Properties, click **Process Definition**.
   d. Under Additional Properties, click **Java Virtual Machine**.
   e. Under Additional Properties, click **Custom Properties**.
   f. If the ws.ext.dirs property has already been defined, click it, append a colon (:) to the value, and add the fully-qualified path of the directory that contains your implementation class.

      If the ws.ext.dirs property is not listed, click **New**. Specify `ws.ext.dirs` as the name of the property, and specify the directory which contains your implementation class or JAR file.
   g. Click **OK**.
   h. Click **Save**.

   To add the directory to the server.policy file, edit the server.policy file that is located in the properties subdirectory of your instance. Specify the following permission:

   ```
   grant codeBase "file:/CustomRegistry/-" {
     permission java.security.AllPermission;
   };
   ```

   For more information about server.policy files, see "Configure the server.policy file" on page 168.

5. Select the **Ignore Case** checkbox for the authorization to perform a case-insensitive check. Enabling this option is necessary only when your registry is case insensitive and does not provide a consistent case when queried for users and groups.
6. Click **Apply** if you have any other additional properties to enter for the registry initialization. Otherwise click **OK** and complete the steps required to enable security.
7. If you need to enter additional properties to initialize your implementation, click **Custom Properties** at the bottom of the panel. Click **New**. Enter the property name and value. Click **OK.** Repeat this step to add other additional properties.

   For the sample, enter the following two properties: (assuming the users.props and groups.props are in myDir directory under the product installation directory).

Property name	**Property value**
`usersFile`	`${USER_INSTALL_ROOT}/myDir/users.props`
`groupsFile`	`${USER_INSTALL_ROOT}/myDir/groups.props`

   Note: The QEJBSVR user profile must have Execute (*X) authority for the directory that contains user.props and groups.props. Additionally, QEJBSVR must have Read and Execute (*RX) authority for the user.props and groups.props files.

   The **Description**, **Required**, and **Validation Expression** fields are not used and can be left blank.
8. If you are enabling security for the first time, complete the remaining steps and then go to the Global Security panel. Select `Custom` as the **Active User Registry**. If security is already enabled but information on this panel is changed, make sure to go to the Global Security panel and click **OK** or **Apply** to validate your changes. If your changes are not validated, the server may not be able to start.

Configure the authentication mechanism: WebSphere Application Server - Express provides these authentication mechanisms:

- "Simple WebSphere Authentication Mechanism" on page 99
  By default, WebSphere Application Server - Express uses SWAM as the authentication mechanism. SWAM is intended for single-server topologies. If you want to use SWAM, no configuration is necessary.
- "Lightweight Third Party Authentication" on page 99
  Use LTPA to support single sign-on (SSO) and the ability to forward credentials to other application server processes. To configure LTPA, use the administrative console.

For more information about how the authentication mechanism works, see "Authentication mechanism" on page 98.

Configure the LTPA authentication mechanism

Perform these steps in the WebSphere administrative console to enable LTPA as the authentication mechanism for WebSphere security:
1. Expand **Security —> Authentication Mechanisms**, and click **LTPA**.
2. Enter the password and confirm it in the password fields. This password is used to encrypt and decrypt the LTPA keys when they are exported and imported. You need to enter this password again when you export the keys to another cell. For more information about LTPA keys, see "Configure LTPA keys" on page 100.
3. Enter a positive integer value in the **Timeout** field. This timeout refers to how long a LTPA token is valid, in minutes. The token contains this expiration time so that any server that receives this token can make sure that this token is valid before proceeding further. When the token expires, the user is prompted to login. An optimal value for this depends on your configuration. The default value is 30 minutes.
4. Click **Apply**. The LTPA configuration is now set. You should not generate the LTPA keys in this step because they are automatically generated later.

5. If your applications contain form-based login, you may want to enable single sign-on support. For more information, see “Configure single signon” on page 101.
6. (Optional) “Configure a trust association interceptor” on page 113.
7. Complete the information in the Global Security panel and press **OK**. When **OK** or **Apply** is clicked in the Global Security panel the LTPA keys are generated automatically the first time, and therefore, you should not generate the keys manually.

   If you later need to generate keys, see “Configure LTPA keys” on page 100.
8. Stop and then start your servers for the changes to take effect.

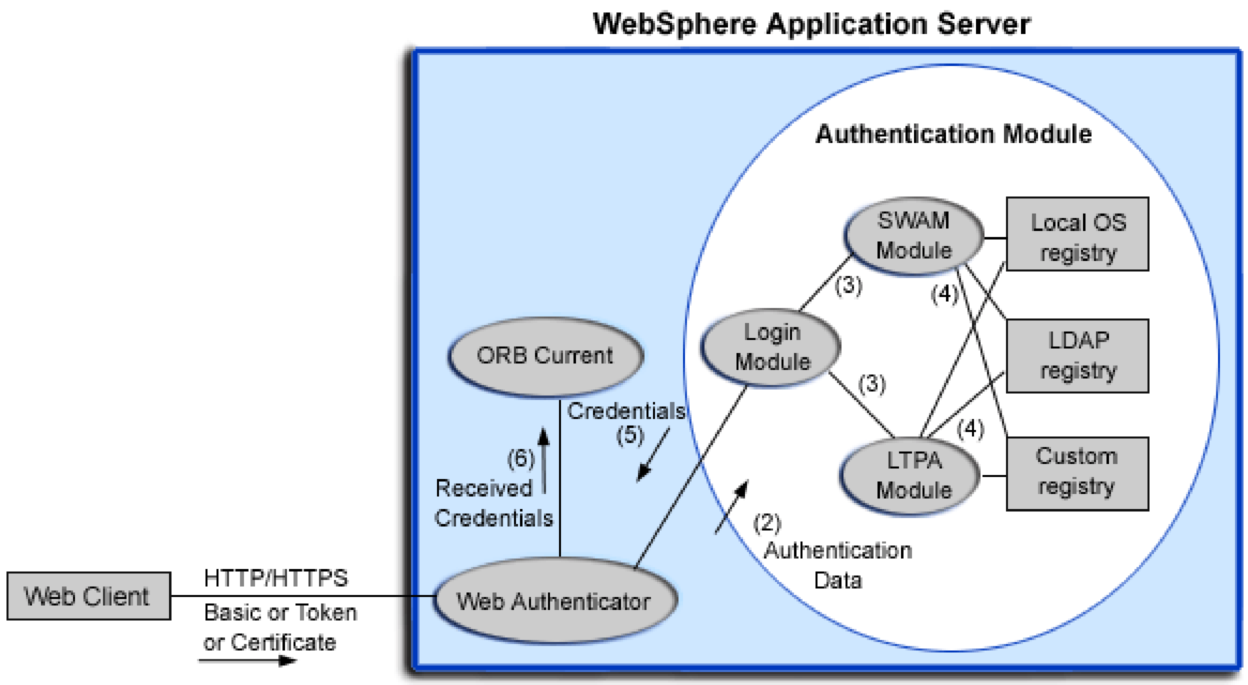
Authentication mechanism: Authentication is the process of establishing whether a client is valid in a particular context. A client can be either an end user, a machine, or an application. An authentication mechanism defines rules about security information (for example, whether a credential is can be forwarded to another Java process), and the format of how security information is stored in both credentials and tokens.

An authentication mechanism in WebSphere Application Server - Express typically collaborates closely with a user registry. The user registry is the repository of user and groups accounts that the authentication mechanism consults when it performs authentication. The authentication mechanism is responsible for creating a credential, which is an internal product representation of a successfully authenticated client user. The abilities of the credential are determined by the configured authentication mechanism.

The WebSphere Application Server - Express provides two authentication mechanisms: Simple WebSphere Authentication Mechanism (SWAM) and Lightweight Third Party Authentication (LTPA). These two authentication mechanisms differ primarily in the distributed security features each supports. Only one configured authentication mechanism can be active at a given time. The active authentication mechanism is selected when you configure WebSphere global security.

Authentication process

This figure shows the authentication process:



These steps describe what occurs during the authentication process:

1. Authentication is required for Web clients when they access protected resources. Web clients use the HTTP or HTTPS protocol to send the authentication information. The authentication information can be either basic authentication (user ID and password), credential token (in case of LTPA), or client certificate. The Web authentication is performed by the Web authentication module.
2. The Authentication module is implemented using Java Authentication and Authorization Service (JAAS) login module. The Web authenticator passes the authentication data to the login module.
3. The login module can use either Lightweight Third Party Authentication (LTPA) or Simple WebSphere Authentication Mechanism (SWAM) for authentication.
4. The authentication module uses the user registry that is configured on the system to perform the authentication. There are three types of registries supported: the local operating system (LocalOS), Lightweight Directory Access Protocol (LDAP), and custom registries.
5. The login module creates a JAAS subject after authentication and stores the CORBA credential that is derived from the authentication data in the public credentials list of the subject. The credential is returned to the Web authenticator.
6. The Web authenticator stores the received credentials in the ORB that is current for the authorization service and uses it to perform further access-control checking.

Simple WebSphere Authentication Mechanism: The SWAM authentication mechanism is intended for simple, non-distributed, single application server runtime environments. The single application server restriction is due to the fact that SWAM does not support forwardable credentials. This means that if an object in one application server process invokes a method on an object that resides in a second process, the identity of the caller in the first process is not transmitted to the second process. What is transmitted is an unauthenticated credential, which, depending on the security permissions configured on the methods, may cause authorization failures.

Because SWAM is intended for a single application server process, single-sign-on (SSO) is not supported.

The SWAM authentication mechanism is suitable for simple environments, software development environments, or other environments that do not require a distributed security solution.

Lightweight Third Party Authentication: Lightweight Third Party Authentication (LTPA) is intended for distributed, multiple application server and machine environments. The LTPA protocol enables WebSphere Application Server - Express to provide security in a distributed environment, using cryptography. Application servers distributed in multiple nodes and cells can securely communicate using this protocol.

LTPA also provides the single sign-on (SSO) feature. SSO allows users to be authenticated once in a DNS domain, without being prompted for authentication information every time they access a resource. This protocol uses cryptographic keys (LTPA keys) to encrypt and decrypt user data that is passed between the servers. These keys must be shared between the different cells for the resources in one cell to access resources in other cells (this assumes all the cells involved use the same LDAP or custom registry).

When you use LTPA, a token is created, which is signed by the keys. The token contains user information and an expiration time. Because of this, the LTPA token is time-sensitive. All product servers that participate in a protection domain must have synchronized time, date, and time zone. If not, LTPA tokens appear to be prematurely expired and cause authentication or validation failures.

This LTPA token is then passed to other servers (in the same cell or in a different cell) either through cookies (for Web resources when SSO is enabled) or through the authentication layer. If the receiving server or servers share the same keys as the originating server, the token can be decrypted to obtain the user information, validated to make sure that it has not expired and that the user information in the token is valid in its registry. Upon successful validation, resources in the receiving servers can be accessed.

All WebSphere Application Server - Express processes in a cell share the same set of keys. If keys need to be shared between different cells, they need to be exported from one cell and imported to the other. For security purposes the keys that are exported are encrypted with a user-defined password. This same password is required when keys are imported into another cell.

LTPA requires that the configured user registry is a central, shared repository.

This table summarizes the authentication mechanism capabilities and user registries with which LTPA can work:

	Forwardable credentials	SSO	LocalOS User Registry	LDAP User Registry	Custom User Registry
SWAM	No	No	Yes	Yes	Yes
LTPA	Yes	Yes	Yes	Yes	Yes

Note: For LTPA, the LocalOS user registry must be centrally managed so that the users and groups are the same, regardless of the machine.

Configure LTPA keys: **Generating keys**

LTPA keys are automatically generated when a password change is detected. The first time you set the LTPA password, (as part of enabling security) the LTPA keys are automatically generated when **OK** or **Apply** is clicked in the LTPA panel. You do not have to click **Generate Keys** in this situation.

Perform these steps in the WebSphere administrative console:
1. Click **Security —> Authentication mechanisms —> LTPA** in the navigation menu.
2. Click **Generate Keys** if you want to use the existing password. This action generates a new set of keys that will be encrypted with the same password as the old set of keys.

   Note: Regardless of the password change, a new set of keys are generated when G**enerate Keys** is clicked. Because these new set of keys are not propagated to the run time unless saved, save the files immediatley.

   To use a new password to generate keys, enter the new password and confirm it. Click **OK** or **Apply**. A new set of keys are generated. A message indicating that a new set of keys are generated shows up on the console. Do not click **Generate Keys**. These new keys are propagated to the run time after you save them.
3. Click **Save** to save the keys.

After a new set of keys are generated and saved, the key propagation is dynamic. All the processes running at that time (cell, node agents, application servers) are updated with the new set of keys. The next topics describe the process of exporting and importing the keys.

Exporting keys

To support single sign on (SSO) in WebSphere Application Server - Express across multiple WebSphere Application Server - Express domains (cells) the LTPA keys and the password should be shared among the domains. The times on the domains should be similar to prevent the tokens from appearing as expired between the cells. The Export Keys button can be used to export the LTPA keys to other domains or cells. Complete the following steps in the administrative console to export key files for LTPA.

Perform these steps in the WebSphere administrative console:
1. Click **Security —> Authentication mechanisms —> LTPA** in the navigation menu.
2. In the **Key File Name** field, enter the full path of a file where the keys need to be stored. The file should have write permissions.

3. Click **Save** to save the file.
4. Click **Export Keys**. A file is created with the LTPA keys in it. Exporting keys fails if a new set of keys was generated or imported and not saved prior to exporting. To avoid failure, make sure you save the new set of keys (if any) before you export them.
5. Click **Save** to save the configuration.

Importing keys

To support single sign on (SSO) in WebSphere Application Server - Express across multiple WebSphere Application Server - Express domains (cells) the LTPA keys and the password should be shared among the domains. The **Import Keys** button can be used to import the LTPA keys from other domains. The key files should have been exported from one of the cells involved into a file.

Importing keys is a dynamic operation. All the servers that are running at this time are updated with the new set of keys and any back-level tokens signed with the back-level keys fail validation and the user is prompted to login again.

Perform these steps in the WebSphere administrative console:
1. Click **Security —> Authentication mechanisms —> LTPA** in the navigation menu.
2. Change the password in the password fields to match the password in the cell from which you are importing the keys.
3. Click **Save** to save the new set of keys in the repository. This is an important step to be completed before importing the keys. If the password and the keys do not match, the servers fails to start. In that case, you would have to turn off security and complete this process again.
4. In the **Key File Name** field, enter the full path of a file where the keys need to be stored. The file should have read permissions.
5. Click **Import Keys**. The keys are now imported into the system.
6. Click **Save** to save the new set of keys in the repository. It is important to save the new set of keys to match the new password so that the servers start.

Configure single signon: With single signon support, Web users can authenticate once when accessing both WebSphere resources (such as JSP files, servlets, HTML files) and Domino resources (such as documents in a Domino database), or when accessing resources in multiple WebSphere domains. This authentication is supported only when LTPA is the authentication mechanism. Single signon uses HTTP cookies to achieve this functionality.

When single signon is enabled, a cookie is created with the LTPA token in it. When the user accesses some other Web resource or Domino resource in any other WebSphere or Domino process in the same DNS domain, the cookie is sent in the request. The LTPA token is then extracted from the cookie and is validated. For more information, see "Prerequisites and conditions for single signon" on page 102.

The LTPA authentication mechanism requires that single signon is enabled if any of the Web applications use form login as the authentication method.

Configure single signon between multiple WebSphere Application Server domains

Complete these steps to configure single signon for multiple WebSphere Application Server domains:
1. "Prerequisites and conditions for single signon" on page 102.
2. "Configure single signon and LTPA for WebSphere Application Server - Express" on page 103.

Configure single signon between WebSphere Application Server - Express and Lotus Domino

Complete these steps to configure single signon for WebSphere Application Server and Domino:

1. "Prerequisites and conditions for single signon."
2. "Configure single signon and LTPA for WebSphere Application Server - Express" on page 103.
3. "Configure single signon for Lotus Domino" on page 106.
4. "Verify single signon between WebSphere Application Server - Express and Lotus Domino" on page 109.

For more information, see "Troubleshooting single signon configurations" on page 110.

Prerequisites and conditions for single signon: To take advantage of support for single signon between WebSphere Application Servers or between WebSphere Application Server - Express and Domino, applications must meet the following prerequisites and conditions:

- The URL for every request must contain the same DNS domain. For example, if the DNS domain is specified as mycompany.com, then single signon is effective for `http://server1.mycompany.com/fred` and `http://server2.mycompany.com/bill`.
- All servers must share the same user registry. This registry can be either a supported LDAP directory server or, if single signon is configured between two WebSphere application servers, a custom user registry. Domino does not support the use of custom registries, but you can use a Domino-supported registry as a custom registry within WebSphere Application Server - Express. For more information, see "Custom user registries" on page 29.

  You can use a Domino Directory (configured for LDAP access) or other LDAP directory for the user registry. The LDAP directory product must be one that is supported by WebSphere Application Server - Express. Supported products include both Domino and all IBM SecureWay LDAP directory servers. Regardless of the choice to use an LDAP or custom registry, the single signon configuration is the same. The difference is in the configuration of the registry.
- All users must be defined in a single LDAP directory. Using LDAP referrals to connect more than one directory together is not supported. Using multiple Domino directory assistance documents to access multiple directories is not supported.
- Users must enable their browsers to accept HTTP cookies because the authentication information that is generated by the server is transported to the browser in a cookie. The cookie is then used to propagate the user's authentication information to other servers, exempting the user from entering the authentication information for every request to a different server.
- The Domino product must meet the following requirements:
  - Domino for iSeries 5.0.6a (or later) is supported.
  - Domino 5.0.5 (or later) for other platforms are supported.
  - A Lotus Notes 5.0.5 (or later) administrator client is required for configuring the Domino server for single signon.
  - You can share authentication information across multiple Domino domains.

  Note: The Domino 6.0 LDAP server is supported as a user registry for WebSphere Application Server - Express version 5.0.2 (or later). Therefore, single signon between a WebSphere application server and a Domino 6.0 server is only supported for WebSphere Application Server - Express version 5.0.2 (or later).
- The WebSphere Application Server products must meet the following requirements:
  - WebSphere Application Server Version 3.5 (or later) for all platforms is supported.
  - You can use any HTTP Web server that is supported by WebSphere Application Server.
  - You can share authentication information across multiple product administrative domains.
  - Basic authentication (user ID and password) using the basic and form-login mechanisms is supported.
  - By default WebSphere Application Server - Express does a case-sensitive comparison for authorization. This implies that the a user who is authenticated by Domino should match exactly the entry (including the base distinguished name) in the WebSphere Application Server authorization

table. If case sensitivity should not be considered for the authorization, the **Ignore Case** property should be enabled in the LDAP user registry settings.

Configure single signon and LTPA for WebSphere Application Server - Express: To use single signon between WebSphere Application Server - Express and Domino or between two WebSphere application servers, you must first configure single signon for WebSphere Application Server - Express. Single signon for WebSphere Application Server allows authentication information to be shared across multiple WebSphere administrative domains and with Domino servers.

To provide single signon to WebSphere application servers in more than one WebSphere administrative domain, you must configure each of the administrative domains to use the same DNS domain, user registry (using LDAP or a custom registry), and a common set of LTPA keys as described in the detailed sections below:

This topic assumes that you have already installed WebSphere Application Server - Express and configured one or more application servers in one or more WebSphere administrative domains. It is also assumed that you are using LDAP as the user registry. Whether you are using an LDAP registry or a custom registry, the single signon setup is the same. The difference is in the configuration of the registry itself. For more information on custom registries, see “Custom user registries” on page 29.

Before you configure single signon for WebSphere Application Server - Express, verify that WebSphere Application Server - Express is accessible:

1. Verify that the application servers are configured correctly. Use a Web browser to access application resources.
2. Verify that the LDAP directory is available and configured with at least one user. Configuring single signon for WebSphere Application Server - Express requires access to the LDAP directory. You can use the Domino Directory or another LDAP directory.

To configure single signon for WebSphere Application Server - Express, perform the following steps:

1. Modify WebSphere Application Server - Express security settings (page 103).
2. Stop and restart the WebSphere instance (page 105).
3. Export LTPA keys to a file (page 105).
4. Authorize users (page 106).
5. Import the LTPA keys file into other WebSphere administrative domains (page 106).

Modify WebSphere Application Server - Express security settings

Single signon configuration is included as part of the overall security configuration of a WebSphere administrative domain.

To change your WebSphere security configuration to support single signon, perform the following steps in the WebSphere administrative console:

1. In the navigation menu, click **Security —> Authentication mechanisms —> LTPA**.
2. Under **Additional properties**, click **Single Signon (SSO)**. Single signon is enabled by default. If it has been disabled, click **Enable**.
3. Select the **Requires SSL** field if all the requests are expected to come over HTTPS transport.
4. In the **Domain Name** field, enter the name of the DNS domain for which single signon is effective (the single signon cookie is sent for all servers only in this domain). For example, if the domain is ibm.com, single signon works between the domains rochester.ibm.com and austin.ibm.com—but not austin.otherCompany.com.

   Note: The domain field is optional, and, if left blank, the Web browser defaults to the domain name of the single signon cookie, which is the WebSphere application server that created it. In this case, single signon is only be valid for the server that created the cookie. This behavior may be desirable when you have defined multiple virtual hosts and each virtual host needs its own or separate domain to be specified in the single signon cookie.

5. Click **OK**.
6. Before you exit the LTPA settings page, you also need to configure the LTPA keys which are used by the administrative domain that you are configuring. You must perform *one* of the following steps, based on the number of administrative domains you are configuring:
   - If you are configuring the first or only WebSphere administrative domain, generate the LTPA keys:
     a. Type the LTPA password to be associated with these LTPA keys in the **Password** and **Confirm Password** fields. You must use this password when importing these keys into other WebSphere Application Server administrative domain configurations (if any) and when you configure single signon for Domino.
     b. Click **Generate Keys** to generate keys for LTPA.
     c. Click **Save** to save the LTPA keys.
   - If you are configuring an additional WebSphere administrative domain, you must import the LTPA keys used during the configuration of the first administrative domain. See Import the LTPA keys file into other WebSphere administrative domains (page 106) for more information.
7. In the navigation menu, click **Security —> User Registries —> LDAP**. (This topic assumes you are using an LDAP user registry. If you are using a custom registry, click **Custom** instead.)
8. Enter your settings in the LDAP User Registry page:
   - **Server User ID**
     The user ID of the administrator for the WebSphere administrative domain. Use the short name or user ID for a user already defined in the LDAP directory. Do not specify a Distinguished Name by using `cn=` or `uid=` before the value. This field is not case sensitive.

     When you start the WebSphere administrative console, you are prompted to login with an administrative account. You must enter exactly the same value that you specify in this field.
   - **Server User Password**
     The password corresponding to the **Server User ID** field. This field is case sensitive.
   - **Type**
     The type of LDAP server you are using. For example, from the list you can select **SecureWay** for IBM SecureWay LDAP Directory or **Domino** for a Domino LDAP Directory.
   - **Host**
     The fully qualified DNS name of the machine on which the LDAP directory runs, for example `myhost.mycompany.com`.
   - **Port**
     The port on which the LDAP directory server listens. By default, an LDAP directory server using an unsecured connection listens on port 389.
   - **Base Distinguished Name**
     The Distinguished Name (DN) of the directory in which searches begin within the LDAP directory. For example, for a user with a DN of `cn=John Doe, ou=Rochester, o=IBM, c=US` and a base suffix of `c=US`, the base DN can be specified in any of the following ways:
     – `ou=Rochester, o=IBM, c=us`
     – `o=IBM, c=us`
     – `c=us`

     This field is not case sensitive. This field is required for all LDAP directories.
   - **Bind Distinguished Name**
     The DN of the user who is capable of performing searches on the directory. In most cases, this field is not required; typically, all users are authorized to search an LDAP directory. However, if the LDAP directory contents are restricted to certain users, you need to specify the DN of an authorized user, for example, an administrator, `cn=administrator`.

- **Bind Password**
  The password corresponding to the Bind Distinguished Name field. This value is required only if you specified a value for the Bind Distinguished Name field. This field is case sensitive.
- **Ignore Case**
  By default WebSphere Application Server - Express does a case-sensitive comparison for authorization. This implies that a user who is authenticated by Domino should match exactly the entry (including the base distinguished name) in the WebSphere Application Server authorization table. If case sensitivity should not be considered for the authorization, the Ignore Case property should be enabled in the LDAP user registry settings.

9. Click **OK**.
10. In the navigation menu, click **Security —> Global Security**. Enable WebSphere security by checking the **Enabled** check box.
11. Verify that the **Cache Timeout** field is set to a reasonable value for your application. When the timeout is reached, WebSphere Application Server - Express clears the security cache and rebuilds the security data. If the value is set too low, the extra processing overhead can be unacceptable. If the value is set too high, you create a security risk by caching security data for a long period of time. The default value is 600 seconds.
12. For the **Active Authentication Mechanism** setting, select **LTPA**.
13. For the **Active User Registry** setting, select **LDAP**.
14. Click **OK** and save the changes.

Stop and restart the WebSphere instance

Whenver changes are made to the global security settings, the instance must be stopped and restarted for the changes to take effect.

1. Logout from the administrative console.
2. Stop the server instance, and then start it. For more information, see the Start and test your application server topic in the *Administration* section.
3. Start the administrative console. Use the domain that you specified during single signon configuration.

   Note: If the hostname is not fully qualified, you cannot log into the administrative console. If the login fails, the login screen is shown again.
4. Specify the user ID and password, exactly as you specified them in the **Server User ID** and **Server User Password** fields in the Global Security settings.

Export the LTPA keys to a file

To complete the security configuration for single signon, you need to export the LTPA keys to a file. Do this for just one WebSphere administrative server if you are configuring single signon for use with multiple WebSphere administrative domains. This file is subsequently used during the configuration of additional administrative domains and during the configuration of single signon for Domino.

To export the LTPA keys to a file, perform the following steps in the administrative console:

1. In the navigation menu, click **Security —> Authentication mechanisms —> LTPA**.
2. In the **Password** and **Confirm Password** fields, specify the password that is associated with the keys to be exported.
3. In the **Key File Name** field, specify the name and location of the file (in the iSeries integrated file system) to contain the LTPA keys. You can use any file name and extension. Note the name and extension you specify; you must use this file when you configure single signon for any additional WebSphere administrative domains and for Domino.
4. Click **Export Keys** to export the LTPA keys to the specified file.
5. Click **Save** to apply the changes to your server configuration.

Authorize users

Before you can test the single signon configuration for WebSphere Application Server, you must grant users permissions to resources so that their access can be tested. For more information, see "Assign users to administrative roles" on page 120.

Import the LTPA keys file into other WebSphere administrative domains

If you are configuring single signon for use with multiple WebSphere administrative domains, import the LTPA keys file into all the administrative domains, excluding only the administrative domain from which you exported the file. Before proceeding, ensure that you have completed all of the preceeding steps (except **Export the LTPA keys to a file**) for these administrative domains.

To import the LTPA keys file, complete the following steps:
1. Start the WebSphere server for the domain.
2. Start the administrative console.
3. In the navigation menu, click **Security —> Authentication mechanisms —> LTPA**.
4. In the **Password** and **Confirm Password** fields, specify the password that is associated with the keys to be imported.
5. In the **Key File Name** field, specify the name and location of the LTPA keys file.
6. Click **Import Keys** to import the LTPA keys from a file.
7. Click **Save** to apply the changes to the master configuration.
8. Click **Logout** to exit the administrative console.
9. Stop and then restart the application server.

Configure single signon for Lotus Domino: To configure single signon for Domino, you select a new multi-server option in a Server document for session-based authentication, and you create a new domain-wide configuration document, called the **Web SSO Configuration document**, in the Domino Directory. The Web SSO Configuration document, which must be replicated to all Domino servers participating in the single signon domain, is encrypted for participating Domino servers and contains a shared secret that is used by Domino servers for authenticating user credentials.

To complete this procedure, you need the following information from your WebSphere Application Server - Express single signon configuration:
- The path and name of the file that contains the LTPA keys that were created.
- The password you used when LTPA keys from WebSphere Application Server - Express were generated.
- The name of the DNS domain in which WebSphere Application Server - Express is configured.

For more information, see "Configure single signon and LTPA for WebSphere Application Server - Express" on page 103.

To configure single signon for Domino servers, complete the following steps:
1. Create the Web SSO Configuration document. (page 106)
2. Configure the Server document. (page 107)
3. Finish the Domino configuration. (page 108)
4. Verify the single signon for Domino configuration. (page 108)
5. (Optional) Configure additional Domino servers in the original domain. (page 108)
6. (Optional) Configure Domino servers in different domains. (page 108)

Create the Web SSO Configuration document

To create the Web SSO Configuration document, use a Lotus Notes Client 5.0.5 (or later) and follow these steps:

1. In the Domino Directory, select the Servers view.
2. Click on the **Web** pull-down menu item.
3. Select **Create Web SSO Configuration** to create the document.
4. On the Web SSO Configuration document, click the **Keys** pull-down menu.
5. Select **Import WebSphere LTPA Keys** to import the LTPA keys previously created for WebSphere Application Server and stored in a file.
6. Enter the fully-qualified path name of the file that contains the keys for WebSphere Application Server and click **OK**.
7. Enter the password that was used when the LTPA keys were generate. The SSO Configuration document is automatically updated to reflect the information in the imported file.
8. Complete the remaining fields in this document. Groups and wildcards are not allowed in the fields. The following list describes the fields and the expected values:
   - **Token Expiration**
     The number of minutes a token can exist before expiring. A token does not expire based on inactivity; it is valid for only the number of minutes that is specified, from the time of issue.
   - **DNS Domain**
     The DNS domain portion of your system's fully qualified Internet name. This is a required field.

     All servers participating in single signon must reside in the same DNS domain; this value must be the same as the Domain value that you specified when in your WebSphere Application Server configuration. Also, WebSphere Application Server treats the DNS domain as case sensitive, so ensure that the DNS domain value is specified in exactly the same way, including the same case.
   - **Domino Server Names**
     The Domino servers that you want to participate in single signon. This SSO Configuration document is encrypted for the creator of the document, the members of the **Owners** and **Administrators** fields, and the servers that are specified in this field. These servers can be in different Domino domains; however they must be in the same DNS domain.

     You must specify a fully qualified Domino server name, for example, `MyDominoServer/MyOu`. The Domino server name that you specify here must also match the name of the corresponding server's Connection document in your client's Domino Directory.
   - **LDAP Realm**
     The fully qualified DNS host name of the LDAP server. This field is initialized from the information that is provided in the imported LTPA keys file. Change this value only if an port value for the LDAP server was specified for the WebSphere Application Server administrative domain. If a port was specified, an escape character (\) must be inserted into the value before the colon character (:). For example, replace `myhost.mycompany.com:389` with `myhost.mycompany.com\:389`.
9. Save the Web SSO Configuration document. It now appears in the Web Configurations view.

If you are configuring multiple Domino servers for single signon, see Configure additional Domino servers (page 108).

Configure the Server document

To update the Server document for single signon, follow these steps:

1. In the Domino Directory, select the Servers view.
2. Edit the Server document.
3. Select the **Ports —> Internet Ports —> Web** tab.
4. To enable basic authentication for Web users in the HTTP Authentication options section set **Name & password** to `yes`.
5. Select **Internet Protocols —> Domino Web Engine**.

6. Select Multiple Servers (SSO) in the **Session Authentication** field to enable single signon for Domino.
7. Select the **Security** tab.
8. In the Internet Access section select "More name variations with lower security" in the **Internet authentication** field so short names can be used for authentication.
9. Save the Server document.

If you are configuring multiple Domino servers for single signon, see Configure additional Domino servers (page 108).

Finish the Domino configuration

Before continuing, finish configuring the Domino server for use by Web users. The remaining configuration steps are not specific to single signon and are not covered here in detail. See the Security topic in the Domino 5 Administration Help

(http://www-12.lotus.com/ldd/doc/domino_notes/5.0.3/help5_admin.nsf) for information on the following tasks:
- Configuring access to an LDAP directory when the Domino Directory is not being used.
- Authorizing Web users to Domino resources.

Verify the single signon for Domino configuration

To verify the single signon configuration for Domino, ensure that the Domino server is configured correctly and that Web users are authorized to access Domino resources by performing the following steps:
1. To verify that the Domino server is configured correctly, stop and restart the Domino HTTP Web server.

   If single signon is configured correctly, the following message appears on the Domino server console:
   ```
   HTTP: Successfully loaded Web SSO Configuration
   ```
   If a Domino server enabled for single signon cannot find a Web SSO Configuration document or is not included in the Domino Server Names field and therefore cannot decrypt the document, the following message appears on your server's console:
   ```
   HTTP: Error Loading Web SSO configuration. Reverting to single-server session authentication
   ```
2. To verify that users are authorized, attempt to access a Domino resource, such as a Domino Directory. First, attempt the access as a user that is defined in the Domino Directory itself, for local authorization. Then, attempt the access as a user that is defined in the LDAP directory service, for authorization of WebSphere Application Server users.

Configure additional Domino servers in a single domain

If you are using single signon with multiple Domino servers, perform the following steps for each additional server:
1. Replicate the initial Web SSO Configuration document to each additional Domino server.
2. Update the Server document for each additional Domino server.
3. Restart each of the Domino HTTP web servers.

Configure Domino servers in multiple Domino domains

If you are using single signon with Domino servers in multiple Domino domains, you must also set up cross-domain authentication among the Domino servers. For example, assume there are Domino servers in two Domino domains, X and Y.

Use the following procedure to enable the Domino servers to perform single signon between the domains:

1. A Domino administrator must copy the Web SSO Configuration document from the Domino Directory for Domain X and paste it into the Domino Directory for Domain Y. The Domino administrator needs rights to decrypt the Web SSO Configuration document in Domain X and to create documents in the Domino Directory for Domain Y.
2. Ensure that your Lotus Notes client's location home server is set to a Domino server in Domain Y.
3. Edit the Web SSO Configuration document for Domain Y.
4. In the **Participating Domino Servers** field, include only the Domino servers with Server documents in Domain Y that will participate in single signon.
5. Save the Web SSO Configuration document. It is now to be encrypted for the participating Domino servers in Domain Y, so these servers now have the same key information as the Domino servers in domain X. This shared information allows Domino servers in Domain Y to perform single signon with Domino servers in Domain X.

Verify single signon between WebSphere Application Server - Express and Lotus Domino: This topic discusses the verification of single signon between Domino and WebSphere Application Server - Express. Before proceeding, verify that the following conditions are met:

- The LDAP directory contains at least one user that is defined for testing purposes.
- The WebSphere administrative console can be started for each of the WebSphere administrative domains that are involved in single signon.
- A user can authenticate to each administrative domain with a security name that is defined in the LDAP directory.
- At least one user in the LDAP directory is authorized to access at least one Domino resource, such as the Domino Directory.
- At least one user in the LDAP directory is authorized to access at least one WebSphere resource, such as the WebSphere administrative console.
- From a Web browser that is configured to not accept HTTP cookies, you are able to reach the following resources after you enter a user ID and password:
  - WebSphere-protected resources (such as a servlet).
  - Domino-protected resources (suc as a Lotus Notes database).

If all of the preliminary tests succeed, you are ready to verify that single signon is working correctly.

To test single signon between WebSphere Application Server - Express and Domino, perform the following steps:

1. Restart your Web browser.
2. Configure the Web browser to accept HTTP cookies. (If you are using Internet Explorer, enable the per-session (not stored) type of cookies.
3. Configure the browser to notify you before it accepts HTTP cookies. The warning provides visual confirmation that Domino and WebSphere Application Server are generating and returning HTTP cookies to your browser after the server authenticates you. (You can suppress the cookie notifications after you verify that cookies are being exchanged.)
4. From the browser, specify the URL for a resource that is protected by the Domino server; for example, attempt to open a database that does not permit access to anonymous users, as shown in the following example:
   a. Make sure to use a fully qualified DNS host name in the URL; for example, enter `http://myhost.mycompany.com/names.nsf` instead of `http://myhost/names.nsf`.
   b. When you are prompted for a user ID and password, make sure that you specify a user ID that is authorized to resources for both the Domino and WebSphere application servers.

The format of the name depends on the level of restriction that Domino enforces for Web users and whether a Domino directory or another LDAP directory is being used. (For details on the options for basic authentication, see the Domino 5 Administration Help

(http://www-12.lotus.com/ldd/doc/domino_notes/5.0.3/help5_admin.nsf); in particular, see the information on controlling the level of authentication for Web clients.)

The level of restriction that Domino enforces for Web users is set in the Web server authentication field on the Security window of the Server document. If you are using the default configuration settings, you can specify the user's short name or user ID.

c. When you are prompted, accept the HTTP cookie.

If you can successfully access such a resource, the token that is generated by the Domino server is accepted by WebSphere Application Server - Express.

5. From the same browser session, attempt to access a resource that is protected by WebSphere Application Server - Express. If single signon is working correctly, access is granted without prompting you to log in.

   Make sure to use the fully qualified DNS host name in the URL. For example, enter `http://myhost.mycompany.com/snoop` instead of `http://myhost/snoop`.

   Note: If you are getting a message about the session being expired or invalid, a possible cause is that the coordinated universal time offset is not set correctly on one of the systems. Verify that the system value QUTCOFFSET is correct.

6. From the same browser session, attempt to access resources that are managed by any additional Domino and WebSphere domains which are included in your single signon configuration.
7. Restart your browser session and perform the verification steps again; but this time, start by accessing a resource that is protected by WebSphere Application Server - Express. This verifies that the token that WebSphere Application Server generates is accepted by the Domino server or servers. When you are prompted for a user ID and password, use the user's short name or user ID, which is the default naming convention for users in WebSphere Application Server - Express.

Troubleshooting single signon configurations: This article describes common problems in configuring single signon between a WebSphere Application Server - Express and a Domino server and suggests possible solutions.

- **Failure to save the Domino Web SSO Configuration document.**
  The client must be able to find Domino server documents for the Domino servers that are participating in single signon. The Web SSO Configuration document is encrypted for the servers that you specify, so the home server that is indicated by the client location record must point to a server in the Domino domain where the participating servers reside. This pointer ensures that lookups can find the public keys of the servers.

  If you receive a message stating that one or more of the participating Domino servers cannot be found, then those servers cannot decrypt the Web SSO Configuration document or perform single signon.

  When the Web SSO Configuration document is saved, the status bar indicates how many public keys were used to encrypt the document by finding the listed servers, authors, and administrators on the document.
- **Domino server console fails to load the Web SSO Configuration document upon Domino HTTP server startup.**
  During configuration of single signon, the server document is configured for `Multi-Server` in the **Session Authentication** field. The Domino HTTP server tries to find and load a Web SSO Configuration document during startup. The Domino server console reports the following message if a valid document is found and decrypted:

  ```
  HTTP: Successfully loaded Web SSO Configuration.
  ```

  If a server cannot load the Web SSO Configuration document, single signon does not work. In this case, a server reports the following message:

```
HTTP: Error Loading Web SSO configuration.
Reverting to single-server session authentication.
```

Verify that there is only one Web SSO Configuration document in the Web Configurations view of the Domino directory and in the $WebSSOConfigs hidden view. You cannot create more than one document, but you can insert additional documents during replication.

If there is only one Web SSO Configuration document, another condition that can cause the same error message is when the public key of the Server document does not match the public key in the ID file. In this case, attempts to decrypt the Web SSO Configuration document fail and the error message is generated.

This situation can occur when the ID file is created multiple times but the Server document is not updated correctly. Usually, there is an error message displays on the Domino server console stating that the public key does not match the server ID. If this happens, single signon does not work because the document is encrypted with a public key for which the server does not possess the corresponding private key.

To correct a key-mismatch problem, perform the following steps:

1. Copy the public key from the server ID file and paste it into the Server document.
2. Re-create the Web SSO Configuration document.

- **Authentication fails when accessing a protected resource.**
  If a Web user is repeatedly prompted for a user ID and password, single signon is not working because either the Domino or the WebSphere security server is not able to authenticate the user with the Lightweight Directory Access Protocol (LDAP) server. Check the following possibilities:
  – Verify that the LDAP server is accessible from the Domino server machine. Use the TCP/IP ping utility to check TCP/IP connectivity and to verify that the host machine is running.
  – Verify that the LDAP user is defined in the LDAP directory. Use the `ldapsearch` utility to confirm that the user ID exists and that the password is correct. For example, you can run the following command, entered as a single line, from the i5/OS Qshell, a UNIX shell, or a Windows DOS prompt:

```
ldapsearch -D "cn=John Doe, ou=Rochester, o=IBM, c=US"
  -w mypassword -h myhost.mycompany.com -p 389
  -b "ou=Rochester, o=IBM, c=US" (objectclass=*)
```

  A list of directory entries is expected. Possible error conditions and causes follow:
  - **No such object:** This error indicates that the directory entry that is referenced by either the user's distinguished name (DN) value (which is specified after the -D option) or the base DN value (which is specified after the -b option) does not exist.
  - **Invalid credentials:** This error indicates that the password is invalid.
  - **Cannot contact LDAP server:** This error indicates that the host name or port specified for the server is invalid or that the LDAP server is not running.
  - An empty list means that the base directory that is specified by the -b option does not contain any directory entries.

  – If you are using the user's short name (or user ID) instead of the distinguished name, verify that the directory entry is configured with the short name. For a Domino directory, this is the **Short name/UserID** field of the Person document. For other LDAP directories, this is the `userid` property of the directory entry.
  – If Domino authentication fails when using an LDAP directory other than Domino directory, verify the configuration settings of the LDAP server in the Directory Assistance document in the Directory Assistance database. Also verify that the Server document refers to the correct Directory Assistance document.

  The following LDAP values that are specified in the Directory Assistance document must match the values that are specified for the user registry in the WebSphere administrative domain:
  - Domain name
  - LDAP host name

    - LDAP port
    - Base DN

  Additionally, the rules that are defined in the Directory Assistance document must refer to the base DN of the directory containing the directory entries of the users.

  You can trace Domino server requests to the LDAP server by adding the following line to the server notes.ini file:

  ```
  webauth_verbose_trace=1
  ```

  After restarting the Domino server, trace messages displays in the Domino server console as Web users attempt to authenticate to the Domino server.

- **Authorization fails when accessing a protected resource.**
  After authenticating successfully, if a Web user is shown an authorization error message, security is not configured correctly.

  For Domino databases, verify that the user is defined in the access-control settings for the database. Refer to the Domino Administrative documentation for the correct way to specify the user's DN. For example, for the DN `cn=John Doe, ou=Rochester, o=IBM, c=US`, the value on the access-control list must be set as `John Doe/Rochester/IBM/US`.

  For resources that are protected by WebSphere Application Server - Express, verify that the security permissions are set correctly. If granting permissions to selected groups, make sure that the user that is attempting to access the resource is a member of the group. For example, you can verify the members of the groups by using the following URL to display the directory contents:

  ```
  Ldap://myhost.mycompany.com:389/ou=Rochester, o=IBM, c=US??sub
  ```

  If you have changed the LDAP configuration information (host, port, and base DN) in a WebSphere Application Server - Express administrative domain since the permissions were set, the existing permissions are probably invalid and need to be re-created.

- **SSO fails when accessing protected resources.**
  If a Web user is prompted to authenticate with each resource, SSO is not configured correctly. Check the following possibilities:
  – Both WebSphere Application Server - Express and the Domino server must be configured to use the same LDAP directory. The HTTP cookie that is used for single signon stores the full DN of the user (for example, `cn=John Doe, ou=Rochester, o=IBM, c=US`) and the domain name system (DNS) domain.
  – If the Domino Directory is used, define Web users by hierarchical names. For example, update the **User name** field in the Person document to include names of this format as the first value: `John Doe/Rochester/IBM/US`.
  – URLs that are issued to Domino servers and WebSphere Application Server - Express servers that are configured for single signon must specify the full DNS server name, not only the host name or TCP/IP address. For browsers to send cookies to a group of servers, the DNS domain must be included in the cookie, and the DNS domain in the cookie must match the URL. (This requirement means that you cannot use cookies across TCP/IP domains.)
  – Domino and WebSphere Application Server - Express must be configured to use the same DNS domain. Verify that the DNS domain value is exactly the same, including capitalization. The DNS domain value is found on the Configure Global Security Settings panel of the WebSphere administrative console and in the Web SSO Configuration document of a Domino server. If you make a change to the Domino Web SSO Configuration document, replicate the modified document to all Domino servers that are participating in single signon.
  – Clustered Domino servers must have the host name populated with the full DNS server name in the Server document for Domino Internet Cluster Manager (ICM) to redirect to cluster members that are using single signon. If this field is not populated, by default, ICM redirects URLs to clustered Web servers by using only the host name. It cannot send the single signon cookie because the DNS domain is not included in the URL.

    To correct the problem, perform the following steps:
    1. Edit the Server document.

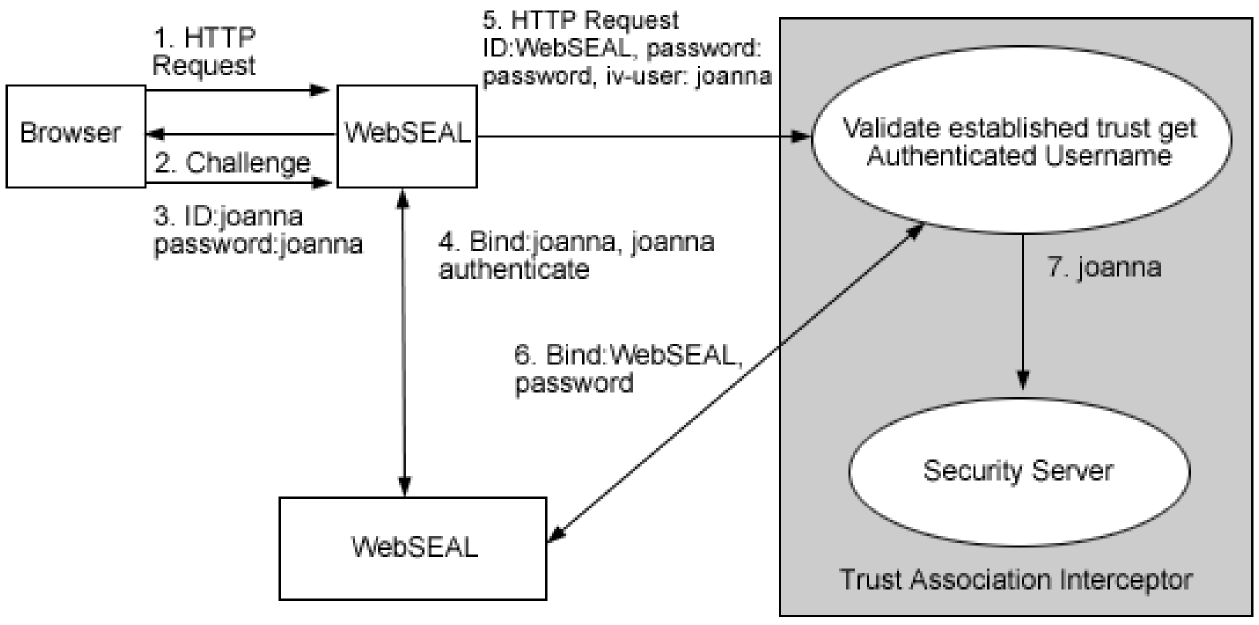
2. Click the **Internet Protocols** tab.
3. Click the **HTTP** tab.
4. Enter the server's full DNS name in the **Host names** field.
5. If a port value for an LDAP server was specified for a WebSphere Application Server administrative domain, edit the Domino Web SSO Configuration document and insert a backslash character (\) into the value of the LDAP Realm field before the colon character (:). For example, replace `myhost.mycompany.com:389` with `myhost.mycompany.com\:389`.

Configure a trust association interceptor: For more information about trust association interceptors, see "Trust associations" on page 115.

A typical scenario where the trust association interceptor (TAI) is used is better understood based on an environment where IBM Tivoli WebSEAL product is deployed and used with WebSphere Application Server. For WebSEAL, there is an implementation of the TAI already provided with the product. These steps outline the typical flow of an HTTP request for a secured WebSphere Application Server resource authenticated by WebSEAL, through a Web trust association.

1. The browser makes a request for a secured WebSphere resource.
2. WebSEAL sends back a challenge, either an HTTP Basic authentication or form-based challenge.
3. User name and password are supplied.
4. WebSEAL authenticates the user.
5. The modified request is forwarded by WebSEAL to the WebSphere Application Server.
6. The plug-in (TAI) uses the validateEstablishedTrust method to establish that WebSphere Application Server trusts the WebSEAL server.
7. The plug-in extracts the end-user name from the iv-user header field and passes it to the WebSphere Application Server to handle authorization.

Note: Versions 3.9 and higher of WebSEAL do not send the user ID and password to the server. Trust is based on a mutual secure sockets layer connection established between WebSEAL and the WebSphere Application Server. As a result, steps 6 and 7 do not apply to versions 3.9 and higher of WebSEAL.


1. HTTP Request
5. HTTP Request ID:WebSEAL, password: password, iv-user: joanna
Browser
WebSEAL
2. Challenge
Validate established trust get Authenticated Username
3. ID:joanna password:joanna
4. Bind:joanna, joanna authenticate
7. joanna
6. Bind:WebSEAL, password
Security Server
WebSEAL
Trust Association Interceptor

When you set up security for the first time, you need to complete the following steps if you want to use WebSEAL Trust Association Interceptor or your own trust association interceptor with a reverse proxy security server.

Perform these steps in the WebSphere administrative console:

1. Start the administrative console.
2. In the topology tree, expand **Security —> Authentication mechanisms** and click **LTPA**.
3. Click **Trust Association** under Additional Properties.
4. Select the **Trust Association Enabled** check box.
5. Click **Interceptors** under Additional Properties.
6. Select the interceptor that you want to configure. For additional information, see Trust association interceptor settings.

   - If you are using WebSEAL Interceptor, select com.ibm.ws.security.web.WebSEALTrustAssociationInterceptor.
   - To set up additional interceptors, perform these steps:
     a. Click **New**.
     b. Specify the classname for the interceptor.
     c. Click **Apply**.
7. To configure an interceptor, click the interceptor classname.
8. Click **Custom Properties**.
9. On the **Custom Properties** page, click **New**.
10. Specify the property name and value pairs. These are the name and value pairs for WebSEAL:

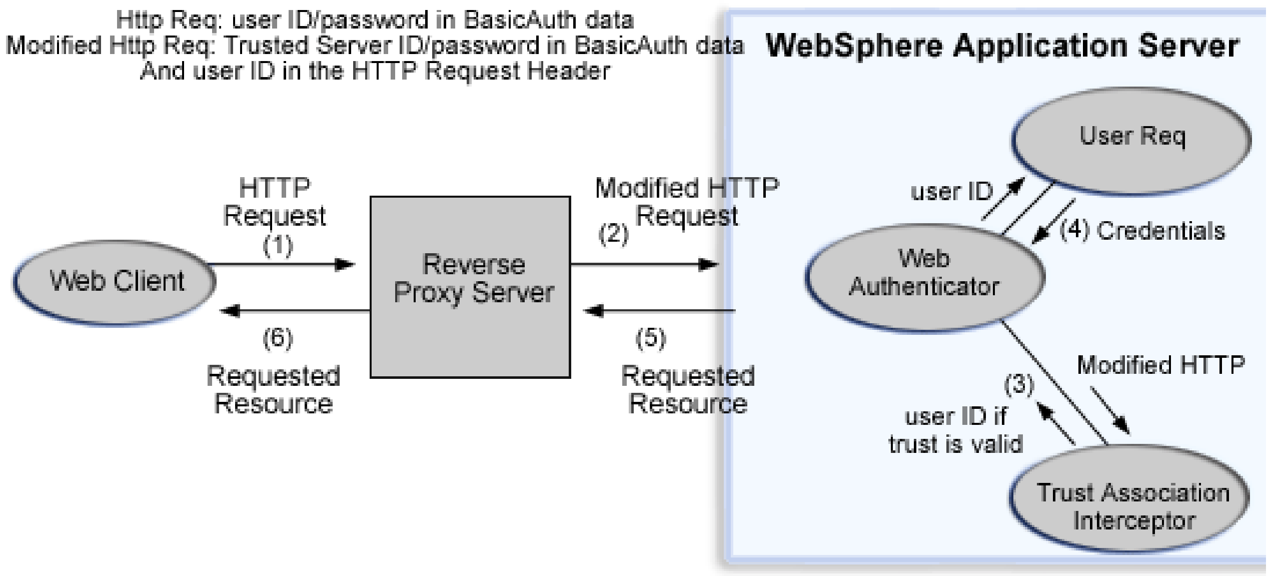
Property name	**Value**
com.ibm.websphere.security.trustassociation.types	WebSEAL
com.ibm.websphere.security.webseal.loginID	The ID of the WebSEAL server.
com.ibm.websphere.security.webseal.id	iv-user. This is a special header field that is sent by WebSEAL with the request to WebSphere Application Server.
com.ibm.websphere.security.webseal.hostnames	The host names (case sensitive) that are expected in the request header (the VIA header). This should also include the proxy host names (if any) unless the com.ibm.websphere.security.webseal.ignoreProxy is set to `true`.
com.ibm.websphere.security.webseal.ports	The corresponding port number of the host names that are expected in the request header (the VIA header). This should also include the proxy ports (if any) unless the com.ibm.websphere.security.webseal.ignoreProxy is set to `true`.
com.ibm.websphere.security.webseal.ignoreProxy	An optional property that if set to `true` or `yes` ignores the proxy host names and ports in the VIA header. By default this property is set to `false`.

11. Click **OK**.
12. Save the configuration.

Trust associations: Trust association enables the integration of IBM WebSphere Application Server - Express security and third-party security servers. More specifically, a reverse proxy server can act as a front-end authentication server while the product applies its own authorization policy onto the resulting credentials that are passed by the proxy server.

Demand for such an integrated configuration has become more and more compelling, especially when a single product cannot meet all customer needs or when migration is not a viable solution. This topic gives a conceptual background behind the approach.

There is a growing demand to provide customers with a trust association solution between IBM WebSphere Application Server - Express and other Web authentication servers that act as reverse proxy security server (IBM Tivoli Security Manager - WebSEAL for Policy Director, ECommerce Server) as an entry point to all service requests. This implementation design intends to have the proxy server as the only exposed entry point. It authenticates all requests that come in and provides coarse, granularity junction point authorization.


Http Req: user ID/password in BasicAuth data
Modified Http Req: Trusted Server ID/password in BasicAuth data
And user ID in the HTTP Request Header
WebSphere Application Server
Web Client
HTTP Request (1)
Reverse Proxy Server
(6) Requested Resource
Modified HTTP Request (2)
(5) Requested Resource
User Req
user ID
(4) Credentials
Web Authenticator
Modified HTTP
(3)
user ID if trust is valid
Trust Association Interceptor

In this setup (shown by the figure), the WebSphere Application Server - Express is used as a back-end server to further exploit its fine-grained access control. The reverse proxy server passes to WebSphere Application Server - Express the HTTP request that includes the credentials of the authenticated user. WebSphere Application Server - Express then uses these credentials to authorize the request.

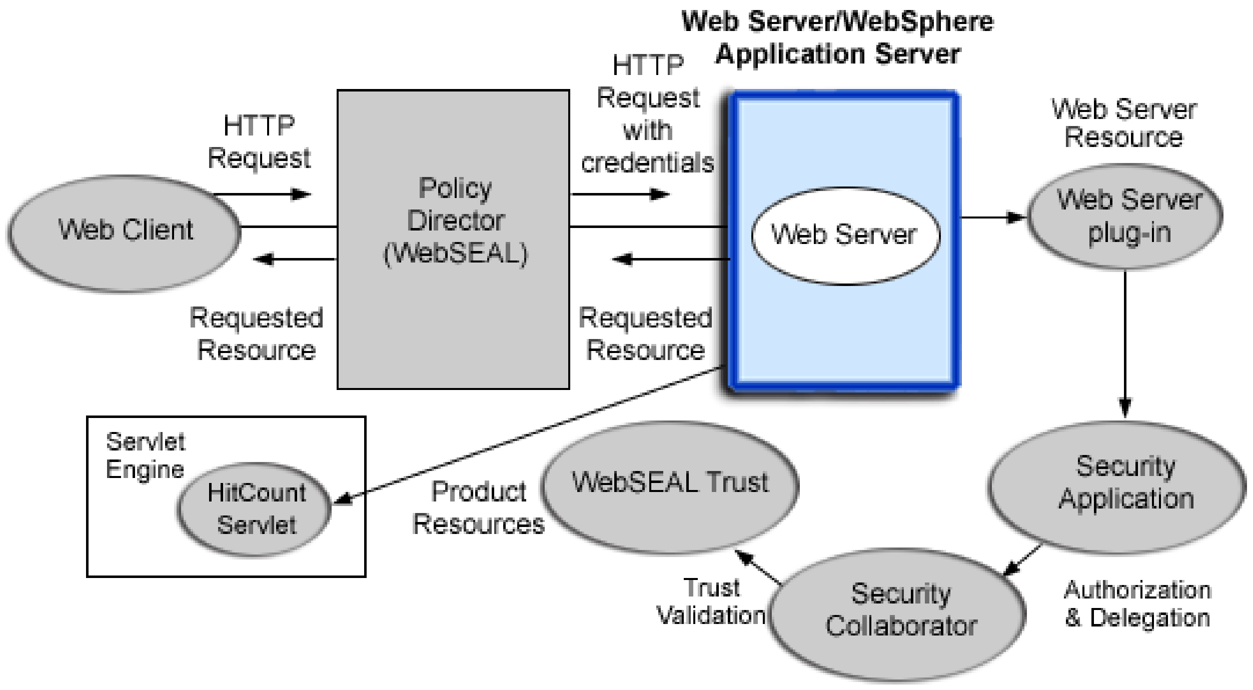
Trust association model

The idea that WebSphere Application Server - Express can support trust association implies that the product application security recognizes and processes HTTP requests that are received from a reverse proxy server. WebSphere Application Server - Express and the proxy server engage in a contract in which the product gives its full trust to the proxy server and the proxy server applies its authentication policies on every Web request that is dispatched to WebSphere Application Server - Express. This trust is validated by the interceptors that reside in the product environment for every request received. The method of validation is agreed upon by the proxy server and the interceptor.

Running in trust association mode does not prohibit WebSphere Application Server - Express from accepting requests that did not pass through the proxy server. In this case, no interceptor is needed for validating trust. It should be possible, however, to configure WebSphere Application Server - Express to strictly require all HTTP requests to go through a reverse proxy server, in which case all requests not coming from a proxy server are immediately denied by WebSphere Application Server - Express.

IBM WebSphere Application Server - Express and WebSEAL integration

The integration of WebSEAL and WebSphere Application Server - Express security is achieved by placing WebSEAL at the front-end as a reverse proxy server (see the figure). From a WebSEAL management perspective, a junction is created with WebSEAL on one end, and the product Web server on the other end. A junction is a logical connection created to establish a path from WebSEAL to another server.



In this setup, a request for Web resources stored in protected domain of the product is submitted to WebSEAL where it is authenticated against the WebSEAL security realm. If the requesting user has access to the junction, the request is transmitted to the WebSphere Application Server - Express HTTP server through the junction, and then to the application server.

Meanwhile, the WebSphere Application Server - Express validates every request that comes through the junction to ensure that the source is a trusted party. This process is referenced as validating the trust and it is performed by a WebSEAL product-designated interceptor. If the validation is successful, the WebSphere Application Server - Express authorizes the request by checking whether the client user has the required permissions to access the Web resource. If so, the Web resource is delivered to WebSEAL, through the Web server, which then gives it to the client user.

WebSEAL

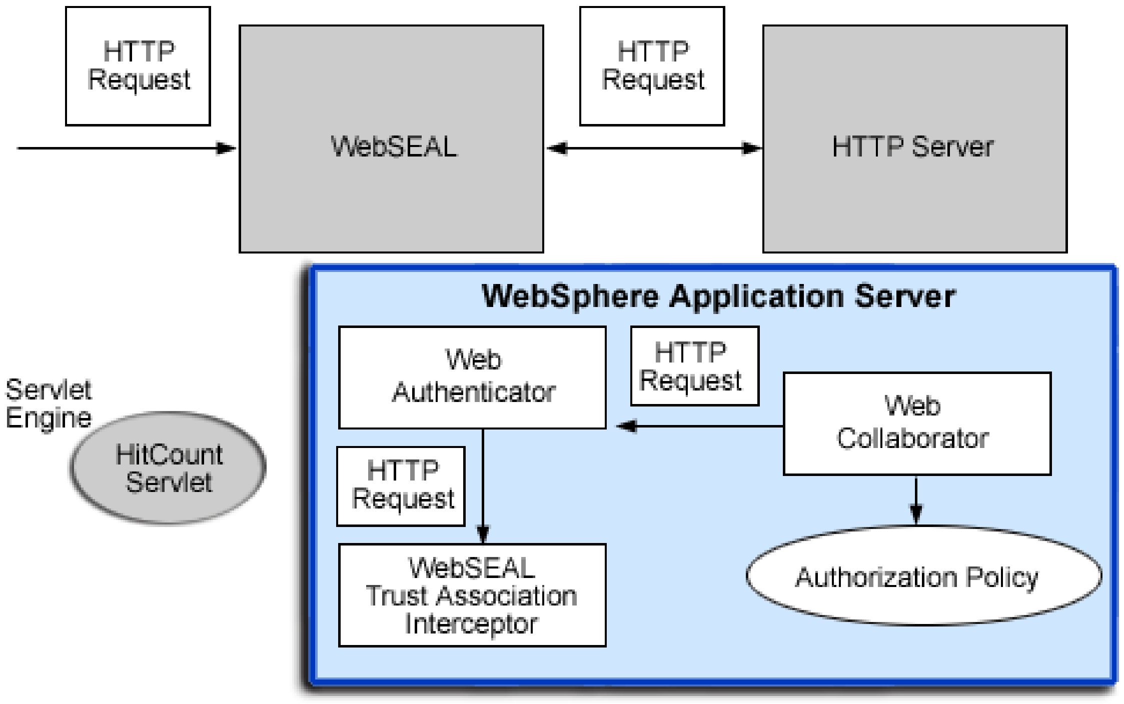
The Policy Director delegates all Web requests to its Web component, the WebSEAL server. One of the major functions of the server is to perform authentication of the requesting user. The WebSEAL server consults an LDAP directory. It can also map the original user ID to another user ID, such as when global single sign-on (GSO) is used.

Policy Director (WebSEAL)

For successful authentication, the server plays the role of a client to WebSphere Application Server - Express when channeling the request. As such, it would need its own user ID and password to identify itself to WebSphere Application Server - Express. This identity must be valid in the security realm of WebSphere Application Server. Thus, WebSEAL replaces the Basic Auth information in the HTTP request with its own user ID and password. In addition, WebSphere Application Server - Express would need to know the user ID of the requesting client so it can base its authorization decision from this user ID, and not from a WebSEAL user ID. This information is transmitted through the HTTP request by creating a header called `iv-user` with the client user ID as its value.

HTTP Server

The junction created in WebSEAL must get to the HTTP server that serves as the product front end. The HTTP Server is shielded however, from knowing that trust association is used. As far as it is concerned, WebSEAL is just another HTTP client, and as part of its normal routines, it sends the HTTP request to the product. The only requirement on the HTTP Server is an SSL configuration using server authentication only. This requirement protects the requests that flow within the junction.



Web collaborator

When trust association is enabled, the Web collaborator manages the interceptors that are configured in the system. It loads and initializes these interceptors when you restart your servers. When a request is passed to WebSphere Application Server - Express by the Web server, the Web collaborator eventually receives the request for a security check. Two actions must be performed:

- The request must be authenticated
- The request must be authorized

The Web authenticator is called to authenticate the request by passing the HTTP request. If successful, a good credential record is returned by the authenticator, which the web collaborator uses to base its authorization on the requested resource. If the authorization succeeds, the Web collaborator indicates to WebSphere Application Server - Express that the security check has succeeded and the requested resource can be served.

Web Authenticator

The Web authenticator is asked by the Web collaborator to authenticate a given HTTP request. Knowing that trust association is enabled, the task of the Web authenticator is to find the appropriate trust association interceptor to direct the request for processing. It does this by querying every available interceptor. If no target interceptor can be found, the Web authenticator processes the request as though trust association is not enabled.

For an HTTP request sent by WebSEAL, the WebSEAL trust association interceptor replies with a positive response to the Web authenticator. Subsequently, the interceptor is asked to validate its trust association with the WebSEAL server and retrieve the user ID of the original user client.

Trust Association Interceptor feature

The intent of the trust association interceptor feature is to have reverse proxy security servers (RPSS) exist as the exposed entry points to perform authentication and coarse-grained authorization, while the WebSphere Application Server - Express enforces further fine-grained access control. Trust associations improve security by reducing the scope and risk of exposure.

In a typical e-business infrastructure, a distributed environment of a company consists of Web application servers, Web servers, legacy systems and one or more RPSS, such as the WebSEAL product from Tivoli. Such reverse proxy servers, front-end security servers or security plug-ins registered within Web servers, guard the HTTP access requests to the Web servers and Web application servers. While protecting access to the Uniform Resource Identifiers (URIs), these RPSS perform authentication, coarse-grained authorization and route the request to the target application server. The credential formed because of the user authentication passing as a part of the request. Unfortunately, most back-end services, including WebSphere Application Server- Express , do not understand the format of the credential information passed to them by the RPSS.

Using the Trust Association Interceptor feature

The following further describes the benefits of the Trust Association Interceptor (TAI) feature:
- Reverse proxy security servers (RPSS) can authenticate WebSphere Application Server - Express users up front and send credential information about the authenticated user to the product so that the product can trust the RPSS to have performed authentication and not prompt the end user for authentication data another time. The strength of the trust relationship between RPSS and the product is based on the criteria of trust association that is particular to a RPSS and enforced through the TAI implementation. This level of trust might need relaxing based on the environment, but the WebSphere Application Server - Express user should be aware of the vulnerabilities in cases where the RPSS is not trusted, based on a security technology.
- The end user credentials most likely are sent in a special format as part of the Hypertext Transfer Protocol (HTTP) headers in the case of RPSS authentication. It may be a special header or a cookie. What data is passed is implementation specific, and the TAI feature considers this fact and accommodates the idea. The TAI implementation works with the credential data and returns a string that represents the end user that WebSphere Application Server - Express uses to enforce security policies.

Change the default SSL keystore and truststore files: To protect the integrity of the messages being sent across the Internet, it is recommended that you change the default SSL keystore and truststore files that are packaged with WebSphere Application Server - Express. A single location is provided where you can specify SSL configurations that can be used among the various WebSphere Application Server - Express features that use SSL including the LDAP user registry, Web Container, and the Authentication Protocol (CSIv2 and SAS). For information on creating new keystore files, see “Using Java keystore files” on page 135.

You can create different keystore and truststore files for different uses or you can create one file that applies to all cases in which the server uses SSL. After you create the new KeyStore and truststore files, specify them in the SSL configuration repertoire. To work with the SSL configuration repertoire, expand **Security** and click **SSL** in the administrative console. You can edit **DefaultSSLConfig** or create a new SSL configuration with a new alias.

If you create a new alias for your new keystore and truststore files, you must also change all of the locations that refer to the SSL configuration alias DefaultSSLConfig. In the administrative console, make the change in each of these locations:
- Security —> User Registries —> LDAP
- Security —> Authentication Protocol —> CSIv2 Inbound Transport
- Security —> Authentication Protocol —> CSIv2 Outbound Transport
- Security —> Authentication Protocol —> SAS Inbound Transport
- Security —> Authentication Protocol —> SAS Outbound Transport
- Servers —> Application Servers —> *app_server* —> Web Container —> HTTP transports —> *host*
- Servers —> Application Servers —> *app_server* —> Server Level Security —> CSIv2 Inbound Transport
- Servers —> Application Servers —> *app_server* —> Server Security —> CSIv2 OutboundTransport
- Servers —> Application Servers —> *app_server* —> Server Security —> SAS Inbound Transport
- Servers —> Application Servers —> *app_server* —> Server Security —> SAS Outbound Transport
- Servers —> Application Servers —> *app_server* —> Administration Services —> JMX Connectors —> SOAPConnector —> Custom Properties —> sslConfig

In this list, *app_server* is the name of your application server and *host* is the value of the **Host** property for an HTTP transport.

Updating the soap.client.props files

The soap.client.props file is used to support secure SOAP connections for administrative tools. See Use wsadmin in a secure environment in the *Administration* topic for more information about configuring secure SOAP connections for administrative tools.

Edit the soap.client.props files to set the following properties for your new client keystore files:
- com.ibm.ssl.keyStore
- com.ibm.ssl.keyStorePassword
- com.ibm.ssl.trustStore
- com.ibm.ssl.trustStorePassword

Note: To encode passwords in your soap.client.props files see Manually encoding passwords in properties files (page 184).

Updating the SSL configuration for the WebSphere Web server plug-in

For more information about updating the SSL configuration for the plug-in, see Configure SSL for WebSphere plug-ins (page 125).

Note: SSL is enabled for the Web server plug-in in the default configuration.

Enable global security: Perform these steps to enable WebSphere Global Security:
1. In the administrative console, click **Security —> Global Security** to configure the rest of the security settings and enable security. When you set up your security configuration, the Global Security configuration should be performed last. This is because this page performs a final validation of the security configuration.
2. Configure the global security settings.

   Make sure that **Enable global security** is selected so security is enabled when the server restarts.

3. Click **OK** or **Apply**. Your configuration is validated. Informational messages appear at the top of the page. If the messages are in red text, a problem has occurred with the security validation. Review your configuration to ensure the user registry settings are accurate. In some cases the LTPA configuration may not be fully specified.
4. If there are no validation problems with your security, then store the configuration for use by the server the next time it is restarted. To save the configuration, click **Save** at the top of the page in the administrative console. This action writes the settings out to the configuration repository.
5. Logoff the administrative console. Click **Logout**.
6. Stop the server:
   a. On an iSeries command line, start Qshell with the STRQSH command.
   b. Change to the /QIBM/ProdData/WebASE/ASE5/bin directory.
   c. Run the stopServer command:
      ```
      /QIBM/ProdData/WebASE/ASE5/bin/stopServer -instance instanceName serverName
      ```
      where *instanceName* is the name of your instance and *server name* is the name of your server.

   Note: After security has been enabled, you need an user ID and password to run the stopServer script.
7. In Qshell, start the server with the startServer command, for example `startServer` *`myServer`*.

If you have any problems restarting the server, review the output logs in the `logs` subdirectory of your instance. See Troubleshooting: Security in the *Troubleshooting* topic for more information.

Assign users to administrative roles

WebSphere Application Server - Express extended J2EE security role based access control to protect the WebSphere Application Server - Express administrative subsystem. Four administrative roles have been defined to provide degrees of authority needed to perform certain WebSphere Application Server - Express administrative functions from either the Web-based administrative console or the system management scripting interface. The authorization policy is only enforced when global security is enabled. The four administrative security roles are defined in the following table:

Role	Description
monitor	The least privileged role which basically allows a user to view the WebSphere Application Server - Express configuration and current state.
configuration	This role has monitor privilege plus the ability to change the WebSphere Application Server - Express configuration.
operator	This role has monitor privilege plus the ability to change runtime state, such as starting or stopping services for example.
administrator	This role has operator and configuration privilege and the permission that is required to access sensitive data including server password, LTPA password and keys, and so on.

When WebSphere Application Server - Express global security is enabled, the administrative subsystem role-based access control is enforced. The administrative subsystem includes Security Server, UserRegistry, and all JMX MBeans. When security is enabled, both the Web-based administrative console and the administrative scripting tool requires users to provide the required authentication data. Moreover, the administrative console is designed so that the control functions that are displayed on the pages are adjusted according to the security roles a user has. For example, a user who has only the monitor role can only see non-sensitive configuration data. A user with the operator role can access buttons to change the system state.

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

In addition to mapping user or groups, a special subject can also be mapped to the administrative roles. A special subject is a generalization of a particular class of users. The AllAuthenticated special subject means that the access check of the administrative role ensures that the user who makes the request has at least been authenticated. The Everyone special subject means that anyone, authenticated or not, can perform the action, as if no security were enabled.

When global security is enabled, WebSphere Application Server - Express servers run under the server identity which is defined under the active user registry configuration. Although it is not shown on administrative console and other tools, a special Server subject is mapped to the administrator role. This is why the WebSphere Application Server - Express server runtime code, which runs under the server identity, would have the required authorization to execute runtime operations. If no other user has been assigned administrative roles, one can login to administrative console or to wsadmin scripting using server identity to perform administrative operations and to assign other users or groups to administrative roles. Because the server identity is assigned to the administrative role by default, the administrative security policy requires administrative role to perform the following operations:

- Change server ID and server password
- Enable or disable WebSphere Application Server - Express global security
- Enforce or disable Java 2 Security
- Change LTPA password or generate keys
- Assign users and groups to administrative roles

When enabling security for the first time, you may perform the following steps to assign one or more users and groups to administrative roles. When global security is enabled, the following steps can be performed by users who have the administrative role. Before performing the following steps, one must configure the active user registry because user and group validation in the following steps depends on active user registry.

To assign users to administrative roles, perform these steps in the WebSphere administrative console:

1. In the administrative console, expand **System Administration**, and click either **Console Users** or **Console Groups**.

   Perform the necessary tasks:
   - To add a user or a group, click **Add**.
   - To add a new administrative user, enter a user identity to the user text box and highlight one administrative role, and then click **OK**. If there is no validation error, the specified user is displayed with the assigned security role. To add a new administrative group, either enter a group name or select either EVERYONE or ALLAUTHENTICATED special subject, and then click **OK**. If there is no validation button, the specified group or special subject is displayed with the assigned security role.
   - To remove a user or group assignment, click the remove button on the Console Users or Console Groups panel. On the users or groups panel, click the check box of the user or group to be removed and then click **OK**.
   - To manage the set of users or groups to be displayed, expand the filter folder on the right side panel, and modify the filter text box. For example, setting the filter to `user*` allows only users with the `user` prefix to be displayed.
2. After modifications have been made, click **Save**.

3. Stop and restart your servers for the changes to take effect. After the server is restarted, all administrative resources are protected. Because the administrative security configuration is at the cell level, restart all servers.

Assign users to naming roles

WebSphere Application Server - Express extended J2EE security role based access control to protect the WebSphere Application Server - Express naming subsystem. CosNaming security offers increased granularity of security control over CosNaming functions. CosNaming functions are available on CosNaming servers such as the WebSphere Application Server - Express. They affect the content of the WebSphere Application Server - Express Name Space. There are generally two ways in which client programs will result in CosNaming calls. The first is through the JNDI interfaces. The second is CORBA clients invoking CosNaming methods directly.

Four security roles are introduced : CosNamingRead, CosNamingWrite, CosNamingCreate, and CosNamingDelete. The name of the four roles are the same with WebSphere Application Server Advanced Edition v4.0.2 However, the roles now have authority level from low to high as follows:

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

Additionally, a Server special subject is assigned to all the four CosNaming roles by default. The Server special subject allows a WebSphere Application Server - Express server process, which runs under the server identity, to have access to all the CosNaming operations. Note that the Server special subject is not displayed and cannot be modified through the administrative console nor other administrative tools.

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

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

In WebSphere Application Server Version 4.0.2, each CosNaming function is assigned to only one role. Therefore, users who have been assigned CosNamingCreate role are not able to query the Name Space unless they have also been assigned CosNamingRead. And in most cases, a creator would need to be assigned three roles: CosNamingRead, CosNamingWrite, and CosNamingCreate. This has been changed in the release. The CosNamingRead and CosNamingWrite roles assignment for the creator example in above have been included in CosNamingCreate role. In most of the cases, WebSphere Application Server administrators do not have to change the roles assignment for every user or group when they move to this release from previous one.

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

To assign users to naming roles, perform these steps:

1. In the administrative console, expand **Environment —> Naming**. Click **CORBA Naming Service Users** or **CORBA Naming Service Groups**.
2. Perform the necessary tasks:
   - Click **Add** on the CORBA Naming Service Users or CORBA Naming Service Groups panel.
   - To add a new naming service user, enter a user identity in the **User** field, highlight one or more naming roles, and click **OK**. If there are no validation errors, the specified user displays with the assigned security role.
   - To add a new naming service group, either select **Specify group** and enter a group name or select **Select from special subject** and then select either EVERYONE or ALL AUTHENTICATED. Click **OK**. If there are no validation errors, the specified group or special subject displays with the assigned security role.
   - To remove a user or group assignment, go to the CORBA Naming Service Users or CORBA Naming Service Groups panel. Select the check box next to the user or group that you want to remove and click **Remove**.
   - To manage the set of users or groups to display, expand the filter folder on the panel on the right side, and modify the filter text box. For example, setting the filter to `user*` allows only users with the ″user″ prefix to be displayed.
3. After modifications have been made, click **Save** to save the mappings. You must restart the server for the changes to take effect.

Configure SSL in WebSphere Application Server - Express

Several WebSphere Application Server - Express components use Secure Socket Layer (SSL) to provide secure communication. In particular, SSL is used by:

- HTTPS, the application server’s built-in HTTPS transport.
- ORB, the application server’s client and server Object Request Broker.
- LDAPS, the administrative server’s secure connection to the LDAP registry used for authentication. This is available only in WebSphere Application Server - Express.

Also, WebSphere applications can be configured and implemented to use SSL. The digital certificates used by such applications can be stored in either Java KeyStore files (.jks files) or i5/OS certificate containers (.kdb files).

Note: Use of i5/OS certificate containers with WebSphere applications is deprecated in Version 5.0 and can only be used with certain limitations. For more information, see “Migrate applications to use Java keystores” on page 150.

Note: The Federal Information Processing Standards (FIPS)-approved Java Secure Socket Extension (JSSE) and Java Cryptography Extension (JCE) providers are not supported on iSeries. Other WebSphere Application Server - Express platforms may support FIPS-approved cryptographic algorithms.

See these topics for more information about configuring SSL:

“Configure SSL for the browser” on page 124

“Configure SSL for Web servers” on page 124

"Configuring IBM HTTP Server for i5/OS for SSL client authentication"

"Configure SSL for WebSphere Application Server" on page 125

"Configure SSL for WebSphere applications" on page 135

"Configure SSL connections between WebSphere Application Server - Express and an LDAP server" on page 152

Configure SSL for the browser: Configuring SSL for the browser is browser-specific. Consult your browser documentation for instructions.

Generally speaking, when the you type `https://...` instead of `http://...`, the browser creates an SSL connection instead of a simple TCP connection to the Web server. The browser then typically either prompts the user or fails to connect if it was unable to validate the Web server or to agree upon the level of security options (the strength of the encryption algorithm to use).

Configure SSL for Web servers: Configuring SSL for the Web server depends on the type of Web server. Consult your Web server documentation for instructions.

Generally speaking, when SSL is enabled, an SSL key file is required. This key file should contain both the CA certificates (signer certificates) as well as any client or server certificates. Client authentication can also be enabled; by default, it is disabled.

When SSL client authentication is enabled in the Web server, the client certificate (the certificate from the browser) is forwarded by the WebSphere Web server plug-in to WebSphere Application Server - Express. When application deployment descriptors specify that client certificate authentication method is used, the application server accepts the forwarded certificate as authentication credentials if the distinguished name attribute of the certificate can be mapped to a principal in the user registry. For more information about mapping client certificates, see "Configure LDAP search filters" on page 94.

Note: The client certificate is forwarded to the application server, regardless of whether SSL is enabled on the plug-in transport. Because SSL authentication of the browser client certificate actually occurs in the Web server, it is strongly recommended that you configure SSL in the plug-in transport. This requires client authentication of the plugin itself when application deployment descriptors specify the use of the client certificate authentication method, and thus, the possibility of receiving a client certificate from an untrusted source is eliminated. For more information about configuring the plug-in transport to require SSL client authentication, see "Configuring SSL for the application server's HTTPS transport" in "Configure SSL for WebSphere Application Server" on page 125.

Configuring IBM HTTP Server for i5/OS for SSL client authentication: Use the Configuration and Administration forms of the IBM HTTP Server for i5/OS to configure your IBM HTTP Server for secure sockets layer (SSL) client authentication. For more information, including prerequisites for SSL security, see these topics in the iSeries Information Center:

- V5R4 Secure Sockets Layer (SSL)
- V5R3 Secure Sockets Layer (SSL)
- V5R2 Secure Sockets Layer (SSL)

Use the IBM HTTP Server for i5/OS configuration and administration forms to create a virtual host and configure the port for SSL:

1. In the **Server** field, select your HTTP server instance.
2. In the left pane, click **General Server Configuration**.
3. In the right pane, click the **General Settings** tab.
4. Under **Server IP addresses and ports to listen on**, click **Add**. Specify values for these fields:

- **IP address**: Type `*` or select `All IP addresses` from the menu.
- **Port**: Enter the port number you want to protect with SSL.
- **FRCA**: If this field is present, set it to `Disabled`.

5. Click **OK**.
6. In the right pane, select **Global Configuration** in the **Server area**.
7. In the left pane, click **Container Management**.
8. In the right pane, select the **Virtual Hosts** tab.
9. Click **Add**.
10. For the **IP address or host name** field, select **All IP addresses**.
11. For the **Port** field, enter the port number you wish to protect with SSL.
12. Click **OK**.
13. Select your new virtual host in the **Server area**.
14. In the left pane, click **Security**.
15. In the right pane, select **Enable SSL**.
16. For the **Server certificate application name** field, select the automatically generated Application ID (QIBM_HTTP_SERVER_LDH for example).
17. Select **Require client certificate for connection**.
18. Click **OK**.

To complete this task you need the Application ID you selected above to install a server certificate for your Web server. Use the IBM Digital Certificate Manager (DCM) to install certificates on your Web server and Web browsers. See these topics in the iSeries Information Center:
- V5R4 Digital Certificate Manager
- V5R3 Digital Certificate Manager
- V5R2 Digital Certificate Manager

Configure SSL for WebSphere Application Server: See these topics for instructions on configuring SSL for WebSphere Application Server:
- Configure SSL for WebSphere plug-ins (page 125)
  - Using the product-provided certificates to configure SSL for WebSphere plug-ins (Version 5.0.1 and later) (page 126)
  - Creating an SSL key file for the WebSphere Web server plug-in (page 126)
- Configuring SSL for the application server's HTTPS transport (page 127)
  1. Create an SSL key file without the default signer certificates (page 127)
  2. Add the signer certificate of the application server to the plug-in's SSL key file (page 128)
  3. Grant access to the key files (page 128)
  4. (Optional) Configure an alias for the SSL port (page 129)
  5. Configure HTTPS transport for the Web container (page 129)

Note: For these steps, it is assumed that you have a network drive mapped from your workstation to your iSeries system.

Configure SSL for WebSphere plug-ins

A WebSphere plug-in interfaces with a Web server to handle client requests for server-side resources and routes them to the application server for processing. WebSphere Application Server - Express includes plug-ins for IBM HTTP Server for i5/OS and Domino Web Server for iSeries.

After SSL is working between your browser and Web server, proceed to configure SSL between the Web server plug-in and the WebSphere Application Server product. This is not required if the link between the plug-in and application server is known to be secure or if your applications are not sensitive. If privacy of application data is a concern, however, this connection should be an SSL connection.

Using the product-provided certificates to configure SSL for WebSphere plug-ins (Version 5.0.1 and later)

WebSphere Application Server - Express Version 5.0.1 (and later) updates server instances with an SSL key file. The pathname for the key file is /QIBM/UserData/WebASE/ASE5/*instance*/etc/plugin-key.kdb, where *instance* is the name of your server instance. (The remainder of this topic refers to this path as USER_INSTALL_ROOT/etc.) However, server instances are updated only if the plugin-key.kdb file is not already present (that is, if the key file was not manually created prior to the Version 5.0.1 installation).

The plugin-key.kdb file that is used in the update process contains a digital certificate that is required for the Web server plug-in to trust the signer of the Web container's certificate when an HTTPS transport is configured with the default SSL repertoire.

Using the product-provided certificates to configure SSL for the WebSphere plug-ins significantly reduces configuration complexity, but they should not be used for production servers. The tasks below demonstrate how to create your own certificates. Alternatively, you can obtain certificates from a commercial certificate authority.

Creating an SSL key file for the WebSphere Web server plug-in

If you are using the key file that is provided with the product (as of Version 5.0.1) to configure SSL for WebSphere plug-ins, skip this task and proceed to Configure an alias for the SSL port (page 129). However, first you should ensure that you have the Cryptographic Access Provider licensed program (5722-AC3) installed on the iSeries system that hosts your Web server.

When configuring SSL, you must first create an SSL key file.

The following is an example of how to create an SSL key file for your WebSphere plug-in:

1. "Start the Digital Certificate Manager" on page 130.
   Procedures vary depending on the release of Digital Certificate Manager (DCM) you have installed on your iSeries system. The release of DCM used in this topic is V5R1M0.
2. "Create a local certificate authority" on page 131.
   Skip this step if you already have a certificate authority (CA) created on you iSeries system.
3. Create a key store for the HTTP server plug-in:
   a. In the left pane, click **Create New Certificate Store**.
   b. Select **Other System Certificate Store** and click the **Continue** button.
   c. On the Create a Certificate in New Certificate Store page, select **Yes - Create a certificate in the certificate store**, and click **Continue**.
   d. On the Select a Certificate Authority (CA) page, select **Local Certificate Authority** and click the **Continue** button.
   e. Fill in the form to create a certificate and certificate store. Use this pathname for the certificate store:

      `/QIBM/UserData/WebASE/ASE5/`*`instance`*`/etc/plugin-key.kdb`

      where *instance* is the name of your instance. (The remainder of these instructions refers to the directory above `etc` as USER_INSTALL_ROOT.)

      Use `MyPluginCert` as the key label. Fill in the other required fields, and then click **Continue**.
4. Set the default system certificate:
   a. In the left pane, click to expand **Fast Path**.

   b. Select **Work with server and client certificates**.
   c. Select certificate MyPluginCert.
   d. Click **Set default**.
5. Remove all trusted signers except the Local CA:
   a. On the left pane, click **Select a Certificate Store**
   b. Select **Other System Certificate Store** and click **Continue**.
   c. On the Certificate Store and Password page, enter the **Certificate store path and filename** (USER_INSTALL_ROOT/etc/plugin-key.kdb) and the password. Click **Continue**.
   d. On the left pane, click **Fast Path**.
   e. Select **Work with CA certificates** and click **Continue**.
   f. On the Work with CA Certificates page, for all CA certificates except the LOCAL_CERTIFICATE_AUTHORITY, select the certificate and then click **Delete**. Respond with **Yes** when asked if you are sure you want to delete this certificate.
6. Extract the Local CA certificate so that you can import the certificate into the application server key file later:
   a. In the left pane, click **Install CA certificate on your PC**.
   b. In the right pane, click **Copy and paste certificate**.
   c. Create text file USER_INSTALL_ROOT/etc/myLocalCA.txt on your workstation's mapped drive to the iSeries, then paste the CA certificate into `myLocalCA.txt` and save the file. For example, if you want to configure the instance that is named `myInstance`, and your workstation's F: drive is mapped to the iSeries system, the path is F:\QIBM\UserData\WebASE\ASE5\myInstance\etc\myLocalCA.txt. Ensure that the copy of the CA certificate ends with the new line character.
   d. Click **Done**.

Use SSL configuration repertoires to manage SSL settings for resources in the administrative domain. The default repertoire is DefaultNode/DefaultSSLSettings. You can use DefaultNode/DefaultSSLSettings for testing or create new SSL configuration repertoires for production applications and associate them with individual resources. For more information, see "Use SSL configuration repertoires" on page 131.

Configuring SSL for the application server's HTTPS transport

To configure SSL, you must first create an SSL key file. The contents of this file depend on whom you want to allow to communicate directly with the application server over the HTTPS port (in other words, you are defining the HTTPS server security policy).

This topic presents a restrictive security policy, in which only a well-defined set of clients (those whose certificates are signed by your local certificate authority) are allowed to connect to the application server HTTPS port. It is recommended that you follow this security policy when your application's deployment descriptor specifies the use of the client certificate authentication method. The procedure for creating an SSL key file without the default signer certificates conforms to this policy.

To configure SSL for the application server's HTTPS transport, follow these steps:

Step 1: Create an SSL key file without the default signer certificates.
1. Start iKeyman on your workstation. For more information, see "The iKeyman utility" on page 132.
2. Create a new key database file:
   a. Click **Key Database File** and select **New**.
   b. Specify settings:
      - **Key database type**: JKS
      - **File Name**: appServerKeys.jks

    - **Location**: your etc directory, such as USER_INSTALL_ROOT/etc
   c. Click **OK**.
   d. Enter a password (twice for confirmation) and click **OK**.
3. Delete all of the signer certificates.
4. Click **Signer Certificates** and select **Personal Certificates**.
5. Add a new self-signed certificate:
   a. Click **New Self-Signed** to add a self-signed certificate.
   b. Specify settings:
    - **Key Label**: appServerTest
    - **Common Name**: use the DNS name for your iSeries server
    - **Organization**: IBM
   c. Click **OK**.
6. Extract the certificate from this self-signed certificate so that it can be imported into the plug-in's SSL key file:
   a. Click **Extract Certificate**.
   b. Specify settings:
    - **Data Type**: Base64-encoded ASCII data
    - **Certificate file name**: appServer.arm
    - **Location**: the path to your etc directory
   c. Click **OK**.
7. Import the Local CA public certificate:
   a. Click **Personal Certificates** and select **Signer Certificates**.
   b. Click **Add**.
   c. Specify settings:
    - **Data Type**: Base64-encoded ASCII data
    - **Certificate file name**: myLocalCA.txt
    - **Location**: the path to your etc directory
   d. Click **OK**.
8. Enter `plug-in` for the label and click **OK**.
9. Click **Key Database File**.
10. Select **Exit**.

Step 2: Add the signer certificate of the application server to the plug-in's SSL key file.
1. "Start the Digital Certificate Manager" on page 130
2. On the left pane, click **Select a Certificate Store**
3. Select **Other System Certificate Store** and click **Continue**.
4. On the Certificate Store and Password page, enter the **Certificate store path and filename** (USER_INSTALL_ROOT/etc/plugin-key.kdb) and the password, then click **Continue**.
5. On the left pane, click **Fast Path**.
6. Select **Work with CA certificates** and click **Continue**.
7. Click **Import**.
8. Specify USER_INSTALL_ROOT/etc/appServer.arm for the **Import file** field value and click **Continue**.
9. Specify appServer for the **CA certificate label** field value and click **Continue**.

Step 3: Grant access to the key files.

It is very important to protect your key files from unauthorized access. Set the following protections by using the i5/OS Change Authority (CHGAUT) command:

- appServerKeys.jks

PROFILE	ACCESS
*PUBLIC	*EXCLUDE
QEJBSVR	*R

- plugin-key.kdb

PROFILE	ACCESS
*PUBLIC	*EXCLUDE
QTMHHTTP	*RX

- All other files you created in the USER_INSTALL_ROOT/etc directory should have *EXCLUDE authority set for *PUBLIC.

Note: QTMHHTTP is the default user profile for the IBM HTTP Server for i5/OS. If your Web server runs under another profile, grant that profile *RX authority for plug-inKeys.kdb instead of QTMHHTTP.

For example, to grant read and execute (*RX) authority for plugin-key.kdb to the QTMHHTTP user profile, run the Change Authority (CHGAUT) command. For example:

```
CHGAUT OBJ('/QIBM/UserData/WebASE/ASE5/etc/plugin-key.kdb')
       USER(QTMHHTTP) DTAAUT(*RX)
```

Step 4: (Optional) Configure an alias for the SSL port

If you have not already configured an alias for your Web server's SSL port in your WebSphere virtual host, do so now.

Step 5: Configure HTTPS transport for the Web container

For more information, see "Configure HTTPS transport for your application server's Web container" on page 134.

Note: Configuring the WebSphere Web plug-in for SSL can require manual updates to the plug-in configuration file. Manual changes can be lost when the plug-in configuration file is regenerated. If you have manually changed the plug-in configuration file, check the file to see determine if your changes have been lost, and reapply them if necessary.

No manual update of the plug-in configuration file is required if you are using the key file that is provided with the product (as of Version 5.0.1) to configure SSL for the Web server plug-in. Your regenerated plug-in configuration file should contain an entry that is similar to the following:

```
<Transport Hostname="MYISERIES" Port="10175" Protocol="https">
  <Property name="keyring" value="/QIBM/UserData/WebASE/ASE5/myInstance/etc/plugin-key.kdb"/>
  <Property name="stashfile" value="/QIBM/UserData/WebASE/ASE5/myInstance/etc/plugin-key.sth"/>
</Transport>
```

The configuration is complete.

As an alternative, you can implement an even more restrictive security policy by configuring the plugin to use a self signed certificate for authenticating to the application server's Web container. Assuming you have successfully completed all steps in the above task, follow these steps to implement this more restrictive policy:

1. Use iKeyman to create a keystore.

2. Create a self signed certificate in the keystore.
3. Export the self signed certificate (with the private key) from the keystore.
4. Extract the self signed certificate (also known as a signer certificate since it doesn't contain the the private key) from the keystore.
5. Again using iKeyman, add the extracted signer certificate to the HTTPS transport's trust store (appServerKeys.jks in the above example).
6. Remove all other signer certificates from the HTTPS transport's trust store.
7. Using DCM, import the self signed certificate (with the private key) into the plugin's key store (plugin-key.kdb). Record the label you use when importing the certificate.

   Note: DCM treats self signed certificates as signer certificates and adds the certificate to the list of signer certificates, even though the certificate contains a private key.
8. Restart the application server.
9. Regenerate the Web plugin configuration file.
10. Specify the certificate the plugin is to use for authenticating to the Web container by manually adding the certLabel property to the HTTPS transport in the Web plugin configuration file (USER_INSTALL_ROOT/config/cell/plugin-cfg.xml). Set the certLabel property value to the label you used when importing the self signed certificate into the plugin's key store. For example:

```
<Transport Hostname="MYISERIES" Port="10175" Protocol="https">
  <Property name="keyring"
   value="/QIBM/UserData/WebASE/ASE5/myinst/etc/plugin-key.kdb"/>
  <Property name="certLabel" value="selfsigned"/>
</Transport>
```
11. Restart the Web server.

Start the Digital Certificate Manager: Digital Certificate Manager (DCM) is an iSeries software product that allows you to create and manage digital certificates through a Web-based administration tool. For more information about DCM, see these topics in the iSeries Information Center:

- V5R4 Digital Certificate Manager
- V5R3 Digital Certificate Manager
- V5R2 Digital Certificate Manager

To start the DCM, perform these steps:

1. Start the *ADMIN instance of IBM HTTP Server for i5/OS.

   You can start the *ADMIN server instance from the CL command line or from iSeries Navigator:
   - **From the CL command line**

     To start the *ADMIN instance from the CL command line, type:

     ```
     STRTCPSVR SERVER(*HTTP) HTTPSVR(*ADMIN)
     ```

     and press **Enter**.
   - **From iSeries Navigator**

     iSeries Navigator is the graphical interface to the iSeries server. iSeries Navigator is part of the iSeries Access for Windows product.

     Note: For more information on iSeries Access for Windows and iSeries Navigator, see iSeries Access for Windows

     .

     To start the *ADMIN instance from iSeries Navigator, follow these steps:

     a. Start iSeries Navigator.
     b. Double-click your iSeries server.
     c. Select **Network —> Servers —> TCP/IP**.

   d. Right-click **HTTP Administration** in the right frame.
   e. Select **Start**.

      Note: Do not select the **Start Instance** option from the menu. If the server is already started, the **Start** option is disabled on the menu.
2. Start your JavaScript-enabled browser. Enter this URL in the URL location or address window:

   `http://your.server.name:2001`

   where *your.server.name* is the host name of your iSeries system.
3. You are prompted for an i5/OS user profile name and password. Your i5/OS user profile must have all object (*ALLOBJ) special authority. The iSeries Tasks page appears.
4. Click **Digital Certificate Manager**. The Digital Certificate Manager page appears.

Create a local certificate authority: This procedure varies depending on the release of DCM you have installed on your iSeries system. The following procedure is for version 5 release 1 modification level 0 (V5R1M0). See the following topics for later release procedures:
- V5R4 Create and operate a Local CA
- V5R3 Create and operate a Local CA
- V5R2 Create and operate a Local CA

To create a local certificate authority (CA), perform these steps:
1. In the left pane, click **Create a Certificate Authority (CA)**.
2. Fill in the fields and click **Continue**. The system creates a certificate authority certificate and stores it in the Local Certificate Authority (CA) certificate store.

   Note: Record the password used in this procedure in a secure location for future reference.
3. On the Install Local CA Certificate page, click **Continue**.
4. On the Certificate Authority (CA) Policy Data page, specify the policy data you want and click **Continue**.
5. On the Policy Data Accepted page, click **Continue**.
6. Fill in the fields and click **Continue**.

   The system creates a certificate with a private key and store the certificate in the default server certificate store (*SYSTEM).
7. On the Select Applications page, click **Continue**.
8. On the Select Applications Status page, click **Continue**.
9. Fill in the fields and click **Continue**.

   The system creates a certificate with a private key and stores the certificate in the default object signing certificate store (*OBJECTSIGNING).
10. On the Select Applications page, click the **OK** button.

    This completes the process of setting up your system as a Certificate Authority (CA).

Use SSL configuration repertoires: An SSL repertoire contains the details necessary for building an SSL connection, such as the location of the key files, their type and the available ciphers. WebSphere Application Server - Express provides a default repertoire called DefaultSSLSettings. To view this page in the administrative console, click **Security —> SSL** to see the list of SSL repertoire settings.

Note: It is not recommended to use the default repertoire in a production environment. For more information, see “Change the default SSL keystore and truststore files” on page 118.

The appropriate repertoire is referenced during the configuration of a service that sends and receives requests encrypted using SSL, such as the Web container. Before deleting SSL configurations from the repertoire, remember that if an SSL configuration alias is referenced somewhere, and it is deleted here, an SSL connection fails if the deleted alias is accessed.

The SSL configuration repertoire allows administrators to define any number of SSL settings which can be used to make HTTPS, IIOPS, or LDAPS connections. You can pick one of the SSL settings defined here from any location within the administrative console which allows SSL connections. This simplifies the SSL configuration process since you can reuse many of these SSL configurations by simply specifying the alias in multiple places.

To create an SSL repertoire, perform these steps in the WebSphere administrative console:

1. In the navigation menu, expand **Security** and then click **SSL**.
2. From the SSL Configuration Repertoire window, click **New**. Type an Alias by which the configuration is known. Click **OK**.
3. Select the new SSL configuration repertoire by clicking the link.
4. Now click **Secure Sockets Layer (SSL)** under in Additional Properties. The new configuration details can be entered in the window that appears.
5. Type the location of the key file name.
6. Type the password for the key file.
7. Repeat the above two steps for the trust file.
8. If Client Authentication is supported by this configuration, then select **Client Authentication**. This only affects HTTP and LDAP requests.
9. The appropriate security level must be set. Valid values are as follows:
   - **Low**
     Specifies only digital signing ciphers (no encryption).
   - **Medium**
     Specifies only 40-bit ciphers (including digital signing).
   - **High**
     specifies only 128-bit ciphers (including digital signing).
10. If the preset security level does not define the required cipher, it can be manually added to the cipher suite option.
11. Note that hardware or software cryptographic support is not available on the iSeries system. The **Cryptographic Token** setting is not applicable to iSeries.
12. Select **IBMJSSE** as the JSSE provider.
13. Select an SSL protocol version.
14. Click **OK** to apply the changes.
15. If there are no errors, save the changes to the master configuration and restart WebSphere Application Server - Express.

The iKeyman utility: The iKeyman utility is a graphical user interface (GUI) based tool that you can use to manage your digital certificates. With iKeyman, you can create a new key database or test a digital certificate, add certificate authority (CA) roots to your database, copy certificates form one database to another, request and receive a digital certificate from a CA, set default keys, and change passwords.

The iKeyman utility is a part of the IBM Java Security Socket Extension package and is shipped with the WebSphere Application Server - Express product. It is recommended that you download the iKeyman utility to a workstation that supports graphical interfaces.

Set up the iKeyman utility

To set up the iKeyman utility to work with your digital certificates, follow these steps:

1. If you have not already done so, install one of these Java environments on your workstation:
   - IBM Developer Kit for Java, Version 1.3 or later
   - IBM Runtime Environment, Java Edition, Version 1.3 or later
   - Sun Microsystems, Inc. Java 2 Software Development Kit, Version 1.3 or later

- Sun Microsystems, Inc. Java 2 Runtime Environemnt, Version 1.3 or later

2. Download the iKeyman program files to your workstation.

   You can map a network drive to your iSeries system or use file transfer protocol (FTP) to copy the files to your workstation system.

   These are the iKeyman program files:
   - /QIBM/ProdData/WebASE/ASE5/lib/gskikm.jar
   - /QIBM/ProdData/WebASE/ASE5/java/ext/ibmjcefw.jar
   - /QIBM/ProdData/WebASE/ASE5/java/ext/ibmjceprovider.jar
   - /QIBM/ProdData/WebASE/ASE5/java/ext/ibmpkcs.jar
   - /QIBM/ProdData/WebASE/ASE5/java/ext/ibmpkcs11.jar
   - /QIBM/ProdData/WebASE/ASE5/java/ext/local_policy.jar
   - /QIBM/ProdData/WebASE/ASE5/java/ext/US_export_policy.jar

   Place the files in the jre/lib/ext subdirectory of your Java environment product directory. For example, on a Windows 32-bit system:
   - C:\Program Files\IBM\Java13\jre\lib\ext
   - D:\jdk13\jre\lib\ext

3. On your workstation, update the java.security file for your Java environment.

   The java.security file is located in the jre/lib/security subdirectory of your Java environment. Open the file in a text editor, and look for an entry similar to this one:

   ```
   security.provider.1=sun.security.provider.Sun
   security.provider.2=com.sun.rsajca.Provider
   ```

   Add this line to the end of the entry:

   ```
   security.provider.3=com.ibm.crypto.provider.IBMJCE
   ```

   If you use PKCS11 hardware cryptography support, also add this entry:

   ```
   security.provider.4=com.ibm.crypto.pkcs11.provider.IBMPKCS11
   ```

   Save the java.security file.

4. (Windows workstations only) Create a batch (BAT) file to run iKeyman.

   If your workstation is a Windows system, you can create a batch file to start iKeyman. Create a batch file similar to the following:

   ```
   setlocal
   set JAVA_HOME=java_root
   set PATH=%JAVA_HOME%\jre\bin;%JAVA_HOME%\bin;%PATH%
   java  com.ibm.gsk.ikeyman.Ikeyman
   endlocal
   ```

   where *java_root* is the root directory of your Java environment, for example, C:\jdk1.3.0_02.

Start the iKeyman utility

If you created a batch file to start iKeyman, run the batch file.

If you did not create a batch file, you can start iKeyman from a prompt by entering the following command:

1. Open a command prompt on your workstation.
2. Change to the directory that contains the iKeyman program files. This is *java_root*/jre/lib/ext, where *java_root* is the root directory of your Java environment product directories.
3. If your Java utilities (such as the `java` command) are not configured in your system path, enter these commands, where *java_root* is the root directory of your Java environment, for example, C:\jdk1.3.0_02.:

```
set JAVA_HOME=java_root

set PATH=%JAVA_HOME%\jre\bin;%JAVA_HOME%\bin;%PATH%
```

4. Enter this command:
   ```
   java com.ibm.gsk.ikeyman.Ikeyman
   ```

Using the iKeyman utility

For more information about using the iKeyman utility, see the iKeyman User Guide, which is located in the WebSphere Application Server - Express product directories: /QIBM/ProdData/WebASE/ASE5/web/docs/ikeyman/ikmuserguide.pdf

Configure HTTPS transport for your application server's Web container: Perform these steps in the WebSphere administrative console:

1. Start the administrative console.
2. If you are using the key file that is provided with the product (Version 5.0.1 and later) to configure SSL for the Web server plug-in, skip to step Configure an HTTPS transport (page 134).

   Perform the following steps to create an SSL repertoire:
   a. In the left pane, expand **Security**
   b. Click **SSL**
   c. In the right hand pane, click **New**
   d. Specify the following configuration settings:
      - **Alias**: (the name of your SSL repertoire, for example `mySSLSettings`)
      - **Key File Name**: *USER_INSTALL_ROOT*/etc/appServerKeys.jks, for example /QIBM/UserData/WebASE/ASE5/myInstance/etc/appServerKeys.jks.
      - **Key File Password**: Enter your password
      - **Key File Format**: Select JKS
      - **Trust File Name**: *USER_INSTALL_ROOT*/etc/appServerKeys.jks

        Note: Typically, you would create a separate trust file for your signer certificates. However, a previous step added the certificate for the CA that signed the plug-in's certificate to appServerKeys.jks, we use appServerKeys.jks here, also.
      - **Trust File Password**: Enter your password
      - **Trust File Format**: Select **JKS**
      - **Client Authentication**: selected
   e. Click **OK**
3. Configure an HTTPS transport:
   a. In the left pane, expand **Servers**
   b. Click **Application Servers**
   c. Int the right hand pane, click your application server name
   d. Click the **Configuration** tab
   e. Click **Web Container**
   f. Click **HTTP transports**
   g. Click **New**
   h. Specify the following configuration settings:
      - **Host**: *
      - **Port**: Enter the port number to use for your Web container's SSL port
      - **SSL Enabled**: select **Enable SSL**

- **SSL**: If you are using the key file that is provided with the product (as of Version 5.0.1) to configure SSL for the Web server plug-in, select the DefaultSettings SSL repertoire. Otherwise, select mySSLSettings.

4. Click **OK**
5. Save your changes.
6. Restart your application server.
7. Start the administrative console.
8. In the left hand pane of the administrative console, expand **Environment** and click **Update Web Server Plugin**.
9. Click **OK**.
10. If you previously made manual changes to the Web server plugin configuration file (*USER_INSTALL_ROOT*/config/cell/plugin-cfg.xml), you may need to manually reapply those changes before restarting the Web server.
11. Restart the Web server to immediately pick up changes to the Web server plugin configuration file.

Configure SSL for WebSphere applications: With SSL, client applications use digital certificates to determine whether server applications are to be trusted. Likewise, server applications may require client applications to present certificates to determine whether client applications are to be trusted. In either case, applications must have access to digital certificates to open secure network connections, and these certificates must be stored in safe and secure containers that are accessable to the client applications. Additionally, mechanisms must be provided so applications can determine which certificates are to be used.

For Java applications running on WebSphere Application Server - Express for iSeries, certificates may be stored in these types of certificate stores:

- **Java keystore files (.jks files)**
  Use either the IBM Key Management tool (iKeyman) or the Keytool utility to create and manage Java keystore files. Java keystore files are the recommended certificate stores for WebSphere Application Server - Express 5.0. For more information, see "Using Java keystore files."
- **i5/OS certificate stores**
  Use of i5/OS certificate containers (.kdb files) with WebSphere applications is deprecated in Version 5.0 and can only be used with certain limitations. For more information, see "Migrate applications to use Java keystores" on page 150.

Using Java keystore files: If you are porting a JSSE application from another platform, or require the Java JSSE interfaces to certificate storage using Java keystore files, or require access to miscellaneous SSL implementation classes such as com.ibm.net.ssl.SSLContext, use the configuration steps below to use Java JSSE. Also, you may use Java keystore files for applications that use the java.net.URL class to provide a direct connection to the Web server through HTTPS protocol. For more information, see "Configure SSL for java.net.URL HTTPS protocol" on page 138.

Configure the client Java keystore

This step may be omitted if you already have a client Java keystore file populated with the required personal and signer certificates.

To configure the client Java keystore, create an SSL key file that is used for both trust validation and key storage. Peform these steps:

1. Start the iKeyman utility on your workstation. For more information, see "The iKeyman utility" on page 132.
2. Create a new key database file:
   a. Click **Key Database File** and select **New**.
   b. Specify settings:

        - **Key database type**: JKS
        - **File Name**: clientAppKeys.jks
        - **Location**: your myKeys directory, such as `WAS_INSTANCE_ROOT/myKeys`
    c. Click **OK**.
    d. Enter a password (twice for confirmation) and click **OK**.
3. Click **Signer Certificates** and select **Personal Certificates**.
4. Add a new self-signed certificate:
    a. Click **New Self-Signed** to add a self-signed certificate.
    b. Specify settings:
        - **Key Label**: clientAppTest
        - **Common Name**: use the DNS name for your iSeries server
        - **Organization**: IBM
    c. Click **OK**.
5. Extract the certificate from this self-signed certificate so that it can be imported into the server application's SSL key file:
    a. Click **Extract Certificate**.
    b. Specify settings:
        - **Data Type**: Base64-encoded ASCII data
        - **Certificate file name**: clientAppsCA.arm
        - **Location**: the path to your myKeys directory
    c. Click **OK**.
6. Import the server application's CA certificate from the serverAppKeys.jks file:
    a. Click **Personal Certificates** and select **Signer Certificates**.
    b. Click **Add**.
    c. Specify settings:
        - **Data Type**: Base64-encoded ASCII data
        - **Certificate file name**: serverAppsCA.arm
        - **Location**: the path to your myKeys directory
    d. Click **OK**.
7. Enter serverAppsCA for the label and click **OK**.
8. Click **Key Database File**.
9. Select **Exit**.

Configure the server Java keystore

This step may be omitted if you already have a server Java keystore file populated with the required personal and signer certificates.

To configure the server Java keystore, create an SSL key file used for both trust validation and key storage. Perform these steps:

1. Start iKeyman on your workstation. For more information, see "The iKeyman utility" on page 132.
2. Create a new key database file:
    a. Click **Key Database File** and select **New**.
    b. Specify settings:
        - **Key database type**: JKS
        - **File Name**: serverAppKeys.jks
        - **Location**: your myKeys directory, such as `WAS_INSTANCE_ROOT/myKeys`

   c. Click **OK**.
   d. Enter a password (twice for confirmation) and click **OK**..
3. Click **Signer Certificates** and select **Personal Certificates**.
4. Add a new self-signed certificate:
   a. Click **New Self-Signed** to add a self-signed certificate.
   b. Specify settings:
      - **Key Label**: serverAppTest
      - **Common Name**: use the DNS name for your iSeries server
      - **Organization**: IBM
   c. Click **OK**.
5. Extract the certificate from this self-signed certificate so that it can be imported into the client application's SSL key file:
   a. Click **Extract Certificate**.
   b. Specify settings:
      - **Data Type**: Base64-encoded ASCII data
      - **Certificate file name**: serverAppsCA.arm
      - **Location**: the path to your myKeys directory
   c. Click **OK**.
6. Import the client application's CA certificate from the clientAppKeys.jks file:
   a. Click **Personal Certificates** and select **Signer Certificates**.
   b. Click **Add**.
   c. Specify settings:
      - **Data Type**: Base64-encoded ASCII data
      - **Certificate file name**: clientAppsCA.arm
      - **Location**: the path to your myKeys directory
   d. Click **OK**.
7. Enter clientAppsCA for the label and click **OK**.
8. Click **Key Database File**.
9. Select **Exit**.

Example client JSSE application code

Note that your application code cannot use `SocketFactory socketFactory = SSLSocketFactory.getDefault()` to obtain the SocketFactory unless `os400.jdk13.jst.factories=true` is specified as either a command line Java virtual machine system property or a security property in the java.security file.

For fully supported use of Java keystore files, two other properties which can only be specified in the java.security file must also be set as follows:

```
ssl.SocketFactory.provider=com.ibm.jsse.JSSESocketFactory
ssl.ServerSocketFactory.provider=com.ibm.jsse.JSSEServerSocketFactory
```

The default java.security file in the properties directory provided for each user instance sets the three properties as follows:

```
os400.jdk13.jst.factories=true
ssl.SocketFactory.provider=com.ibm.jsse.JSSESocketFactory
ssl.ServerSocketFactory.provider=com.ibm.jsse.JSSEServerSocketFactory
```

See "Example: JSSE client servlet" on page 141. The client keystore must be placed in the working directory of WebSphere Application Server - Express.

Example server JSSE application code

Your application code cannot use `ServerSocketFactory serverSocketFactory = SSLServerSocketFactory.getDefault()` to obtain the ServerSocketFactory unless `os400.jdk13.jst.factories=true` is specified as either a command line Java virtual machine system property or a security property in the java.security file.

For fully supported use of Java keystore files, two other properties which can only be specified in the java.security file must also be set as follows:

```
ssl.SocketFactory.provider=com.ibm.jsse.JSSESocketFactory
ssl.ServerSocketFactory.provider=com.ibm.jsse.JSSEServerSocketFactory
```

The default java.security file in the properties directory provided for each user instance sets the three properties as follows:

```
os400.jdk13.jst.factories=true
ssl.SocketFactory.provider=com.ibm.jsse.JSSESocketFactory
ssl.ServerSocketFactory.provider=com.ibm.jsse.JSSEServerSocketFactory
```

See "Example: JSSE server servlet" on page 146. The server keystore must be placed in the working directory of the WebSphere Application Server - Express.

Configure SSL for java.net.URL HTTPS protocol: The java.net.URL class provides a direct connection to the Web server to retrieve the specified URL using the HTTPS protocol.

Configuring SSL for the Web server depends on the type of Web server. Consult your Web server documentation for instructions.

Configure the client Java keystore

If you already have a client Java keystore file that is populated with the required personal and signer certificates, you can omit this step.

To configure the client Java keystore, use Digital Certificate Manager (DCM) to extract the Local Certificate Authority (CA) certificate that is used by the Web server. You can then import the certificate into the client Java keystore file.

Perform these steps:
1. "Start the Digital Certificate Manager" on page 130.
2. Create a Local Certificate Authority (CA). If you already have a certificate authority created on your iSeries system, skip this step.
3. On the left pane, click **Select a Certificate Store**.
4. Select ***System** and click **Continue**.
5. On the Certificate Store and Password page, enter the password, then click **Continue**.
6. In the left pane, click **Install CA certificate on your PC**.
7. In the right pane, click **Copy and paste certificate**.
8. Create text file `*USER_INSTALL_ROOT*/etc/myLocalCA.txt` on your PC, then paste the CA certificate into `myLocalCA.txt` and save the file. Ensure that the copy of the CA certificate ends with the new line character.
9. Click the **Done** button.

Next, create a new key database file for the client Https application:

1. Start iKeyman on your workstation. For more information, see "The iKeyman utility" on page 132.
2. Create a new key database file:
   a. Click **Key Database File** and select **New**.
   b. Specify settings:
      - **Key database type**: JKS
      - **File Name**: httpsClientKeys.jks
      - **Location**: your etc directory, such as
        `USER_INSTALL_ROOT/etc/myKeys`
   c. Click **OK**.
   d. Enter a password (twice for confirmation) and click **OK**.
3. Delete all of the signer certificates.
4. Click **Signer Certificates** and select **Personal Certificates**.
5. Add a new self-signed certificate:
   a. Click **New Self-Signed** to add a self-signed certificate.
   b. Specify settings:
      - **Key Label**: httpsClientTest
      - **Common Name**: use the DNS name for your iSeries server
      - **Organization**: IBM
   c. Click **OK**.
6. Extract the certificate from this self-signed certificate so that it can be imported into the Web server's SSL key file:
   a. Click **Extract Certificate**.
   b. Specify settings:
      - **Data Type**: Base64-encoded ASCII data
      - **Certificate file name**: httpsClient.arm
      - **Location**: the path to your etc directory
   c. Click **OK**.
7. Import the Web server's certificate:
   a. Click **Personal Certificates** and select **Signer Certificates**.
   b. Click **Add**.
   c. Specify settings:
      - **Data Type**: Base64-encoded ASCII data
      - **Certificate file name**: myLocalCA.txt
      - **Location**: the path to your etc directory
   d. Click **OK**.
8. Enter `web-server` for the label and click **OK**.
9. Click **Key Database File**.
10. Select **Exit**.

Configure the Web server's certificate store

Add the signer certificate of the client HTTPS application to the Web server's SSL key file and to the list of trusted CA certificates for the Web server's secure application. This step is needed if the Web server configuration requires client authentication:

1. "Start the Digital Certificate Manager" on page 130.
2. On the left pane, click **Select a Certificate Store**

3. Select ***SYSTEM** and click **Continue**.
4. On the Certificate Store and Password page, enter the password, then click **Continue**.
5. On the left pane, click **Fast Path**.
6. Select **Work with CA certificates**.
7. Click the **Import** button.
8. Specify `USER_INSTALL_ROOT/etc/httpsClient.arm` for the **Import file:** field value and click **Continue**.
9. Specify `httpsClient` for the **CA certificate label** field value and click **Continue**.
10. On the left pane, select **Work with server applications**. On this page, select the application used by the Web server's configuration, and click **Work with Application**.
11. Click **Define CA Trust List**.
12. Click the check box for the httpsClient **CA**, then click **OK**.

Configure WebSphere Application Server - Express

You must specify some Java virtual machine properties for the application server. Use the WebSphere administrative console to perform these steps:

1. In the navigation menu, expand **Servers**, and click **Application Servers**.
2. In the Application Servers page, click the name of your server.
3. Under Additional Properties, click **Process Definition**.
4. Under Additional Properties, click **Java Virtual Machine**.
5. Under Additional Properties, click **Custom Properties**.
6. Click **New** to a new property. Add these properties:

Name	**Value**
`java.protocol.handler.pkgs`	`com.ibm.net.ssl.internal.www.protocol`
`javax.net.ssl.trustStore`	`USER_INSTALL_ROOT/etc/httpsClientKeys.jks`, where *USER_INSTALL_ROOT* is the root directory of your instance
`javax.net.ssl.trustStorePassword`	(Enter your password.)

If the Web server requires client authentication, you need to additionally specify these properties:

Name	**Value**
`javax.net.ssl.keyStore`	`USER_INSTALL_ROOT/etc/httpsClientKeys.jks`, where *USER_INSTALL_ROOT* is the root directory of your instance.
`javax.net.ssl.keyStorePassword`	(Enter your password.)

Normally, javax.net.ssl.keyStore would be a different keystore file.

Click **OK**.

For a code example of a servlet that uses HTTPS, see "Example: HTTPS servlet." The servlet retrieves the URL to display entered as a query string or as a servlet initialization parameter.

Example: HTTPS servlet:

```
/*
 *  This material contains programming source code for your
 *  consideration.  These examples have not been thoroughly
 *  tested under all conditions.  IBM, therefore, cannot
 *  guarantee or imply reliability, serviceability, or function
 *  of these program.  All programs contained herein are
 *  provided to you "AS IS".  THE IMPLIED WARRANTIES OF
 *  MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 *  ARE EXPRESSLY DISCLAIMED.  IBM provides no program services for
 *  these programs and files.
 */
import java.io.DataInputStream;
import java.security.*;
import java.net.URLConnection;
import java.net.URL;
import java.net.URLDecoder;
import java.io.PrintWriter;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


public class HttpsSampleServlet extends HttpServlet
{

  public void doGet(HttpServletRequest req, HttpServletResponse res) {

    res.setContentType("text/html");

    // url passed in as browser query string
    String url  = req.getParameter("httpsURL");
    if (null != url)
      url = URLDecoder.decode(url);
    else {
      // url passed in as servlet init parameter
      url = getInitParameter("httpsURL");
    }

    URLConnection conn = null;
    URL connectURL = null;

    // send result to the caller
    try {

      PrintWriter out = res.getWriter();
      if (null == url || url.length() == 0) {
        out.println("No Https URL provided to retrieve");
      }
        else {
          connectURL = new URL(url);
          conn = connectURL.openConnection();
          DataInputStream theHTML = new DataInputStream(conn.getInputStream());
          String thisLine;
          while ((thisLine = theHTML.readLine()) != null) {
            out.println(thisLine);
          }
        }
        out.flush();
        out.close();
      }
    catch (Exception e) {
      System.out.println("Exception in HttpsSampleServlet: " + e.getMessage());
      e.printStackTrace();
    }
  }//end goGet(...)
}//end class
```

Example: JSSE client servlet:

```
/*
 *
 *  This material contains programming source code for your
 *  consideration.  These examples have not been thoroughly
 *  tested under all conditions.  IBM, therefore, cannot
 *  guarantee or imply reliability, serviceability, or function
 *  of these program.  All programs contained herein are
 *  provided to you "AS IS".  THE IMPLIED WARRANTIES OF
 *  MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 *  ARE EXPRESSLY DISCLAIMED.  IBM provides no program services for
 *  these programs and files.
 *
 */
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
import java.security.Security;
import java.security.KeyStore;
import javax.net.ssl.SSLServerSocketFactory;
import javax.net.ssl.SSLSocketFactory;
import javax.net.ssl.SSLServerSocket;
import javax.net.ssl.SSLSocket;
import javax.net.ssl.SSLSession;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.ibm.net.ssl.SSLContext;
import com.ibm.net.ssl.TrustManagerFactory;
import com.ibm.net.ssl.TrustManager;
import com.ibm.net.ssl.KeyManagerFactory;

/*
 * ServerJsse and ClientJsse
 *
 * Sample applications to demonstrate a simple client/server interaction using JSSE.
 * They communicate using all enabled cipher suites.
 *
 * The ServerJsse and ClientJsse use different KeyStore files called
 * "clientAppKeys.jks" and "serverAppKeys.jks" protected by a password and assumed
 * to be in the current working directory. In a real application, the client and
 * server should also have their own KeyStore files.
 *
 * You can build these sample program classes by issuing the following command but
 * first make sure that the ibmjsse.jar file shipped in the
 * "WAS_PRODUCT_ROOT/java/ext" is in your CLASSPATH or extensions directory.
 * This is a servlet, so you also need "WAS_PRODUCT_ROOT/lib/j2ee.jar to compile.
 *
 *     javac ServerJsse.java
 *     javac ClientJsse.java
 *
 *
 * Running the client/server application requires that the Server is started first
 * and then the Client application.
 *
 * To start the ServerJsse, run the following command:
 *
 *     java ServerJsse
 *
 * The ServerJsse uses port 8050 by default. Once ServerJsse has started it waits
 * for the ClientJsse connection.
 *
 * To start the client, issue the following command:
 *
 *     java ClientJsse hostname:port
 *
 * where hostname:port is the optional host:port running SeverJsse.
 * The default is localhost:8050.
 *
 */

public class ClientJsse extends HttpServlet {
  static int N = 50, L = 100, R = 2;
  static PrintWriter out = new PrintWriter(System.out);
  static SSLContext sslContext;


  // Open the KeyStore to obtain the Trusted Certificates.
  // KeyStore is of type "JKS". Filename is "clientAppKeys.jks"
  // and password is "myKeys".
  public static void initContext() {
    try {
      Security.addProvider(new com.ibm.jsse.JSSEProvider());

      KeyStore ks = KeyStore.getInstance("JKS");
      ks.load(new FileInputStream("clientAppKeys.jks"), "myKeys".toCharArray());
      KeyManagerFactory kmf = KeyManagerFactory.getInstance("IbmX509");
      kmf.init(ks, "myKeys".toCharArray());

      TrustManager[] tm;
      TrustManagerFactory tmf = TrustManagerFactory.getInstance("IbmX509");
      tmf.init(ks);
      tm = tmf.getTrustManagers();

      sslContext = SSLContext.getInstance("SSL");
      sslContext.init(kmf.getKeyManagers(), tm, null);
    }
    catch (Throwable t) {
      out.println("Failed to read clientAppKeys.jks file.");
      t.printStackTrace();
    }
  }

  public void doGet(HttpServletRequest q, HttpServletResponse r) {
    r.setContentType("text/html");
    String[] args = new String[1];
    args[0] = getInitParameter("hostname");

    // send result to the caller
    try {
      out = r.getWriter();
      out.println("<html>");
      out.println("<head>title>Client JSSE Session</title></head>");
      out.println("<body>");
      out.println("<h1>Client JSSE Session (starting)</h1>");
      out.flush();
      out.println("<pre>");

      main(args);

      out.println("</pre>");
      out.println("</body></html>");
      out.flush();
      out.close();

    }
    catch (Exception e) {
      out.println("Exception in ClientJsse Servlet: " + e.getMessage());
      e.printStackTrace();
    }
  }//end  public void doGet(...)



 /** Starts the ClientJsse application as stand alone application.
   * @param args the name of the host the ClientJsse should connect to
   */
  public static void main(String args[]) throws Exception {

    int i, j, len, ci_nonanon;
```

```
String host = "127.0.0.1";
int port = 8050;

if ((args != null) && (args.length > 0)) {
  host = args[0];
  i = host.indexOf(':');
  if (i != -1) {
    try {
      port = Integer.parseInt(host.substring(i+1));
      host = host.substring(0,i);
    }
    catch (Exception e) {
      System.err.println("ClientJsse: wrong address format");
      e.printStackTrace();
      throw e;
    }
  }
}

if (args.length > 1)
  N = Integer.parseInt(args[1]);
if (args.length > 2)
  L = Integer.parseInt(args[2]);
byte buffer[] = new byte[L];

out.println("ClientJsse: started.");

initContext();

try {
  if (sslContext == null) {
    out.println ("ClientJsse:  SSL client context NOT created.");
    return;
  }

  out.println("ClientJsse: SSL client context created.");
  SSLSocketFactory factory = sslContext.getSocketFactory();
  String[] cipher_suites =
     sslContext.getSocketFactory().getSupportedCipherSuites();

  out.println("ClientJsse: enabled cipher suites");
  for (i=0, ci_nonanon=0; i < cipher_suites.length; i++) {
    if (cipher_suites[i].indexOf("_anon_") < 0) {
      ci_nonanon++;
    }
    out.println("   " + cipher_suites[i]);
  }
  out.flush();
  SSLSocket ssl_sock;
  OutputStream ostr;
  InputStream istr;
  long t;
  String[] ecs = new String[1];

  out.println("\n ");
  for (int csi = 0; csi < ci_nonanon; csi++) {
    // Remove anonymous cipher suites.
    // TrustManager requires a personal certificate.
    ecs[0] = cipher_suites[csi];
    if (ecs[0].indexOf("_anon_") >= 0) {
      continue;
    }
    for (int n = 0; n < R; n++) {
      t = System.currentTimeMillis();
      try {
        ssl_sock = (SSLSocket) factory.createSocket(host, port);
```

```
          ssl_sock.setEnabledCipherSuites(ecs);

          ssl_sock.startHandshake();
          SSLSession session = ssl_sock.getSession();
          out.println(" ");
          out.println("\nClientJsse: SSL connection established");
          out.println("   cipher suite:       " + session.getCipherSuite());
        }
        catch (Exception se) {
          out.println(" ");
          out.println("\nClientJsse: can't connect using: " +
                      cipher_suites[csi] + "\n" + se);
          break;
        }

        if (L > 0) {
          istr = ssl_sock.getInputStream();
          ostr = ssl_sock.getOutputStream();

          for (j=0; j < N; j++) {
            if ((j==N-1) && (n == R-1) && (csi == ci_nonanon - 1))
              buffer[0] = (byte)-1;
            ostr.write(buffer, 0, L);
            for (len = 0;;) {
              try {
                if ((i = istr.read(buffer, len, L-len)) == -1) {
                  out.println("ClientJsse: SSL connection dropped by partner.");
                  istr.close();
                  return;
                }
                // out.println("ClientJsse: " + i + " bytes received.");
                if ((len+=i) == L) break;
              }
              catch (InterruptedIOException e) {
                // out.println("ClientJsse: timeout.\n");
              }
            }
          }
        }
        out.println("Messages = " + N*2 + "; Time = " +
                    (System.currentTimeMillis() - t));
        ssl_sock.close();
        out.println("ClientJsse: SSL connection closed.");
        out.flush();
      }
    }

    Thread.sleep(1500);

    // Example using plain sockets
    if (L > 0) {
      Socket sock = new Socket(host, port);
      out.println(" ");
      out.println("\nClientJsse: Plain Socket connection established.");
      istr = sock.getInputStream();
      ostr = sock.getOutputStream();
      t = System.currentTimeMillis();
      for (j=0; j < N; j++) {
        ostr.write(buffer, 0, L);
        for (len = 0;;) {
          if ((i = istr.read(buffer, len, L-len)) == -1) {
            out.println("ClientJsse: connection dropped by partner.");
            return;
          }
          if ((len+=i) == L) break;
        }
      }
```

```
        out.println("Messages = " + N*2 + "; Time = " +
                   (System.currentTimeMillis() - t));
        sock.close();
        out.println("ClientJsse: Plain Socket connection closed.");
        Thread.sleep(1500);
      }
    }
    catch (Exception e) {
      out.println(e);
    }
    out.println(" ");
    out.println("ClientJsse: terminated.");
    out.flush();
  }//end main(...)
}//end class
```

Example: JSSE server servlet:

```
/*
 *
 *  This material contains programming source code for your
 *  consideration.  These examples have not been thoroughly
 *  tested under all conditions.  IBM, therefore, cannot
 *  guarantee or imply reliability, serviceability, or function
 *  of these program.  All programs contained herein are
 *  provided to you "AS IS".  THE IMPLIED WARRANTIES OF
 *  MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 *  ARE EXPRESSLY DISCLAIMED.  IBM provides no program services for
 *  these programs and files.
 *
 */
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
import java.security.Security;
import java.security.KeyStore;
import javax.net.ssl.SSLServerSocketFactory;
import javax.net.ssl.SSLServerSocket;
import javax.net.ssl.SSLSocket;
import javax.net.ssl.SSLSession;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.ibm.net.ssl.SSLContext;
import com.ibm.net.ssl.TrustManagerFactory;
import com.ibm.net.ssl.KeyManagerFactory;

/*
 * ServerJsse and ClientJsse
 *
 * Sample applications to demonstrate a simple client/server interaction using JSSE.
 * They will communicate using all enabled cipher suites.
 *
 * The ServerJsse and ClientJsse use different KeyStore files called
 * "clientAppKeys.jks" and "serverAppKeys.jks" protected by a password and assumed
 * to be in the current directory. In a real application, the client and server
 * should also have their own KeyStore files.
 *
 * You can build these sample program classes by issuing the following command but
 * first make sure that the ibmjsse.jar file shipped in the
 * "WAS_PRODUCT_ROOT/java/ext" is in your CLASSPATH or extensions directory.
 * This is a servlet, so you also need "WAS_PRODUCT_ROOT/lib/j2ee.jar to compile.
 *
 *     javac ServerJsse.java
 *     javac ClientJsse.java
 *
 *
 * Running the client/server application requires that the Server is started first
 * and then the Client application.
 *
 * To start the ServerJsse, run the following command:
 *
 *     java ServerJsse  port
 *
 * The ServerJsse uses port 8050 by default. Once ServerJsse has started it waits
 * for the ClientJsse connection.
 *
 * To start the client, issue the following command:
 *
 *     java ClientJsse hostname:port
 *
 * where hostname:port is the name of the host:port running SeverJsse.
 * The default is localhost:8050.
 *
 */

public class ServerJsse extends HttpServlet {
  static int N = 50, L = 100, R = 2;
  static PrintWriter out = new PrintWriter(System.out);
  static SSLContext sslContext;


  // Open up the KeyStore to obtain the Trusted Certificates.
  // KeyStore is of type "JKS". Filename is "serverAppKeys.jks"
  // and password is "myKeys".
  private static void initContext() throws Exception {
    try {
      Security.addProvider(new com.ibm.jsse.JSSEProvider());

      KeyStore ks = KeyStore.getInstance("JKS");
      ks.load(new FileInputStream("serverAppKeys.jks"), "myKeys".toCharArray());
      KeyManagerFactory kmf = KeyManagerFactory.getInstance("IbmX509");
      kmf.init(ks, "myKeys".toCharArray());
      TrustManagerFactory tmf = TrustManagerFactory.getInstance("IbmX509");
      tmf.init(ks);
      sslContext = SSLContext.getInstance("SSL");
      sslContext.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);
    }
    catch (Exception e) {
      out.println("Failed to read serverAppKeys.jks file.");
      e.printStackTrace();
      throw e;
    }
  }


  public void doGet(HttpServletRequest q, HttpServletResponse r) {
    r.setContentType("text/html");

    String[] args = new String[1];
    args[0] = getInitParameter("port");

    // send result to the caller
    try {
      out = r.getWriter();
      out.println("<html>");
      out.println("<head><title>Server JSSE Session</title></head>");
      out.println("<body>");
      out.println("<h1>Server JSSE Session (starting) </h1>");
      out.flush();
      out.println("<pre>");
      main(args);

      out.println("</pre>");
      out.println("</body></html>");
      out.flush();
      out.close();

    }
    catch (Exception e) {
      out.println("Exception in ServerJsse Servlet: " + e.getMessage());
      e.printStackTrace();
    }
  }//end  public void doGet(...)


  /* Entry point for ServerJsse as stand alone application.
   * @param args no parameters required
   */
  public static void main(String args[]) throws Exception {
    int i, j, len, portint;
    String port = "8050";

    if ((args != null) && (args.length > 0)) {
      port = args[0];
    }

    portint = (new Integer(port)).intValue();

    if (args.length > 1)
      N = Integer.parseInt(args[0]);
    if (args.length > 2)
      L = Integer.parseInt(args[1]);

    byte buffer[] = new byte[L];

    out.println("SERVER: started.");

    initContext();

    try {

      out.println("ServerJsse: SSL server context created.");

      SSLServerSocketFactory factory = sslContext.getServerSocketFactory();

      // Get and print the list of supported enabled cipher suites
      String enabled[] = factory.getSupportedCipherSuites();
      out.println("ServerJsse: enabled cipher suites");
      for (i=0; i < enabled.length; i++)
        out.println(enabled[i]);
      out.flush();

      // Create an SSL session over port 8050
      SSLServerSocket ssl_server_sock =
         (SSLServerSocket)factory.createServerSocket(portint);
      SSLSocket ssl_sock;
      InputStream istr;
      OutputStream ostr;
      long t;
      while (buffer[0] != -1) {
        for (int n=0; n<R; n++) {
          try {
            ssl_sock = (SSLSocket)(ssl_server_sock.accept());
            ssl_sock.setEnabledCipherSuites(enabled);

            ssl_sock.startHandshake();
            SSLSession session = ssl_sock.getSession();
            out.println(" ");
            out.println("\nServerJsse: SSL connection established");
            out.println("   cipher suite:       " + session.getCipherSuite());
          }
          catch (IOException se) {
            out.println("ServerJsse: client connection refused\n" + se);
            break;
          }

          if (L > 0) {
            istr = ssl_sock.getInputStream();
            ostr = ssl_sock.getOutputStream();

            t = System.currentTimeMillis();
            for (j=0;j<N;j++) {
              for (len = 0;;) {
                try {
                  if ((i = istr.read(buffer, len, L-len)) == -1) {
                    out.println("ServerJsse: connection dropped by partner.");
                    ssl_sock.close();
                    return;
                  }
                }
                catch (InterruptedIOException e) {
                  out.println("waiting");
                  continue;
                }
                if ((len+=i) == L) break;
              }
              ostr.write(buffer, 0, L);
            }
            out.println("Messages = " + N*2 + "; Time = " +
                       (System.currentTimeMillis() - t));
          }
          ssl_sock.close();
        }
      }
      ssl_server_sock.close();
      out.println("ServerJsse: SSL connection closed.");
      out.flush();
      Thread.sleep(1000);

      // Example using plain sockets
      if (L > 0) {
        ServerSocket server_sock = new ServerSocket(portint);
        Socket sock = server_sock.accept();
        out.println(" ");
        out.println("\nServerJsse: Plain Socket connection accepted.");
        istr = sock.getInputStream();
        ostr = sock.getOutputStream();

        t = System.currentTimeMillis();
        for (j=0;j<N;j++) {
          for (len = 0;;) {
            if ((i = istr.read(buffer, len, L-len)) == -1) {
              out.println("ServerJsse: connection dropped by partner.");
              return;
            }
            if ((len+=i) == L) break;
          }
          ostr.write(buffer, 0, L);
        }
        out.println("Messages = " + N*2 + "; Time = " +
                   (System.currentTimeMillis() - t));
        sock.close();
        server_sock.close();
        out.println("ServerJsse: Plain Socket connection closed.");
        Thread.sleep(1000);
      }
    }
```

```
    catch (IOException e) {
      e.printStackTrace();
    }
    catch (InterruptedException f) {
      f.printStackTrace();
    }
    out.println(" ");
    out.println("ServerJsse: terminated.");
    out.flush();

  }//end main(...)

}//end class
```

Migrate applications to use Java keystores: Prior to WebSphere Application Server Version 5.0, the use of i5/OS *SYSTEM certificate stores (.kdb files) is supported for WebSphere applications that use the JSSE and SSL program interfaces. With Version 5.0, only Java keystores (.jks files) are fully supported. It is recommended that you migrate your applications to use Java keystores.

With Version 5.0, i5/OS *SYSTEM certificate stores can only be used by applications that are deployed within an administrative domain where global security is disabled. To continue using i5/OS *SYSTEM certificate stores with your applications, perform these steps:

1. Ensure that security is disabled for the administrative domain where your application is deployed. Use the WebSphere administrative console to disable global security.

   Note: If you disable WebSphere security, your WebSphere resources may no longer be protected. Carefully assess your security needs before you disable security. If you decide that security must remain enabled, migrate your applications to use Java keystores.
2. In the java.security file for your WebSphere instance, set the os400.jdk13.jst.factories property to `false`. The integrated file system pathname of the java.security file for an instance is /QIBM/UserData/WebASE/ASE5/*instance*/properties/java.security.
3. Restart all servers within your administrative domain.

To migrate your applications to use Java keystores (.jks files), perform these steps:

1. Create Java keystores for your certificates. If your application uses a digital certificate to authenticate to itself to a server, use the iKeyman utility to create a Java keystore to contain the certificate. For more information, see "The iKeyman utility" on page 132.

   Note: When you are working with iKeyman, you may find it convenient to map a drive to access keystores and certificate files that are located in the integrated file system of your iSeries system. Otherwise, you can copy the files to your workstation from your iSeries system (for example, by using file transfer protocol), modify them with iKeyman, and then copy the files to the iSeries system.

   Typically, you use one keystore to contain the certificate authority certificates (CA or signer certificates) for your application (a trust keystore) and another keystore to contain the client or server certificate. If you have multiple applications, you use a keystore for each application (to contain the client or server certificate), and you use a single trust keystore for all of the applications.

   If your application uses commercial certificate authority certificates (signer or CA certificates), you may be able to use the cacerts keystore (the default trust keystore) with your application. The integrated file system path for cacerts is /QIBM/ProdData/Java400/jdk13/lib/security/cacerts. However, in no case should you attempt to modify the cacerts keystore. Rather, you should create a private copy of the cacerts file, and then add or remove certificates. The password for cacerts is `changeit`. Be sure to change the password that protects your private copy of the cacerts file. Also, note that initially, all keystores created using iKeyman contain a number of commercial CA certificates.

   You may create your Java keystores in any iSeries integrated file system directory. However, it may be convenient to place them in the same directory as those that are used by your WebSphere instance. This may make it easier to include them in your backup and restore procedure. WebSphere Application Server - Express provides an initial set of Java keystores that are used to secure connections between WebSphere components. These keystores are found in the `etc` directory of your WebSphere instance, for example, /QIBM/UserData/WebASE/ASE5/myInstance/etc.

For an example of how to create a Java keystore, see "Using Java keystore files" on page 135.

2. Use the Digital Certificate Manager (DCM) to export certificates from your i5/OS *SYSTEM certificate stores. Export both the client and server certificates that are used by your applications. Additionally, export the certificate authority (CA) certificates. For more information about DCM, see these topics in the iSeries Information Center:
   - V5R4 Digital Certificate Manager
   - V5R3 Digital Certificate Manager
   - V5R2 Digital Certificate Manager

   Note: DCM uses PKCS12 format to export client and server certificates, and it uses base64 encoded format to export certificate authority certificates. These formats are compatible with iKeyman. However, you must to convert the base64 encoded data from EBCDIC to ASCII format before you use it with iKeyman. To convert the data, perform these steps:
   a. On an iSeries command line, enter the Start Qshell (STRQSH) command.
   b. Use the `cd` command to change to the directory that contains the file that is used to export the CA certificate.
   c. Use the `touch` and `cat` commands to copy the data to another file and convert it to ASCII formate. For example, to convert the data for file myCA to file myCA.arm, enter this command:
   ```
   touch -C 819 myCA.arm; cat myCA > myCA.arm
   ```
3. To import certificates into your Java keystores, perform these steps:
   a. Start iKeyman on your workstation. For more information, see "The iKeyman utility" on page 132.
   b. Open your key database file (Java keystore):
      1) Select **Open** from the Key Database File menu.
      2) Click **Browse...** and select the file containing your key database (keystore).
      3) Click **OK** on the Open panel.

      Note: You may use separate keystores to contain your client or server certificates and your CA certificates.
   c. Import the client or server certificate:
      1) Select **Personal Certificates** from the **Key database content** pulldown menu.
      2) Click **Export/Import...**.
      3) Select **Import Key** for the Action Type.
      4) Select **PKCS12** for the Key file type.
      5) Click **Browse...** and select the file that contains the certificate to be imported.
      6) Click **OK** on the Export/Import Key panel.
   d. Add signer certificates:
      1) Select **Signer Certificates** from the Key Database File menu.
      2) Click **Add...**
      3) Click **Browse...** and select the file that contains the signer certificate to be added.
      4) Click **OK** on the `Add CA's certificate from a file` panel.
   e. Select **Exit** from the Key Database File menu.
4. Grant Read and Execute (*RX) authority to your Java keystores for the i5/OS user profile under which your application runs. For example, enter the Change Authority (CHGAUT) command to grant *RX authority to the QEJBSVR user profile for the myKeys.jks (where myKeys.jks is accessed by a servlet that is deployed on your default instance):
   ```
   CHGAUT OBJ('/QIBM/UserData/WebASE/ASE5/instance/etc/myKeys.jks')
    USER(QEJBSVR) DTAAUT(*RX)
   ```

5. Make the code changes that are required for using your Java keystores. The changes that are required for your applications to use your Java keystores vary, depending on your implementation. See "Using Java keystore files" on page 135 for code examples.
6. Test your applications.
7. After you are convinced that your applications are working correctly, you may want to clean up files, authorities, and properties that your application no longer needs.

   Note: This cleanup applies to both stand-alone Java client applications and to applications that are deployed on your application server.

   One or more of the following properties may be set and should be removed from your application's runtime environment. If the application is running on your application server, the properties are set as custom properties in the runtime configuration of your application server:
   - **os400.certificateContainer**
     If your application used the os400.certificateContainer property, also delete the certificate container (.kdb file), but only if it is no longer used by any other application.
   - **os400.certificateLabel**
   - **os400.secureApplication**
     If your application used the os400.secureApplication property, also use Digital Certificate Manager (DCM) to delete the secure application from the *SYSTEM certificate store.

   Use Operations Navigator to remove authority to the *SYSTEM certificate store for the i5/OS user profile under which your server or client application runs. The user profile that you use to run Operations Navigator must have *SECADM special authority to perform this task.

   Note: This step applies only if your application previously used certificates that were contained in the *SYSTEM certificate store, and the user profile does not require access to the *SYSTEM certificate store for any other application.

   For example, if the user profile the application runs under is QEJBSVR, perform these tasks:
   a. Start Operations Navigator.
   b. Click the iSeries system that contains the secure client application.
   c. Click **Configure Application Administration** in the bottom window.
   d. Click the **Host Applications** tab.
   e. Expand **Digital Certificate Manager**.
   f. Select ***SYSTEM certificate store**.
   g. Click **Customize**.
   h. If Default access is selected or if QEJBSVR is not in the Access allowed list, then stop and cancel out of the tasks. You do not need to customize access.
   i. Select **QEJBSVR** from the Access allowed panel.
   j. Click **Remove** to remove QEJBSVR from the Access allowed panel.
   k. Click **OK**.

Configure SSL connections between WebSphere Application Server - Express 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. If you are using the i5/OS Directory Service, see the Directory Services documentation in the iSeries Information Center:
   - For V5R4
   - For V5R3
   - For V5R2
2. Update your WebSphere Application Server - Express trust store file. The trust store file is the repository for the WebSphere server's trust base. Because it needs to authenticate the LDAP server during SSL initialization, the trust store file must provide information about the LDAP server.

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 CA certificate to your trust store file on the server.

To add the additional certificate to the trust store file, do the following:

a. Obtain the certificate of the CA that issued the LDAP server's certificate. For example, if your LDAP server's certificate was issued by the Local CA on your iSeries system, extract the Local CA's certificate by using the Digital Certificate Manager (DCM):
   1) "Start the Digital Certificate Manager" on page 130

      Note: Procedures vary depending on the release of DCM you have installed on your iSeries system. The release of DCM that is used in this topic is V5R1M0.
   2) In the left pane, click **Install CA certificate on your PC**.
   3) In the right pane, click **Copy and paste certificate**.
   4) Create a text file on your PC, then paste the CA certificate into *myLocalCA.txt* and save the file. Ensure that the copy of the CA certificate ends with the new line character.
   5) Click the **Done** button.

b. Add the CA certificate to the server's trust store file:
   1) Start iKeyman on your workstation. For more information, see "The iKeyman utility" on page 132.
   2) Click **Key Database File** and select **Open**.
   3) Using the browser, navigate to the directory containing the trust store file for your WebSphere server instance, and open the file. For example: *USER_INSTALL_ROOT*/etc/DummyServerTrustFile.jks.
   4) Click **Personal Certificates** and select **Signer Certificates**.
   5) Click **Add**.
   6) Specify settings:
      - **Data Type**: Base64-encoded ASCII data
      - **Certificate file name**: myLocalCA.txt
      - **Location**: the path to the directory containing myLocalCA.txt
   7) Click **OK**.
   8) Enter LocalCA for the label and click **OK**.
   9) Click **Key Database File**.
   10) Select **Exit**.

3. Enable the SSL connection in WebSphere. Use the WebSphere administrative console to modify your LDAP configuration (under **Security —> User Registries —> LDAP**):
   - Set the port to 636. (If you used a different port number, set the port to that number.)
   - Select **SSL Enabled**.
   - Select **DefaultSSLSettings**.
4. Click **OK**.
5. Save your changes.
6. Stop and restart the application server, then start the administrative console. You are prompted to login to the LDAP registry.

Tips

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

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

Configure Java 2 security

Java 2 Security is a new feature in WebSphere Application Server - Express Version 5. It is a new programming model that is very pervasive and has a huge impact on application development. For more information, see "Java 2 Security" on page 155.

Java 2 Security is disabled by default, but is enabled automatically when global security is enabled. However, it is independent of J2EE role-based security, you can disable or enable it separately of global security. Java 2 Security provides an extra level of access control protection in addition to J2EE role-based authorization. It particularly addresses the protection of system resources and APIs. The Java development kit has used Java 2 Security to protect sensitive APIs (such the API to exit the Java virtual machine) and APIs that utilize resources (for example, opening files for reading and writing) since the release of Java 2.

Note: Java 2 Security only restricts Java programs that run in a Java virtual machine that has Java 2 Security enabled. It does not protect system resources if Java 2 Security is disabled, or if system resources are accessed from other programs or commands. Therefore, if you want to protect your iSeries system resources, you need to use iSeries security.

Here are some guidelines for when to use Java 2 Security:

- To enable protection on system resources. For example, when opening or listening to a socket connection, reading or writing to operating system file systems, reading or writing Java virtual machine system properties, and so on.
- To prevent application code from calling destructive APIs. For example, calling System.exit() brings down the application server.
- To prevent application code from obtaining privileged information (passwords) or gaining extra privileges (obtaining server credentials).

To ease the effort of enabling Java 2 Security in a test or production environment, make sure the application is developed with the Java 2 Security programming model in mind. Developers have to know whether or not the APIs used in the applications are protected by Java 2 Security. It is very important that the required permissions for the APIs used are declared in the policy file (was.policy), or the application fails to run when Java 2 Security is enabled. See "Configure Java 2 policy files" on page 158 for more information about was.policy and other Java 2 Security policy files.

The default permission set for applications is the recommended permission set that is defined in the J2EE 1.3 Specification. The default is declared in the app.policy policy file in addition to permissions that are defined in the Java development kit java.policy file that grant permissions to everyone. However, applications are denied permissions that are declared in the filter.policy file. Permissions that are declared in the filter.policy file are filtered out for applications during the permission check.

Define the required permissions for an application in a was.policy file and embed the was.policy file in the application EAR file in the META-INF subdirectory.

For more information about policy files, see "Configure Java 2 policy files" on page 158.

To enable Java 2 Security, perform these steps in the administrative console:

- In the navigation menu, expand **Security**. Click **Global Security**. The Global Security page appears.
- Enable Java 2 Security by selecting **Enforce Java 2 Security**.

- Click **OK** or **Apply** on the Global Security page.
- Click **Save** to save the changes.
- Restart the server for the changes to take effect.

Java 2 Security is enabled and enforced for the servers. Java 2 Security permission is checked when a Java 2 Security protected API is called.

Tracing Java 2 Security

The WebSphere Java 2 Security Manager is enhanced to dump the Java 2 Security permissions granted to all classes on the call stack when an application is denied access to a resource (the java.security.AccessControlException exception is thrown). However, this tracing capability is disabled by default. You can enable it by specifying the server trace service with the `com.ibm.ws.security.core.SecurityManager=all=enabled` trace specification. When the exception is thrown, the trace dump provides hints to determine whether the application is missing permissions or the product runtime code or that third-party libraries used are not properly marked as privileged when accessing Java 2 protected resources.

Java 2 Security: Java 2 Security is a supported feature in WebSphere Application Server - Express. Java 2 Security provides a policy-based, fine-grain access control mechanism that increases overall system integrity by checking for permissions before allowing access to certain protected system resources. Java 2 Security is independent of J2EE role-based authorization. Java 2 Security guards access to system resources such as file input and output, sockets, and properties, whereas J2EE security guards access to Web resources such as servlets and JSP files. WebSphere global security includes J2EE role-based authorization, the CSIv2 authentication protocol, and SSL configuration. Java 2 Security can be disabled and enabled independently of WebSphere global security (the **Enforce Java 2 Security** check box in the Global Security Panel or Server Level Security Panel). However, when WebSphere global security is enabled, by default Java 2 Security is also enabled. Note that Java 2 security can be disabled even though WebSphere Global Security may be enabled.

Because Java 2 Security is relatively new to the industry and new to WebSphere, many existing or even new applications may not be prepared for the very fine-grain access control programming model that Java 2 Security is capable of enforcing. Administrators should understand the possible consequences of enabling Java 2 Security if applications are not prepared for Java 2 Security. Java 2 Security places some new requirements on application developers and administrators.

Java 2 Security for deployers and administrators

Before you enable Java 2 Security, you need to make sure that all the applications are granted the required permissions, otherwise, applications may fail to run. By default, applications are granted the permissions recommended in the J2EE 1.3 Specification. For details of default permissions granted to applications in WebSphere, please refer to the following policy files:
- /QIBM/ProdData/Java400/jdk13/lib/security/java.policy
- /QIBM/UserData/WebASE/ASE5/*instance*/properties/server.policy
- /QIBM/UserData/WebASE/ASE5/*instance*/config/cells/*cell*/nodes/*node*/app.policy

where *instance* is the name of your instance, *cell* is the name of your cell, and *node* is the name of your node.

WebSphere Application Server- Express does not support a more restrictive policy than the default one that is embodied in these policy files. This is because WebSphere may not have the necessary Java 2 Security doPrivileged APIs in place. The result of attempting to make the default policy more restrictive may result in WebSphere throwing Java 2 security access control exceptions and causing applications or WebSphere Application Server - Express itself to fail unexpectedly.

Several policy files are used to define the security policy for Java processes. Some policy files are static (the code base is statically defined in the policy file). An example of a static policy file is the java.policy file that is provided by the Java development kit.

Other policy files are dynamic. WebSphere provides support for both static and dynamic policy files. Dynamic policy files are used with enterprise applications, resources and utility libraries. For dynamic policy files, the codebase is dynamically calculated (based on deployment information) and permissions are granted (based on template policy files) during run time.

For more information about policy files, see “Configure Java 2 policy files” on page 158.

If an application is not prepared for Java 2 Security, if the application provider does not provide a was.policy file as part of the application, or if the application provider does not communicate the expected permissions that should be granted to one of the WebSphere policy files, the application is likely to cause Java 2 security access control exceptions at runtime (if Java 2 Security is enabled in WebSphere) and probably runs incorrectly.

It may not be obvious that a application is or is not prepared for Java 2 security. WebSphere provides several runtime debugging aids to help determine troubleshoot applications that may be experiencing Java 2 Security-related access control exceptions.

Java 2 Security for application developers

Application developers must understand the permissions that are granted in the default WebSphere policy and the permission requirements of the Java SDK APIs. You need to know whether the APIs that your application calls requires additional permissions or not. For more information about which Java APIs require permissions, see Permissions in the Java 2 SDK

(http://java.sun.com/j2se/1.3/docs/guide/security/permissions.html).

Application providers can assume that applications are granted the necessary permissions in the default policy. Applications that access resources that are not covered by the default WebSphere policy are required to grant the additional Java 2 security permissions to the application.

While it is possible to grant the application additional permissions in one of the other dynamic WebSphere policy files or in one of the more traditional static policy files, such as java.policy, the was.policy file (which is embedded in the EAR file) ensures that the additional permissions are scoped to the exact application that requires them. If permissions are scoped to the application code that requires the permission, application code that normally does not have permission to access secured resources is not granted that permission.

If a component of an application (such as a library) is included in more than one EAR file, then the library developer should document the required Java 2 permissions that are needed by the application assembler. There is no was.policy type of file for library type components, the developer must communicate the required permissions through external documentation. If the component library is shared by multiple enterprise applications, the permissions can be granted to all enterprise applications on the node in the app.policy.

If the permission is only used internally by the component library and the application is never granted access to resources that are protected by the permission, then it may be necessary to mark the code as privileged (by inserting a doPrivileged block). (For more information, see “AccessControlException” on page 168.) However, improperly inserting a doPrivileged block could open up security holes, so great care and understanding should be employed when decided whether to use a doPriveleged block or not

See “Configure the was.policy file” on page 164 for a description of how the permissions in was.policy files are granted at runtime.

Developing an application with Java 2 Security in mind may be a new skill and impose a security awareness that was not previously required of application developers. Describing the Java 2 Security model and the implications on application development is beyond the scope of this section. For more information, see The J2SDK 1.3 Security documentation

(http://java.sun.com/j2se/1.3/docs/guide/security/index.html).

Debugging aids

There are two primary aids to debugging Java 2 Security:

- **The WebSphere SystemOut.log file**
  The AccessControl exception is logged in the SystemOut.log, and it contains the permission violation that causes the exception, the exception call stack, and the permissions that are granted to each stack frame. This information is usually enough to determine the missing permission and the code that requires the permission.
- **The com.ibm.websphere.java2secman.norethrow property**
  When Java 2 Security is enabled in WebSphere, by default the Security Manager component throws an java.security.AccessControl exception when a permission violation occurs. If this exception is not handled, it may cause a runtime failure. (This exception is also logged in the SystemOut.log.) However, when the com.ibm.websphere.java2secman.norethrow property is set to `true`, the Security Manager logs the AccessControl exception rather than throw it.

  Note: Because this property instructs the Security Manager to not throw the exception, the Security Manager is technically not enforcing Java 2 Security. The norethrow property should not be used in a production environment.

  This property is valuable in a sandbox or test environment where the application can be thoroughly tested and the SystemOut.log can be inspected for AccessControl exceptions. Because this property causes the AccessControl exception not to be thrown, the exception is not propagated up the call stack and does not cause a failure. Without this property, AccessControl exceptions would have to be found and fixed, one at a time.

Handling applications that are not ready for Java 2 Security

If you use a third-party application and the increased system integrity that is provided by Java 2 Security is important, contact the application provider to request support for Java 2 Security or, at least, to understand the additional permissions beyond the default WebSphere policy that are needed.

The easiest way to deal with applications that do not support Java 2 Security is to disable Java 2 Security in WebSphere. The downside is that this solution applies to the entire system and the integrity of the system is not as strong as it could be. This should be a consideration that is not taken lightly. In some cases, disabling Java 2 Security may be an acceptable risk.

Another approach is to leave Java 2 Security enabled but either grant only enough additional permissions or grant all permissions to only the problematic application. Determining how many permissions are enough may not be trivial. If the application provider has not communicated the required permissions in some way, then there is no easy way to determine the required permissions, so granting all permissions may be the only choice. You can minimize this risk by deploying such an application on a different node; this may help to isolate it from certain resources. Then, you can grant java.security.AllPermission in the was.policy file that is embedded in the application’s EAR file, for example:

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

The server.policy file

This policy defines the policy for the WebSphere classes.

The server.policy file should be used to define the Java 2 security policy for server resources. For enterprise application resources, use the app.policy file for node-scoped permissions or the was.policy file for application-scoped permissions.

The java.policy file

This file represents the default permissions granted to all classes. The policy of this file applies to all the processes that are launched by the Java virtual machine in which WebSphere Application Server - Express runs. For more information, see "Configure the java.policy file" on page 167.

Troubleshooting

- **Symptom**
  Error message `SECJ0314E: Current Java 2 Security policy reported a potential violation of Java 2 Security Permission. Please refer to Problem Determination Guide for further information.{0}Permission\:{1}Code\:{2}{3}Stack Trace\:{4}Code Base Location\:{5}`
- **Problem**
  The Java Security Manager checkPermission() reported a SecurityException on the subject Permission with debugging information. The reported information can be different with respect to the system configuration. This report is enabled by configuring RAS trace into debug mode. For more information, see Enable and disable the trace service in the *Troubleshooting* topic.
- **Recommended response**
  The reported exception may be critical to the secure system. Turn on security trace to determine the potential code that may have violated the security policy. After you determine the violating code, your should verify if the attempted operation is permitted (with respect to Java 2 Security) by examining all applicable Java 2 security policy files and the application code itself.

  Note: If the application uses Java Mail, this message may be benign. You can update the was.policy file to grant the following permissions to the application:

  ```
  permission java.io.FilePermission "${user.home}${/}.mailcap", "read";
  permission java.io.FilePermission "${user.home}${/}.mime.types", "read";
  permission java.io.FilePermission "${java.home}${/}lib${/}mailcap", "read";
  permission java.io.FilePermission "${java.home}${/}lib${/}mime.types", "read";
  ```

  Note: If the application uses the `getResourceAsStream(`*foo*`)` method of the ClassLoader class where *foo* is some file name, this message may be benign. You can update the was.policy file to grant the following permission to the application (or update the app.policy file to grant the permission to all applications on node):

  ```
  permission java.io.FilePermission "/QIBM/ProdData/Java400/foo", "read";
  ```

Messages

- **Message:** `SECJ0313E: Java 2 Security Manager debug message flags are initialized: TrDebug: {0}, Access: {1}, Stack: {2}, Failure: {3}`
  Problem: Configured values of the valid debug message flags for Security Manager.
  Recommended response: None.
- **Message:** `SECJ0307E: Unexpected exception is caught when trying to determine the code base location. Exception: {0}`
  Problem: An unexpected exception is caught when the code base location is being determined.
  Recommended response: Please contact an IBM representative.

Configure Java 2 policy files: The J2EE 1.3 specification has a well-defined programming model of responsibilities between the container providers and the application code. It is recommended that you use the Java 2 Security manager to help enforce this programming model. Certain operations are not allowed in the application code because such operations interfere with the behavior and operation of the containers. The Java 2 Security manager is used in the product to enforce responsibilities of the container and the application code.

WebSphere Application Server - Express provides support for policy file management. There are a number of policy files in the product, which are either static or dynamic. Static policy files provide default permissions. Dynamic policy files are templates of permissions for a particular type of resource. You can use relative file paths in some dynamic policy files. The absolute path is resolved when the application is deployed. For more information, see “Syntax of policy files” on page 160.

Dynamic policy files

These files provide the permissions for an application:

- **app.policy**
  This file contains the default permissions for all of the enterprise applications in the cell. For more information, see “Configure the app.policy file” on page 164.
- **was.policy**
  This file contains application-specific permissions for a WebSphere Application Server - Express enterprise application. This file is packaged within an EAR file. For more information, see “Configure the was.policy file” on page 164.
- **ra.xml**
  This file contains connector-specific permissions for a particular WebSphere Application Server - Express enterprise application. This file is packaged within a RAR file.
- **spi.policy**
  This file contains permissions for a service provider interface (SPI) or third-party resources that are embedded in WebSphere Application Server - Express. For more information, see “Configure the spi.policy file” on page 165.
- **library.policy**
  This file contains permissions for Java library classes that are shared by enterprise applications. By default, this file is empty. For more information, see “Configure the library.policy file” on page 165.
- **filter.policy**
  This file contains a list of permissions that are filtered out of the was.policy and app.policy files in the cell. This filtering mechanism only applies to was.policy and app.policy. For more information, see “Configure the filter.policy file” on page 166.

Static policy files

These files provide default permissions. If permissions are required beyond the application level, you may need to update the static policy files. Note that the static policy file is not a configuration file that is managed by the WebSphere repository and file replication service. Changes to these files are local and are not replicated to other machines.

- **java.policy**
  This file contains default permissions for all of the Java programs that run in the node’s Java virtual machine. (On iSeries, this file is shipped with IBM Development Kit for Java.) By default, permissions are granted to all Java classes. Because this file represents permissions for all JVM processes, it is recommended that you do not modify its contents unless it is absolutely necessary. For more information, see “Configure the java.policy file” on page 167.
- **server.policy**
  This file contains default permissions for all WebSphere Application Server - Express programs on the node. By default, permissions are granted to all the product servers. Because this file represents permissions for all server processes, it is recommended that you do not modify its contents unless it is absolutely necessary. For more information, see “Configure the server.policy file” on page 168.

Here are some considerations when you edit Java 2 Security policy files:

- It is recommended that you update dynamic policy files rather than static policy files because the static policy files grant permissions beyond the application level.
- When you edit a policy file to add required permissions, it is recommended that you use the policy file of smallest scope. That way you can avoid giving a unnecessary permissions to applications, which better protects your resources. For example, update the ra.xml or was.policy file (these files define permissions for a single application) rather than the app.policy file (which defines permissions for all applications in the cell).
- Use specific component symbols, such as `${webComponent}`, `${connectorComponent}`, and `${jars}`, rather than `${application}` symbols.
- If there is any permission that should never be granted to the WebSphere Application Server - Express enterprise application within the cell, add this permission to the filter.policy file.
- After you have modified a dynamic policy file, restart the enterprise application.
- After you have modified a static policy file, restart the application server.

Troubleshooting

If a WebSphere Application Server - Express enterprise application within a cell requires permissions, some of the dynamic policy files may need to be updated. The symptom of a missing permission is a java.security.AccessControlException. For more information, see "AccessControlException" on page 168.

The missing permission is listed in the exception data, for example:

```
java.security.AccessControlException: access denied
(java.io.FilePermission /QIBM/ProdData/WebASE/ASE5/java/ext/mail.jar read)
```

When a Java program receives this exception and adding this permission is justified, add a permission to an adequate dynamic policy file, for example:

```
grant codeBase "file:${application}" {
  permission java.io.FilePermission
    "/QIBM/ProdData/WebASE/ASE5/java/ext/mail.jar", "read";
};
```

Syntax of policy files: This topic describes the syntax of Java 2 Security policy files that are supported by WebSphere Application Server - Express.

Syntax of the policy file

A policy file contains several policy entries. The format of each policy entry follows a specific format:

```
grant [codebase Codebase] {
  permission Permission;
  permission Permission;
  permission Permission;
};
```

where these variables are used:

- ***Codebase***
  This is a URL. For example: `"file:${java.home}/../lib/tools.jar"`
  Usage:
  - If `[codebase Codebase]` is not specified, listed permissions are applied to everything.
  - If the URL ends with a JAR file name, only the classes in the JAR file belong to the codebase.
  - If the URL ends with "/", only the class files in the specified directory belong to the codebase.
  - If the URL ends with "*", all JAR and class files in the specified directory belong to the codebase.
  - If the URL ends with "-", all JAR and class files in the specified directory and its subdirectories belong to the codebase.

Note: A `grant` entry that is specified in the app.policy and was.policy files must have a defined code base. If grant entries are specified without a code base, the policy files are not loaded properly, and the application can fail. If the intent is to grant the permissions to all applications, then use `file:${application}` as a code base in the grant entry.

- *Permissions*
  A permission is defined with these pieces of information:
  - A permission type, which is the class name of the permission.
  - A target name, which specifies the target.
  - Actions, which specify the actions that are allowed to be performed upon the target.

  Here is an example of a permission entry: `java.io.FilePermission "/tmp/xxx", "read,write"`

  For details about the permissions, see Permissions in the Java 2 SDK

  (http://java.sun.com/j2se/1.3/docs/guide/security/permissions.html)

Syntax of dynamic policy files

You can define permissions for specific types of resources in dynamic policy files for an enterprise application. You do this by using product-reserved symbols in the policy files.

The scope of the reserved symbol depends on where it is defined. If you define the permissions in the app.policy file, the symbol applies to all the resources on all of the enterprise applications that run in the node. If you define the permissions in the META-INF/was.policy file in your assembled application, it only applies to the specific enterprise application.

This list contains the valid symbols for the codebase entry:

- **`file:${application}`**
  Permissions apply to all resources within the application.
- **`file:${jars}`**
  Permissions apply to all utility JAR files within the application.
- **`file:${webComponent}`**
  Permissions apply to Web resources within the application.
- **`file:${connectorComponent}`**
  Permissions apply to connector resources both within the application and stand alone connector resources.

For a more granual setting, you can specify a particular module name. For example:

```
grant codebase "file:DefaultWebApplication.war" {
  permission java.security.SecurityPermission "printIdentity";
};

grant codebase "file:IncCMP11.jar" {
  permission java.io.FilePermission
   "${user.install.root}${/}bin${/}DefaultDB${/}-", "read,write,delete";
};
```

A relative codebase can be used only in the was.policy file.

There are embedded symbols provided that specify the path and name for java.io.FilePermission. These symbols enable you to specify more flexible permissions. The absolute file path is not fixed until after the application has been installed.

Here are the embedded symbols that are supported:

- **`${app.installed.path}`**
  Path where the application is installed.
- **`${was.module.path}`**
  Path where the module is installed.
- **`${current.cell.name}`**
  Current cell name.
- **`${current.node.name}`**
  Current node name.
- **`${current.server.name}`**
  Current server name.

Note: The `${was.module.path}` symbol cannot be used in the `${application}` entry.

Carefully determine where to add a new permission. An incorrectly specified permission causes an AccessControlException exception. Because DynamicPolicy resolves the codebase at run time, determining which policy file has a problem can be difficult. Add a permission only to the necessary resources. For example, use `${webComponent}` instead of `${application}`, and update the was.policy file instead of the app.policy file, if possible.

Filtering static policies

The policy model supports some limited static policy filtering. If the app.policy file and the was.policy file have permissions that are defined in filter.policy file, the run time removes the permissions from the applications and an audit message is logged. However, if the permissions defined in app.policy and was.policy are compound permissions (for example, java.security.AllPermission), the permission is not removed, and a warning message is written to the log file. Policy filtering only supports packages whose names begin with `java` or `javax`.

Run time policy filtering support is provided to force filtering that is more strict. If the app.policy file and was.policy file have permissions that are defined in the filter.policy file with the keyword `runtimeFilterMask`, the run time removes the permissions from the applications no matter which permissions are granted to the application. For example, even if a was.policy file grants java.security.AllPermission permission to a module, the specified permissions (such as runtimeFilterMask) are removed from the granted permission during run time.

If **Issue Permission Warning** is enabled in the Global Security panel of the administrative console and if the app.policy file and the was.policy file contain custom permissions (for packages that do not begin with `java` or `javax`), a warning message is logged and the permission is not removed. If the AllPermission permission is listed in both the app.policy file and the was.policy file, a warning message is logged.

Editing policy files

It is recommended that you use the policy tool (policytool) that is provided with the IBM Developer Kit for Java or the Sun Microsystems Java 2 Software Development Kit (in the `bin` subdirectory) to edit the policy files.

Note: If there are syntax errors in the policy files, the enterprise application or server process may fail to start. Extreme care should be taken when editing these policy files.

For more information, see “Create and edit policy files with the policy tool” on page 163.

Troubleshooting

To debug the dynamic policy, you can use these ways to generate the detail report of the AccessControlException exception:

- Enable tracing with the trace specification: `com.ibm.ws.security.policy.*=all=enabled:com.ibm.ws.security.core.SecurityManager=all=enabled`
- Enable first-fail data capture (FFDC). Modify the ffdcRun.properties file and specify `Level=4` and `LAE=true`. Look for a keyword **Access Violation** in the log file.

Create and edit policy files with the policy tool: Java 2 Security uses several policy files to determine the granted permission for each Java program. The Java development kit and the Java runtime environment provides the policytool graphical application to edit these policy files. While the policy tool is available as part of the iSeries IBM Developer Kit for Java, it is recommended that you run the policy tool on a workstation. The policy tool is located in the `bin` subdirectory of the Java development kit installation root or the Java runtime environment installation root.

It is recommended that you always use this tool to edit any policy file to guarantee the syntax of its contents. Syntax errors in the policy file causes an AccessControlException during server startup and application run time. Identifying the cause of an AccessControlException is not an easy task. Extreme care should be taken when editing these policy files.

1. Start the policy tool from a command prompt. For example, on a Windows 32-bit system which has the JRE installed in a directory named java, enter this command on the command line:
   ```
   C:\java\jre\bin\policytool
   ```
2. The PolicyTool window opens. The policy tool looks for the java.policy file in your home directory. If it does not exist, an error message displays. Click **OK**.
3. If you want to edit an existing policy file, click **File —> Open** Navigate to the policy file. Select it, and click **Open**. The code base entries are listed in the window.

   If you want to create a new policy file, click **File —> New**.
4. Create or modify a code base entry:
   - To modify the existing code base entry, select the entry, and click **Edit Policy Entry**. The Policy Entry window opens with the permission list that is defined for the selected code base.
   - To create a new code base entry, click **Add Policy Entry**. The Policy Entry window opens. In the **CodeBase** field, enter the code base information in URL format, for example: `/QIBM/UserData/WebASE/ASE5/`*`instance`*`/InstalledApps/testcase.ear`.
5. Modify or add the permission specification:
   - To modify an existing permission specification, click the entry you want to modify, and click **Edit Permission**. The Permissions window opens with the selecting permission information displayed.
   - To add a new permission, click **Add Permission**. The Permissions window opens.

   Perform these steps in the Permissions window:
   a. Select the permission from the **Permission** list. The selected permission displays. After a permission is selected, the **Target Name**, **Actions, and Signed By** fields automatically show the valid choices, or they enable text input in the right text input area.
   b. Select **Target Name** from the list, or enter the target name in the text field.
   c. Select **Actions** from the list.
   d. Enter a value in the **Signed By**, if necessary.
   e. Click **OK** to close the Permissions window.

   The modified permission entries of the specified code base are displayed.
6. Click **Done** to close the window. The modified code base entries are listed.
7. Repeat steps 4 through 6 until you complete editing.
8. Click **File —> Save** after you finish editing the file.

For more information about the policy tool, see Policy Tool (http://java.sun.com/j2se/1.3/docs/tooldocs/win32/policytool.html).

Configure the app.policy file: Java 2 Security uses several policy files to determine the granted permission for each Java program. The app.policy file is a default policy file that is shared by all of the WebSphere Application Server - Express enterprise applications. The union of the permissions that are contained in the app.policy file, server.policy file, the application's was.policy file, and the ra.xml file are applied to the enterprise application.

If the default permissions for enterprise application are enough, no action is required. If a specific change is required to all of the enterprise application in the cell, the app.policy file must be updated. Note that syntax errors in the policy files can cause the application server fail to start. Extreme care should be taken when editing these policy files.

Modify the app.policy file with policytool. For more information, see "Create and edit policy files with the policy tool" on page 163. The changes are local for the node.

The app.policy file that is supplied by WebSphere Application Server - Express resides at /QIBM/UserData/WebASE/ASE5/*instance*/config/cells/*cell*/nodes/*node*/app.policy, where *instance* is the name of your instance, *cell* is the name of your cell, and *node* is the name of your node.

The app.policy file contains these default permissions:

```
grant codeBase "file:${application}" {
  // The following are required by Java mail
  permission java.io.FilePermission
   "${was.install.root}${/}java${/}extlib${/}mail.jar", "read";
  permission java.io.FilePermission
   "${was.install.root}${/}java${/}extlib${/}activation.jar", "read";
};

grant codeBase "file:${jars}" {
  permission java.net.SocketPermission "*", "connect";
  permission java.util.PropertyPermission "*", "read";
};

grant codeBase "file:${connectorComponent}" {
  permission java.net.SocketPermission "*", "connect";
  permission java.util.PropertyPermission "*", "read";
};
grant codeBase "file:${webComponent}" {
  permission java.io.FilePermission "${was.module.path}${/}-", "read, write";
  permission java.lang.RuntimePermission "loadLibrary.*";
  permission java.lang.RuntimePermission "queuePrintJob";
  permission java.net.SocketPermission "*", "connect";
  permission java.util.PropertyPermission "*", "read";
};
```

If all of the WebSphere Application Server - Express enterprise applications within a cell require permissions that are not defined as defaults in the app.policy file, you may have to update the app.policy file, and possibly the server.policy file.

If you change the app.policy file, you must restart all enterprise applications to ensure that the updated app.policy file takes effect.

Configure the was.policy file: Java 2 Security uses several policy files to determine the granted permission for each Java program. The was.policy file is a application-specific policy file for an enterprise application. It is embedded in the EAR file (META-INF/was.policy).

The union of the permission contained in the java.policy file, server.policy file, the app.policy file, the application's was.policy file, and the permission specification of ra.xml file are applied to the enterprise application.

If the default permissions for enterprise application are enough, no action is required. If an application must access a specific resource, you may need to update the was.policy file. Note that syntax errors in the policy file may cause the application server to fail to start.

To create or update the was.policy file for your application, See "Create and edit policy files with the policy tool" on page 163. After you have created the was.policy file, you must add it to your application. See "Add the was.policy file to applications" on page 81 for more information.

Restart your enterprise application for the changes to take effect.

Configure the spi.policy file: This file contains permissions for a service provider interface (SPI) or third-party resources that are embedded in WebSphere Application Server - Express. Examples of SPIs are JDBC drivers. By default, the content of this file grants permission to everything. You may need to update this file when more permissions are required for SPI resources. However, use care when updating the file because its permissions are applied to all of the SPIs that are defined in resources.xml.

The union of the permissions that are contained in the java.policy file and spi.policy file are applied to the SPI libraries.

The WebSphere Application Server - Express spi.policy file is located in the /QIBM/UserData/WebASE/ASE5/*instance*/config/cells/*cell*/nodes/*node* directory, where *instance* is the name of your instance, *cell* is the name of your cell, and *node* is the name of your node.

The default spi.policy file contains the following default permission:

```
grant {
  permission java.security.AllPermission;
};
```

For the updated spi.policy file to take effect, you must restart all related Java processes.

Configure the library.policy file: The library.policy file is the template for Java library classes. Shared libraries can be defined and be used by multiple enterprise applications. For information about defining and managing shared libraries, see Administer shared libraries in the *Administration* topic.

If the default permissions for shared library (which is the union of the permissions that are defined in the java.policy file, the app.policy file, and the library.policy file) are adequate, no action is required. If a specific change is required to access a shared library in the cell, update the library.policy file.

The union of the permissions that are contained in the java.policy file, the app.policy file and the library.policy file are applied to the shared libraries.

The WebSphere Application Server - Express library.policy file in located in the /QIBM/UserData/WebASE/ASE5/*instance*/config/cells/*cell*/nodes/*node* directory, where *instance* is the name of your instance, *cell* is the name of your cell, and *node* is the name of your node.

By default, the file contains empty permission entry:

```
grant {
};
```

Use the policy tool to update the library.policy file for your instance. For more information, see "Create and edit policy files with the policy tool" on page 163.

Configure the filter.policy file: Java 2 Security uses several policy files to determine the granted permission for each Java programs. Java 2 Security policy filtering is only in effect when Java 2 Security is enabled. For more information about enabling Java 2 Security, see "Configure Java 2 security" on page 154.

The filtering policy that is defined in the filter.policy file is cell wide. The filter.policy file is the only policy file that is used to *restrict* permissions instead of granting permissions. The permissions that are listed in the filter policy file are filtered out of the app.policy and was.policy files. Permissions that are defined in the other policy files are not affected by the filter.policy file.

When a permission is filtered out, an audit message is logged. However, if the permissions that are defined in the app.policy and was.policy files are compound permissions (java.security.AllPermission, for example), the permission is not removed, and a warning message is logged. If **Issue Permission Warning** is enabled (it is enabled by default), and if the app.policy and was.policy files contain custom permissions (permissions for packages that do not start with `java` or `javax`), then a warning message is logged, and the permission is not removed. You can disable the **Issue Permission Warning** setting in the Global Security page of the administrative console. If java.security.AllPermission is specified in the app.policy file or in the was.policy file, a warning message is logged. However, the permission is not removed. It is recommended that you not use AllPermission for your enterprise applications.

Some default permissions are defined in the filter.policy file. These are the minimal permissions that are recommended. If more permissions are added to the filter.policy file, certain operations may fail for enterprise applications.

The WebSphere Application Server - Express filter.policy file is located in the /QIBM/UserData/WebASE/ASE5/*instance*/config/cells/*cell* directory, where *instance* is the name of your instance, and *cell* is the name of your cell.

By default, the filter.policy file contains these permissions:

```
filterMask {
  permission java.lang.RuntimePermission "exitVM";
  permission java.lang.RuntimePermission "setSecurityManager";
  permission java.security.SecurityPermission "setPolicy";
  permission javax.security.auth.AuthPermission "setLoginConfiguration";
};
runtimeFilterMask {
  permission java.lang.RuntimePermission "exitVM";
  permission java.lang.RuntimePermission "setSecurityManager";
  permission java.security.SecurityPermission "setPolicy";
  permission javax.security.auth.AuthPermission "setLoginConfiguration";
};
```

The permissions that are defined in the filterMask block are for static policy filtering. The security run time tries to remove the permissions from applications during application startup. Compound permissions are not removed and a warning is issued. Application deployment is stopped if applications contain permissions that are defined in filterMask and if scripting was used (wsadmin tool).

The runtimeFilterMask block defines the permissions to access resources that the security run time denies to the application thread. Do not add more permissions to the runtimeFilterMask because the application may fail to start or may function incorrectly. Usually, you only need to add permissions to the filterMask block.

WebSphere Application Server - Express itself relies on the filter.policy file to restrict or disallow certain permissions that could compromise the integrity of the system itself. For example, WebSphere Application Server - Express considers the exitVM and setSecurityManager permissions as those permissions that most applications should never have. If you grant these permissions to your application, then the following scenarios are possible:

- **exitVM**
  A servlet, JSP file, or other library could call the System.exit() API and cause the entire WebSphere Application Server - Express process to terminate.
- **setSecurityManager**
  An application could install its own SecurityManager that could either grant more permissions or bypass the default policy that the WebSphere Application Server - Express SecurityManager enforces.

If there are syntax errors in the filter.policy file, it is not loaded by the WebSphere security run time. This implies that there is no filter in place. If there is no filter, enterprise applications can access resources that are normally not allowed. Use extreme care editing the filter.policy file.

In Version 5, there is no tool support for editing the filter.policy file. You must use a text editor to edit the file.

For the updated filter.policy file to take effect, restart all related Java processes.

Configure the java.policy file: The java.policy file is a global default policy file that is shared by all of the Java programs that run on the iSeries system. It is never recommended to modify this file. If some Java programs on a node require permissions that are not defined as defaults in the java.policy file then add the permissions by configuring the server.policy file. For more information, see "Configure the server.policy file" on page 168.

The java.policy file is included as part of the Java development kit, and it is not managed by the WebSphere configuration and file replication services.

The java.policy file that is used by WebSphere Application Server - Express is located in the /QIBM/ProdData/Java400/jdk13/lib/security directory. It contains these default permissions:

```
grant codeBase "file:${java.home}/lib/ext/*" {
  permission java.security.AllPermission;
};

grant codeBase "file:/QIBM/ProdData/OS400/Java400/*" {
  permission java.security.AllPermission;
};

grant codeBase "file:/QIBM/ProdData/OS400/Java400/ext/*" {
  permission java.security.AllPermission;
};

grant codeBase "file:/QIBM/ProdData/Java400/*" {
  permission java.security.AllPermission;
};

grant {
  permission java.lang.RuntimePermission "stopThread";
  permission java.net.SocketPermission "localhost:1024-", "listen";
  permission java.util.PropertyPermission "java.version", "read";
  permission java.util.PropertyPermission "java.vendor", "read";
  permission java.util.PropertyPermission "java.vendor.url", "read";
  permission java.util.PropertyPermission "java.class.version", "read";
  permission java.util.PropertyPermission "os.name", "read";
  permission java.util.PropertyPermission "os.version", "read";
  permission java.util.PropertyPermission "os.arch", "read";
  permission java.util.PropertyPermission "file.separator", "read";
  permission java.util.PropertyPermission "path.separator", "read";
  permission java.util.PropertyPermission "line.separator", "read";
  permission java.util.PropertyPermission "java.specification.version", "read";
  permission java.util.PropertyPermission "java.specification.vendor", "read";
  permission java.util.PropertyPermission "java.specification.name", "read";
  permission java.util.PropertyPermission "java.vm.specification.version", "read";
  permission java.util.PropertyPermission "java.vm.specification.vendor", "read";
```

```
    permission java.util.PropertyPermission "java.vm.specification.name", "read";
    permission java.util.PropertyPermission "java.vm.version", "read";
    permission java.util.PropertyPermission "java.vm.vendor", "read";
    permission java.util.PropertyPermission "java.vm.name", "read";
  };
```

Configure the server.policy file: The server.policy file is a default policy file for the server. If the default permissions for the server are adequate, no action is required. If a specific change is required to some of the server programs on a node, update the server.policy file. If you want to add permissions to an application, use the app.policy and was.policy files.

If some server programs on a node require permissions that are not defined as defaults in the java.policy file and the server.policy file, use the policy tool to update the server.policy file. To decide whether to add a permission, see "AccessControlException."

The server.policy file is located in the /QIBM/UserData/WebASE/ASE5/*instanceName*/properties directory where *instanceName* is the name of your instance. It contains these default permissions:

```
  grant codeBase "file:${was.install.root}/java/extlib/-" {
    permission java.security.AllPermission;
  };

  grant codeBase "file:${was.install.root}/java/tools/ibmtools.jar" {
    permission java.security.AllPermission;
  };

  grant codeBase "file:/QIBM/ProdData/Java400/jdk13/lib/tools.jar" {
    permission java.security.AllPermission;
  };

  grant codeBase "file:${was.install.root}/lib/-" {
    permission java.security.AllPermission;
  };

  grant codeBase "file:${was.install.root}/classes/-" {
    permission java.security.AllPermission;
  };

  grant codeBase "file:${was.install.root}/deploytool/-" {
    permission java.security.AllPermission;
  };
```

Use the policy tool to modify the server.policy file. For more information, see "Create and edit policy files with the policy tool" on page 163.

After you have updated the server.policy file, restart all of the Java processes for the updated server.policy file to take effect.

AccessControlException: The Java 2 Security behavior is specified by its security policy. The security policy is an access-control matrix that specifies which system resources that certain code bases can access and who must sign them. The Java 2 Security policy is declarative and is enforced by the java.security.AccessController.checkPermission() method.

The following code example is the algorithm for the java.security.AccessController.checkPermission() method where caller `m` invoked the java.security.AccessController.checkPermission() method. For the full algorithm, see the API documentation for Class AccessController

```
i = m;
while (i > 0) {
  if (caller i's domain does not have the permission)
```

```
    throw AccessControlException;
  else if (caller i is marked as privileged)
    return;
  i = i - 1;
};
```

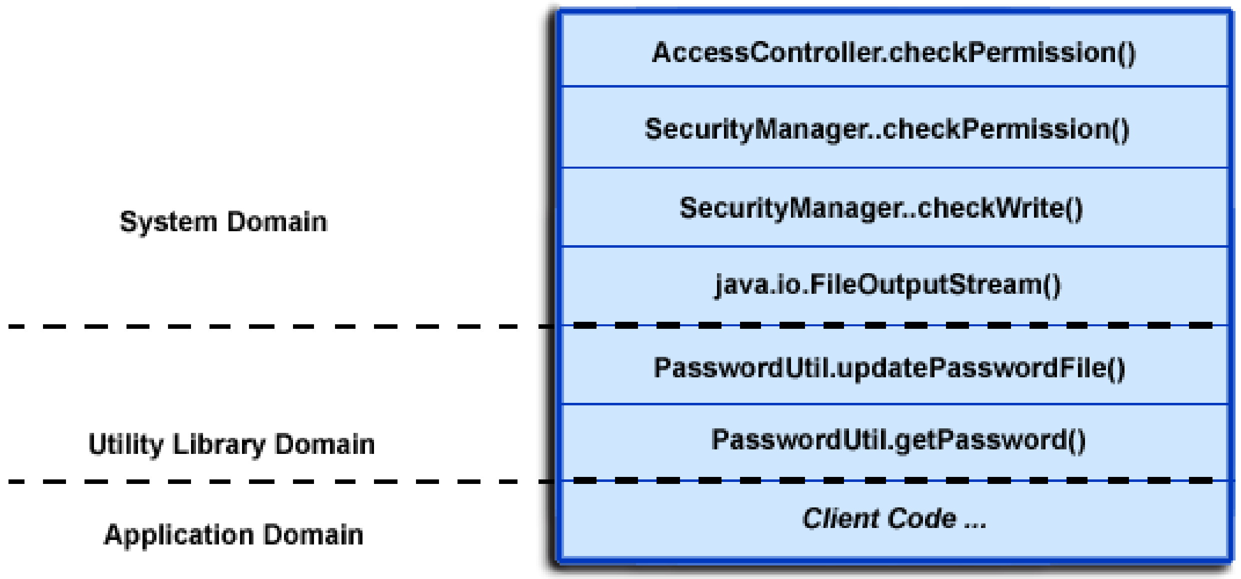
The algorithm requires that all the classes (callers) on the call stack be granted the said permissions when a java.security.AccessController.checkPermission() is performed, or the request is denied (a java.security.AccessControlException is thrown). However, if the caller is marked as privileged and the class (caller) is granted the said permissions, the algorithm returns at that point and does not walk the entire call stack. Subsequent classes (callers) do not need to be granted the required permission.

A java.security.AccessControlException exception is thrown as a result of certain classes on the call stack missing the required permissions during a java.security.AccessController.checkPermission() method. The following are the two possible resolutions to the java.security.AccessControlException exception:

- If the application is calling an API that is protected by Java 2 Security, then grant the required permission to the application Java 2 Security policy.
- If the application does **not** directly call an API that is protected by Java 2 Security, but is third-party code that accesses protected resources, you have to option to grant the application additional privileges. However, if granting the additional permissions gives the application more access than it should have, then it is likely that the application is not properly marked as privileged.

Example call stack

This is an example of a call stack where an application code is using a third party API utility library to update the password. The following is only an example to illustrate the point. It is not the ultimate guide of where to mark the code as privileged. The decision as to where is the appropriate place to mark the code as privileged is application specific and is unique in every situation, requiring great depth of domain knowledge and security expertise to make the correct judgement. There are a number of well written publications and books on this topic, it is highly recommended that you reference these materials for more detailed information.


AccessController.checkPermission()
SecurityManager..checkPermission()
System Domain
SecurityManager..checkWrite()
java.io.FileOutputStream()
PasswordUtil.updatePasswordFile()
Utility Library Domain
PasswordUtil.getPassword()
Application Domain
Client Code ...

You can use the PasswordUtil utility to change the password of a user. You type in the old password and the new password twice to ensure that the correct password is entered. If the old password matches the one stored in the password file, the new password is stored and updates the password file. Lets assume that none of the stack frame is marked as privileged. According to the java.security.AccessController.checkPermission() algorithm, the application fails unless all the classes on the call stack are granted write permission to the password file. The client application should not be granted the permission to write to the password file directly and update the password file at will.

However, if the PasswordUtil.updatePasswordFile() method marks the code that accesses the password file as privileged, then the check permission algorithm does not check for the required permission from classes that call the PasswordUtil.updatePasswordFile() method for the required permission as long as the PasswordUtil class is granted the permission. Then the client application can successfully update a password without granting the permission to write to the password file.

The ability to mark code privileged is very flexible and powerful, yet it is a double edged sword. If this ability is not used correctly, the overall security of the system can be compromised and security holes will be exposed. The ability to mark code privileged must be used with extreme care.

Note: Domain knowledge and security expertise is required to decide where to mark the code as privileged. A security exposure can result from code that is incorrectly marked.

Resolution to java.security.AccessControlException

As described previously, there are two possibilities to resolve a java.security.AccessControlException exception. Judge these case by case to decide which of the following is the best resolution to the problem:

- Grant the missing permission to the application.
- Mark some code as privileged (keeping in mind concerns and risk of doing so).

Configure Java Authentication and Authorization Service login

Java Authentication and Authorization Service (JAAS) is a new feature in WebSphere Application Server - Express. JAAS is a collection of WebSphere Application Server - Express strategic authentication APIs for and replace the CORBA programmatic login APIs. WebSphere Application Server - Express has provided some extensions to JAAS:

- **com.ibm.websphere.security.auth.WSSubject**
  Due to a design oversight in JAAS V1.0, javax.security.auth.Subject.getSubject() does not return the subject associated with the thread of execution inside a java.security.AccessController.doPrivileged() code block. This presents an inconsistent behavior that is problematic and causes undesirable effort. The com.ibm.websphere.security.auth.WSSubject API provides a workaround to associate the subject to thread of execution. The com.ibm.websphere.security.auth.WSSubject API extends the JAAS authorization model to J2EE resources.

  Version 5.0.1 or later: You can retrieve the subjects in a Subject.doAs() block with the Subject.getSubject() call. However, this procedure does not work if there is an AccessController.doPrivileged() call within the Subject.doAs() block. In the following example, s1 is equal to s, but s2 is null:

  ```
  Subject.doAs(s, new PrivilegedAction() {
    public Object run() {
      System.out.println("Within Subject.doAsPrivileged()");
      Subject s1 = Subject.getSubject(AccessController.getContext());
      AccessController.doPrivileged(new PrivilegedAction() {
        public Object run() {
          Subject s2 = Subject.getSubject(AccessController.getContext());
          return null;
        }
      }
      return null;
    }
  }
  ```
- You can configure JAAS login in the WebSphere administrative console. However, WebSphere Application Server - Express still supports the default JAAS login configuration format (plan text file) that is provided by the JAAS default implementation. If there are duplicate login configurations

defined in both the administrative configuration and the plan text file format, the one in the configuration takes precedence. There are advantages to defining the login configuration with the administrative console:

– User interface support in defining JAAS login configuration.
– You can manage the JAAS configuration login configuration centrally.

- **Proxy LoginModule.**
  The default JAAS implementation does not use the thread context class loader to load classes. The LoginModule can not load if the LoginModule class file is not in the application class loader or the Java extension class loader class path. Due to this class loader visibility problem, WebSphere Application Server - Express provides a proxy LoginModule to load JAAS LoginModule using the thread context class loader. You do not need to place the LoginModule implementation on the application class loader or the Java extension class loader classpath with this proxy LoginModule.

Note: Do not remove or delete the pre-defined JAAS login configurations (ClientContainer, WSLogin and DefaultPrincipalMapping). Deleting or removing them could cause other enterprise applications to fail.

Create a new JAAS login configuration

Perform these steps in the administrative console:

1. Click **Security** in the navigation tree.
2. Click **JAAS Configuration —> Application Logins**.
3. Click **New**. The Application Login Configuration panel appears.
4. Specify the alias name of the new JAAS login configuration and click **Apply**. This is the name of the login configuration that you pass in the javax.security.auth.login.LoginContext for creating a new LoginContext.
5. Click **JAAS Login Modules**.
6. Click **New**.
7. Specify the **Module Classname**. It is recommended that you specify WebSphere Proxy LoginModule because of the limitation of the class loader visibility problem.
8. Specify the **LoginModule** implementation as the delegate property of the Proxy LoginModule. The WebSphere Proxy LoginModule classname is com.ibm.ws.security.common.auth.module.proxy.WSLoginModuleProxy.
9. Select **Authentication Strategy** from the drop down list and click **Apply**.
10. Click **Custom Properties**. This will navigate to the Custom Properties panel for the selected LoginModule.
11. Create a new property with the name delegate with the value being the real LoginModule implementation. You can specify other properties like debug with value true. These properties are passed to the LoginModule as options to initialize() method of the LoginModule.
12. Click **Save**.

Change the plain text file

WebSphere Application Server - Express supports the default JAAS login configuration format (plain text file) provided by the JAAS default implementation. However, there is no tool provided to edit plain text file in this format. You can define the JAAS login configuration in the plain text file wsjaas.conf (which is located in the `properties` subdirectory of your instance root, for example /QIBM/UserData/WebASE/ASE5/*instanceName*/properties), any syntax errors can cause the plain JAAS Login Configuration text file to not parse correctly. This could cause other applications to fail.

Note: Do not remove or delete the pre-defined JAAS login configurations (ClientContainer, WSLogin, system.SWAM and system.LTPA). Deleting or removing them could cause other enterprise applications to fail.

For more information about editing the plain-text JAAS configuration file, see JAAS 1.0 Developer's Guide

.

Restart the application servers to validate changes to the plain-text file.

Configure J2C authentication data entries

Java 2 Connector (J2C) authentication data entries are used by resource adapters and JDBC data sources. A Java 2 Connector authentication data entry contains authentication data, which contains the following information:

- **Alias**
  An identifier used to identify the authenticated data entry. When configuring resource adapters or Java database connectivity (JDBC) data sources, the administrator can specify which authentication data to choose for the corresponding alias.
- **User ID**
  A user identity that is used to connect to the resource adapter of the intended security domain.
- **Password**
  The password of the user identity is encoded in the configuration respository.
- **Description**
  A short text description.

Create a new J2C authentication data entry

Perform the following steps in the WebSphere administrative console to create a new J2C authentication data entry:

1. In the navigation menu, expand **Security** and then click **JAAS Configuration —> J2C Authentication Data**.
2. In the J2C Authentication Data Entries panel, click **New**.
3. Enter a unique alias, a value user ID, a valid password, and a short description (optional).
4. Click **OK** or **Apply**. Note that no validation is performed on the user ID and password.
5. Click **Save**.

You can use the newly created entry without restarting the application server process. Specifically, the authentication data is loaded by an application server when starting an application and is shared among applications in the same application server.

Delete a J2C authentication data entry

Perform the following steps in the WebSphere administrative console to delete a J2C authentication data entry:

1. Before you delete or remove an authentication data entry, make sure that it is not used or referenced by any resource adapter or JDBC data source. If the deleted authentication data entry is used or referenced by a resource, the application that uses the resource adapter or JDBC data source fails to connect to the resources.
2. In the navigation menu, expand **Security** and then click **JAAS Configuration —> J2C Authentication Data**.
3. In the J2C Authentication Data panel, select the check boxes for the entries to delete and click **Delete**.

Tune your security configuration

Performance issues typically involve trade-offs between function and speed. Usually, the more function and the more processing involved, the slower the performance will be. Consider what type of security is necessary and what you can disable in your environment. For example, if your application servers are running in a Virtual Private Network (VPN), consider whether you must disable Single Sockets Layer (SSL). If you have a lot of users, can they be mapped to groups and associated the groups to your J2EE roles? These questions are things to consider when designing your security infrastructure.

There is always a trade-off between performance, feature and security. Security typically adds more processing time to your requests, but for a good reason. Not all security features are required in your environment. When you decide on tuning security, you should create a benchmark before making any change to ensure the change is improving performance.

In a large scale deployment, performance is very important. Running benchmark measurements with different combinations of features can help you to determine the best performance versus benefit configuration for your environment.

See these topics for more information about how you can tune your WebSphere security configuration:

"General security tuning tips"
See this topic for tips that can improve the general performance of WebSphere security.

"Tune CSIv2" on page 174
If you are using the Common Security Interoperability Version 2 (CSIv2) authentication protocol, this topic offers considerations that can tune your configuration.

"Tune LDAP authentication" on page 175
If you are using an LDAP user registry, see this topic for pointers to optimize the authentcation process.

"Tune Web authentication" on page 175
If you are using browser-based authentication, see this topic for tuning tips.

"Tune authorization" on page 175
See this topic for ways to speed up the authorization process.

"SSL performance tips" on page 175
If you are using SSL connections with WebSphere Application Server, see this topic for tips to improve performance.

General security tuning tips

The following are some general tuning tips for WebSphere security configurations:

1. Consider disabling the Java 2 Security Manager, if you know exactly what code is put onto your server and you do not need to protect process resources. Remember that in doing so you are putting your local resources at some risk.
2. If you feel your environment is secure enough, consider increasing the cache and token time-out settings. (These settings are available as general properties on the Global Security panel in the WebSphere administrative console.) By doing so, re-authentication is less frequently required. This action allows subsequent requests to more frequently reuse the credentials that are already created. The downside of increasing the token time-out is the exposure of having a token highjacked. The higher time-out setting provides the highjacker more time to hack into the system before the token expires. You can use security cache properties to determine the initial size of the primary and secondary Hashtable caches, which affect the frequency of rehashing and the distribution of the hash algorithms. See "Security cache properties" on page 174 for a list of these properties.

3. Consider changing your administrative connector from Simple Object Access Protocol (SOAP) to Remote Method Invocation (RMI). RMI uses stateful connections while SOAP is completely stateless. Run a benchmark test to determine if the performance has been improved in your environment. This control only affects the performance of the administrative application.
4. Use the wsadmin script to complete the access IDs for all the users and or groups to speed up the application startup. Complete this action if applications contain many users and or groups or if applications are stopped and started frequently. For more information, see The wsadmin administrative tool in the *Administration* topic.
5. Consider whether you really need SSL to be enabled for connections that are used by the Web server plug-in. These are long-lived connections, while those used to connect browsers to the Web server are typically short lived. Hence, enabling SSL for connections that are used by the Web server plug-in generally has a smaller impact on performance than enabling SSL for connections between browsers and the Web server. However, SSL protection for your plug-in connections may not be required if sufficient security is already provided. For example, you may decide plug-in connections are sufficiently secure if the Web server and application server are protected by a firewall.

Security cache properties: The following system properties determine the initial size of the primary and secondary hashtable caches, which affect the frequency of rehashing and the distribution of the hash algorithms. The larger the number of available hash values, the less likely a hash collision occurs, and the more likely a slower retrieval time. If several entries compose a hashtable cache, creating the table in a larger capacity supports more efficient hash entries than letting automatic rehashing determine the growth of the table. Rehashing causes every entry to move each time.

- **com.ibm.websphere.security.util.authCacheSize**
  This cache stores basic authentication credentials at the security server. Whenever a Lightweight Third Party Authentication (LTPA) token expires, a new token generates from the basic authorization credentials in this cache. If no basic authorization credentials exist, the requesting browser must send the basic authorization credentials to the security server. The browser prompts the user for a user ID and password if no cookie exists that contains the credentials.
- **com.ibm.websphere.security.util.tokenCacheSize**
  This cache stores LTPA credentials in the cache, using the LTPA token as a lookup value. When using an LTPA token to login, the LTPA credential is created at the security server for the first time. This cache prevents the need to access the security server on subsequent logins using an LTPA token.
- **com.ibm.websphere.security.util.CredentialCacheSize**
  Given the user ID and password for login, this cache returns the concrete credential object, either LocalOS or LTPA, without the need to repeat authentication at the security server. If the credential object has expired, repeat authentication is required.
- **com.ibm.websphere.security.util.LTPAValidationCacheSize**
  Given the credential token for login, this cache returns the concrete LTPA credential object, without the need to revalidate at the security server. If the token has expired, revalidation is required.

Tune CSIv2

To tune the Common Security Interoperability Version 2 (CSIv2) authentication protocol, consider the following tasks:

1. If you send a large amount of data that is not very sensitive, reduce the strength of your ciphers. A strong cipher takes longer to encrypt data in bulk. If the data is not sensitive, processing with 128-bit ciphers may not be worth the effort.
2. Consider putting just an asterisk (*) in the trusted server ID list (this means that all servers are trusted) when you use Identity Assertion for downstream delegation. Use SSL mutual authentication between servers to provide this trust. Adding this extra step in the SSL handshake performs better than having to fully authenticate the upstream server and check the trusted list. When an asterisk is used, the identify token is trusted. The SSL connection trusts the server by way of client certificate authentication.
3. Ensure that stateful sessions are enabled for CSIv2. This is the default, but it only requires authentication on the first request and on any subsequent token expirations.

4. If you are only communicating with WebSphere Application Server Version 5 servers, specify only **CSI** rather than **CSI and SAS** for the **Active Authentication Protocol** setting. This action removes an interceptor invocation for every request on both the client and server sides.

Tune LDAP authentication

To tune the LDAP authentication process, consider the following steps:

1. When case-sensitivity is not critical, select the **Ignore Case** check box in the LDAP User Registry panel in the WebSphere administrative console.
2. Select **Reuse Connections** in your LDAP User Registry configuration.
3. Check to see what caching features your LDAP server provides, and take advantage of them. This action is best suited to LDAP servers that do not change frequently.
4. Starting with WebSphere Application Server - Express for iSeries Version 5.0.1, you can choose the directory type of `IBM_Directory_Server` instead of `SecureWay` if you are using the i5/OS Directory Services product on V5R2 and have upgraded to LDAP 4.1. With LDAP 4.1, i5/OS Directory Services yields improved performance because it is programmed to use the new group membership attributes to optimize group membership searches. However, it is required that authorization is case-insensitive, so also select **Ignore Case** when you choose the directory type of `IBM_Directory_Server`. See iSeries LDAP: What's New

   (http://www-1.ibm.com/servers/eserver/iseries/ldap/whatsnew.htm) for information regarding the PTFs that are required for LDAP 4.1.

Tune Web authentication

To tune the Web authentication process, consider the following steps:

1. If you feel your environment is secure enough, consider increasing the cache and token time-out settings. (These settings are available as general properties on the Global Security panel in the WebSphere administrative console.) By doing so, re-authentication is less frequently required. This action allows subsequent requests to more frequently reuse the credentials that are already created. The downside of increasing the token time-out is the exposure of having a token highjacked. The higher time-out setting provides the highjacker more time to hack into the system before the token expires. You can use security cache properties to determine the initial size of the primary and secondary Hashtable caches, which affect the frequency of rehashing and the distribution of the hash algorithms. See "Security cache properties" on page 174 for a list of these properties.
2. Consider enabling Single sign-on (SSO). SSO is only available when you select LTPA as the authentication mechanism in the Global Security panel of the WebSphere administrative console. When you select SSO, a single authentication to one application server is adequate to make requests to multiple application servers in the same SSO domain. There are some situations where SSO is not desirable, so SSO should not be used in these situations. For more information about SSO, see "Configure single signon" on page 101.

Tune authorization

To tune the authorization process, consider the following steps:

1. Consider mapping your users to groups in the user registry. Then, associate the groups with your J2EE roles. This association greatly improves performance as the number of users increase.
2. Judiciously assign security-constraints for servlets. For example, you can use the URL pattern `*.jsp` to apply the same authentication data constraints to indicate all JSP files. For a given URL, the exact match in the deployment descriptor takes precedence over longest path match. Use the extension match (*.jsp , *.do , *.html) if there is no exact match and longest path match for a given URL in the security constraints.

SSL performance tips

The following are two types of Secure Sockets Layer (SSL) performance:

- Handshake

- Bulk encryption and decryption

When an SSL connection is established, an SSL handshake occurs. After a connection is made, SSL performs bulk encryption and decryption for each read-write. The performance cost of an SSL handshake is much larger than that of bulk encryption and decryption.

Using a single SSL connection to send multiple requests (rather than opening a connection for each request) can enhance SSL performance. Additionally, it is important to reduce the the number of times a new connection is opened. Decreasing the overall number of connections increases performance for secure communication through SSL connections, as well as non-secure communication through simple TCP/IP connections. One way to send multiple requests over a single SSL connection us to use a browser that supports HTTP 1.1.

Another common approach is to decrease the number of connections (both TCP/IP and SSL) between two WebSphere Application Server components. The following guidelines help to verify the HTTP transport of the application server is configured so that the Web server plug-in does not repeatedly open new connections to the application server:

- Verify that the maximum number of keep-alives are, at minimum, as large as the maximum number of threads supported by the Web server. Make sure that the Web server plug-in is capable of obtaining a keep-alive connection for every possible concurrent connection to the application server. Otherwise, the application server closes the connection after a single request is processed. Also, the maximum number of threads in the Web container thread pool should be larger than the maximum number of keep-alive connections, to prevent the keep-alive connections from consuming the Web container threads. For more information about this setting, see "Example: Setting custom properties for an HTTP transport" on page 177.
- Consider increasing the maximum number of requests per keep-alive connection. The default value is 100, which means the application server closes the connection from the plug-in after 100 requests. The plug-in then must open a new connection. The purpose of this parameter is to prevent denial-of-service attacks when connecting to the application server and preventing continuous send requests to tie up threads in the application server. See Example: Setting custom properties for an HTTP transport for more information about this setting.
- Use a hardware accelerator if the system performs several SSL handshakes.

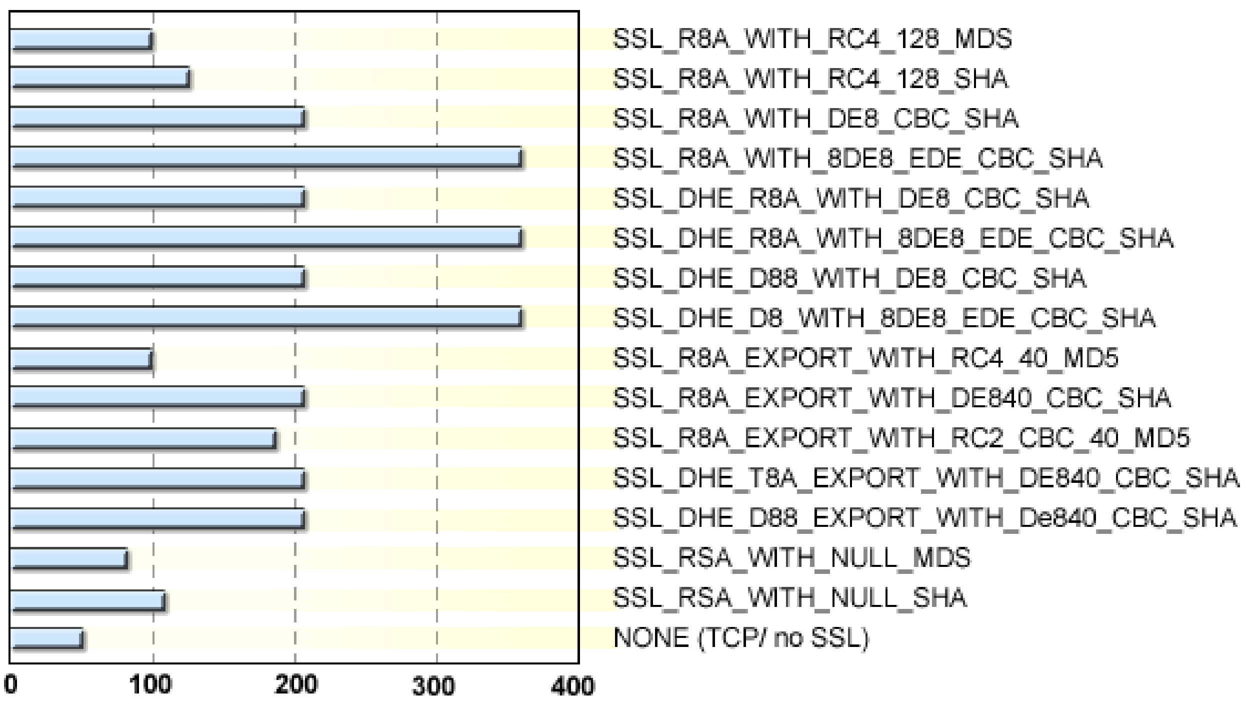
  Note: The IBM Cryptopgraphic Coprocessor for iSeries is not supported for use with WebSphere Application Server. However, you can use the IBM Cryptographic Coprocessor to improve SSL performance for other iSeries products, such as IBM HTTP Server for i5/OS (powered by Apache).

  An accelerator typically only benefits the Web server because Web server connections are short-lived. Generally, WebSphere Application Server connections are more long-lived, so using a hardware accelerator does not significantly improve SSL performance for WebSphere Application Server.

Another way to improve performance is to use an alternative cipher suite that more quickly encrypts and decrypts data.

The performance of a cipher suite is different with software and hardware. Just because a cipher suite performs better in software does not mean a cipher suite performs better with hardware. Some algorithms are typically inefficient in hardware (for example, DES and 3DES), however, specialized hardware can provide efficient implementations of these same algorithms.

The performance of bulk encryption and decryption is affected by the cipher suite used for an individual SSL connection. The following chart displays the performance of each cipher suite:



The test software for calculating the data uses Java Secure Socket Extension (JSSE) for both the client and server software, which uses no cryptographic hardware support. The test did not include the time for establishing a connection, but only the time to transmit data through an established connection. Therefore, the data reveals the relative SSL performance of various cipher suites for long running connections.

Before establishing a connection, the client enables a single cipher suite for each test case. After the connection is established, the client measures how long it takes to write an integer to the server and for the server to write the specified number of bytes back to the client. Varying the amount of data has negligible effects on the relative performance of the cipher suites.

An analysis of the above reveals the following analysis:
- Bulk encryption performance is only affected by what follows the WITH in the cipher suite name. This is expected since the portion before the WITH identifies the algorithm used only during the SSL handshake.
- MD5 and SHA are the two hash algorithms that are used to provide data integrity. MD5 is 25% faster than SHA, however, SHA is more secure than MD5.
- DES and RC2 are slower than RC4. Triple DES is the most secure, but the performance cost is high when using only software.
- The cipher suite providing the best performance while still providing privacy is SSL_RSA_WITH_RC4_128_MD5. Even though SSL_RSA_EXPORT_WITH_RC4_40_MD5 is cryptographically weaker than RSA_WITH_RC4_128_MD5, the performance for bulk encryption is the same. Therefore, as long as the SSL connection is a long-running connection, the difference in the performance of high and medium security levels is negligible. It is recommended that a security level of high be used, instead of medium, for all components participating in communication only among WebSphere Application Server products. Make sure that the connections are long running connections.

Example: Setting custom properties for an HTTP transport: WebSphere Application Server - Express has several transport properties that are not shown in the WebSphere administrative console in the settings page for an HTTP transport. They are as follows:

- **ConnectionIOTimeout**
  Specifies the maximum number of seconds to wait when trying to read or process data during a request. Data type: Integer.
- **ConnectionKeepAliveTimeout**
  Specifies the maximum number of seconds to wait for the next request on a keep-alive connection. Data type: Integer.
- **MaxKeepAliveConnections**
  Specifies the maximum number of concurrent keep-alive (persistent) connections across all HTTP transports. To make a particular transport close connections after a request, you can set MaxKeepAliveConnections to `0` (zero) or you can set KeepAliveEnabled to `false` on that transport.

  The Web server plug-in keeps connections open to the application server as long as it can. However, if the value of this property is too small, performance is negatively impacted because the plug-in has to open a new connection for each request instead of sending multiple requests through one connection. The application server may not accept a new connection under a heavy load if there are too many sockets in TIME_WAIT state. If all client requests are going through the Web server plug-in and there are many TIME_WAIT state sockets for port 9080, the application server is closing connections prematurely, which decreases performance. The application server closes the connection from the plug-in, or from any client, for any of the following reasons:
  - The client request was an HTTP 1.0 request when the Web server plug-in always sends HTTP 1.1 requests.
  - The maximum number of concurrent keep-alives was reached. A keep-alive must be obtained only once for the life of a connection, that is, after the first request is completed, but before the second request can be read.
  - The maximum number of requests for a connection was reached, preventing denial of service attacks in which a client tries to hold on to a keep-alive connection forever.
  - A time out occurred while waiting to read the next request or to read the remainder of the current request.

  Data type: Integer. Default: 90% of the maximum number of threads in the Web container thread pool. This prevents all of the threads from being held by keep alive connections so that there are threads available to handle new incoming connect requests.
- **MaxKeepAliveRequests**
  Specifies the maximum number of requests which can be processed on a single keep alive connection. This parameter can help prevent denial-of-service attacks when a client tries to hold on to a keep-alive connection. The Web server plug-in keeps connections open to the application server as long as it can, providing optimum performance. Data type: Integer. Default: 100.
- **KeepAliveEnabled**
  Specifies whether to keep connections alive or not. Data type: Boolean. Default is `true`.

  You can set these properties on either the Web Container or HTTP Transport Custom Properties pages. When set on the Web container Custom Properties page, all transports inherit the properties. Setting the same properties on a transport overrides like settings defined for a Web container.

To specify values for these custom properties for a specific transport on the HTTP Transport Custom Properties page:
1. Access the settings page for transport properties in the WebSphere administrative console:
   a. In the navigation menu, click **Servers —> Application Servers —>** *server* **—> Web Container —> HTTP Transport**, where *server* is the name of your application server.
   b. Click the host whose properties you want to set.
   c. Under **Additional Properties** click **Custom Properties**.
   d. On the Custom Properties page, click **New**.

2. On the settings page for a new property, enter the name of the transport property and the value to which you want it set. For example, if you want the transport to wait a maximum of 60 seconds when trying to read or write data during a request, enter `ConnectionIOTimeout` for the name and `60` for the value. Then click **OK**.
3. Click **Save** on the console taskbar and save the changes to the configuration.
4. Restart the server.
5. Regenerate the Web server plug-in.

Run application servers under specific user profiles

You can run an application server under a user profile other than the default QEJBSVR user profile.

1. Choose an existing user profile, or create a new user profile with the Create User Profile (CRTUSRPRF) command.

   The new user profile must have authority to the same objects to which QEJBSVR has authority. Specify QEJBSVR as the new profile's group profile. Run this command from the iSeries command line:

   ```
   CHGUSRPRF USRPRF(profile) GRPPRF(QEJBSVR)
   ```

   where *profile* is the name of the new user profile.
2. (Optional) If the application server is currently running under a user profile other than QEJBSVR, run the following commands in Qshell (where *instance* is the name of your instance):
   a.
   ```
   chown -R QEJBSVR /QIBM/UserData/WebASE/ASE5/instance
   ```
   b.
   ```
   cd /QIBM/ProdData/WebASE/ASE5/bin
   ```
   c.
   ```
   grtwasaut -user QEJBSVR -dtaaut RWX -objaut ALL -recursive
   ```
   d. (Optional) If the old user profile is no longer used to run any of the servers in the instance, you can revoke the authorities that it has by running the following command:
   ```
   rvkwasaut -instance instance -user profile -recursive
   ```
   where *profile* is the name of the old user profile.
3. "Enable user profiles to run application servers with iSeries Navigator."
4. Specify the new user profile name in the Run As User property of the application server:
   a. Start the WebSphere administrative console.
   b. In the navigation menu, expand **Servers**, and click **Application Servers**.
   c. In the Application Servers page, click the name of your application server.
   d. Under Additional Properties, click **Process Definition**.
   e. Under Additional Properties, click **Process Execution**.
   f. Enter the name of the user profile in the **Run As User** field.
   g. Click **OK**
   h. Save your configuration.
   i. Restart the application server.

Enable user profiles to run application servers with iSeries Navigator

With the exception of the default user profile (QEJBSVR), all user profiles that are to be used to run an application server must be enabled through iSeries Navigator for the WebSphere application.

The user profile that you use to run Operations Navigator must have *SECADM special authority to perform this task.

1. Open iSeries Navigator.

2. Right click the iSeries icon that contains the Application Server.
3. Select **Application Administration** on the pop-up menu.
4. Click the **Host Applications** tab.
5. Expand **QIBM_EJB_PRODUCT**.
6. Expand **QIBM_EJB_GROUP_OF_FUNCS**.
7. Select **QIBM_EJB_SERVER_FUNC**, and click **Customize**.
8. In the **Users and groups** list box, select the user profile under which you want the application server to run.
9. Click the top **Add** button to add the user profile to the **Usage allowed** list and then click **OK**.
10. On the Application Administration panel, click **OK**.

Securing iSeries objects and files

This topic discusses the various iSeries objects and files that contain sensitive information and need to be protected.

Secure integrated file system files

In addition to servlets and JSP files, the WebSphere administrative application and application servers access integrated file system stream files. The following files may contain sensitive information and should be given close consideration to ensure no unauthorized access is granted:

- Some files that are located in the `properties` subdirectory of your instance (for example, /QIBM/UserData/WebASE/ASE5/*instance*/properties) can contain user IDs and passwords.

  By default, these files are shipped with *PUBLIC authority set to *EXCLUDE. The QEJBSVR user profile is granted *RX authority to these files. Additional protection is available through password encoding. For more information, see "Password encoding" on page 181.
- In the `etc` subdirectory if your instance, all key (KDB) files and trust (JKS) files that you create for your WebSphere Application Server - Express instance should be protected:
  - For the JKS files, the QEJBSVR user profiles should have *R authority and *PUBLIC should have *EXCLUDE authority.
  - For the KDB files, the user profile that the Web server is running under should have *RX authority and *PUBLIC should have *EXCLUDE authority.

Secure the WebSphere server

When you enable WebSphere security, the server's user profile and password are placed into server configuration files which should be maintained in a secure way using i5/OS system security. Additionally, some WebSphere resources can be password-protected, and these passwords are also placed in server configuration files. The server automatically encodes passwords to deter casual observation, but password encoding alone is not sufficient protection.

These files are located in the `config` subdirectory of your instance, and they can contain user identifiers and passwords:

- config/cells/*cell_name*/security.xml
- config/cells/*cell_name*/nodes/*node_name*/resources.xml
- config/cells/*cell_name*/nodes/*node_name*/servers/*server_name*/server.xml

where *cell_name* is the name of the cell, *node_name* is the name of the node, and *server_name* is the name of the application server.

The server's user profile and password are used for authenticating the server when it initializes. This authentication is required for these reasons:

- The user ID and password are used as the System Identity for the server when a bean's security has been deployed to use SYSTEM_IDENTITY for method delegation. In this case, the user ID and password are used when method calls are are made from one enterprise bean to another.
- The user ID and password are used to authenticate servers for inter server communication. Because security for these files may be compromised, use a non-default user profile for the server identity and password. The default user profile is QEJBSVR. If you use the Local Operating System (LocalOS) user registry (i5/OS), you may choose to create and use an iSeries user profile that has no special authorities. For more information, see "Run application servers under specific user profiles" on page 179.

WebSphere user profiles

When it is first installed, by default WebSphere Application Server - Express uses the following iSeries user profiles:

- **QEJB**
  This profile provides access to some administrative data, including passwords.
- **QEJBSVR**
  This profile provides the context in which your WebSphere application server runs. For security or administrative purposes, you may want to create other user profiles under which to run various parts of WebSphere Application Server - Express. For more information, see "Run application servers under specific user profiles" on page 179.

Password encoding

- What password encoding is (page 181)
- What password encoding is not (page 182)
- Issues to consider when using the OS400 password encoding algorithm (page 182)
- Enabling the OS400 password encoding algorithm for a WebSphere instance (page 182)
- Manually encoding passwords in properties files (page 184)
- Protecting plain text passwords (page 184)
- Administration of validation list objects (page 185)
- Switching algorithms (page 185)
- Problem resolution (page 186)

What password encoding is

The purpose of password encoding is to deter casual observation of passwords in server configuration and property files. By default, passwords are automatically encoded with a simple masking algorithm in various ASCII WebSphere server configuration files. Additionally, passwords can be manually encoded in properties files used by Java clients and by WebSphere administrative commands.

The default encoding algorithm is referred to as XOR. An alternate OS400 encoding algorithm can be used only with WebSphere Application Server - Express for iSeries and exploits native validation list (*VLDL) objects. With the OS400 algorithm, passwords are stored in encrypted form within a validation list, and the configuration files then contain indexes to the the stored passwords instead of the masked passwords themselves as is done with the XOR algorithm.

Properties of the WebSphere Application Server - Express instance control which algorithm to use for encoding the passwords.

Encoded passwords are of this form:

```
{algorithm}encoded_password
```

where {*algorithm*} is a tag that specifies the algorithm used to encode the password (either XOR or OS400), and *encoded_password* is the encoded value of the password. When a server or client needs to decode a password, it uses the tag to determine what algorithm to use and then uses that algorithm to decode the encoded password.

WebSphere administrative commands use passwords from the soap.client.props file (also located in the properties subdirectory) for SOAP connections, and some administrative commands optionally use passwords from the sas.client.props file (in the properties subdirectory) for RMI connections. To use password encoding with WebSphere administrative commands, the passwords must be manually encoded in the soap.client.props and sas.client.props files using the PropFilePasswordEncoder tool.

What password encoding is not

Whether you select to use the OS400 encoding algorithm or the default encoding algorithm, encoding is not sufficient to fully protect passwords. Native security is the primary mechanism for protecting passwords used in WebSphere configuration and property files.

Issues to consider when using the OS400 password encoding algorithm

There are important issues to consider before deciding to use the OS400 password encoding algorithm:

- Use of the OS400 password algorithm requires that the i5/OS system value QRETSVRSEC be set to 1, on the system hosting the WebSphere server, to allow retrieval of the encrypted passwords from the validation list. Note that the QRETSVRSEC system value effects access to the encrypted data in all of the validation lists on your iSeries system.

  Note: Do not use the OS400 password encoding algorithm if this setting is not consistent with your iSeries system security policy.
- Only use the OS400 algorithm with server instances when all server instances within the WebSphere administrative domain reside on the same iSeries system:
  - WebSphere administrative domains can extend across multiple iSeries systems. The OS400 password algorithm may only be used when all of the servers within an administrative domain reside on the same iSeries system.
  - Server configuration (XML) files contain encoded passwords. If the passwords contained in XML files are encoded using the OS400 encoding algorithm, those encodings can only be valid for WebSphere server instances on the same iSeries system on which the passwords were originally encoded. Thus, copies of configuration files containing passwords encoded using the OS400 encoding algorithm cannot be used to configure servers on other iSeries systems.
  - Additionally, all WebSphere server instances within an administrative domain must be configured to use the same native validation list (*VLDL) object.
- If an error occurs while encoding a password using the OS400 encoding algorithm, the XOR encoding algorithm will be used to encode the password. Such an error would occur for instance if an administrator manually creates the validation list object and grants insufficient authority to the validation list object for the i5/OS user profile QEJB.

Enabling the OS400 password encoding algorithm for a WebSphere instance

To enable the OS400 password encoding algorithm for a WebSphere instance, perform these steps:

1. Set the os400.security.password properties to turn on the OS400 password encoding algorithm and to specify the validation list object to use.

   It is recommended that you use the same validation list object for all WebSphere instances on the iSeries system. An exception would be if you do not backup the objects and data for all instances simultaneously. Consider your backup and restore policy when deciding what validation list object to use for each WebSphere instance.

   To set the properties, perform one of these steps:

- Use the -os400passwords and -validationlist options of the crtwasinst utility (located in the bin subdirectory of the product installation, for example /QIBM/ProdData/WebASE/ASE5/bin) to set the properties when creating the instance. For example, to create a WebSphere instance named `prod` and enable that instance for the OS400 encoding algorithm using the validation list object /QSYS.LIB/QUSRSYS.LIB/WAS.VLDL, you would do this:
  a. Run the Start Qshell (STRQSH) command on the CL command line.
  b. In Qshell, run this command:
     ```
     /QIBM/ProdData/WebASE/ASE5/bin/crtwasinst -instance prod -portblock 10150
        -os400passwords -validationlist /QSYS.LIB/QUSRSYS.LIB/WAS.VLDL
     ```
- Set Java system properties in the setupCmdLine Qshell script of the WebSphere instance (which is located in the bin directory of the user instance). For example, to enable the OS400 password encoding algorithm for an instance, edit /QIBM/UserData/WebASE/ASE5/*instance*/bin/setupCmdLine where *instance* is the name of your instance:
  a. Set the property **os400.security.password.encoding.algorithm** to `OS400`. The default setting is `XOR`.
  b. Set the property **os400.security.password.validation.list.object** to the absolute name of the validation list you wish to use. The default setting is `/QSYS.LIB/QUSRSYS.LIB/EJSADMIN.VLDL`.
  c. Save the file.

2. Grant the QEJB user profile execute authority (*X) to the library that contains the validation list. If QEJB already has the minimum required authority (*X) to the library then proceed to the next step.

   For example, if the validation list is created in /QSYS.LIB/WSADMIN.LIB, use the Display Authority (DSPAUT) to check for the minimum required authority:
   ```
   DSPAUT OBJ('/QSYS.LIB/WSADMIN.LIB')
   ```
   Then use the Change Authority (CHGAUT) command to grant execute authority to QEJB (only if QEJB does not already have execute authority). For example:
   ```
   CHGAUT OBJ('/QSYS.LIB/WSADMIN.LIB') USER(QEJB) DTAAUT(*X)
   ```
3. Create a native validation list object (*VLDL). This step is optional for server instances. The validation list object will be created when the server is started. For remote instances, create the validation list if the validation list does not already exist on the system hosting the remote instance. Also, consider your backup and restore policy when deciding what validation list object to use with each remote instance.

   Note: When using the OS400 password encoding algorithm, the Java client is not required to reside on the same iSeries system as the WebSphere server instance that the client accesses.

   To create a validation list object, perform these steps with an i5/OS user profile with *ALLOBJ special authority:
   a. Signon the iSeries server with a user profile that has *ALLOBJ special authority.
   b. Use the Create Validation List (CRTVLDL) command to create the validation list object. For example, to create validation list object WSVLIST in library WSADMIN.LIB, use the following command:
      ```
      CRTVLDL VLDL(WSADMIN/WSVLIST)
      ```
   c. Grant the QEJB user profile *RWX authority to the validation list object. For example, to grant *RWX authority to the validation list object WSVLIST in library WSADMIN, use the following command:
      ```
      CHGAUT OBJ('/QSYS.LIB/WSADMIN.LIB/WSVLIST.VLDL') USER(QEJB) DTAAUT(*RWX)
      ```
4. Use the Change System Value (CHGSYSVAL) command to set the QRETSVRSEC system value to 1. For example:
   ```
   CHGSYSVAL SYSVAL(QRETSVRSEC) VALUE('1')
   ```
5. For server instances, start (or restart) the server and wait until the server is ready for service before attempting to manually encode passwords in properties files belonging to the instance.

Manually encoding passwords in properties files

Use the PropFilePasswordEncoder utility to encode the passwords in properties files. This Qshell script is available in WebSphere Application Server - Express. To run the script, your user profile must have *ALLOBJ authority.

For example, to encode the passwords for properties in the sas.client.props file in your instance, perform the following steps:

1. Signon the iSeries server with a user profile that has all object (*ALLOBJ) special authority.
2. Run the Start Qshell (STRQSH) command on an CL command line to start the Qshell environment.
3. In Qshell, enter this command as a single line:
   ```
   /QIBM/ProdData/WebASE/ASE5/bin/PropFilePasswordEncoder -instance instance
    /QIBM/UserData/WebASE/ASE5/instance/properties/sas.client.props -SAS
   ```
   where *instance* is the name of your server instance.

To encode passwords for properties in the soap.client.props file in an instance named `myInst`:

1. Run the Start Qshell (STRQSH) command on an CL command line to start the Qshell environment.
2. In Qshell, enter this command as a single line:
   ```
   /QIBM/ProdData/WebASE/ASE5/bin/PropFilePasswordEncoder -instance myInst
    /QIBM/UserData/WebASE/ASE5/myInst/properties/soap.client.props
    com.ibm.SOAP.loginPassword,com.ibm.ssl.keyStorePassword,com.ibm.ssl.trustStorePassword
   ```

For more information, see The PropFilePasswordEncoder script in the *Administration* topic.

If you use a text file to store user IDs and passwords for applications that run outside the WebSphere Application Server - Express process (such as another server), use the EncAuthDataFile script to encode the passwords. For more information, see The EncAuthDataFile script in the *Administration* topic.

Protecting plain text passwords

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

File name	**Additional information**
security.xml	The following fields contain encoded passwords: • **LTPA password** • **JAAS Auth Data** • **User Registry server password** • **LDAP User Registry bind password** • **Key file password** • **Trust file password** • **Crypto token device password**
war/WEB-INF/ibm_web_bnd.xml	Specify passwords for the default basic authentication for the `resource-ref` bindings within all descriptors (except in the Java crytography architecture).
server.xml	The following fields contain encoded passwords: • **key file password** • **trust file password** • **crypto token device password** • **auth target password**

File name	Additional information
resource.xml (for cells, servers, and nodes)	The following fields contain encoded passwords: • **mailTransport password** • **mailStore password**
ws-security.xml	
ibm-webservices-bnd.xmi	
ibm-webservicesclient-bnd.xmi	
/properties/soap.client.props	
/properties/sas.tools.properties	
/properties/sas.stdclient.properties	
wsserver.key	

Administration of validation list objects

Validation lists may be shared between multiple WebSphere instances. For example, if you have two instances of WebSphere Application Server - Express, default and prod, both instances may use the validation list /QSYS.LIB/QUSRSYS.LIB/EJSADMIN.VLDL.

You should periodically save your validation list objects along with the other configuration data objects used by WebSphere Application Server - Express. See Save and restore: Security in the *Administration* topic for additional information.

Restore or replace damaged validation list objects. To replace a validation list object:

1. For all WebSphere instances that use the validation list object:
   a. Stop the servers.
   b. Set the property os400.security.password.validation.list.object for all servers to the absolute name of the new validation list you wish to use. You may use an existing validation list object or specify a new one. For new validation list objects, either create them manually, as specified above or they will be created when the server is restarted.
   c. Edit the configuration files and set each encoded password to the appropriate clear text value
2. Edit the sas.client.props and soap.client.props files. Set each encoded password to the appropriate clear text value, then manually encode the passwords.
3. Restart the servers for all WebSphere instances whose validation list objects were replaced.

Switching algorithms

To switch the encoding algorithm for a WebSphere instance:

1. For all WebSphere instances that use the validation list object, set the property **os400.security.password.encoding.algorithm** to `OS400` or `XOR`. If using OS400 then complete enablement as described above in Enabling the OS400 password encoding algorithm for a WebSphere instance (page 182). If you stop with this step, then all passwords remain encoded with the old algorithm, but after restarting the server, future changes to passwords made using the administrative console results in those passwords being encoded with the new encoding algorithm.
2. Stop all servers.
3. Edit the configuration files and change all encoded passwords to their clear text values.
4. Edit the properties files and change all encoded passwords to their clear text values.
5. Restart the servers.
6. Encode the passwords in the properties files using the instructions in Manually encoding passwords in properties files (page 184).

Problem resolution

If a password cannot be decoded, for instance if a password encoding has become corrupted, do the following:

- If the password is contained in a server configuration file, then edit the file and set the password to the clear text value. Then restart the server.
- If the password is contained in the sas.client.props file or soap.client.props file, then edit the file and set the password to the appropriate clear text value. Finally, encode the password using the PropFilePasswordEncoder utility.

The instance-specific setupCmdLine QShell (located in the `bin` directory of your instance) script contains a property that allows you to obtain trace information when using the OS400 algorithm with Java clients and WebSphere administrative commands. To obtain the trace, set `os400.security.password.debug=true`. The trace is printed to standard output.

Appendix. Notices

Programming Interface Information

This WebSphere Application Server - Express publication documents intended Programming Interfaces that allow the customer to write programs to obtain the services of IBM i5/OS.

Trademarks

Terms and conditions

Code license and disclaimer information

Printed in USA