Configuring Java CAPS for SSL Support

Part No: 820–3503–11
June 2010

ORACLE

100624@24378

Contents

1 Configuring Java CAPS for SSL Support 5

Secure Sockets Layer (SSL) Overview 5

Public Keys, Private Keys, and Certificates 6

Keytool Program 8

OpenSSL Project 8

SSL and Adapters 9

KeyStores and TrustStores 10

Generating a KeyStore and TrustStore 11

Configuring a Sun JMS IQ Manager to Use SSL 14

Configuring the Message Server URL 15

External JMS Clients 16

Changing the Self-Signed Server Certificate 16

Configuring the Repository to Use SSL 18

Generating a Key Pair and a Self-Signed Certificate 18

Obtaining a Signed Certificate 19

Importing the Certificate 20

Configuring the server.xml File 20

Testing the New SSL Connection 21

Configuring Enterprise Manager to Use SSL 22

Creating the Keystore and Trust Store 22

Importing the Domain Certificate 23

Enabling Security on the Application Server 25

Logging In to Enterprise Manager 25

Using SSL With the WebSphere MQ Adapter 26

Creating a Certification Authority 26

Using the OpenSSL Utility for the LDAP and HTTPS Adapters 29

Creating a Sample CA Certificate 29

Signing Certificates With Your Own CA 31

Windows OpenSSL.cnf File Example 33

Index 37

◆ ◆ ◆ CHAPTER 1

Configuring Java CAPS for SSL Support

The topics listed here provide information about how to configure the Sun Java™ Composite Application Platform Suite (Java CAPS) for Secure Sockets Layer (SSL) support.

If you have any questions or problems, see the Java CAPS web site at `http://goldstar.stc.com/support`.

- "Secure Sockets Layer (SSL) Overview" on page 5
- "Configuring a Sun JMS IQ Manager to Use SSL" on page 14
- "Configuring the Repository to Use SSL" on page 18
- "Configuring Enterprise Manager to Use SSL" on page 22
- "Using SSL With the WebSphere MQ Adapter" on page 26
- "Using the OpenSSL Utility for the LDAP and HTTPS Adapters" on page 29

Secure Sockets Layer (SSL) Overview

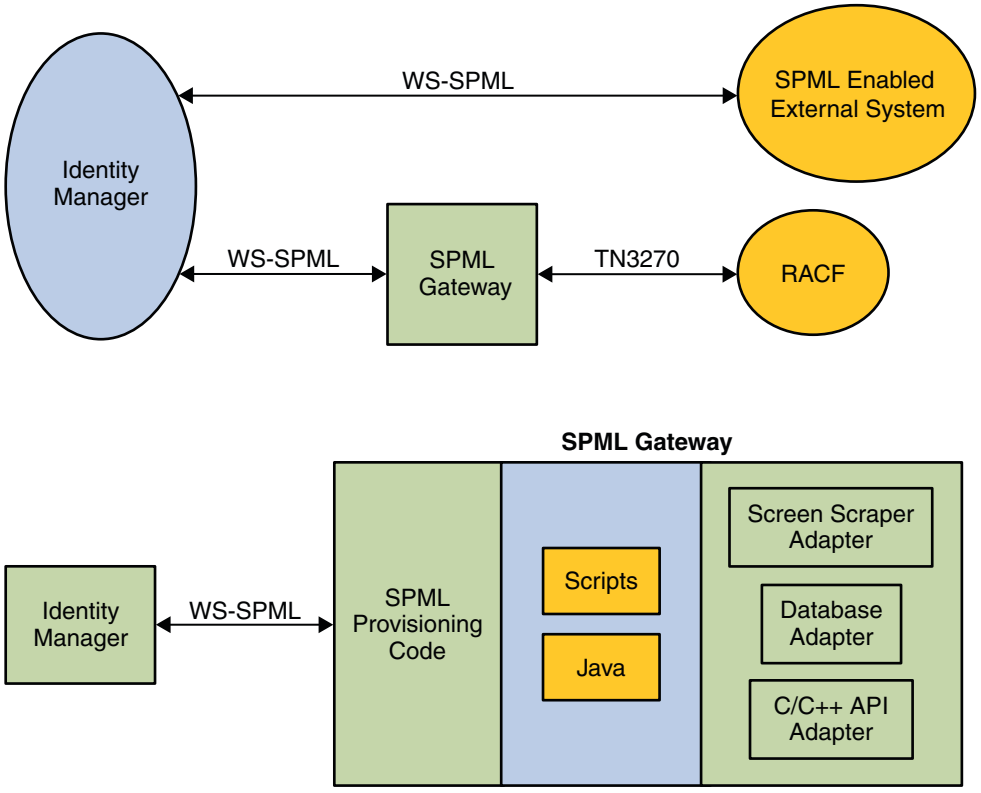
You can use the Secure Sockets Layer (SSL) protocol to protect communication between clients and servers over the Internet.

SSL provides such features as server authentication, client authentication, and data encryption. *Authentication* confirms the identity of a server or client. *Encryption* converts data into an unreadable form before the data is sent.

The scheme of a URL that uses SSL is `https`. For example:

```
https://www.onlinebooks.com/creditcardinfo.html
```

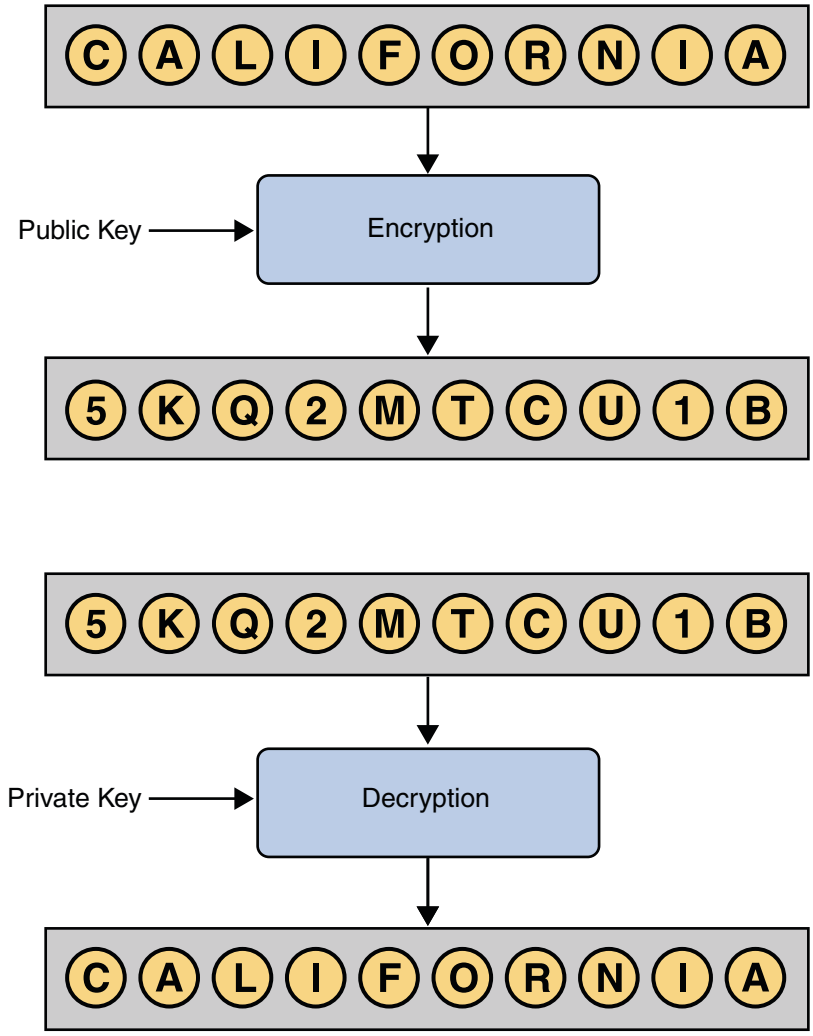
The latest version of SSL is called Transport Layer Security (TLS). The Internet Engineering Task Force (IETF) maintains the TLS standard.



Public Keys, Private Keys, and Certificates

When performing authentication, SSL uses a technique called *public-key cryptography*.

Public-key cryptography is based on the concept of a key pair, which consists of a *public key* and a *private key*. Data that has been encrypted with a public key can be decrypted only with the corresponding private key. Conversely, data that has been encrypted with a private key can be decrypted only with the corresponding public key.



The owner of the key pair makes the public key available to anyone, but keeps the private key secret.

A *certificate* verifies that an entity is the owner of a particular public key.

Certificates that follow the X.509 standard contain a data section and a signature section. The data section includes such information as:

- The Distinguished Name of the entity that owns the public key
- The Distinguished Name of the entity that issued the certificate
- The period of time during which the certificate is valid
- The public key itself

You can obtain a certificate from a Certificate Authority (CA) such as VeriSign. Alternately, you can create a *self-signed certificate*, in which the owner and the issuer are the same.

An organization that issues certificates can establish a hierarchy of CAs. The root CA has a self-signed certificate. Each subordinate CA has a certificate that is signed by the next highest CA in the hierarchy. A *certificate chain* is the certificate of a particular CA, plus the certificates of any higher CAs up through the root CA.

Keytool Program

The `keytool` program is a security tool included in the `bin` directory of the Java SDK.

This tool manages a type of database called a *keystore*. Keystores contain two types of entries:

- A *key entry* consists of a private key and the certificate chain for the associated public key.
- A *trusted certificate entry* is a public key certificate that belongs to another entity and that the owner of the keystore has determined to be trustworthy.

Each entry in the keystore is identified by a unique *alias*. When you add an entity to the keystore, you must specify an alias.

The available commands of the `keytool` program include the following:

- The `genkey` command generates a key pair. If you specify a keystore that does not exist, then the keystore is created.
- The `certreq` command generates a Certificate Signing Request (CSR).
- The `import` command adds a certificate to a keystore. If you specify a keystore that does not exist, then the keystore is created.
- The `export` command exports a certificate to a file.
- The `list` command prints the contents of a keystore entry.

For more information about the `keytool` program, go to http://java.sun.com/j2se/1.5.0/docs/tooldocs/index.html.

OpenSSL Project

The OpenSSL Project is an effort to develop an open-source toolkit that implements the SSL and TLS protocols, as well as a cryptographic library.

The toolkit includes the `openssl` command-line tool, which enables you to use various functions of the cryptographic library.

The available commands of the `openssl` tool include the following:

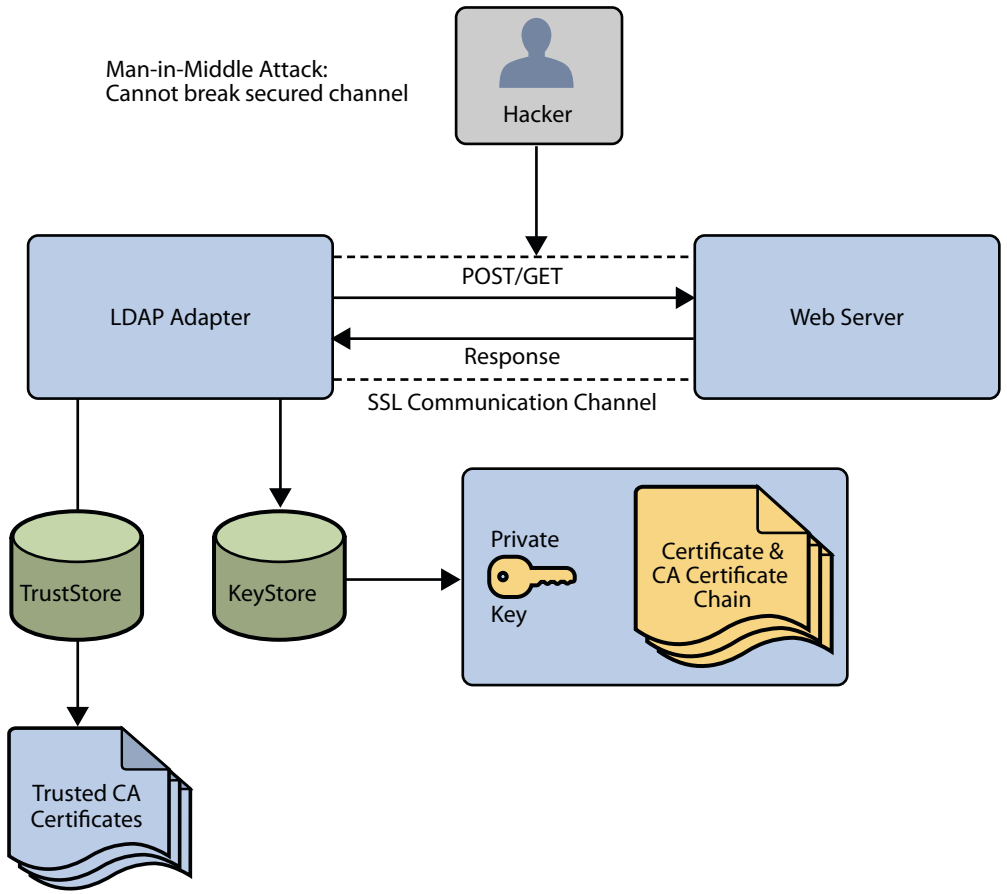
- The `pkcs12` command parses or generates a PKCS #12 file.
- The `req` command creates and processes certificate requests in PKCS #10 format.

You can download the current version of OpenSSL at http://www.openssl.org.

SSL and Adapters

The use of SSL with HTTP, LDAP, and WebSphere MQ enables data exchanges that are secure from unauthorized interception from hackers or other entities. The adapter's SSL feature provides a secure communications channel for the data exchanges.

The following diagram illustrates the use of SSL with the LDAP Adapter.



This SSL feature is supported through the use of Java Secure Socket Extension (JSSE) version 1.0.3.

Currently, the JSSE reference implementation is used. JSSE is a provider-based architecture, which means there is a set of standard interfaces for cryptographic algorithms, hashing algorithms, secured-socket-layered URL stream handlers, and so on.

Because the user is interacting with JSSE through these interfaces, the different components can be mixed and matched as long as the implementation is programmed under the published interfaces. However, some implementations might not support a particular algorithm.

The JSSE 1.0.3 API can support SSL versions 2.0 and 3.0 and TLS version 1.0. These security protocols encapsulate a normal bidirectional stream socket. The JSSE 1.0.3 API adds transparent support for authentication, encryption, and integrity protection. The JSSE reference implementation implements SSL version 3.0 and TLS version 1.0.

The following options available for setting up SSL connectivity with a web server:

- Server-side Authentication. The majority of e-commerce web sites are configured for server-side authentication. The adapter requests a certificate from the web server and authenticates the web server by verifying that the certificate can be trusted. Essentially, the adapter performs this operation by looking into its TrustStore for a CA certificate with a public key that can validate the signature on the certificate received from the web server.
- Dual Authentication. This option requires authentication from both the adapter and web server. The server side (web server) of the authentication process is the same as that described previously. In addition, the web server requests a certificate from the adapter. The adapter sends its certificate to the web server. The server authenticates the adapter by looking into its TrustStore for a matching trusted CA certificate. The communication channel is established by the process of both parties requesting certificate information.

KeyStores and TrustStores

The JSSE makes use of files called *KeyStores* and *TrustStores*. The KeyStore is used by the adapter for client authentication, while the TrustStore is used to authenticate a server in SSL authentication.

- A *KeyStore* consists of a database containing a private key and an associated certificate, or an associated certificate chain. The certificate chain consists of the client certificate and one or more certification authority (CA) certificates.
- A *TrustStore* contains only the certificates trusted by the client (a "trust" store). These certificates are CA root certificates, that is, self-signed certificates. The installation of the Logical Host includes a TrustStore file named **cacerts.jks** in the location:

  ```
  <c:\JavaCAPS>\appserver\domains\<MyDomain>\config
  ```

  where `<c:\JavaCAPS>` is the directory where Java CAPS is installed and `<MyDomain>` is the name of your domain. This file is recommended as the TrustStore for the Sun Adapters.

Both KeyStores and TrustStores are managed by means of a utility called **keytool**, which is a part of the Java SDK installation.

Generating a KeyStore and TrustStore

The following sections explain how to create both a KeyStore and a TrustStore (or import a certificate into an existing TrustStore such as the default Logical Host TrustStore in the location:

```
<c:\JavaCAPS>\appserver\domains\<MyDomain>\config\cacerts.jks
```

where `<c:\JavaCAPS>` is the directory where Java CAPS is installed and `<MyDomain>` is the name of your domain. The primary tool used is **keytool**, but **openssl** is also used as a reference for generating **pkcs12** KeyStores.

For more information on **openssl** and available downloads, visit the following web site:

`http://www.openssl.org`.

Creating a KeyStore in JKS Format

This section explains how to create a KeyStore using the JKS format as the database format for both the private key, and the associated certificate or certificate chain. By default, as specified in the java.security file, **keytool** uses JKS as the format of the key and certificate databases (KeyStore and TrustStores). A CA must sign the certificate signing request (CSR). The CA is therefore trusted by the server-side application to which the Adapter is connected.

Note – It is recommended to use the default KeyStore

```
<c:\JavaCAPS>\appserver\domains\<MyDomain>\config\keystore.jks
```

where `<c:\JavaCAPS>` is the directory where Java CAPS is installed and `<MyDomain>` is the name of your domain.

▼ To Generate a KeyStore

1. **Perform the following command.**

   ```
   keytool -keystore clientkeystore -genkey -alias client
   ```

2. **Once prompted, enter the information required to generate a CSR. A sample key generation section follows.**

   ```
   Enter keystore password: javacaps
   What is your first and last name?
   [Unknown]: development.sun.com
   ```

```
What is the name of your organizational unit?
[Unknown]: Development
what is the name of your organization?
[Unknown]: Sun
What is the name of your City or Locality?
[Unknown]: Monrovia
What is the name of your State or Province?
[Unknown]: California
What is the two-letter country code for this unit?
[Unknown]: US
Is<CN=development.sun.com, OU=Development, O=Sun, L=Monrovia, ST=California,
C=US> correct?
[no]: yes

Enter key password for <client>
    (RETURN if same as keystore password):
```

If the KeyStore password is specified, then the password must be provided for the adapter.

3 **Press RETURN when prompted for the key password (this action makes the key password the same as the KeyStore password).**

This operation creates a KeyStore file **clientkeystore** in the current working directory. You must specify a fully qualified domain for the "first and last name" question. The reason for this use is that some CAs such as VeriSign expect this properties to be a fully qualified domain name.

There are CAs that do not require the fully qualified domain, but it is recommended to use the fully qualified domain name for the sake of portability. All the other information given must be valid. If the information cannot be validated, a CA such as VeriSign does not sign a generated CSR for this entry.

This KeyStore contains an entry with an alias of **client**. This entry consists of the generated private key and information needed for generating a CSR as follows:

```
keytool -keystore clientkeystore -certreq -alias client -keyalg rsa -file client.csr
```

This command generates a certificate signing request which can be provided to a CA for a certificate request. The file **client.csr** contains the CSR in PEM format.

Some CA (one trusted by the web server to which the adapter is connecting) must sign the CSR. The CA generates a certificate for the corresponding CSR and signs the certificate with its private key. For more information, visit the following web sites:

`http://www.thawte.com`

or

`http://www.verisign.com`

If the certificate is chained with the CA's certificate, perform step 4; otherwise, perform step 5 in the following list:

4 **Perform the following command.**

```
keytool -import -keystore clientkeystore -file client.cer -alias client
```

The command imports the certificate and assumes the client certificate is in the file **client.cer** and the CA's certificate is in the file **CARoot.cer**.

5 **Perform the following command to import the CA's certificate into the KeyStore for chaining with the client's certificate.**

```
keytool -import -keystore clientkeystore -file CARoot.cer -alias theCARoot
```

6 **Perform the following command to import the client's certificate signed by the CA whose certificate was imported in the preceding step.**

```
keytool -import -keystore clientkeystore -file client.cer -alias client
```

The generated file **clientkeystore** contains the client's private key and the associated certificate chain used for client authentication and signing. The KeyStore and/or **clientkeystore**, can then be used as the adapter's KeyStore.

Creating a KeyStore in PKCS12 Format

This section explains how to create a PKCS12 KeyStore to work with JSSE. In a real working environment, a customer could already have an existing private key and certificate (signed by a known CA). In this case, JKS format cannot be used, because it does not allow the user to import/export the private key through **keytool**. It is necessary to generate a PKCS12 database consisting of the private key and its certificate.

The generated PKCS12 database can then be used as the Adapter's KeyStore. The **keytool** utility is currently lacking the ability to write to a PKCS12 database. However, it can read from a PKCS12 database.

Note – There are additional third-party tools available for generating PKCS12 certificates, if you want to use a different tool.

For the following example, **openssl** is used to generate the PKCS12 KeyStore:

```
cat mykey.pem.txt mycertificate.pem.txt>mykeycertificate.pem.txt
```

The existing key is in the file **mykey.pem.txt** in PEM format. The certificate is in **mycertificate.pem.txt**, which is also in PEM format. A text file must be created which contains the key followed by the certificate as follows:

```
openssl pkcs12 -export -in mykeycertificate.pem.txt -out mykeystore.pkcs12
-name myAlias -noiter -nomaciter
```

This command prompts the user for a password. The password is required. The KeyStore fails to work with JSSE without a password. This password must also be supplied as the password for the Adapter's KeyStore password.

This command also uses the **openssl pkcs12** command to generate a PKCS12 KeyStore with the private key and certificate. The generated KeyStore is **mykeystore.pkcs12** with an entry specified by the **myAlias** alias. This entry contains the private key and the certificate provided by the **-in** argument. The **noiter** and **nomaciter** options must be specified to allow the generated KeyStore to be recognized properly by JSSE.

Creating a TrustStore

For demonstration purposes, suppose you have the following CAs that you trust: **firstCA.cert, secondCA.cert, thirdCA.cert**, located in the directory **C:\cascerts**. You can create a new TrustStore consisting of these three trusted certificates.

▼ To Create a New TrustStore

1 Perform the following command.

```
keytool -import -file C:\cascerts\firstCA.cert -alias firstCA -keystore myTrustStore
```

2 Enter this command two more times, but for the second and third entries, substitute secondCA and thirdCA for firstCA. Each of these command entries has the following purposes:

- The first entry creates a KeyStore file named **myTrustStore** in the current working directory and imports the **firstCA** certificate into the TrustStore with an alias of **firstCA**. The format of **myTrustStore** is JKS.
- For the second entry, substitute **secondCA** to import the **secondCA** certificate into the TrustStore, **myTrustStore**.
- For the third entry, substitute **thirdCA** to import the **thirdCA** certificate into the TrustStore.

Once completed, myTrustStore is available to be used as the TrustStore for the adapter.

Configuring a Sun JMS IQ Manager to Use SSL

Sun JMS IQ Manager provides a self-signed server certificate.

You can set the authentication mode to `Authenticate` or `TrustAll`.

- If the mode is `Authenticate`, then clients authenticate the server certificate that the message server sends. The clients need to use their trust store.
- If the mode is `TrustAll`, then clients always trust the message server that they connect to. The clients do not need to use their trust store.

The default mode is `TrustAll`.

You can replace the Sun JMS IQ Manager's self-signed server certificate with your own server certificate.

Configuring the Message Server URL

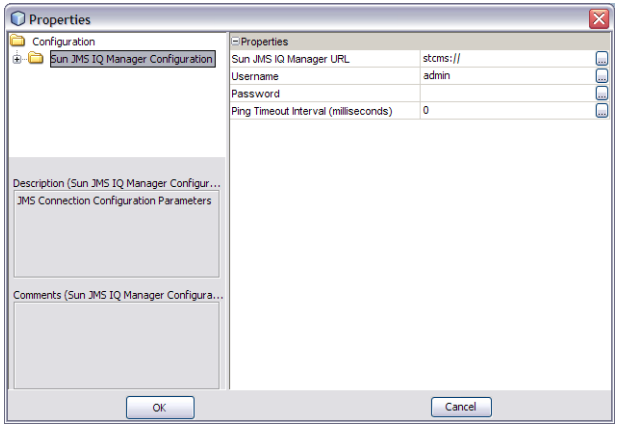
You can configure SSL for the Sun JMS IQ Manager by editing an Environment property.

▼ To Configure the Message Server URL

1. **In the Services window of the NetBeans IDE, expand the CAPS Environment in which the JMS IQ Manager is located.**

2. **Right-click the JMS IQ Manager and choose Properties.**

   The Properties dialog box appears.



3. **Ensure that the Sun JMS IQ Manager URL property begins with `stcmss` and includes the SSL port number of the JMS IQ Manager. For example:**

   ```
   stcmss://localhost:18008
   ```

4. **If you want clients to authenticate the server certificate that the message server sends, then append the string `com.stc.jms.ssl.authenticationmode=Authenticate` to the Sun JMS IQ Manager URL property. For example:**

   ```
   stcmss://localhost:18008?com.stc.jms.ssl.authenticationmode=Authenticate
   ```

5 **If you want clients to always trust the message server that they connect to, then append the string** `com.stc.jms.ssl.authenticationmode=TrustAll` **to the Sun JMS IQ Manager URL property. For example:**

```
stcmss://localhost:18008?com.stc.jms.ssl.authenticationmode=TrustAll
```

6 **Click OK.**

External JMS Clients

By default, JMS clients that are deployed inside the Sun Java™ System Application Server use the default keystore and trust store.

External JMS clients must set the following properties in the connection factory:

- `com.stc.jms.ssl.authenticationmode`
- `javax.net.ssl.trustStore`

Enterprise Service Bus API Kit for JMS IQ Manager (Java Edition) describes how to instantiate connection factories and set the properties.

Changing the Self-Signed Server Certificate

You can replace the Sun JMS IQ Manager's self-signed server certificate with your own server certificate.

This procedure makes the following assumptions:

- You have a server certificate in PEM format. The file name is `mycacert.pem`. The common name of the owner and issuer is `mycertuserid`. The password is `mycertpassword`.
- You have a private key in PEM format. The file name is `mycakey.pem`.

▼ To Change the Self-Signed Server Certificate

1 **Import your server certificate into the default trust store of the Sun Java System Application Server. The trust store is located in the** `JavaCAPS-install-dir/appserver/domains/domain-name/config` **directory.**

```
keytool -import -alias stcmscert -file mycacert.pem -keystore cacerts.jks
```

For the `-alias` option, you can use any value.

2 **Convert your server certificate and private key from PEM format to PKCS #12 format. You can use the** `pkcs12` **command of the** `openssl` **command-line tool to export a file that contains both the server certificate and the private key.**

```
openssl pkcs12 -export -in mycacert.pem -inkey mycakey.pem -out mycert.p12
-name "stcmscert"
```

3. **Make the following changes to the files:**

   a. **Change the name of the server certificate file from** `mycacert.pem` **to** `stcmscert.pem`**.**

   b. **Change the name of the private key file from** `mycakey.pem` **to** `stcmskey.pem`**.**

   c. **(UNIX only) Copy the** `stcmscert.pem` **file to a new file called** `stcmscert.cer`**.**

   d. **(Windows only) Change the name of the PKCS #12 file from** `mycert.p12` **to** `stcmscert.cer`**.**

4. **Copy the** `stcmscert.pem`**,** `stcmskey.pem`**, and** `stcmscert.cer` **files into the** `JavaCAPS-install-dir/appserver/addons/stcms/templates` **directory.**

5. **If you already created an instance, then you must also copy the** `stcmscert.pem`**,** `stcmskey.pem`**, and** `stcmscert.cer` **files into the** `JavaCAPS-install-dir/appserver/domains/domain-name/addons/stcms/instance-name/config` **directory.**

6. **Open the** `stcms.default.Properties` **file in the** `JavaCAPS-install-dir/appserver/addons/stcms/templates` **directory.**

7. **Add the** `STCMS.SSL.UserId` **and** `STCMS.SSL.Password` **properties.**

   ```
   STCMS.SSL.UserId=mycertuserid
   STCMS.SSL.Password=mycertpassword
   ```

8. **(Windows only) Set the value of the** `STCMS.SSL.CertificateFileStore.Option` **property.**

   - If you want the JMS IQ Manager to install the certificate automatically, then set the value to `On`.
   - If you want to install the certificate by using the `certmgr` tool or Internet Explorer, then set the value to `Off`.

9. **If you already created an instance, then copy the** `stcms.default.Properties` **file into the** `JavaCAPS-install-dir/appserver/domains/domain-name/addons/stcms/instance-name/config` **directory.**

10. **If the domain is running, then restart the domain.**

Configuring the Repository to Use SSL

The HTTPS service of the Repository will not run unless a server certificate has been installed. Use the following procedure to set up a server certificate that can be used by the Repository to enable SSL.

Note – If you configure the Repository to use SSL, then NetBeans IDE users cannot connect to the Repository.

To enable the Repository to use SSL, perform the tasks in the following sections:

- "Generating a Key Pair and a Self-Signed Certificate" on page 18
- "Obtaining a Signed Certificate" on page 19
- "Importing the Certificate" on page 20
- "Configuring the server.xml File" on page 20
- "Testing the New SSL Connection" on page 21

Note – The instructions in this topic use port number 8443 as the SSL port. The instructions in "Configuring Enterprise Manager to Use SSL" on page 22 also use port number 8443 as the SSL port. If you are configuring the Repository and Enterprise Manager on the same computer, then ensure that the port numbers are different.

Generating a Key Pair and a Self-Signed Certificate

The `genkey` command of the `keytool` program enables you to generate a key pair.

▼ To Generate a Key Pair and a Self-Signed Certificate

1. **Navigate to the `JAVA_HOME/bin` directory, where `JAVA_HOME` is the installation directory of the Java SDK.**

2. **Enter the following command:**

   ```
   keytool -genkey -keyalg RSA -alias CAPS -keystore keystore_filename
   ```

3. **When prompted, enter your keystore password.**

4. **When prompted, enter the Distinguished Name information.**

   a. **What is your first and last name?**

Caution – When prompted for your first and last name, make sure you enter the machine hostname.

b. What is the name of your organizational unit?

c. What is the name of your organization?

d. What is the name of your City or Locality?

e. What is the name of your State or Province?

f. What is the two-letter country code for this unit?

g. Is CN=first_and_last_name, OU=organizational_unit, O=organization_name, L=city_or_locality, ST=state_or_province, C=two_letter_country_code correct?

5 When prompted, enter a password for the keystore entry. If the password is same as the keystore password, press Return.

Note – If you want to use a keystore, it is recommended to use the `sbyn.keystore` file in the `JavaCAPS-install-dir/repository/repository/server` directory.

Obtaining a Signed Certificate

You must obtain either a digitally signed certificate from a certificate authority or a self-signed certificate from a local keystore.

▼ To Obtain a Digitally Signed Certificate from a Certificate Authority

1 Enter the following command to generate a Certificate Signing Request (CSR):

`keytool -certreq -alias CAPS -keyalg RSA -file` *csr_filename* `-keystore` *keystore_filename*

2 Send the CSR for signing.

3 Store the signed certificate in a file.

Note – If you want to use a keystore, it is recommended to use the `sbyn.keystore` file in the `JavaCAPS-install-dir/repository/repository/server` directory.

▼ To Obtain a Self-Signed Certificate from a Local Keystore

- **Enter the following command to generate a self-signed certificate:**

  `keytool -export -alias CAPS -keystore` *keystore_filename* `-rfc -file` *self_signed_cert_filename*

Importing the Certificate

If you are using a self-signed certificate or a certificate signed by a CA that your browser does not recognize, a dialog box will appear the first time you try to access the server. You can then choose to trust the certificate for this session only or permanently.

▼ To Import the Certificate

- **Enter the following command to install the certificate:**

```
keytool -import -trustcacerts -alias CAPS -file ca-certificate-filename
-keystore keystore_filename
```

Note – You must have the required permissions to modify the `JAVA_HOME/jre/lib/security/cacerts` file. You must import your certificate into the `cacerts` file also.

If you want to use a keystore, it is recommended to use the `sbyn.keystore` file in the `JavaCAPS-install-dir/repository/repository/server` directory.

Configuring the server.xml File

You now edit the `server.xml` file in the Repository to enable SSL support.

▼ To Configure the server.xml File

1. **If the Repository is running, then shut down the Repository.**
2. **Using a text editor, open the `server.xml` file in the `JavaCAPS-install-dir/repository/repository/server/conf` directory.**
3. **Within the `<Service>` element, comment out the first `<Connector>` element.**

4 **Comment in the second** `<Connector>` **element.**

```
<!--  Define an SSL Coyote HTTP/1.1 Connector on port 8443  -->
<Connector className="org.apache.coyote.tomcat4.CoyoteConnector"
    port="8443" minProcessors="5" maxProcessors="75"
    enableLookups="true"
    acceptCount="100" debug="0" scheme="https" secure="true"
    useURIValidationHack="false" disableUploadTimeout="true">
<Factory
    className="org.apache.coyote.tomcat4.CoyoteServerSocketFactory"
    clientAuth="false" protocol="TLS"
    keystoreFile="sbyn.keystore" keystorePass="changeit" />
</Connector>
```

5 **Save the** `server.xml` **file.**

6 **Start the Repository.**

Testing the New SSL Connection

This procedure verifies that SSL support has been correctly installed.

▼ To Test the New SSL Connection

1 **Load the default Repository server introduction page with the following URL:**

`https://localhost:8443/`

The `https` portion indicates that the browser should use the SSL protocol.

The port 8443 is where the SSL Connector was created in "Configuring the server.xml File" on page 20.

2 **The first time that you load this application, the New Site Certificate dialog box appears. Select Next to move through the series of New Site Certificate dialog boxes. Select Finish when you reach the last dialog box.**

Note – You should still have the option to use HTTP to connect to the NetBeans IDE. System administrators should not block the HTTP port.

Configuring Enterprise Manager to Use SSL

To enable Enterprise Manager to use SSL, perform the tasks in the following sections:

- "Creating the Keystore and Trust Store" on page 22
- "Importing the Domain Certificate" on page 23
- "Enabling Security on the Application Server" on page 25
- "Logging In to Enterprise Manager" on page 25

Note – The instructions in this topic use port number 8443 as the SSL port. The instructions in "Configuring the Repository to Use SSL" on page 18 also use port number 8443 as the SSL port. If you are configuring the Repository and Enterprise Manager on the same computer, then ensure that the port numbers are different.

Creating the Keystore and Trust Store

The first task involves creating a keystore and a trust store on the computer where the Enterprise Manager server is installed.

For basic information about keystores and trust stores, see "Secure Sockets Layer (SSL) Overview" on page 5.

The examples in this topic show passwords being entered on the command line. You can omit these passwords, and be prompted to enter them.

▼ To Create the Keystore and Trust Store

1. **Go to the computer where the Enterprise Manager server is installed.**
2. **If the Enterprise Manager server is running, then shut down the Enterprise Manager server.**
3. **Set the path variable to include the Java™ Runtime Environment (JRE™) software used by the Enterprise Manager server. For example:**

   ```
   set PATH="C:\Program Files\Java\jdk1.6.0_06\jre\bin";%PATH%
   ```

4. **Create a directory for the keystore and trust store. For example:**

   ```
   C:\JavaCAPS6\keystore
   ```

5. **Navigate to the directory that you created, and use the `keytool` program to create a certificate in a new keystore.**

   ```
   keytool -genkey -alias mykey -keyalg RSA -keypass changeit -keystore keystore.jks
   -storepass changeit
   ```

When you are prompted to enter your first and last name, do not enter your first and last name. Instead, enter the fully qualified computer name. For example:

```
What is your first and last name?
  [Unknown]:  example.company.com
What is the name of your organizational unit?
  [Unknown]:  Development
What is the name of your organization?
  [Unknown]:  Sun Microsystems
What is the name of your City or Locality?
  [Unknown]:  Monrovia
What is the name of your State or Province?
  [Unknown]:  California
What is the two-letter country code for this unit?
  [Unknown]:  US
Is CN=example.company.com, OU=Development, O=Sun Microsystems, L=Monrovia,
ST=California, C=US correct?
  [no]:  yes
```

6 **Export the certificate to a file.**

```
keytool -export -alias mykey -file mykey.cer -keystore keystore.jks -storepass changeit
```

The certificate is stored in the file that you specified.

7 **Import the certificate into a new trust store.**

```
keytool -import -v -trustcacerts -alias mykey -keypass changeit -file mykey.cer
-keystore cacerts.jks -storepass changeit
```

The trust store is created. The trust store includes the imported certificate.

Importing the Domain Certificate

After you perform the steps in "Creating the Keystore and Trust Store" on page 22, you must import the certificate of the application server domain into the trust store.

In addition, you must perform the following edits to files on the Enterprise Manager server:

- Enable the SSL connector in the `server.xml` configuration file. A *connector* represents an endpoint by which requests are received and responses are returned.
- Add an option to the `startserver` batch file.

The examples in this topic show passwords being entered on the command line. You can omit these passwords, and be prompted to enter them.

▼ To Import the Domain Certificate

1. **Go to the computer where the application server is installed.**

2. **Navigate to the** `JavaCAPS-install-dir/appserver/domains/domain-name/config` **directory.**

3. **Export the domain certificate to a file.**

   ```
   keytool -export -alias s1as -file ascert.cer -keystore keystore.jks -storepass changeit
   ```

   The certificate is stored in the file that you specified.

4. **Copy the file to the directory that you created in** **"Creating the Keystore and Trust Store" on page 22.**

5. **Import the domain certificate into the trust store that you created in** **"Creating the Keystore and Trust Store" on page 22.**

   ```
   keytool -import -v -trustcacerts -alias s1as -keypass changeit -file ascert.cer
   -keystore cacerts.jks -storepass changeit
   ```

   The certificate is added to the trust store.

6. **Using a text editor, open the** `server.xml` **file in the** `JavaCAPS-install-dir/emanager/server/conf` **directory.**

7. **Within the** `<Service>` **element, comment out the first** `<Connector>` **element.**

8. **Comment in the second** `<Connector>` **element. Add the** `keystoreFile` **and** `keystorePass` **attributes.**

   Set the value of the `keystoreFile` attribute to the fully qualified name of the keystore that you created in "Creating the Keystore and Trust Store" on page 22. Set the value of the `keystorePass` attribute to the corresponding password.

   ```
   <Connector port="8443"
      maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
      enableLookups="false" disableUploadTimeout="true"
      keystoreFile="C:\JavaCAPS6\keystore\keystore.jks"
      keystorePass="changeit"
      acceptCount="100" debug="0" scheme="https" secure="true"
      clientAuth="false" sslProtocol="TLS" />
   ```

9. **Save the** `server.xml` **file.**

10. **Using a text editor, open the** `startserver` **batch file in the** `JavaCAPS-install-dir/emanager` **directory.**

11 **Add the `javax.net.ssl.trustStore` option. Set the value to the fully qualified name of the trust store that you created in "Creating the Keystore and Trust Store" on page 22.**

```
set JAVA_OPTS=-Xmx512m -Djavax.net.ssl.trustStore=C:\JavaCAPS\keystore\cacerts.jks
```

12 **Save the `startserver` batch file.**

Enabling Security on the Application Server

In the final configuration task, you enable security for one of the default HTTP listeners on the Sun Java™ System Application Server. The listener is called `admin-listener`.

▼ To Enable Security on the Application Server

1 **Log in to the Admin Console.**

2 **In the left pane, expand the Configurations node, the HTTP Service node, and the HTTP Listeners node.**

3 **In the left pane, select the `admin-listener` node.**

4 **Click the SSL tab.**

5 **In the Certificate NickName field, type `s1as`.**

6 **Click Save.**

7 **Click the Edit HTTP Listener tab.**

8 **Select the Enable check box to the right of the Security label.**

9 **Click Save.**

10 **Stop and restart the application server.**

Logging In to Enterprise Manager

When Enterprise Manager has been configured to use SSL, the URL that you use to log in has a different scheme and port number.

For detailed information about Enterprise Manager user names and passwords, see *Managing Java CAPS Users*.

▼ To Log In to Enterprise Manager

1. **Start the Enterprise Manager server.**
2. **In a supported browser, enter the following URL:**

   ```
   https://hostname:portnumber
   ```

   The scheme must be `https`. The port number must be the value used in the `<Connector>` element in the `server.xml` file. For example:

   ```
   https://example.company.com:8443/
   ```

   The Enterprise Manager Security Gateway screen appears.
3. **In the User ID field, enter an Enterprise Manager user name.**
4. **In the Password field, enter the corresponding password.**
5. **Click Login.**

   Enterprise Manager appears.

Using SSL With the WebSphere MQ Adapter

Perform the following tasks:

- "To Create a Certification Authority" on page 26
- "To Issue a Certificate to a Queue Manager" on page 27
- "To Issue a Certificate to Java CAPS" on page 28

Creating a Certification Authority

The following steps describe how to create a Certification Authority (CA) using the command-line utilities supplied with WebSphere MQ.

▼ To Create a Certification Authority

1. **Create a key repository for the CA.**
2. **Create a directory and in that directory, create a key repository file by entering the text shown below:**

   ```
   C:\> mkdir \myCAdir
   C:\> cd \myCAdir
   C:\myCAdir> runmqckm -keydb -create -db myCA.kdb -type cms
   ```

When prompted to create a password, type the password you want to use for the CA's key repository.

3 Create a self-signed CA certificate, which will be used to identify your CA:

```
C:\myCAdir> runmqckm -cert -create -db myCA.kdb -type cms -label "myCAcertificate"
-dn "CN=myCAName,O=myOrganisation,OU=myDepartment,L=myLocation,C=IN" -expire 1000
-size 1024
```

4 Extract the CA certficate into a file called myCAcertfile.cer, which you will later transfer to the key repositories of the queue manager and client application:

```
C:\myCAdir> runmqckm -cert -extract -db myCA.kdb -type cms -label "myCAcertificate"
-target myCAcertfile.cer -format ascii
```

▼ To Issue a Certificate to a Queue Manager

Each queue manager in your infrastructure should have its own certificate, with an appropriate Distinguished Name (DN). The DN should be unique within the WebSphere MQ network.

1 Create the queue manager's key repository

```
C:\myCAdir> mkdir \REPOS
C:\myCAdir> cd \REPOS
```

2 Issue the following command to create a key database for the queue manager:

```
C:\REPOS> runmqckm -keydb -create -db myqmgr.kdb -type cms -stash
```

When prompted to create a password, type the password you want to use for the queue manager's key repository.

The `-stash` option is important, as it causes a stash file to be created. This file is called `myqmgr.sth`. It allows the queue manager to open the key repository without requesting a password from the user.

3 Generate a certificate request file for the queue manager, along with a private key:

```
C:\REPOS> runmqckm -certreq -create -db myqmgr.kdb -type cms
-dn "CN=QMNAME,O=SUN,OU=BI,L=BLR,C=IN" -label "ibmwebspheremqmyqmgr" -file myqmgr.req
```

The label (as specified with the `-label` parameter) must be of the form `ibmwebspheremqmyqmgr`, all in lower case. This is important, as otherwise the queue manager will fail to find the certificate.

4 Transfer the certificate request file, myqmgr.req , to the directory where the CA files are located. Then change to the following directory:

```
C:\REPOS> copy myqmgr.req \myCAdir
C:\REPOS> cd \myCAdir
```

5 **Sign the queue manager's certificate by running the following command:**

```
C:\myCAdir> runmqckm -cert -sign -db myCA.kdb -label "myCAcertificate" -expire 365
-format ascii -file myqmgr.req -target myqmgr.cer
```

When prompted for the password, supply the CA key repository's password. Refer to the first step in "To Create a Certification Authority" on page 26.

6 **Transfer the signed certificate (myqmgr.cer) and the public certificate of the CA (myCAcertfile.cer) back to C:\REPOS**

```
C:\myCAdir> copy myqmgr.cer \REPOS
C:\myCAdir> copy myCAcertfile.cer \REPOS
C:\myCAdir> cd \REPOS
```

7 **Add the public certificate of the CA to the key repository of the queue manager:**

```
C:\REPOS> runmqckm -cert -add -db myqmgr.kdb -type cms -file myCAcertfile.cer
-label "theCAcert"
```

When prompted for a password, supply the queue manager key repository's password.

8 **Receive the certificate (now signed by the CA) into the queue manager's key repository:**

```
C:\REPOS> runmqckm -cert -receive -db myqmgr.kdb -type cms -file myqmgr.cer
```

When prompted for a password, supply the queue manager key repository's password. Refer to step 1 (above).

▼ To Issue a Certificate to Java CAPS

1 **Create a certificate request to the application server domain default keystore.jks.**

```
<JavaCAPS>\appserver\domains\<domain_name>\config> runmqckm -certreq -create
-db keystore.jks -type jks -dn "CN=Client Identifier,O=SUN,OU=BI,L=BLR,C=IN"
-label "ibmwebspheremqmyuserid" -file myappj.req
```

When prompted to create a password, type the default password changeit for the application server. The certificate label chosen was `ibmwebspheremqmyuserid`.

2 **Transfer the certificate request file (myappj.req) to the directory where the CA files are located, then change to this directory:**

```
<JavaCAPS>\appserver\domains\<domain_name>\config> copy myappj.req C:\myCAdir
<JavaCAPS>\appserver\domains\<domain_name>\config> cd  C:\myCAdir
```

3 **Sign the application's certificate by running the following:**

```
C:\myCAdir> runmqckm -cert -sign -db myCA.kdb -label "myCAcertificate" -expire 365
-format ascii -file myappj.req -target myappj.cer
```

When prompted for a password, supply the CA key repository's password. Refer to the first step in "To Create a Certification Authority" on page 26.

4. **Transfer the signed certificate (myappj.cer) and the public certificate of the CA (myCAcertfile.cer) back to C:\MYAPPJ:**

```
C:\myCAdir> copy myappj.cer <JavaCAPS>\appserver\domains\<domain_name>\config\
C:\myCAdir> copy myCAcertfile.cer<JavaCAPS>\appserver\domains\<domain_name>\config
C:\myCAdir> cd <JavaCAPS>\appserver\domains\<domain_name>\config
```

5. **Add the CA certificate to the Java CAPS keystore.**

```
<JavaCAPS>\appserver\domains\<domain_name>\config> runmqckm -cert -add
-db keystore.jks -type jks -file myCAcertfile.cer -label "theCAcertificate"
```

When prompted for a password, supply the Java CAPS keystore password as `changeit`.

6. **Receive the certificate (now signed by the CA) into the Java CAPS keystore:**

```
<JavaCAPS>\appserver\domains\<domain_name>\config> runmqckm -cert -receive
-db keystore.jks -type jks -file myappj.cer
```

When prompted for a password, supply the Java CAPS keystore password as `changeit`.

7. **Add the CA certificate to truststore:**

```
<JavaCAPS>\appserver\domains\<domain_name>\config> runmqckm -cert -add
-db cacerts.jks -type jks -file myCAcertfile.cer -label "theCAcertificate"
```

Using the OpenSSL Utility for the LDAP and HTTPS Adapters

The **OpenSSL** utility is a free implementation of cryptographic, hashing, and public key algorithms such as 3DES, SHA1, and RSA respectively. This utility has many options including certificate signing, which **keytool** does not provide. You can download **OpenSSL** from the following Web site:

`http://www.openssl.org`

Follow the build and installation instruction for **OpenSSL**.

Creating a Sample CA Certificate

The sample given in this section demonstrates the use of the **OpenSSL** utility to create a CA. This generated CA is then used to sign a CSR (see "Signing Certificates With Your Own CA" on page 31), whether it is generated from **keytool** or **OpenSSL**.

▼ To Create a Sample CA Certificate

For testing purposes, a sample CA can be generated. To avoid spending additional funds to have a commercial CA sign test certificates, a sample is generated and used to sign the test certificate.

1 **Perform the following operations from the command line:**

```
openssl  req  -config c:\openssl\bin\openssl.cnf
 -new  -x509  -keyout  ca-key.pem.txt -out  ca-certificate.pem.txt  -days  365

Using properties from c:\openssl\bin\openssl.cnf
Loading 'screen' into random state: done
Generating a 1024 bit RSA private key
.................++++++
.....................++++++
writing new private key to 'ca-key.pem.txt'
Enter PEM pass phrase:
Verifying password: Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be
    incorporated into your certificate request.
What you are about to enter is what is called a Distinguished Name
    or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) []:US
State or Province Name (full name) []:California
Locality Name (eg, city) []:Monrovia
Organization Name (eg, company) []:Sun
Organizational Unit Name (eg, section) []:Development
Common Name (eg, your websites domain name) []
    :development.sun.com
Email Address []:development@sun.com
```

You are prompted for password information.

2 **Enter a password and remember this password for signing certificates with the CA's private key.**

This command creates a private key and the corresponding certificate for the CA. The certificate is valid for 365 days starting from the date and time it was created.

The properties file **C:\openssl\bin\openssl.cnf** is needed for the **req** command. The default **config.cnf** file is in the **OpenSSL** package under the **apps** sub-directory.

Note – To use this file in Windows, you must change the paths to use double back-slashes. See "Windows OpenSSL.cnf File Example" on page 33 for a complete **Config.cnf** file example, which is known to work in a Windows environment.

Signing Certificates With Your Own CA

The example in this section shows how to create a Certificate Signing Request with **keytool** and generate a signed certificate for the Certificate Signing Request with the CA created in the previous section. The steps shown in this section, for generating a **KeyStore** and a Certificate Signing Request, were already explained under "Creating a KeyStore in JKS Format" on page 11.

Note – No details are given here for the **keytool** commands. See "Creating a KeyStore in JKS Format" on page 11 for more information.

▼ To Create a CSR with keytool and Generate a Signed Certificate for the Certificate Signing Request

1 Perform the following operations from the command line.

```
keytool –keystore clientkeystore –genkey –alias client

Enter keystore password:  javacaps
What is your first and last name?
[Unknown]:  development.sun.com
What is the name of your organizational unit?
[Unknown]:  Development
What is the name of your organization?
[Unknown]:  Sun
What is the name of your City or Locality?
[Unknown]:  Monrovia
What is the name of your State or Province?
[Unknown]:  California
What is the two-letter country code for this unit?
[Unknown]:  US
Is <CN=development.sun.com, OU=Development, O=Sun, L=Monrovia, ST=California,
C=US> correct?
[no]:  yes

Enter key password for <client>
(RETURN if same as keystore password):
```

2 Generate the Certificate Signing Request.

```
keytool –keystore clientkeystore –certreq –alias client –keyalg rsa
–file client.csr
```

3 Generate a signed certificate for the associated Certificate Signing Request.

```
openssl  x509  -req  -CA
    ca-certificate.pem.txt -CAkey ca-key.pem.txt
    -in client.csr -out client.cer  -days 365  -CAcreateserial
```

4 Use the keytool to import the CA certificate into the client keystore.

```
keytool -import -keystore clientkeystore -file ca-certificate.pem.txt
-alias theCARoot

Enter keystore password:  javacaps
Owner: EmailAddress=development@sun.com, CN=development.sun.com, OU=Development,
 O=Sun, L=Monrovia, ST=California, C=US
Issuer: EmailAddress=development@sun.com, CN=development.sun.com,
OU=Development, O=Sun, L=Monrovia, ST=California, C=US
Serial number: 0
Valid from: Tue May 08 15:09:07 PDT 2007 until: Wed May 08
    15:09:07 PDT 2008
Certificate fingerprints:
MD5:  60:73:83:A0:7C:33:28:C3:D3:A4:35:A2:1E:34:87:F0
SHA1: C6:D0:C7:93:8E:A4:08:F8:38:BB:D4:11:03:C9:E6:CB:9C:D0:72:D0
Trust this certificate? [no]:  yes
Certificate was added to keystore
```

5 Use the keytool to import the signed certificate for the associated client alias in the keystore.

```
keytool –import –keystore clientkeystore –file client.cer –alias client

Enter keystore password: javacaps
Certificate reply was installed in keystore
```

Caution – The following error will be generated if there is no certificate chain in the client certificate.

```
keytool -import -keystore clientkeystore -file client.cer -alias client

Enter keystore password: javacaps
keytool error: java.lang.Exception: Failed to establish chain from reply
```

This error is because the CA's certificate was not imported into the **KeyStore** first. You must import the CA's certificate (step 4), then import the client.cer file itself to form a certificate chain (step 5).

Now that we have a private key and an associating certificate chain in the **KeyStore clientkeystore**, we can use it as a **KeyStore** for client (adapter) authentication. The only warning is that the CA certificate must be imported into the trusted certificate store of the web server to which you will be connecting. Moreover, the web server must be configured for client authentication (**httpd.conf** for Apache, for example).

Windows OpenSSL.cnf File Example

This section contains the contents of the `openssl.cnf` file that can be used on Windows. Be sure to make the appropriate changes to the directories.

```
#
# SSLeay example properties file.
# This is mostly being used for generation of certificate requests.
#

RANDFILE        = .rnd

####################################################################
[ ca ]
default_ca    = CA_default        # The default ca section

####################################################################
[ CA_default ]

dir        = G:\\openssl\\\bin\\demoCA    # Where everything is kept
certs        = $dir\\certs                # Where the issued certs are kept
crl_dir    = $dir\\crl                # Where the issued crl are kept
database    = $dir\\index.txt            # database index file.
new_certs_dir    = $dir\\newcerts            # default place for new certs.

certificate    = $dir\\cacert.pem                # The CA certificate
serial        = $dir\\serial                # The current serial number
crl        = $dir\\crl.pem                # The current CRL
private_key    = $dir\\private\\cakey.pem       # The private key
RANDFILE    = $dir\\private\\private.rnd     # private random number file

x509_extensions    = x509v3_extensions    # The extentions to add to the cert
default_days        = 365            # how long to certify for
default_crl_days    = 30            # how long before next CRL
default_md        = md5            # which md to use.
preserve        = no            # keep passed DN ordering

# A few difference way of specifying how similar the request should look
# For type CA, the listed attributes must be the same, and the optional
# and supplied fields are just that :-)
policy        = policy_match

# For the CA policy
[ policy_match ]
countryName            = match
stateOrProvinceName        = match
organizationName        = match
organizationalUnitName    = optional
```

```
commonName            = supplied
emailAddress            = optional

# For the 'anything' policy
# At this point in time, you must list all acceptable 'object'
# types.
[ policy_anything ]
countryName         = optional
stateOrProvinceName    = optional
localityName        = optional
organizationName    = optional
organizationalUnitName    = optional
commonName            = supplied
emailAddress            = optional

######################################################################
[ req ]
default_bits        = 1024
default_keyfile     = privkey.pem
distinguished_name    = req_distinguished_name
attributes        = req_attributes

[ req_distinguished_name ]
countryName            = Country Name (2 letter code)
countryName_min        = 2
countryName_max        = 2

stateOrProvinceName        = State or Province Name (full name)

localityName            = Locality Name (eg, city)

0.organizationName        = Organization Name (eg, company)

organizationalUnitName    = Organizational Unit Name (eg, section)

commonName            = Common Name (eg, your website's domain name)
commonName_max        = 64

emailAddress            = Email Address
emailAddress_max        = 40

[ req_attributes ]
challengePassword        = A challenge password
challengePassword_min    = 4
challengePassword_max    = 20

[ x509v3_extensions ]
```

Note – The following copyright notices apply: Copyright © 2004-2008 The OpenSSL Project. All rights reserved.Copyright © 2005-2008 World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved. **http://www.w3.org/Consortium/Legal/**

Index

A

adapters
- HTTPS, 29-35
- LDAP, 29-35
- WebSphere MQ, 26-29

admin-listener, 25
alias, defined, 8
authentication, defined, 5
authentication mode, 14

C

cacerts.jks file, 10, 16
certificate
- defined, 7
- importing, 20, 23-25
- obtaining, 19-20
- self signed, 16-17

Certificate Authority (CA), 8
certificate chain, defined, 8
Certificate Signing Request (CSR), 8

E

encryption, defined, 5
Enterprise Manager, 22-26

H

HTTPS Adapter, 29-35
https scheme, 5

I

Internet Engineering Task Force, 5

J

JMS clients, external, 16

K

keystore
- defined, 8, 10
- generating, 11, 13

keystore.jks file, 11
keytool program, 8

L

LDAP Adapter, 29-35

O

openssl.cnf file, example, 33-35
OpenSSL Project, 8-9
openssl tool, 8

P
passwords, keystore, 18
PEM format, 16
PKCS #12 format, 16
public-key cryptography, 6-8

R
Repository, 18-21

S
sbyn.keystore file, 19
self-signed certificate, defined, 8
server.xml file
  Enterprise Manager, 23
  Repository, 20-21
SSL
  Enterprise Manager, 22-26
  overview, 5-14
  Repository, 18-21
  Sun JMS IQ Manager, 14-17
startserver file, 23
stash file, 27
stcms.default.Properties file, 17
Sun JMS IQ Manager, 14-17

T
TLS, 5
trust store
  defined, 10
  generating, 14

W
WebSphere MQ Adapter, 26-29

X
X.509 standard, 7