PART I - The Keystore

Step 1 - Creating the Keystore

The keys Tomcat will use for SSL transactions are stored in a password-protected file called, creatively, the "keystore." The first step to enabling SSL on your server is to create and edit this file. You can create this file in one of two ways - by importing an existing key into the keystore, or by creating an entirely new key.

A program called keytool, which is included with your JDK, will do the actual work of creating your new keystore. To create a new keystore using this program, enter the following command at the command-line, substituting syntax appropriate for your OS:

$JAVA_HOME/bin/keytool -genkey -alias [youralias] -keyalg RSA -keystore [/preferred/keystore/path]

Use an [alias] and [path] of your choice.

Next, keytool will ask you to enter the password you want to use for the keystore. Again, choose whatever you like (but don't forget it).

After you choose the keystore password, you will enter the information required for the Certificate, such as your company and your name. Make sure this information is accurate, as you will have to submit this file to the Certificate Authority of your choice to obtain a certificate.

The last thing keytool will ask you to specify is the key password, which is the password specific to this specific certificates. Rather than enter anything at this prompt, just press ENTER.

This will cause keytool to set the key password to a value equivalent to the keystore password. Matching passwords are REQUIRED for Tomcat to access the certificate. If you choose two different passwords, any attempts to access the keystore will result in a crash (so don't do it).

Step 2 - Creating the Certificate Signing Request

Now that you've created your keystore, it's time to create a file called the Certificate Signing Request, or CSR, which will be used by the Certificate Authority of your choice to generate the Certificate SSL will present to other parties during the handshake.

You can use the keytool to create this file, as well. To do so, enter the following at the command line:

```
keytool -certreq -keyalg RSA -alias [youralias] -file
[yourcertificatname].csr -keystore [path/to/your/keystore]
```

If you follow the instructions correctly, keytool will create a file called yourcertificatename.csr, which you can submit to the CA you've chosen via the process they provide on their website. Using this file, they will generate a custom certificate for your server, which you can download according to the instructions they provide on their website.

Step 3 - Installing Your New Certificate

SSL verifies the authenticity of a site's certificate by using something called a "chain of trust," which basically means that during the handshake, SSL initiates an additional handshake with the Certificate Authority specified in your site's certificate, to verify that you haven't simply made up your own CA.

In order to "anchor" your certificate's chain of trust, you have to download an additional certificate, called a "Root Certificate," from your CA, and then import both this certificate and your site's new certificate into your keystore. Your CA should provide information about obtaining a Root Certificate on their website.

Once you've downloaded both your own Certificate and the Root certificate provided by your CA, import them into your keystore with the following commands, replacing the [placeholders]:

To import the Root Certificate -

```
keytool -import -alias root -keystore [path/to/your/keystore] -
trustcacerts -file [path/to/the/root_certificate]
```

To import your new Certificate -

```
keytool -import -alias [youralias] -keystore [path/to/your/keystore] -
file [path/to/your_keystore]
```

PART II - Configuring Tomcat to use SSL

Now that we have a functional keystore populated with valid certificates, it's time to configure Tomcat to use SSL. First, we'll configure Tomcat's SSL connectors, and then we'll specify which webapps we want to use SSL by default.

Step 1 - Configuring Tomcat's SSL Connectors

```
<!-- Define a SSL HTTP/1.1 Connector on port 8443

This connector uses the JSSE configuration, when using APR, the

connector should be using the OpenSSL style configuration

described in the APR documentation -->

<!--

<Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"

maxThreads="150" scheme="https" secure="true"

clientAuth="false" sslProtocol="TLS"/>

-->
```

You'll notice that the comment enclosing this connector talks about a choice between APR and JSSE configurations. This refers to the implementation of SSL you are intending to use. JSSE, which is Tomcat's default configuration, is supported by default, and included in all JDKs after version 1.4. So if you don't even know what APR is, you only need to uncomment this entry, and add some additional information to allow Tomcat to find your keystore:

```
<Connector port="8443" maxThreads="150" scheme="https"
secure="true" SSLEnabled="true"
keystoreFile="path/to/your/keystore"
keystorePass="YourKeystorePassword" clientAuth="false"
keyAlias="yourAlias" sslProtocol="TLS"/>
```

If, on the other hand, you know that using the Apache Portable Runtime (APR), also known as Tomcat's "native library," is by far the best practice to follow, especially when using Tomcat as a standalone web server (which you probably are), and have already installed it on your server, then you'll need to alter this entry as follows to allow Tomcat to use APR's OpenSSL implementation in place of JSSE, or trying to use SSL will generate an error:

```
<Connector port="8443" scheme="https" secure="true" SSLEnabled="true"
SSLCertificateFile="/path/to/your/certificate.crt"
SSLCertificateKeyFile="/path/to/your/keyfile"
SSLPassword="YourKeystorePassword"
SSLCertificateChainFile="path/to/your/root/certificate"
keyAlias="yourAlias" SSLProtocol="TLSv1"/>
```

Notice that if you are using APR, the "SSLCertificateFile" and "SSLCertificateKey"-type attributes are used in place of the keystoreFile attribute. For more information on the differences between using APR in place of JSSE, consult Apache's Tomcat APR Documentation.

Restart Tomcat. Once you're up and running again, test your configuration by connecting to a secure page, using a URL such as https:/[yourhost]:8443. If you followed the directions correctly, you should be able to view the page over a secure HTTPS connection!

Limiting SSL Usage (Optional)

Enabling SSL in Tomcat's server.xml file causes all files to be run both as secure and insecure pages, which can cause unnecessary server load. You can choose which applications offer SSL connections on a per-application basis by adding the following <security-constraint> element to the application's WEB-INF/web.xml file:

```
<security-constraint>

<web-resource-collection>

<web-resource-name>YourAppsName</web-resource-name>

<url-pattern>/*</url-pattern>

</web-resource-collection>

<user-data-constraint>

<transport-guarantee>CONFIDENTIAL</transport-guarantee>

</user-data-constraint>

</security-constraint>
```

This configuration allows you to set SSL options for all an application's pages in one place. For example, to disable SSL for all your application's pages, change "CONFIDENTIAL" to "NONE".

Additional Considerations

In the interest of simplicity, this guide does not cover all of the elements of SSL configuration (although they are covered extensively in Apache's Tomcat SSL Documentation). We will, however, provide you with a short list of other options and important areas of note to consider as you tweak your SSL configuration.

Specifying Implementation

If you have configured connectors for both APR and JSSE, Tomcat will use APR by default if you have installed the native libraries. You can force it to use JSSE by modifying a Connector's "protocol" attribute as follows:

```
<Connector protocol="org.apache.coyote.http11.HTTP11NioProtocol">
```

If you want to force APR, you can do so with a similar edit:

```
<Connector protocol="org.apache.coyote.http11.Http11AprProtocol">
```

Common Errors Caused By Aliases and Passwords

If you encounter any errors with your SSL configuration, make sure that you have correctly entered settings such as keystore passwords and aliases. These values are case sensitive for some of the supported keystore formats.