# Creating a Keystore From Your Client Certificate

1. Make sure you have a JDK (preferably Java 8) installed
2. Copy your client certificate (yourcertificate.pfx) to the bin directory of the JDK. Remove all white spaces from the certificate name before proceeding
3. From the Windows start menu, go to Start > All Programs > Accessories. Rick click Command Prompt, then click Run as administrator (requires admin rights)
4. CD to the JDK bin folder where you placed yourcertificate.pfx
5. Run the following command  
   keytool –v –list –storetype pkcs12 –keystore yourcertificate.pfx > clientcert.txt  
   Enter the password for yourcertificate.pfx when prompted. This will create a text file named clientcert.txt in the working directory with information about your certificate that will be used in subsequent steps
6. Run the following command  
   keytool –importkeystore –srckeystore yourcertificate.pfx –srcstoretype pkcs12 –srcstorepass yourcertificatepassword –srcalias yourcertificatealias –destkeystore clientcert.jks –deststoretype jks –deststorepass changeit –destalias clientcert  
   Your certificate alias can be found in clientcert.txt created in the previous step. Executing this command will create keystore clientcert.jks in the working directory. The Java client we create in the next section will need this keystore file

# Setting Up a Maven Project in Eclipse

1. In Eclipse, go to File > New > Project
2. Select Maven Project and click Next
3. Check “Use default Workspace location” and click Next
4. On the archetype selection screen, choose “maven-archetype-quickstart” and click Next
5. Choose a Group Id suitable to your organization (such as “com.ercot”) and type “ews-client” for Artifact Id. Click Finish
6. Eclipse will generate a Maven quickstart project for you with the appropriate structure

# Generating Client Classes with Maven

1. Right click on your maven project and select New > Source Folder. Name the folder “src/main/schemas” and click Finish
2. Unzip “External Web Services XSD Vx.xxB”. Copy all files ending with .xsd and .wsdl to the “src/main/schemas” folder of your Maven project
3. Update your pom.xml file so the <parent>, <properties>, <dependencies>, and <build> sections look like the pom.xml supplied in the example source code.
4. In the plugins section of pom.xml, you should find two plugins listed. Comment out the spring-boot-maven-plugin around the <plugin> start and end tags. Leave the maven-jaxb2-plugin active.
5. Right click your Maven project, then select Run As > Run Configurations. In the Goals textbox type “package” without quotes, then press Apply, followed by Run
6. The maven-jaxb2-plugin will use “Nodal.wsdl” (as specified in the pom) to create proxy classes in the “src/main/java” source folder

# Setting Up Springboot Security

1. If your project doesn’t have a source folder named “src/main/resources”, create it by right clicking on your Maven project and choosing New > Source Folder
2. Copy the “clientcert.jks” keystore you created in the first section of this document into the “src/main/resources” folder in your project
3. Copy the “SecurityPolicy.xml” file from the supplied project code into the same folder

# Creating Springboot Classes (Almost done!)

1. Right click on your project’s “src/main/java” source folder, then choose New > Package. Name the package something relevant. It will contain 3 client classes to configure and run the application as a Springboot app
2. Copy the supplied “Application.java”, “EwsClient.java”, and “EwsConfiguration.java” classes from the code bundle (or create and fill them in yourself).
3. Open “EwsConfiguration.java” and check the following.

* Make sure the context path of the marshaller() method is set to the package where “RequestMessage.java” lives. If you didn’t specify otherwise, the maven-jaxb2-plugin will have placed it in “com.ercot.schema.\_2007\_06.nodal.ews.message” and no change will be necessary
* The keyStore() method sets parameters relevant to the keystore you created in the first section. Make sure the filename matches what you actually named the .jks file, and that it is in the “src/main/resources” folder in your project. The password should be whatever you used when you were creating the keystore in the first section (default is changeit)
* The keyStoreHandler() method sets a private key password. This is the password for yourcertificate.pfx. The default alias should match the –destalias used in the first section when generating the keystore. SecurityPolicy.xml in the resources folder should also use this same alias. It’s “clientcert” by default if you’ve been following this guide
* The securityInterceptor() method should configure itself with the “SecurityPolicy.xml” file in your project’s resources folder
* The ewsClient() method should set a default URI. Observe the soap:operation elements in Nodal.wsdl for wsdl:operation MarketTransactions, MarketInfo, and Alerts. The default URI should be the portion of the soap:operation string that they have in common (up to “HttpEndPoint” in this example)

1. “EwsClient.java” is a bare bones client class that uses Spring’s WebServiceTemplate to create SOAP messages and make web service calls. It doesn’t need to be modified but certainly can be to meet your needs
2. Open “Application.java” and observe the following

* The formRequest() method creates a RequestMessage object and populates it with the required fields (as specified by the wsdl) to make a successful request. It is currently set up to make a “getSystemStatus” web service call. Make sure the source and user id of the HeaderType object are aligned with what’s specified in yourcertificate.pfx
* The CommandLineRunner bean makes a call to the EwsClient. The “soap\_address\_x” parameter should match the URL of the EWS server you’re trying to interact with, the “soap\_action\_x” parameter should match the endpoint of the type of web service call you’re trying to make (in the case of “getSystemStatus” this is the market info endpoint), and the third argument is the RequestMessage for the web service call

# Building the Application with Maven

1. Open your pom.xml file. In the build > plugins section of the pom, comment out the maven-jaxb2-plugin around the <plugin> start and end tags, as there is no reason to generate Java proxy classes a second time
2. Uncomment the spring-boot-maven-plugin by removing the comment start and end points around the <plugin> tags that you created earlier in this tutorial
3. Right click your project, then choose Run As > Maven build. It should default to using the Maven run configuration you created earlier
4. A .jar file should be produced in the target folder of your project

# Running the Client

1. From the Windows start menu, go to All Programs > Accessories > Cmd to open a command prompt
2. Run the following command

java –jar jarname.jar  
where jarname.jar is the full path to the jar in the target folder of your project that you created in the previous step (for example C:\MyWorkspace\MyProject\target\jarname.jar)