# How to Use This Guide

The code presented in this guide is for educational purposes only and is intended to provide a working example of a client that consumes ERCOT Web Services. It is not intended to be a production level application.

To get started, download and unzip the source code, or clone the repository using Git. From there, you can either create a new keystore with your company’s certificate and just drop it into the existing project, or follow the steps below which illustrate how to create a client project from scratch.

Regardless of which route you choose, the remainder of this guide will assume that you have access to the provided source code.

# Prerequisites

1. It is recommended but not required that you complete this Spring tutorial on consuming a SOAP web service, as we will be building a Spring Boot application to do the same for ERCOT web services.   
   <https://spring.io/guides/gs/consuming-web-service/>

The source code that you can download in the tutorial will look very similar to ours in the end but lacks the added complexity of setting up certificate authentication, so it’s a good starting point for getting a sample Spring Boot application up and running to consume SOAP web services.

1. Make sure you have a JDK (preferably Java 8) installed and that your PATH variable contains its bin directory so you can use the keytool program.
2. It is assumed that you are using Eclipse as an IDE. If not, instructions for setting up, configuring, and running the project may need to be adjusted.
3. It is assumed that an MP interested in this implementation has already established a relationship with ERCOT and has established a USA for their company.

# Creating a Keystore From Your Client Certificate

1. Contact your company’s USA to request an API certificate for the Market Operations Test Environment (MOTE).
2. Copy your client certificate (yourcertificate.pfx) to any working directory of your choosing
3. Open a command prompt and cd to your working directory
4. Run the following command  
   keytool –v –list –storetype pkcs12 –keystore yourcertificate.pfx > clientcert.txt  
   Note that if your certificate filename has white spaces that you must enclose it in quotes. 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
5. 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, so make sure it’s placed in the src/main/resources directory of the project.

# Create a Truststore

1. You will need to create a truststore that contains all of the certificates in the MOTE server’s certificate chain. As of the time of this writing, the certificates listed in this section are accurate, but it’s possible they may change as new certificates are generated for the MOTE environment. Please make sure your truststore has the correct certificate chain imported or the client will not work.
2. Go to <http://www.ercot.com/services/mdt/webservices> and obtain a copy of the Symantec SHA256 Intermediate SSL certificate. Also obtain a copy of the VeriSign Universal Root Certification Authority certificate. These should comprise the certificate chain for the EWS server’s SSL certificate.
3. Unzip (if necessary) and move the certificates into your working directory
4. Run the following command

keytool –import –file VeriSignUniversalRootCertificationAuthority.cer –alias VeriSignUniversalRootCertificationAuthority –keystore truststore.jks

When prompted for a password, enter “changeit”, then enter yes to trust this certificate. A truststore.jks file should be created in the working directory with the Verisign Root CA certificate added to it

1. Run the following command

keytool –import –file SymantecSHA256Intermediate.cer –alias SymantecSHA256Intermediate –keystore truststore.jks

When prompted for a password, enter “changeit”. The Symantec Intermediate CA certificate should now be added to truststore.jks

1. If you encounter errors with messages like “unable to find valid certification path to requested target” or “PKIX path building failed” when running the application it’s likely that the server’s certificate chain was not successfully added to the truststore.
2. If you’re having trouble, the truststore has been included in the sample source code in the src/main/resources folder and can be used as-is assuming the MOTE server certificate hasn’t changed since the time of this writing.

# 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. Download the EWS specification, XSDs, and WSDL from <http://www.ercot.com/services/mdt/xsds/index.html>. The necessary file is zipped and named “External Web Services XSD Vx.xxB”.
3. 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
4. Update your pom.xml file so the <parent>, <properties>, <dependencies>, and <build> sections look like the pom.xml supplied in the example source code.
5. 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.
6. 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
7. 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 earlier in this guide into the “src/main/resources” folder in your project
3. Copy the “SecurityPolicy.xml” file from the supplied project code into the same folder
4. By default “truststore.jks” is also included in this folder. Its path must be set in the run configuration that you create later.

# 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” and “ResponseMessage.java” reside. 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 client keystore you created earlier. 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 (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 section regarding 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

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. If you open clientcert.txt that you created earlier in this tutorial, the source will correspond to the certificate’s CN (common name), and the user id will correspond to the EmployeeID that is found listed as an OU (organizational unit).
* The CommandLineRunner bean makes a call to the EwsClient. The “soap\_address” 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. In eclipse, go to Run > External Tools > External Tools Configuration
2. In the left pane, right click on Program and click New
3. Enter “ews-client-jar” as the run configuration name
4. For the location, click the “Browse File System” button and choose the Java executable from the bin directory of your java home installation (usually JAVA\_HOME\bin\java.exe)
5. For the working directory, click the “Browse File System” button and choose the “target” folder in your project (where maven creates the compiled jar for your project)
6. Type the following in the arguments box

-Djavax.net.ssl.trustStoreType=JKS –Djavax.net.ssl.trustStore=pathToTruststore\truststore.jks –Djavax.net.ssl.trustStorePassword=changeit -jar jarname.jar

where jarname.jar is the name of the jar in the target folder

1. Hit the Apply button to save settings for this run configuration
2. Hit the Run button to run the springboot application. Output should be directed to the eclipse console