SDL Trisoft Java WCF-Client

I Development System

[I. Development System 1](#_Toc355083298)

[A. Java Development Kit (JDK) 2](#_Toc355083299)

[B. UnlimitedJCEPolicy 2](#_Toc355083300)

[C. Maven 2](#_Toc355083301)

[D. Eclipse 3](#_Toc355083302)

[1. JDK Configuration 3](#_Toc355083303)

[2. Maven Eclipse-plugin 4](#_Toc355083304)

[3. Maven Eclipse Configuration 5](#_Toc355083305)

II SDL Trisoft WCF Client Example

[A. Creating a New Project 6](#_Toc355083306)

[B. Maven Project Configuration 7](#_Toc355083307)

[C. Initialize Project 8](#_Toc355083308)

[D. Client Configuration 9](#_Toc355083309)

[*1.* *WCF Client Configuration* 9](#_Toc355083310)

[*2.* *STS Client Configuration* 9](#_Toc355083311)

[E. Client Example Code 11](#_Toc355083312)

[F. Extended Client Example 11](#_Toc355083313)

III Troubleshooting

[A. CA Certificates 13](#_Toc355083314)

[1. Download the CA certificate. 13](#_Toc355083315)

[2. Install the CA certificate 15](#_Toc355083316)

[B. CXF Code Adjustments 17](#_Toc355083317)

1. Development System

This section will provide the setup for a development system which is necessary to run the example on a Windows machine.

# Java Development Kit (JDK)

Download and install the latest version of **JDK 7**<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

Remark:  
The JDK installation directory will be refered to as **<DIR\_JDK>**  
Wherever you need this directory, change<DIR\_JDK> according to your local JDK installation.  
Eg. **C:\Program Files\Java\jdk1.7.0**

Create the environment variable **JAVA\_HOME** with value **<DIR\_JDK>**  
Change the environment variable **PATH** by adding **“;%JAVA\_HOME%/bin”** at the end.

# UnlimitedJCEPolicy

Due to import control restrictions of some countries, the version of the Java Cryptography Extension policy files that are bundled in the Java Runtime Environment (JRE) 7 environment allow "strong" but limited cryptography to be used.  
  
Download the UnlimitedJCEPolicy bundle which provides "unlimited strength" policy files which contain no restrictions on cryptographic strengths.

<http://www.oracle.com/technetwork/java/javase/downloads/jce-7-download-432124.html>  
  
This bundle contains the following 2 files:  
local\_policy.jar  
US\_export\_policy.jar

You install the unlimited strength policy JAR files by placing them in this directory:  
<DIR\_JDK>\jre\lib\security

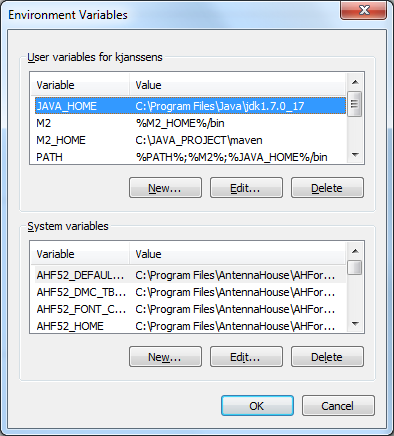
Remark:  
In case you later decide to revert to the original "strong" but limited policy versions, first make a copy of the original JCE policy files (US\_export\_policy.jar and local\_policy.jar). Then replace the strong policy files with the unlimited strength versions extracted from the downloaded bundle.

# Maven

Download and install the latest version of Maven 3.  
<http://maven.apache.org/download.cgi>

Unzip the distribution archive, i.e. apache-maven-3.0.5-bin.zip to the directory you wish to install Maven 3.0.5.   
Remark:The Maven installation directory will be refered to as <DIR\_MAVEN>  
Wherever you need this directory, change<DIR\_ MAVEN > according to your local Maven installation.

Create the environment variable **M2\_HOME**, pointing to the Maven installation directory.  
Create the environment variable **M2** with value “**%M2\_HOME%/bin”.**  
  
After steps **A** and **C** your system/user environment variables should look something like:

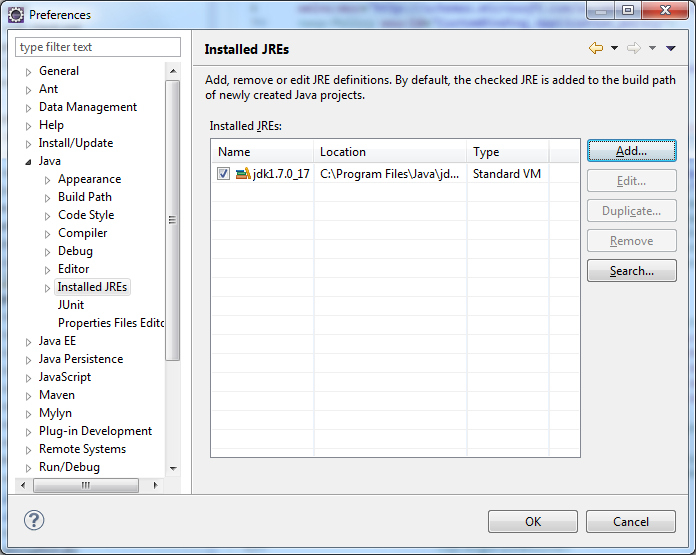


# Eclipse

Download and install **“Eclipse ID for EE Developers”**   
<http://www.eclipse.org/downloads/>

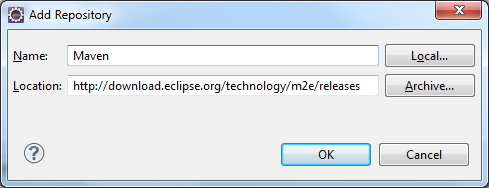
### JDK Configuration

Start Eclipse 🡪 Go to Window 🡪 Preferences 🡪 Java 🡪 Installed JREs 🡪 Click “Add...” 🡪 Choose “Standard VM” and click “Next >” 🡪 Click “Directory...” 🡪 Choose <DIR\_JDK> 🡪 Click “Finish”  
In the window “Installed JREs” check the checkbox next the newly added jdk.

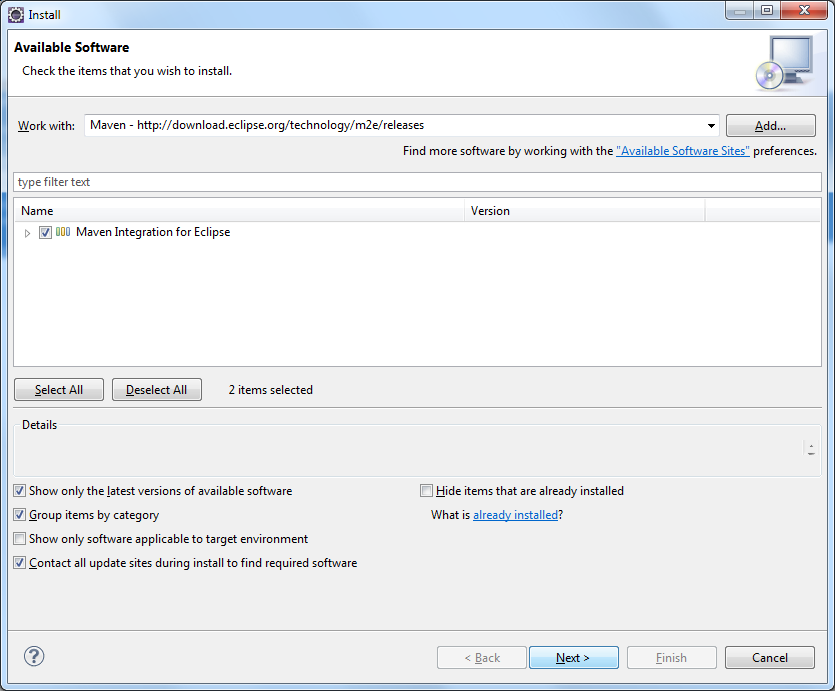
  
This will make Eclipse use the installed JDK.

### Maven Eclipse-plugin

Help 🡪 Install New Software … 🡪 Click “Add…”



<http://download.eclipse.org/technology/m2e/releases>

Complete the installation by click “Next >” and finally “Finish”  
  


### Maven Eclipse Configuration

Go to Window 🡪 Preferences 🡪 Maven 🡪 Installation 🡪 Click “Add...” 🡪 Choose <DIR\_MAVEN> 🡪 Click “Ok” 🡪 Check “External <DIR\_MAVEN>”

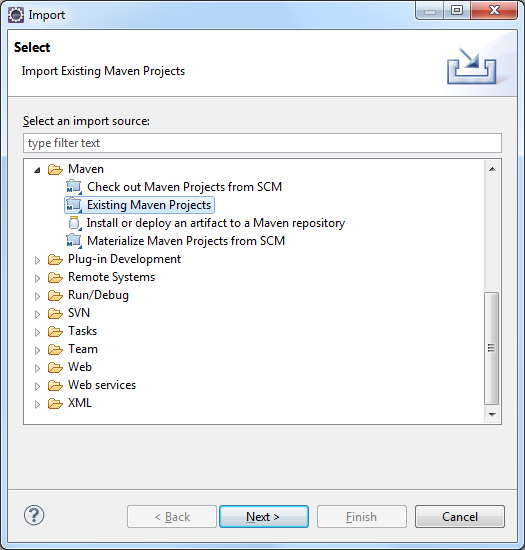
1. SDL Trisoft WCF Client Example

# Creating a New Project

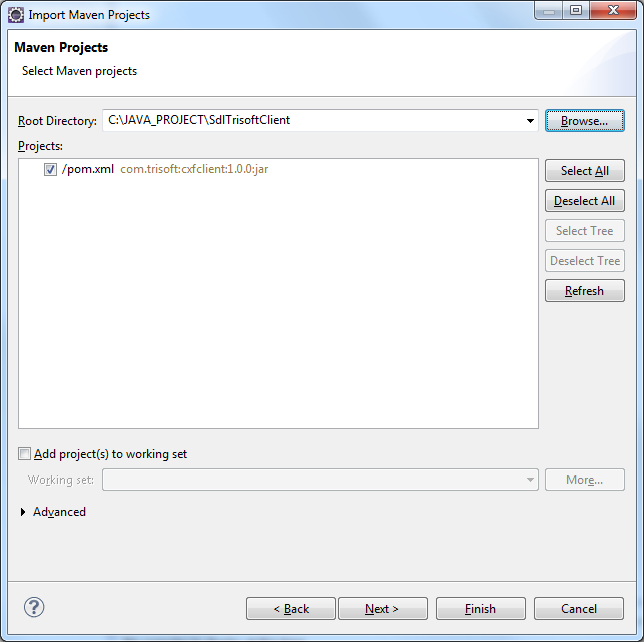
Extract the provided **SdlTrisoftClient.zip**  to a location of your choice.

Remark:  
The project extraction directory will be refered to as <DIR\_CLIENT>

In Eclipse you import this project:  
File 🡪Import ... 🡪 Selecht Maven 🡪 Select “Existing Maven Projects” 🡪 Click “Next >”



Click “Browse...” 🡪 Select the extracted project 🡪 Click “Ok” 🡪 Click “Finish”



# Maven Project Configuration

A Maven project is configured through its **“pom.xml”** file.

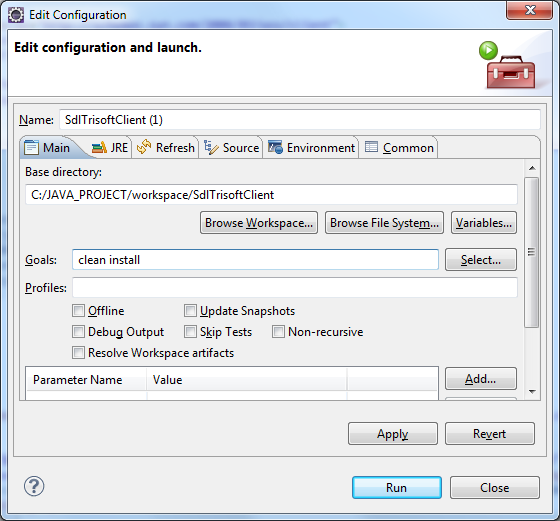
Open the pom.xml (with Eclipse) and go to the **<properties>** section

Complete the values for the following tags  
<wcf.url> Hostname of the WCF service, eg. https://service.sdl.corp  
<wcf.path> Path to the WCF service, eg. InfoShareWS/API25/Application.svc  
<wcf.package> The main package name of the generated proxy classes  
<sts.url> Hostname of the STS service   
<sts.path> Path to the STS WSDL file, eg. InfoShareSTS/issue/wstrust?wsdl  
 OR the MEX file, eg. adfs/services/trust/mex

# Initialize Project

Before we can continue with the configuration of the WCF client, we’ll need the WSDL files of the given webservices. These will be obtained during the Maven “clean” and “install” lifecycle.

Right click the project 🡪 Select “Run As” 🡪 Select “Maven Build**…**” 🡪 Fill in “clean install” in the “Goals” field 🡪 Click “Run”  
  
Remark:  
Maven is casesensitive, so keep the same case as given in this document.



During this step, the following things will happen:

* + All webservice proxies are generated *(target/source-generated)*
  + All necessary libraries will be downloaded *(Maven Dependencies)*
  + The configured WSDL files will be downloaded *(src/main/resources)*

Remark:  
If the hostnames are secured (https) this step will very likely fail and give the following error message:  
 **[ERROR] Failed to execute goal org.codehaus.mojo:wagon-maven-plugin:1.0-beta-4:download-single (download-wcf-wsdl) on project cxfclient: Error handling resource: peer not authenticated**

This is because Java needs the CA certificate to be able to access the secure host. To make this CA certificate available to the JDK, please follow the instructions in **troubleshoot section A.** After the installation of the CA certificates, repeat this step.

# Client Configuration

In Eclipse, open the file **“src/main/resources/cxf.xml”**

### WCF Client Configuration

Look for the **<jaxws:client>** tag and complete the “name” attribute with:

**{<TARGET\_NAMESPACE>}<TARGET\_PORT>**  
Eg. {http://tempuri.org/}CustomBinding\_Application1

***Remark:****This information can be found in the WCF webservice WSDL file:* ***<DIR\_CLIENT>/src/main/resources/wsdl/service.wsdl***

### STS Client Configuration

Look for the **<bean id=”stsClient”>** tag and complete the “**value**” attributes for:

serviceName **{<TARGET\_NAMESPACE>}<SERVICE>**  
endpointName **{<TARGET\_NAMESPACE>}<PORT>**  
ws-security.username **<USERNAME>**  
ws-security.password **<PASSWORD>**

*Example:*

*serviceName {http://schemas.microsoft.com/ws/2008/06/ide ntity/securitytokenservice}SecurityTokenService*

*endpointName {http://schemas.microsoft.com/ws/2008/06/ide  
ntity/securitytokenservice}UserNameWSTrust  
Binding\_IWSTrust13Async*

***Remark I:****This information can be found in the WCF webservice WSDL file:* ***<DIR\_CLIENT>/src/main/resources/wsdl/trust.wsdl****🡺 To find the portname for an “****infoShareSTS****”, search for the value “/mixed/username”:*

|  |
| --- |
| *<wsdl:port name="UserNameWSTrustBinding\_IWSTrust13Sync"  binding="i0:UserNameWSTrustBinding\_IWSTrust13Sync">  <soap12:address location="https://host/InfoShareSTS/  issue/wstrust/mixed/username"/>  <wsa10:EndpointReference>  <wsa10:Address>  https://host/InfoShareSTS/issue/wstrust/mixed/username  </wsa10:Address>  </wsa10:EndpointReference> </wsdl:port>* |

*🡺 To find the portname for an “****ADFS****”, search for the value “/usernamemixed”:*

|  |
| --- |
| *<wsdl:port name="UserNameWSTrustBinding\_IWSTrust13Async2"  binding="tns:UserNameWSTrustBinding\_IWSTrust13Async2">*  *<soap12:address location="https://host/adfs/services/  trust/13/usernamemixed"/>*  *<wsa10:EndpointReference>*  *<wsa10:Address>*  *https://host/adfs/services/trust/13/usernamemixed*  *</wsa10:Address>*  *</wsa10:EndpointReference>*  *</wsdl:port>* |

***Remark II:****The usercredentials can also be provided in code. See the commented code lines in the client example code*

# Client Example Code

The following code snippet is the bootstrap code for your client.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13 | Bus bus = new SpringBusFactory().createBus("cxf.xml");  /\* PROVIDE USERNAME & PASSWORD IN CODE \*/  // STSClient stsClient = (STSClient) bus  // .getExtension(BusApplicationContext.class)  // .getBean("stsClient");  // stsClient.getRequestContext()  // .put(SecurityConstants.USERNAME, “user");  // stsClient.getRequestContext()  // .put(SecurityConstants.PASSWORD, "secret");  Service service = new Service();  IService port = service.getCustomBindingApplication1();  System.out.println(port.getVersion()); |

Line 1 is for loading the CXF configuration.

The used classes at lines 11 and 12 are the generated WCF proxy classes. These are available in the directory “**<DIR\_CLIENT>/target/source-generated”**.

The service class can be found by searching for the class that extends “**javax.xml.ws.Service”**

# Extended Client Example

The following code snippet shows how an actual SDL Trisoft webservice is accessed, using different datatypes and ByRef parameters.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | new SpringBusFactory().createBus("cxf.xml");  String LOGICAL\_ID = "ISHPUBLILLUSTRATIONMISSING";    DocumentObj\_Service service = new DocumentObj\_Service();  DocumentObj port = service.getCustomBindingDocumentObj1();  System.out.println("DocumentObj25.GetFolderLocation(...)");  Holder<BaseFolder> baseFolder = new Holder<BaseFolder>();  Holder<ArrayOfstring> folderPath =   new Holder<ArrayOfstring>();  Holder<ArrayOflong> folderRefs = new Holder<ArrayOflong>();  port.folderLocation(LOGICAL\_ID, baseFolder,   folderPath, folderRefs);  System.out.println(baseFolder.value.toString());  for (String folder : folderPath.value.getString()) {  System.out.println(folder);  }  for (Long folderRef : folderRefs.value.getLong()) {  System.out.println(folderRef);  }    System.out.println("DocumentObj25.GetMetadata(...)");  String language = "en";  String resolution = "Low";  Holder<String> version = new Holder<String>("1");  Holder<String> xmlObjectList = new Holder<String>();  port.getMetadata(LOGICAL\_ID, version, language,   resolution, "", xmlObjectList);  System.out.println(version.value);  System.out.println(xmlObjectList.value); |

1. Troubleshooting

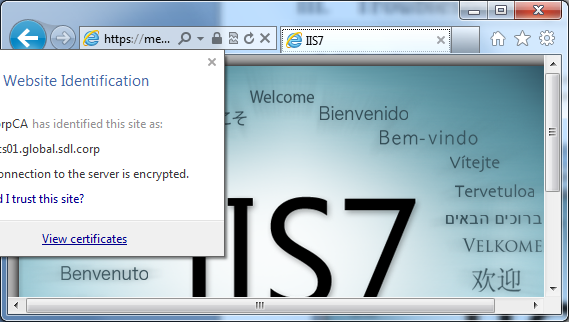
# CA Certificates

Due to security reasons the JDK isn’t able to access secures hosts by itself. It must be explicitly granted to do so by giving gim the necessary CA certificates.

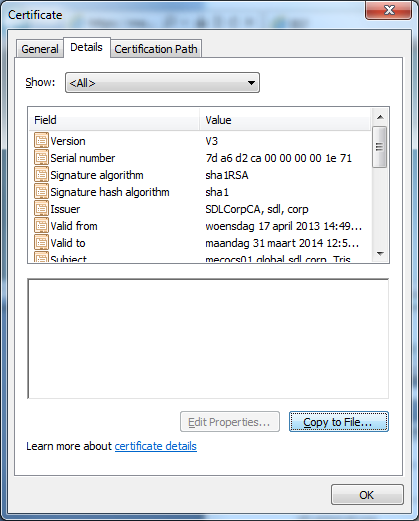
### Download the CA certificate.

****

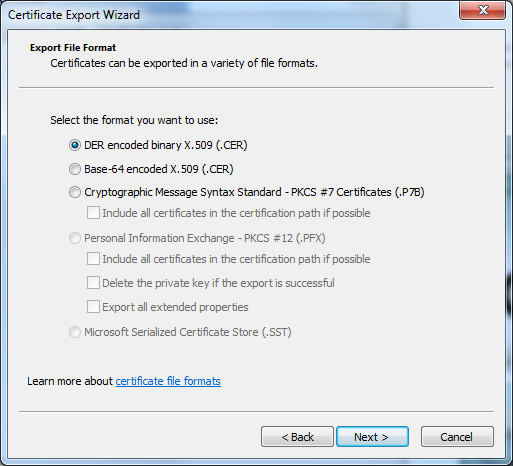
* Click on the lock

****

* Click “View certificates”

****

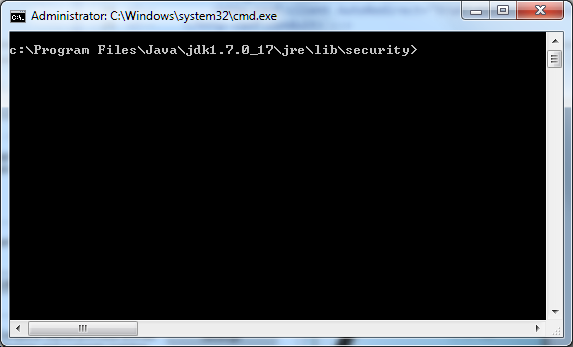
* Click “Copy to File …”



* Save the .CER file at a location of your choice.

### Install the CA certificate

Open a command prompt and go to <DIR\_JDK>/jre/lib/security



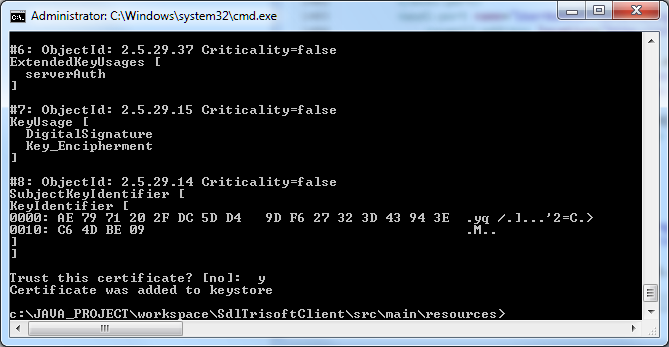
* Execute the following command  
    
  **keytool -importcert -trustcacerts -alias <ALIAS\_HOST> -keystore cacerts -file c:\host.cer -storepass <STORE\_PASSWORD>**

<ALIAS\_HOST> Replace this by a meaningfull alias for the host

<STORE\_PASSWORD> By default the keystore password is **“changeit”**

* Press “Enter”
* Answer yes on the question **“Trust this certificate”** by typing the letter “**y**” and press “Enter”
* The message **“Certificate was added to keystore”**
* Copy paste the value of this tag in a new file (**filename: x509.cer**) and save it in “**<DIR\_CLIENT>/src/main/resources”**:

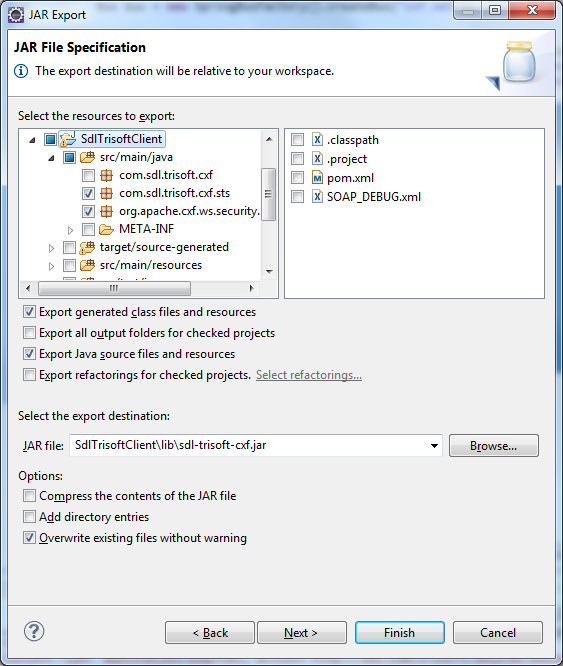
|  |
| --- |
| **-----BEGIN CERTIFICATE----- <VALUE\_OF\_THE\_ X509CERTIFICATE \_TAG>**  **-----END CERTIFICATE-----** |

* Open a command prompt and go to **“<DIR\_CLIENT>/src/main/resources”**
* Execute the following command  
    
  **keytool -import -keystore encryption.jks -storepass sdlpass -file x509.cer**
* 
* Answer yes on the question **“Trust this certificate”** by typing the letter “**y**” and press “Enter”
* The message **“Certificate was added to keystore”**

# CXF Code Adjustments

To be able to get the CXF library working with the SDL Trisoft WCF webservices, certain code overrides had to be made to the CXF library. These overrides are included in the example zip file and can be found in the file “**sdl-trisoft-cxf.jar**”.

If the CXF libraries are updated and these code overrides causes troubles, adjustements can be made by extracting the content of this jar and put in “src\main\java” of your existing Client Project.   
After you made the necessary adjustments, you export these changes to a new jar file:  
Right click the project 🡪 Click “Export…” 🡪 Select “Java” 🡪 Select JAR file 🡪 Select only the overriding class source files in the file explorer window:



Fill in the JAR file location (you can immediately overwrite the one in your client project) 🡪 Click “Finish” 🡪 Refresh the project and the new class files should become active