JAVA - DRI Connection Test Manual

This is a step by step guide on how to access the DRI Web service with native Java.

1. Install the NetBeans IDE

Since the DRI Web service is developed in .NET WCF, interoperability with Java is a big issue. The NetBeans IDE offers a lot of help in this regard out of the box which makes it a requirement for this guide. NetBeans Version 8.2 was used for this guide.

Download the **Java EE** edition from here: https://netbeans.org/downloads/

2. Setup a new Project

In NetBeans, create a new Java Project and name it: DRIConnectionJava

Right click the Project and add 3 new Folders: New -> Folder…

- WSDL
- Certificates
- TestData

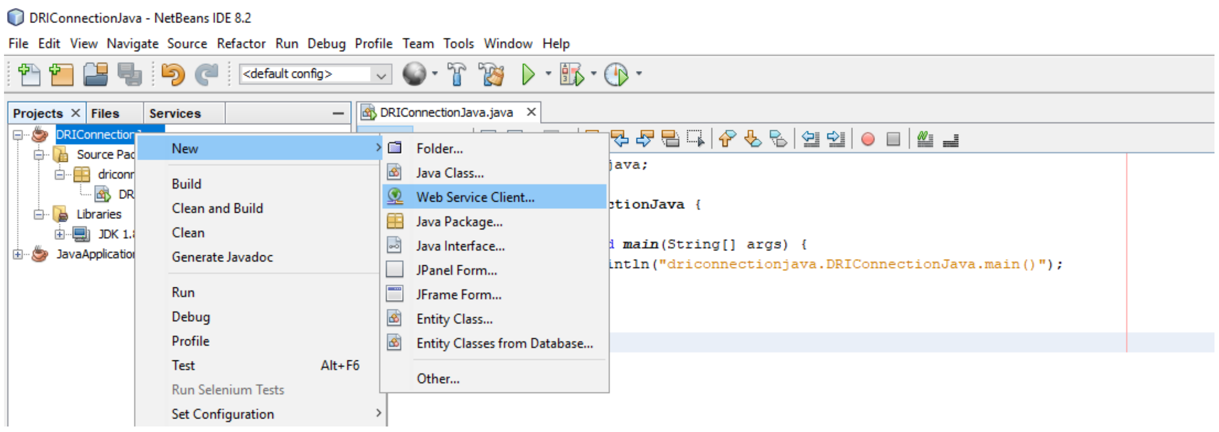
3. Import the WSDL

From now on, the DRI Web service environment (PROD, INT, TEST) will matter. This guide will only use the TEST environment. The steps however can easily be transferred to other environments.

Download the **singleWsdl** file from the DRI Webservice and save it to the WSDL Folder you created earlier. The URL is: https://dri-test.fma-li.li/DRITest/DataReception.svc?singleWsdl

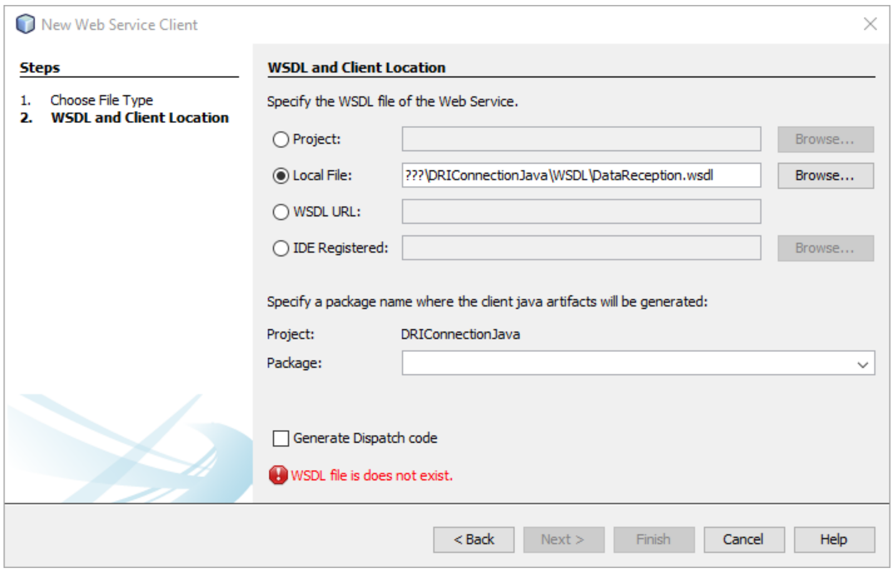
Note: The access to the WSDL is protected with TLS 1.2 with Client Certificate. You will need to install a valid Client Certificate in order to be able to download the singleWsdl. You can use the TESTKUNDE-TEST.pfx file for this purpose (provided with this sample). Alternatively the WSDL is also contained in this example as a file.

Make sure you save the file to the WSDL folder with the ending .wsdl in order for NetBeans to recognize it. Now create a new Web service Client by right-clicking the project and choosing: New -> Web Service Client…



```
java;
tionJava {
 main(String[] args) {
intln("driconnectionjava.DRIConnectionJava.main()");
```

Select the WSDL File you downloaded earlier as a **Local File** as shown below (replace ??? with your project location):



New Web Service Client

Steps
1. Choose File Type
2. **WSDL and Client Location**

WSDL and Client Location

Specify the WSDL file of the Web Service.

( ) Project:
(●) Local File: ???\DRIConnectionJava\WSDL\DataReception.wsdl
( ) WSDL URL:
( ) IDE Registered:

Specify a package name where the client java artifacts will be generated:

Project: DRIConnectionJava

Package:

☐ Generate Dispatch code

WSDL file is does not exist.

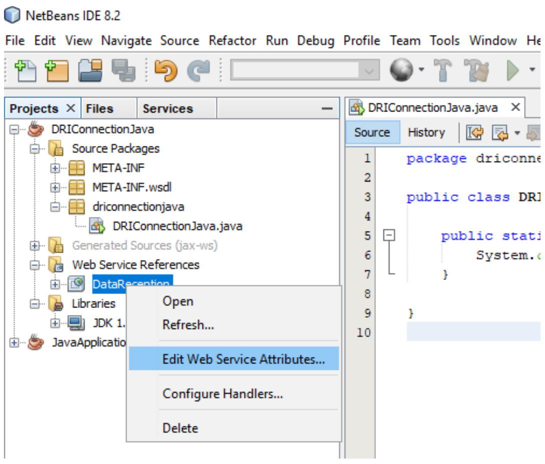
Make sure the project builds at this point by pressing the hammer icon:

Note: It might be necessary to change the encoding for source files of your project from utf-8 to ISO-8859-1 because of the “Umlaute” (right-click project -> Properties -> Sources -> Encoding)

4. Tweaking the WSDL

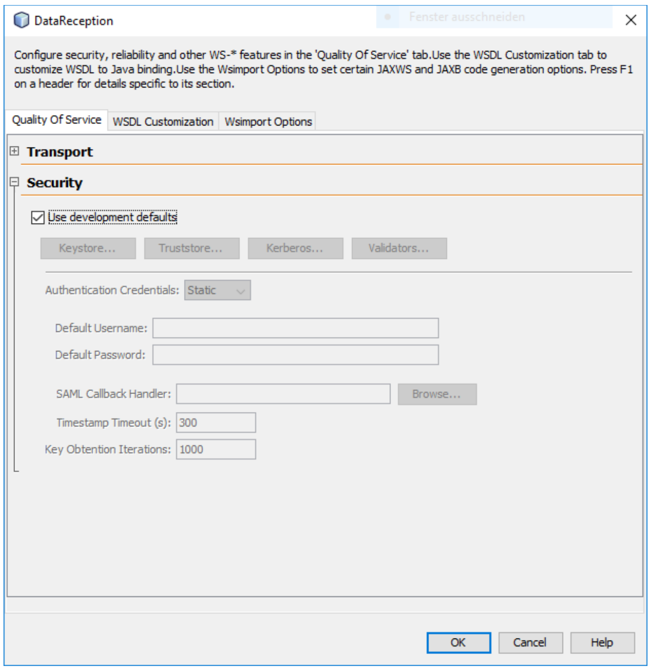
This next step is required to make NetBeans use the **Metro** JAX-WS library which has built in .NET interop capabilities.

Expand the Web Service Reference Folder and right-click the Web Service. Select the **Edit Web Service Attributes…** entry:



```
package driconne

public class DRI

    public stati
        System.
    }

}
```

In the Service Attribute Window *check* the **Use development defaults** checkbox.



DataReception

Configure security, reliability and other WS-* features in the 'Quality Of Service' tab.Use the WSDL Customization tab to customize WSDL to Java binding.Use the Wsimport Options to set certain JAXWS and JAXB code generation options. Press F1 on a header for details specific to its section.

Quality Of Service | WSDL Customization | Wsimport Options

Transport

Security

☑ Use development defaults

Keystore... Truststore... Kerberos... Validators...

Authentication Credentials: Static

Default Username:

Default Password:

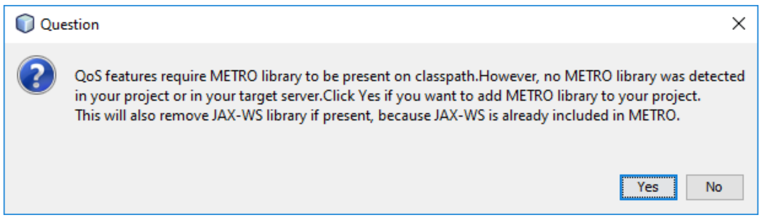
SAML Callback Handler: Browse...

Timestamp Timeout (s): 300

Key Obtention Iterations: 1000

OK Cancel Help

You will now be prompted if you want to add the METRO library to your classpath; this is exactly what we want – to use METRO instead of the native JAX-WS library -> Answer with **Yes**

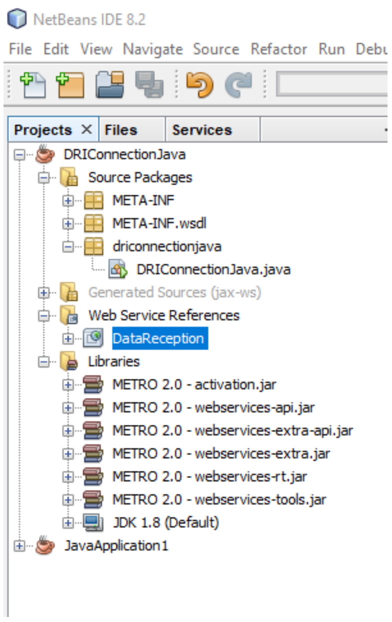


Question

QoS features require METRO library to be present on classpath.However, no METRO library was detected in your project or in your target server.Click Yes if you want to add METRO library to your project.
This will also remove JAX-WS library if present, because JAX-WS is already included in METRO.

Yes No

Afterwards, expand the Libraries Folder and it should look like this:



NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debu

Projects | Files | Services

```
DRIConnectionJava
  Source Packages
    META-INF
    META-INF.wsdl
    driconnectionjava
      DRIConnectionJava.java
  Generated Sources (jax-ws)
  Web Service References
    DataReception
  Libraries
    METRO 2.0 - activation.jar
    METRO 2.0 - webservices-api.jar
    METRO 2.0 - webservices-extra-api.jar
    METRO 2.0 - webservices-extra.jar
    METRO 2.0 - webservices-rt.jar
    METRO 2.0 - webservices-tools.jar
    JDK 1.8 (Default)
JavaApplication1
```

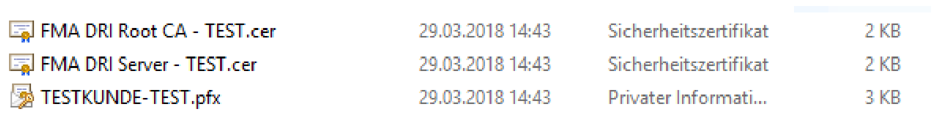
You should still be able to build the project.

5. Generating the Key & Truststore

The next step is to use Mutal Certificates for the Custom WCF binding. For this you need to convert the certificates using the Java Key Tools.

Extract the Content of the TESTKUNDE-TEST .zip archive which is contained in this example to the Certificate Folder you created earlier.

The content of your certificate folder should look like this:



FMA DRI Root CA - TEST.cer	29.03.2018 14:43	Sicherheitszertifikat	2 KB
FMA DRI Server - TEST.cer	29.03.2018 14:43	Sicherheitszertifikat	2 KB
TESTKUNDE-TEST.pfx	29.03.2018 14:43	Privater Informati...	3 KB

Open a command line window and navigate to the certificate folder. Now you can use the **Java Key Tool** to generate the correct certificate stores.

*Note: Make sure your **Java bin folder** is added to the path environment variable in order for the **keytool** command to be found -> Google knows how*

The Key store command

```
keytool -importkeystore -srckeystore TESTKUNDE-TEST.pfx -srcstoretype
pkcs12 -destkeystore TESTKUNDE-TEST.jks -deststoretype JKS
```

- As the Target-Keystore-Password, choose a secure password
- Repeat the new Password in the second prompt
- Enter the password stated in *TESTKUNDE-TEST.Password.txt* in the third prompt

The Trust store command

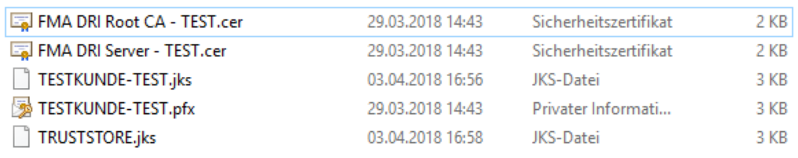
```
keytool -importcert -keystore TRUSTSTORE.jks -alias "FMA DRI Root CA -
TEST" -srcstoretype pkcs12 -file "FMA DRI Root CA - TEST.cer" -
deststoretype JKS
```

- As the Keystore-Password, choose a secure password
- Repeat the new Password in the second prompt
- Enter **Yes** to trust the self-signed FMA Root Certificate

```
keytool -importcert -keystore TRUSTSTORE.jks -alias "FMA DRI Server - TEST"
-srcstoretype pkcs12 -file "FMA DRI Server - TEST.cer" -deststoretype JKS
```

- Enter the password you just choose

Your certificate folder should now look like this:

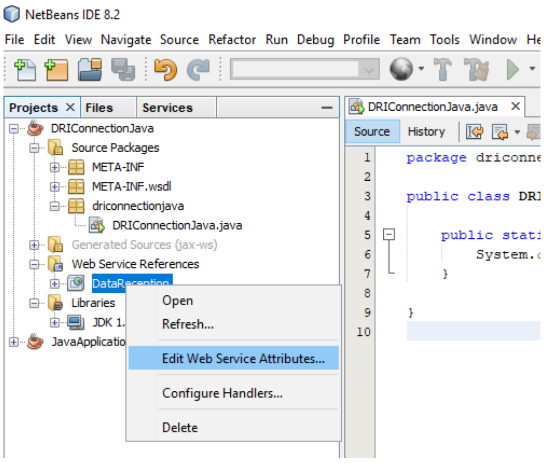


FMA DRI Root CA - TEST.cer	29.03.2018 14:43	Sicherheitszertifikat	2 KB
FMA DRI Server - TEST.cer	29.03.2018 14:43	Sicherheitszertifikat	2 KB
TESTKUNDE-TEST.jks	03.04.2018 16:56	JKS-Datei	3 KB
TESTKUNDE-TEST.pfx	29.03.2018 14:43	Privater Informati...	3 KB
TRUSTSTORE.jks	03.04.2018 16:58	JKS-Datei	3 KB

6. Using the Certificate Stores

Now you have to use the Certificates you just generated for the Mutal Certificate Custom Binding.

Right click the Web Service again and select the **Edit Web Service Attributes…** entry:

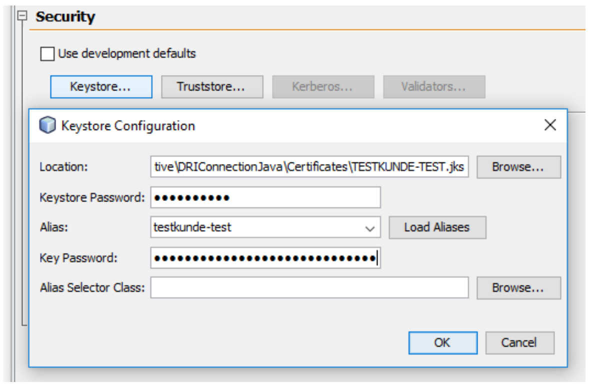


```
package driconne

public class DRI

    public stati
        System.(
    }

}
```

Uncheck the Use development defaults checkbox.

Edit the Keystore…

Select the location of the Keystore you created in the last step. Enter the password you choose as the **Keystore Password** and enter the password stated in *TESTKUNDE-TEST.Password.txt* as **Key Password**. Load Aliases and the correct alias should already be selected:

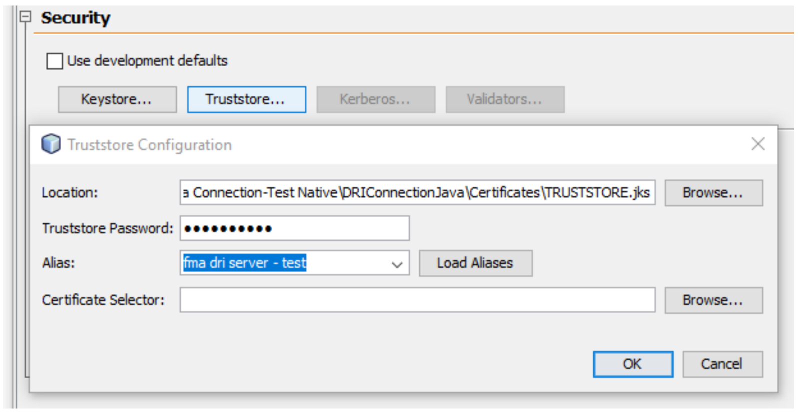


☐ Use development defaults

Location: tive\DRIConnectionJava\Certificates\TESTKUNDE-TEST.jks
Keystore Password: ••••••••••
Alias: testkunde-test
Key Password: ••••••••••••••••••••••••••••••
Alias Selector Class:

Edit the Truststore…

Select the location of the Truststore you created in the last step and enter the password you choose as Truststore Password. Load Aliases and select the **fma dri server –test** alias. DO **NOT** SELECT THE ROOT ALIAS!!!

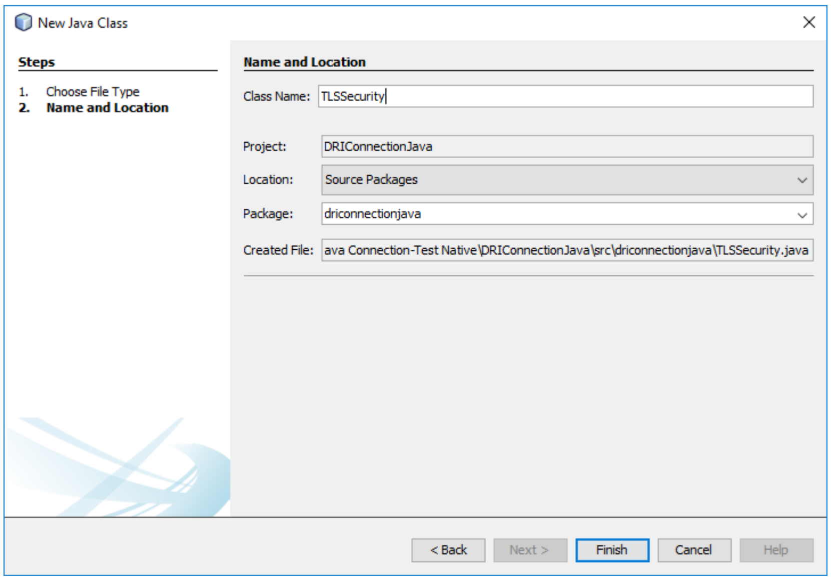


☐ Use development defaults

Location: a Connection-Test Native\DRIConnectionJava\Certificates\TRUSTSTORE.jks
Truststore Password: ••••••••••
Alias: fma dri server - test
Certificate Selector:

7. Code of the Sample

Now let's get to the code. The first step is to prepare for using TLS. For this purpose add a new Java class to the project and name it **TLSSecurity.java**:



Class Name: TLSSecurity

Project: DRIConnectionJava

Location: Source Packages

Package: driconnectionjava

Created File: ava Connection-Test Native\DRIConnectionJava\src\driconnectionjava\TLSSecurity.java

Paste the following source code into the class:

```
package driconnectionjava;

import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
import java.security.KeyStore;
import java.security.SecureRandom;
import java.security.Security;
import javax.net.ssl.KeyManagerFactory;
import javax.net.ssl.SSLContext;

/**
 * Helper Class to build a TLS 1.2 Context
 */
public class TLSSecurity {

    /**
     * Creates a TLS 1.2 Context using a Client Certificate
     * @param clientCertificatePath Relative path to the Client Certificate File (.pfx)
     * @param clientCertificatePassword Password fot the Client Certificate
     * @return The TLS 1.2 Context to be used for the HTTPS Connection
     * @throws Exception TODO - Add sophisticated Exception handling
     */
    public static SSLContext getContext(String clientCertificatePath,
          String clientCertificatePassword) throws Exception {

        // Enable Oracle Strong Encryption
        Security.setProperty("crypto.policy", "unlimited");

        // Load the Client Certificate to be used for the TLS Handshake
        KeyStore keyStore = KeyStore.getInstance("PKCS12");

        File keyFile = new File(clientCertificatePath);
        try (InputStream keyInput = new FileInputStream(keyFile)) {
            keyStore.load(keyInput, clientCertificatePassword.toCharArray());
        }

        KeyManagerFactory keyManagerFactory = KeyManagerFactory.getInstance("SunX509");
        keyManagerFactory.init(keyStore, clientCertificatePassword.toCharArray());

        // Create the TSL Context for Transport Security
        SSLContext context = SSLContext.getInstance("TLSv1.2");
        context.init(keyManagerFactory.getKeyManagers(), null, new SecureRandom());

        return context;
    }
}
```

Create another new Java class in the same package and name it **DRIConnectionTest.java**.

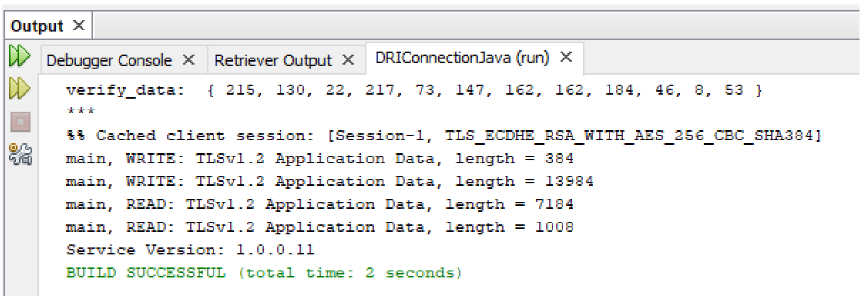
Paste the following code into the file:

```
package driconnectionjava;

import de.amana_consulting.dri.DataReception;
import de.amana_consulting.dri.IDataReception;
import
de.amana_consulting.dri.IDataReceptionFileTransactionReportArgumentExceptionFaultFaultMessage;
import
de.amana_consulting.dri.IDataReceptionGetTransactionReportFeedbackArgumentExceptionFaultFaultM
essage;
import
de.amana_consulting.dri.IDataReceptionGetTransactionReportFeedbackFeedbackNotAvailableFaultFau
ltFaultMessage;
import
de.amana_consulting.dri.IDataReceptionIsTransactionFeedbackAvailableArgumentExceptionFaultFaul
tMessage;
import de.amana_consulting.dri.ReceptionResult;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.function.Consumer;
import javax.net.ssl.SSLContext;
import javax.xml.datatype.DatatypeConfigurationException;
import javax.xml.datatype.DatatypeFactory;
import javax.xml.datatype.XMLGregorianCalendar;
import javax.xml.ws.soap.SOAPFaultException;
import javax.xml.ws.BindingProvider;

/**
 * DRI Connection Test in Java
 */
public class DRIConnectionTest {

    // Constants for teh Webservice Communication
    private static final String WEBSERVICE_ADDRESS = "https://dri-test.fma-
li.li/DRITest/DataReception.svc";
    private static final String LEI_CODE = "529900UHDH7QLRMEJQ86";
    private static final String FILENAME_UPLOAD =
"TestData/LI_529900UHDH7QLRMEJQ86_2018_42.zip";
    private static final String FILENAME_DOWNLOAD = "TestData/DOWNLOAD.zip";
    private static final String DATE_DOWNLOAD = "2017-12-08T16:00:00.000Z";

    // Constants for the TLS Transport Security
    private static final String CLIENT_CERTIFICATE_PATH = "Certificates/TESTKUNDE-TEST.pfx";
    private static final String CLIENT_CERTIFICATE_PASSWORD = "333f804c-0d04-458a-9dec-
6d6df622008e";

    /**
     * @param args the command line arguments
     * @throws java.lang.Exception TODO - Add sophisticated Exception handling
     */
    public static void main(String[] args) throws Exception {

        // Show TLS debug information - useful for testing; disable for production
        System.setProperty("javax.net.debug", "ssl");

        // Enabling TLS Transport Security within the JVM
        SSLContext context = TLSSecurity.getContext(CLIENT_CERTIFICATE_PATH,
CLIENT_CERTIFICATE_PASSWORD);
        SSLContext.setDefault(context);

        // Call the Webservice - Comment or Uncomment to Test different methods

        testWebservice();

        //uploadReport();

        //isFeedbackAvailable();

        //downloadFeedback();
    }

    /**
     * Calls the get Version Testmethod
     */
    private static void testWebservice() {
        callWebservice(dataReceptionService ->{
            String moduleVersion = dataReceptionService.getModuleVersion();
            System.out.println("Service Version: " + moduleVersion);
        });
    }

    /**
     * Uploads a Transaction Report for LEI_CODE from path FILENAME_UPLOAD
     */
    private static void uploadReport() {
        callWebservice(dataReceptionService ->{
            try {
                byte[] data = Files.readAllBytes(Paths.get(FILENAME_UPLOAD));
                ReceptionResult result =  dataReceptionService.fileTransactionReport(LEI_CODE,
data);
                System.out.println("Upload Result: " + result.getMessage().getValue());
            } catch (IOException |
IDataReceptionFileTransactionReportArgumentExceptionFaultFaultMessage e) {
                System.err.println(e.getMessage());
            }
        });
    }

    /**
     * Checks if Feedback is available for for LEI_CODE and DATE_DOWNLOAD
     */
    private static void isFeedbackAvailable() {
        callWebservice(dataReceptionService ->{
            try {
                XMLGregorianCalendar feedbackDate =
DatatypeFactory.newInstance().newXMLGregorianCalendar(DATE_DOWNLOAD);
                boolean feedbackAvailable =
dataReceptionService.isTransactionFeedbackAvailable(LEI_CODE, feedbackDate);
                System.out.println("Transaction Report from " + DATE_DOWNLOAD + " is
available: " + feedbackAvailable);
            } catch (DatatypeConfigurationException |
IDataReceptionIsTransactionFeedbackAvailableArgumentExceptionFaultFaultMessage e) {
                System.err.println(e.getMessage());
            }
        });
    }

    /**
     * Downloads Feedback for for LEI_CODE on DATE_DOWNLOAD
     */
    private static void downloadFeedback() {
        callWebservice(dataReceptionService ->{
            try {
                XMLGregorianCalendar feedbackDate =
DatatypeFactory.newInstance().newXMLGregorianCalendar(DATE_DOWNLOAD);
                byte[] data = dataReceptionService.getTransactionReportFeedback(LEI_CODE,
feedbackDate);
                Files.write(Paths.get(FILENAME_DOWNLOAD), data);
                System.out.println("Transaction Report from " + DATE_DOWNLOAD + " downloaded
to: " + FILENAME_DOWNLOAD);
            } catch
(IDataReceptionGetTransactionReportFeedbackArgumentExceptionFaultFaultMessage |

IDataReceptionGetTransactionReportFeedbackFeedbackNotAvailableFaultFaultFaultMessage |
                    DatatypeConfigurationException | IOException e) {
                System.err.println(e.getMessage());
            }
        });
    }

    /**
     * Wrapper Function for Webservice Access - Wraps Setup and SOAPFaultException
     * @param webserviceAction Inject Web Service Access as Lambda
     */
    private static void callWebservice(Consumer<IDataReception> webserviceAction) {
        // Open the Webservice Proxy
        DataReception dataReception = new DataReception();
        IDataReception dataReceptionService = dataReception.getDataReceptionService();

        // Inject the correct Webservice Endpoint (PROD, INT, TEST)
        BindingProvider webserviceBinding = (BindingProvider) dataReceptionService;
        webserviceBinding.getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
WEBSERVICE_ADDRESS);

        // Call the Webservice Function
        try {
            webserviceAction.accept(dataReceptionService);
        } catch (SOAPFaultException e) {
            System.err.println(e.getMessage());
        }
    }
}
```

8. Test the Web Service Connection

Now everything should be in place. Run the **DRIConnectionTest.java** class using the *Run Project* button in the toolbar:

If everything went well, the console will show an output similar to this, indicating that the service was accessible by printing its version:



```
verify_data:  { 215, 130, 22, 217, 73, 147, 162, 162, 184, 46, 8, 53 }
***
%% Cached client session: [Session-1, TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384]
main, WRITE: TLSv1.2 Application Data, length = 384
main, WRITE: TLSv1.2 Application Data, length = 13984
main, READ: TLSv1.2 Application Data, length = 7184
main, READ: TLSv1.2 Application Data, length = 1008
Service Version: 1.0.0.11
BUILD SUCCESSFUL (total time: 2 seconds)
```

9. Testing with Data

You can now also test to upload Transaction Reports and download Feedbacks.

Copy the LI_529900UHDH7QLRMEJQ86_2018_42.zip file contained in this example to the TestData folder you created earlier.

You can now comment/uncomment the test methods you like to test. For example to test the upload use a configuration like this:

```
// Call the Webservice - Comment or Uncomment to Test different methods

//testWebservice();

uploadReport();

//isFeedbackAvailable();

//downloadFeedback();
```

You will now receive feedback from the DRI Server about your Reports.

10. Using the JAVA – DRI Connection-Test as a Library

It is most likely that you will want to the use the DRI Connection Functionality in a development environment other than NetBeans. For this purpose it is recommended to use it as a .jar file.

Net beans automatically generates a .jar file for the project when a main method is detected. This Jar is saved in the **/dist** Folder of the Project. You can test this in the command line:

- Navigate to the DRIConnectionJava/dist folder
- Copy your *Certificates* and *TestData* folder to this directory
- Run the command: java -jar "DRIConnectionJava.jar"

11. Customization

You now have everything up and running.

From now on, it' all about customizing the solution for your own needs. The following steps might be mandatory to integrate this solution into your own software solutions:

- Build the Key- and Truststore with your own Certificate you downloaded from the FMA e-Service
- Develop an API, based on this Sample Application which can be integrated in your own system – Use the JAR file for this purpose and import it in other projects
- Optionally replace the file-based solution (for transaction reports) with a in memory solution

12. The Provided Sample Code

The Sample Project which is developed in this Guide is included as FULL_SAMPLE.zip file.

Note: It won't work out of the box, because the generated Client Proxy stores some hard coded Paths that might differ on your system. To be use the sample, just delete everything in the META-INF folder and Web Service Reference. Then import the WSDL again and link the certificates as explained in sections 3 and 6.