LegStar COBOL Transformers Generator User Guide

LegStar COBOL Transformers Generator User Guide

1.6.0

Table of Contents

List of Figures

Chapter 1. Introduction

COBOL Transformers Generator overview

LegStar COBOL Transformers Generator, (coxbgen for short), is a code generator that produces COBOL to Java Transformers.

The COBOL Transformers Generator is available either as ant scripts or as an Eclipse plugin.

The major input to the generation process is an XML Schema with COBOL annotations. LegStar has 2 tools that generate such XML Schemas:

- COBOL to XML Schema Translator [http://code.google.com/p/legstar-cob2xsd/]: used when the starting point is a COBOL structure.
- XML Schema to COBOL Translator [http://code.google.com/p/legstar-xsd2cob/]: used when the starting point is a Java class, an XML Schema or a WSDL.

How to get it

coxbgen is one of the several LegStar modules. As such it is generally easier to download the complete product [http://www.legsem.com/legstar/legstar-distribution].

However, if you feel comfortable enough, you can also download a standalone distribution by following this link [download.html].

The COBOL Transformers generator can be installed as an Eclipse plugin. More details on this option are available on the LegStar Eclipse site [http://www.legsem.com/legstar/legstar-eclipse/index.html].

COBOL Transformers

Transformers are java classes that you can use to transform mainframe data to Java, XML or JSON. These same Transformers can also turn Java objects, XML or JSON to mainframe data.

Transformers don't make any assumption on where the mainframe data originated from. As far as Transformers are concerned, mainframe data is the content of a byte array.

Transformers have detailed knowledge of the mainframe data internal structure and can handle a large variety of mainframe data types such as compressed numerics, variable size arrays and redefines.

Internally, Transformers use JAXB [https://jaxb.dev.java.net] Coxbgen produces JAXB Classes with COBOL annotations. These classes are not different from regular JAXB Classes apart from additional annotations providing meta-data to map Java types with COBOL types.

An important class is CobolElement [http://www.legsem.com/legstar/legstar-core/legstar-coxbapi/apidocs/com/legstar/coxb/CobolElement.html] which is a java annotation that represent the mapping between Java properties and a Cobol data elements.

Transformers use LegStar runtime libraries which provide low-level conversions for simple types such as strings, Cobol packed decimals, Cobol zoned decimals, simple arrays, etc.

Learn more

For instructions on how to use coxbgen follow this link [legstar-coxbgen-user-guide.html].

For answers to common questions about LegStar coxbgen, see the FAQ [faq.html].

Chapter 2. Standalone Install instructions

Installing COBOL Transformers Generator standalone

These instructions apply if you are installing from the coxbgen standalone distribution. If you are installing the complete LegStar distribution please refer to legstar distribution [http://www.legsem.com/legstar/legstar-distribution].

Alternatively, the COBOL Transformers generator can be installed as an Eclipse plugin which contains all the necessary components. More details on this option are available on the LegStar Eclipse site [http://www.legsem.com/legstar/legstar-eclipse/index.html]. However, the plugin does not include the samples. Furthermore, the libraries that you would need to run your Transformers are not easily accessible with the plugin so we still recommend that you perform a standalone installation as described in this document.

Pre-requisites

Java 1.5+ and ant 1.6.5+ are both prerequisites for LegStar. This module requires a JDK, or an international version of the JRE, that includes charsets.jar. Make sure JAVA_HOME and ANT_HOME environment variables are set and that $JAVA_HOME/bin (%JAVA_HOME%/bin on Windows) and $ANT_HOME/bin (%ANT_HOME%/bin on Windows) are both in your system path.

Coxbgen relies on Sun's JAXB [https://jaxb.dev.java.net/]. The JAXB implementation must be at least 2.1 compliant. Such an implementation is part of the LegStar distribution.

You will likely need the LegStar tools that generate XML Schemas with COBOL annotations as this is the major input to COBOL Transformers generator:

- COBOL to XML Schema Translator [http://code.google.com/p/legstar-cob2xsd/]: used when the starting point is a COBOL structure.
- XML Schema to COBOL Translator [http://code.google.com/p/legstar-xsd2cob/]: used when the starting point is a Java class, an XML Schema or a WSDL.

Installing

Download the latest coxbgen standalone distribution [download.html].

Unzip the binary distribution package into the directory of your choice, referred to as <installDir> in the following steps.

The directory tree should look like this:

```
<installDir>
  |-->LICENSE
  |-->NOTICE
  |-->readme.html
```

```
|--><samples>
    |--><quickstarts>
        |--><adapter_lsfileae>
            |-->build-jaxb.xml
            |-->build-coxb.xml
            |--><schema>
                |-->lsfileae.xsd
        |--><proxy_pojo_jvmquery>
            |-->build-jaxb.xml
            |-->build-coxb.xml
            |--><src>
                |-->**/*.java
            |--><schema>
                |-->jvmquery.xsd
        |--><proxy_ws_cultureinfo>
            |-->build-jaxb.xml
            |-->build-coxb.xml
            |--><schema>
                |-->jvmquery.xsd
|--><lib>
    |-->*.jar
```

At this stage, you will probably want to run the samples provided

Uninstalling

To uninstall, remove the <installDir> folder.

Running the Transformers ant samples

Generate Transformers for the lsfileae adapter

run command *ant -f build-coxb.xml* from the samples/quickstarts/adapter_lsfileae folder and check the result.

build-coxb.xml uses the sample XML Schema in the schema folder. It generates JAXB classes in the src folder in the com.legstar.test.coxb.lsfileae package.

The JAXB classes are identical to the classes generated using Sun's XJC utility apart from additional Java 5 annotations holding COBOL meta-data.

Under the com.legstar.test.coxb.lsfileae.bind package, you will find the Transformers classes. They are generated by reflecting on the JAXB classes.

The Transformers classes provide a high performance alternative to reflecting on the JAXB classes at runtime. They are used by higher level LegStar modules.

Generate Transformers for the jvmquery proxy

run command *ant -f build-coxb.xml* from the samples/quickstarts/proxy_pojo_jvmquery folder and check the result.

You will notice that there are 2 jaxbRootClass parameters in the generateCOXB target. This is because the input and output structures from the jvmQuery POJO are different.

Generate Transformers for the cultureinfo proxy

run command *ant -f build-coxb.xml* from the samples/quickstarts/proxy_ws_cultureinfo folder and check the result.

You will notice that there are 2 jaxbRootClass parameters in the generateCOXB target. This is because the input and output structures from the cultureinfo Web service are different.

Chapter 3. Using the COBOL Transformers Generator

We assume you have an XML Schema with COBOL annotations you are ready to turn into Transformers.

We show 2 use cases in the following sections. The Adapter use case is one where you started from a commarea-driven COBOL CICS program (LSFILEAE). The Proxy use case is one where you started from a WSDL (Microsoft LIVE search).

You have the choice between using ant scripts and Eclipse plugins.

Using ant scripts

COBOL Transformers Generation for an Adapter

You can use the *build-coxb.xml* ant script from the samples/quickstarts/adapter_lsfileae folder as your starting point. It generates Transformers for the LSFILEAE COBOL program which takes the same input and output structure called Dfhcommarea.

Generally, there are 4 things you will need to change in *build-coxb.xml* in order to adapt it to your needs:

- Make sure the classpath setting is correct. The fileset should point to the location where you installed LegStar.
- The xsdFile parameter of the jaxbgen ant task must point to the location of you XML Schema with COBOL annotations.
- The jaxbPackageName parameter, for both jaxbgen and coxbgen ant tasks, must be set to the same value conforming to your naming standards.
- The jaxbRootClass parameter of the coxbgen ant task must designate one, or more, JAXB classes that you want to turn into Transformers.

build-coxb.xml executes 3 steps (or targets in ant parlance):

- The first target, generateJAXB, runs the jaxbgen ant task and turns an XML Schema with COBOL annotations into JAXB classes with COBOL annotations.

  For a complete list of options for the jaxbgen task, you can refer to CobolJAXBGenerator [http://www.legsem.com/legstar/legstar-core/legstar-jaxbgen/apidocs/com/legstar/jaxb/gen/CobolJAXBGenerator.html].

- The second target, compileJAXB, is a regular java compilation step for the JAXB classes previously generated.
- The third target, generateCOXB, runs the coxbgen ant task and produces the actual Transformers. The generation process reflects on the JAXB classes compiled at the previous step.

  You will notice that the coxbgen ant task takes one or more jaxbRootClass elements. These are needed to designate which JAXB class (or classes) should become a Transformer. You would generally pickup the higher classes in the hierarchy but you don't have to.

By default you get Java to Host Transformers only. In addition, you can get Host to XML and Host to JSON Transformers (use the xmlTransformers and jsonTransformers options).

For a complete list of options for the jaxbgen task, you can refer to CoxbBindingGenerator [http://www.legsem.com/legstar/legstar-core/legstar-coxbgen/apidocs/com/legstar/coxb/gen/CoxbBindingGenerator.html].

The Using generated tranformers section gives examples of code you could write to run Transformers.

COBOL Transformers Generation for a Proxy

You can use the *build-coxb.xml* ant script from the samples/quickstarts/proxy_ws_cultureinfo folder as your starting point. It generates Transformers for the cultureinfo Web Service which takes the GetInfo structure as its input and the GetInfoResponse structure as its output.

Refer to COBOL Transformers Generation for an Adapter for a description of what you need to customize in *build-coxb.xml* and how it works.

Assuming you want to customize *build-coxb.xml* for the Microsoft LIVE search SOAP service. Follow these steps:

- Change xsdFile parameter of the jaxbgen ant task to point to the location the XML Schema that was produced with the WSDL/XSD Structures Mapping tool.
- Change the jaxbPackageName parameter, for both jaxbgen and coxbgen, to something like *com.microsoft.schemas.msnsearch*.
- Change the jaxbRootClass parameters to Search and SearchResponse, which are the wrapper elements expected and produced by the target LIVE Web Service.

if you execute the ant script as is, you will get a JAXB error:

```
[ERROR] A class/interface with the same name "com.microsoft.schemas.msnsearch.Se
```

This is because Microsoft uses the same names for Complex Types and Elements in their XML Schemas which confuses JAXB. To solve this, you can use the typeNameSuffix parameter on the jaxbgen task. The Task should now look like this:

```
<jaxbgen
    xsdFile="LIVESearch.xsd"
    targetDir="src"
    jaxbPackageName="com.microsoft.schemas.msnsearch"
    generateIsSetMethod="true"
    serializableUid="1"
    typeNameSuffix="Type"
/>
```

You can now execute the ant script.

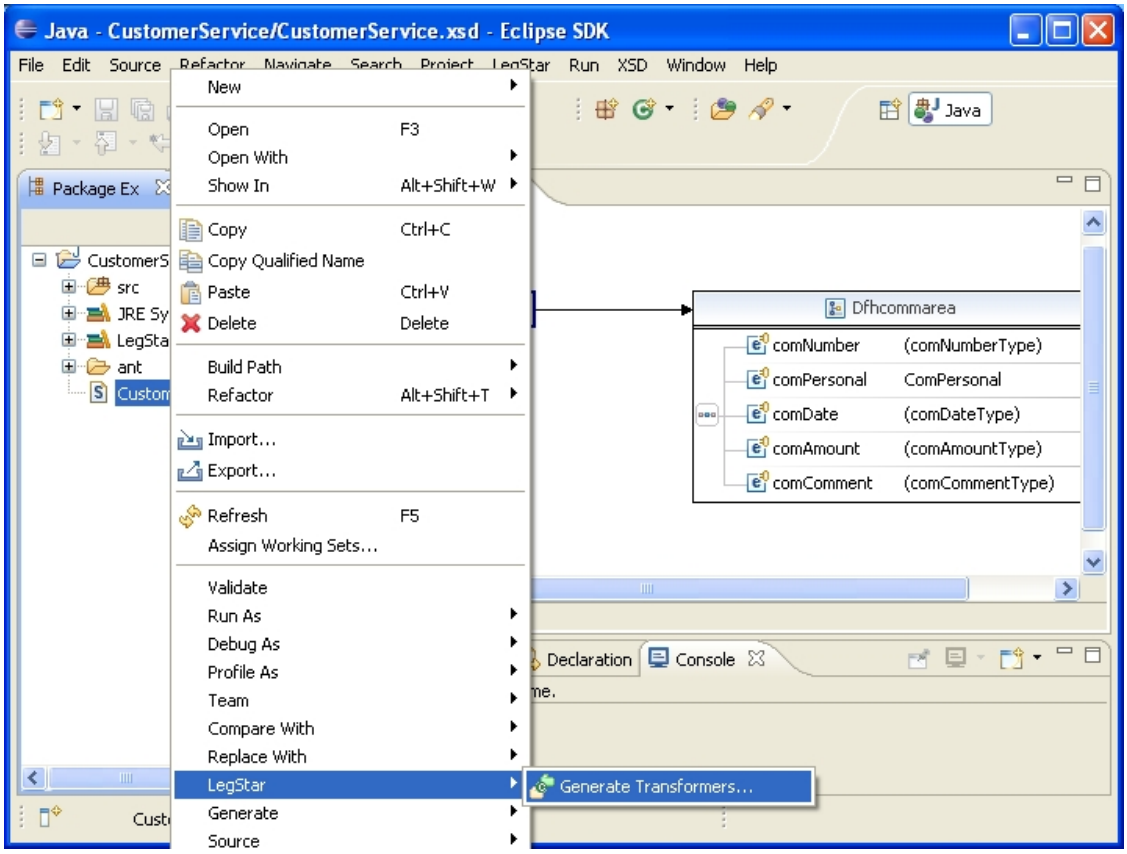
The Using generated tranformers section gives examples of code you could write to run Transformers.

Using Eclipse plugins

COBOL Transformers Generation for an Adapter

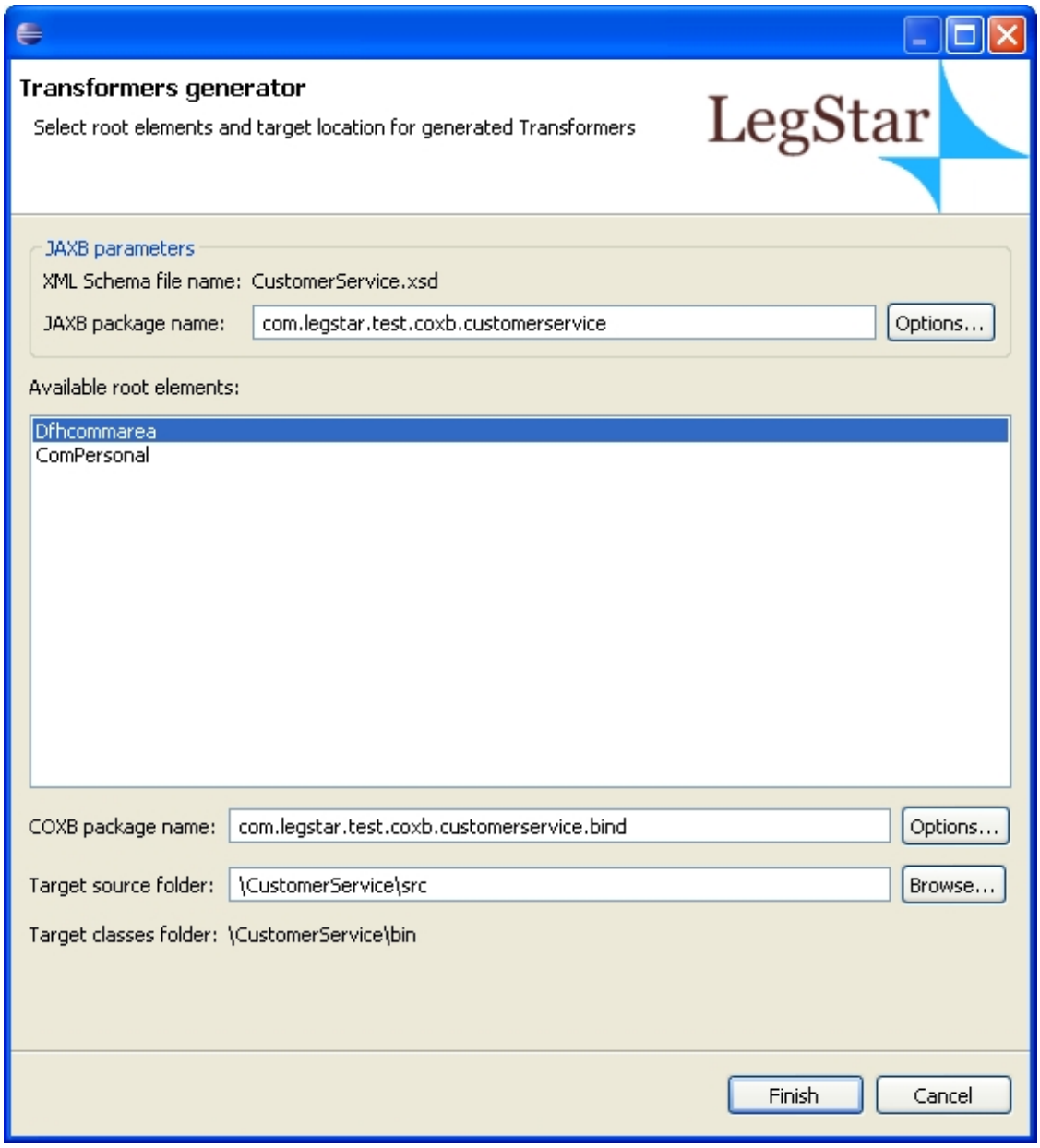
The wizard is started from the package explorer, by right clicking on the previously generated XML Schema:

Figure 3.1. Adapter COBOL Transformers generation menu



The next page allows you to specify which elements from the source XML Schema will need to be bound. All elements are displayed here but if you select a parent element, this will automatically select all children for you, so all you need to do is to select root elements:

Figure 3.2. Adapter COBOL Transformers generation parameters screen



In the Adapter case, the mainframe program expects a Dfhcommarea and also produces a Dfhcommarea so that's the only element we need to select.

The first options button allows you to customize the JAXB classes that will be generated.

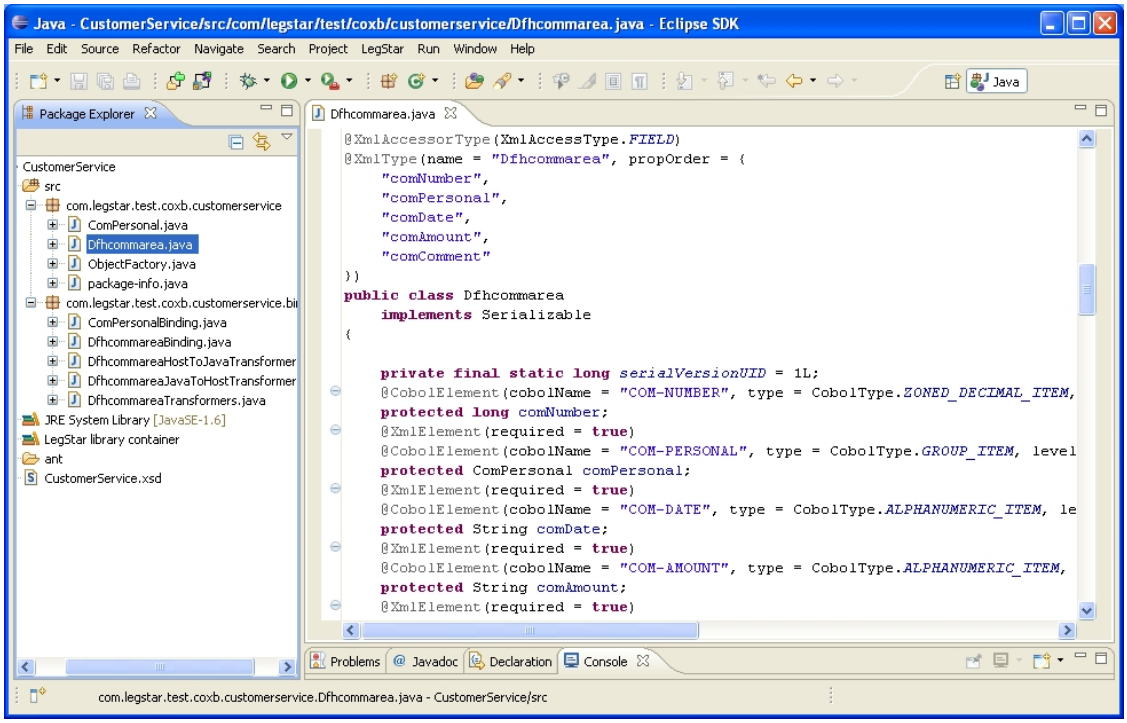
The second options button allows you to specify additional Transformers to be generated such as XML and JSON Transformers.

When you click the finish button, an ant script with a name similar to build-coxb-CustomerService.xsd.xml is generated and launched. There are two different java packages that are generated by the ant script:

- *com.legstar.test.coxb.customerservice* contains JAXB classes as generated by Sun's JAXB XJC utility but with special COBOL annotations as shown on the next screen.

- *com.legstar.test.coxb.customerservice.bind* contains the Transformers classes that can be used for fast marshaling/unmarshaling. Using these classes, there is no need for reflection on the JAXB classes to get the COBOL meta-data at runtime.

Figure 3.3. Adapter COBOL Transformers generation screen results

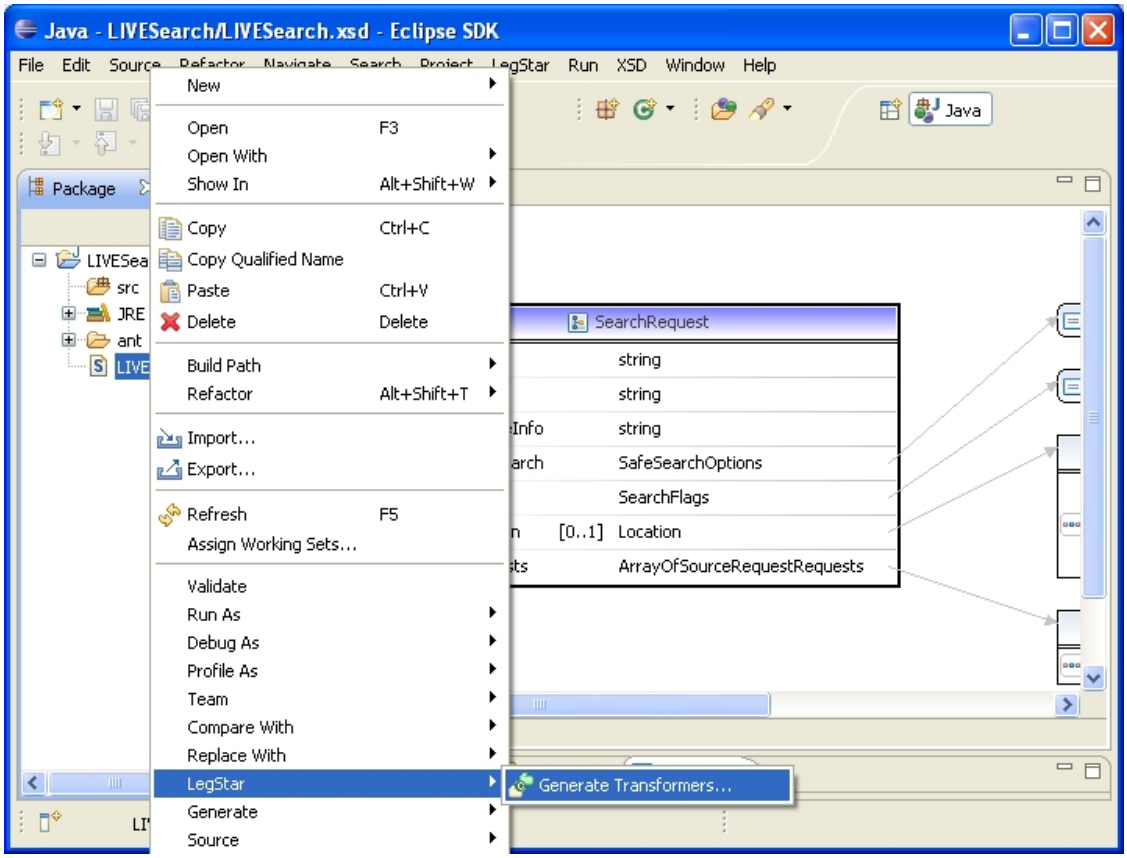


The Using generated tranformers section gives examples of code you could write to run Transformers.

COBOL Transformers Generation for a Proxy

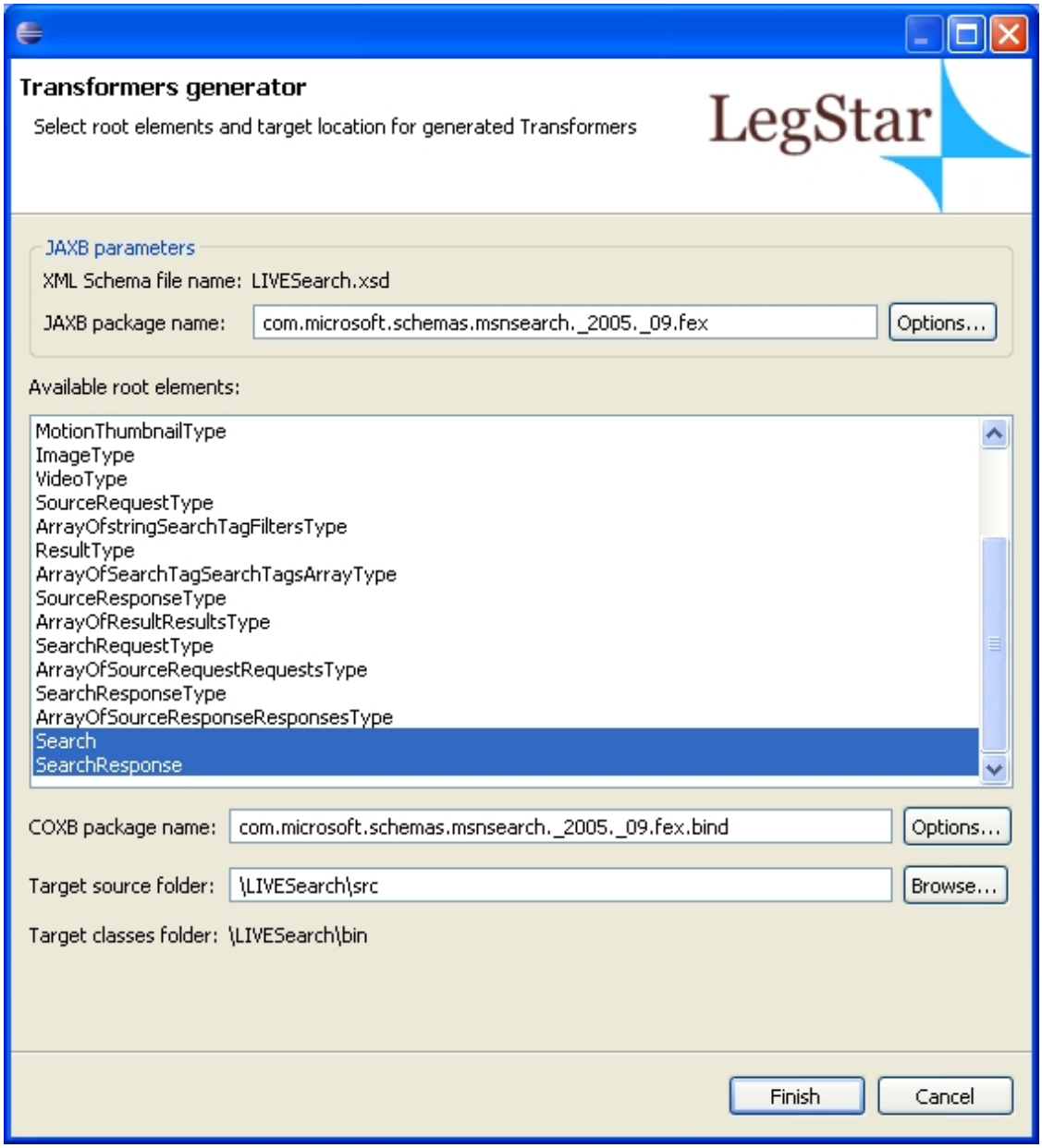
The wizard is started from the package explorer, by right clicking on a previously generated XML Schema and then selecting LegStar->Generate Transformers:

Figure 3.4. Proxy COBOL Transformers generation menu



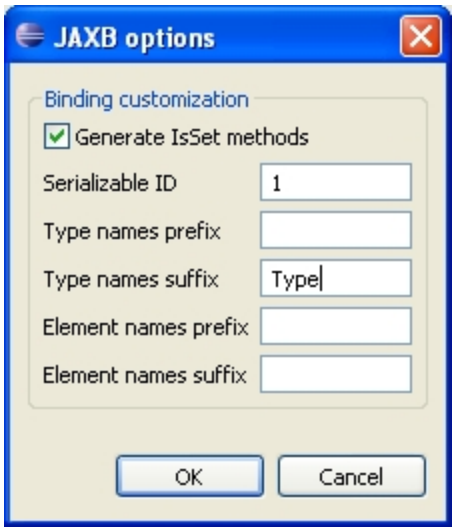
In this case, the root structures we are interested in are Search and SearchResponse, which are the wrapper elements expected and produced by the target Web Service. We select them both.

Figure 3.5. Proxy COBOL Transformers generation screen parameters



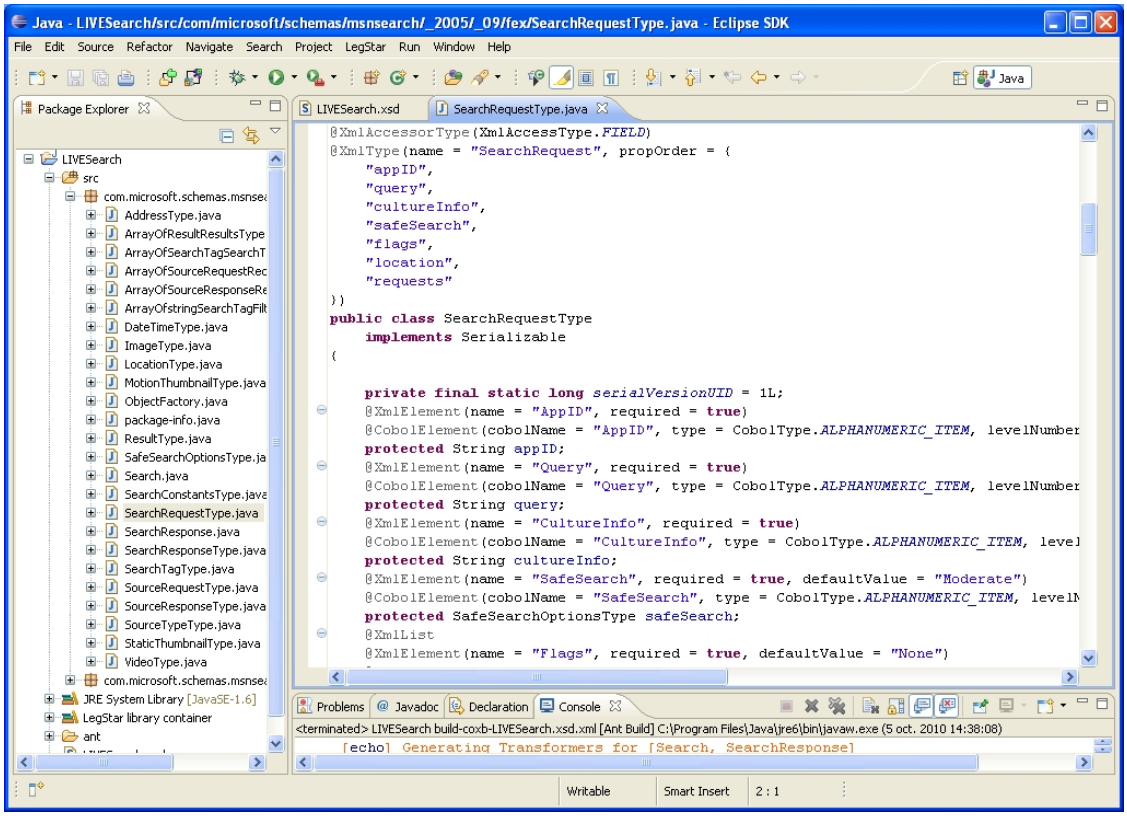
Because Microsoft uses the same names for both Elements and Complex Types, JAXB may complain about name conflicts. To avoid this, you can customize JAXB using the upper options button and specify that all Complex Types should be suffixed with characters "Type":

Figure 3.6. Proxy COBOL Transformers generation screen JAXB options



After you click finish, two Java packages are created, one for JAXB classes with COBOL annotations and one for the optimized Transformers classes.

Figure 3.7. Proxy COBOL Transformers generation screen results



The Using generated tranformers section gives examples of code you could write to run Transformers.

Using generated tranformers

Running Host/Java Transformers

The COBOL Transformers Generator produces a set of java classes that you can easily use to turn mainframe payloads to java data objects.

This is sample code showing how you would use a generated mainframe to java transformer assuming you just generated a transformer class called com.legstar.test.coxb.lsfileae.DfhcommareaTransformers.

```
/**
 * Transform host data and test java data object result.
 *
 * @param hostBytes a byte array holding the mainframe payload
 * @throws HostTransformException if transforming fails
 */
public void hostToJavaTransform(final byte[] hostBytes)
        throws HostTransformException {

    DfhcommareaTransformers transformers = new DfhcommareaTransformers();
    Dfhcommarea dfhcommarea = transformers.toJava(hostBytes);
    assertEquals(100, dfhcommarea.getComNumber());
    assertEquals("TOTO", dfhcommarea.getComPersonal().getComName().trim());
    assertEquals("LABAS STREET", dfhcommarea.getComPersonal()
            .getComAddress().trim());
    assertEquals("88993314", dfhcommarea.getComPersonal().getComPhone()
            .trim());
    assertEquals("100458", dfhcommarea.getComDate().trim());
    assertEquals("00100.35", dfhcommarea.getComAmount().trim());
    assertEquals("A VOIR", dfhcommarea.getComComment().trim());
}
```

Conversely, you would produce a byte array with mainframe data from a java data object with code similar to this:

```
/**
 * Creates a java data object and returns the host data result.
 *
 * @return a byte array holding the mainframe payload
 * @throws HostTransformException if transforming fails
 */
public byte[] javaToHostTransform() throws HostTransformException {
    Dfhcommarea dfhcommarea = new Dfhcommarea();
    dfhcommarea.setComNumber(100L);
    ComPersonal comPersonal = new ComPersonal();
    comPersonal.setComName("TOTO");
    comPersonal.setComAddress("LABAS STREET");
    comPersonal.setComPhone("88993314");
    dfhcommarea.setComPersonal(comPersonal);
    dfhcommarea.setComDate("100458");
    dfhcommarea.setComAmount("00100.35");
```

```
        dfhcommarea.setComComment("A VOIR");
        DfhcommareaTransformers transformers = new DfhcommareaTransformers();
        return transformers.toHost(dfhcommarea);
    }
```

Generated transformers use the default IBM01140 US EBCDIC character set for conversions.

Methods toHost and toJava also accept a character set name as a second parameter if you need to use a different one (just make sure your JRE charsets.jar supports your character set).

Running Host/XML Transformers

In addition to Host/Java transformers, you can generate Host/XML transformers by turning the xmlTransformers generation option on.

Using these transformers, this is sample code to turn host data to XML:

```
    /**
     * Transform host data and test XML result.
     *
     * @param hostBytes a byte array holding the mainframe payload
     * @throws HostTransformException if transforming fails
     */
    public void hostToXmlTransform(final byte[] hostBytes)
            throws HostTransformException {

        DfhcommareaXmlTransformers transformers =
                new DfhcommareaXmlTransformers();
        StringWriter writer = new StringWriter();
        transformers.toXml(hostBytes, writer);
        assertEquals(
                "<?xml version=\"1.0\" encoding=\"UTF-8\" "
                        + "standalone=\"yes\"?>"
                        + "<Dfhcommarea xmlns="
                        + "\"http://legstar.com/test/coxb/lsfileae\">"
                        + "<ComNumber>100</ComNumber>"
                        + "<ComPersonal>"
                        + "<ComName>TOTO</ComName>"
                        + "<ComAddress>LABAS STREET</ComAddress>"
                        + "<ComPhone>88993314</ComPhone>"
                        + "</ComPersonal>"
                        + "<ComDate>100458</ComDate>"
                        + "<ComAmount>00100.35</ComAmount>"
                        + "<ComComment>A VOIR</ComComment>"
                        + "</Dfhcommarea>", writer.toString());
    }
```

This is code to turn XML into host data:

```
    /**
```

```
 * Turns an XML into host data.
 *
 * @return a byte array holding the mainframe payload
 * @throws HostTransformException if transforming fails
 */
public byte[] xmlToHostTransform() throws HostTransformException {
    StringReader reader = new StringReader(
            "<?xml version=\"1.0\" encoding=\"UTF-8\" "
                    + "standalone=\"yes\"?>"
                    + "<Dfhcommarea xmlns="
                    + "\"http://legstar.com/test/coxb/lsfileae\">"
                    + "<ComNumber>100</ComNumber>"
                    + "<ComPersonal>"
                    + "<ComName>TOTO</ComName>"
                    + "<ComAddress>LABAS STREET</ComAddress>"
                    + "<ComPhone>88993314</ComPhone>"
                    + "</ComPersonal>"
                    + "<ComDate>100458</ComDate>"
                    + "<ComAmount>00100.35</ComAmount>"
                    + "<ComComment>A VOIR</ComComment>"
                    + "</Dfhcommarea>");
    DfhcommareaXmlTransformers transformers =
            new DfhcommareaXmlTransformers();
    return transformers.toHost(new StreamSource(reader));
}
```

Running Host/JSON Transformers

In addition to Host/Java transformers, you can generate Host/JSON transformers by turning the jsonTransformers generation option on.

Using these transformers, this is sample code to turn host data to JSON:

```
/**
 * Transform host data and test JSON result.
 *
 * @param hostBytes a byte array holding the mainframe payload
 * @throws HostTransformException if transforming fails
 */
public void hostToJsonTransform(final byte[] hostBytes)
        throws HostTransformException {

    DfhcommareaJsonTransformers transformers =
            new DfhcommareaJsonTransformers();
    StringWriter writer = new StringWriter();
    transformers.toJson(hostBytes, writer);
    assertEquals("{\"ComNumber\":100,"
            + "\"ComPersonal\":"
            + "{\"ComName\":\"TOTO\","
            + "\"ComAddress\":\"LABAS STREET\","
            + "\"ComPhone\":\"88993314\"},"
            + "\"ComDate\":\"100458\","
```

```
                + "\"ComAmount\":\"00100.35\","
                + "\"ComComment\":\"A VOIR\"}",
                writer.toString());
    }
```

This is code to turn JSON into host data:

```
    /**
     * Turns JSON into host data.
     *
     * @return a byte array holding the mainframe payload
     * @throws HostTransformException if transforming fails
     */
    public byte[] jsonToHostTransform() throws HostTransformException {
        StringReader reader = new StringReader(
                "{\"ComNumber\":100,"
                        + "\"ComPersonal\":"
                        + "{\"ComName\":\"TOTO\","
                        + "\"ComAddress\":\"LABAS STREET\","
                        + "\"ComPhone\":\"88993314\"},"
                        + "\"ComDate\":\"100458\","
                        + "\"ComAmount\":\"00100.35\","
                        + "\"ComComment\":\"A VOIR\"}");
        DfhcommareaJsonTransformers transformers =
                new DfhcommareaJsonTransformers();
        return transformers.toHost(reader);
    }
```