

Migrating Java CAPS 5/6 Assets to Oracle SOA Suite 11g

HL7 JCD to Document Callout Migration

michal@czapski.id.au, February 2013, Rev 1.0

Table of Contents

Introduction.....	1
Prerequisites and Assumptions.....	2
Migration Process	3
Anatomy of a Java CAPS HL7 Solution.....	3
Analysing HL7 JCD Solution	4
Analysing HL7 processing JCD.....	6
Analysing Enterprise Archive	6
Building and exercising a Stand-alone Java Application.....	10
Dependency Issues.....	20
Basic SOA Suite for healthcare integration Solution.....	21
Stage I: Implement End-to-End HL7 Passthrough with JMS	22
Stage II: Develop Java Callout with JCD code.....	22
Stage III: Create and Populate File System Directories.....	30
Stage IV: Restart WebLogic Server	30
Stage V: Set the location of callouts directory	30
Stage VI: Configure the callout using the callout JAR	31
Stage VII: Configure the callout in the inbound Endpoint	31
Stage VIII: Add HL7 v2.4 ADT A01 Document.....	32
Stage IX: Make HL7JMSOut endpoint use the HL7 v2.4 ADT A01.....	33
Stage X: Modify JMSPassThrough project to support HL7 v2.4 ADT A01	34
Stage XI: Exercise the solution	34
Summary	35
Appendix - HL702Transformer JCD Source	36

Introduction

This article leverages discoveries from the 2+ year old article “Migrating Java CAPS 5/6 Assets to Oracle SOA Suite 11g – HL7 JCD to Spring Component Migration”, at <http://blogs.czapski.id.au/2010/10/migrating-java-caps-56-assets-to-oracle-soa-suite-11g-hl7-jcd-to-spring-component-migration>, and presents a different method of recusing “HL7 Transformer JCD” code in a SOA or OSB solution. This method uses SOA Suite for healthcare integration, slightly modified Java code for a “HL7 Transformer JCD” and WebLogic JMS. The Java CAPS part of the discussion is substantially the same, except the JCD code is wrapped in a SOA Suite for healthcare integration Callout, rather than being converted to a Spring Component embedded in a SOA Campsite. The “HL7 Adapter” part of the article is different in that SOA Suite for healthcare

integration is used instead of the SOA Suite B2B HL7 functionality.

This article is of potential interest to these Sun/SeeBeyond customers who have an investment in moderate and large Java Collaboration Definition-based transformation and mapping rules, and who are looking for ways to reuse as much as possible of the Java code involved, when migrating to the Oracle SOA Suite or the Oracle Service Bus. The example developed in this article comes from the healthcare domain and uses the HL7 OTDs (Object Type Definitions). This is a deliberate choice because all but the most trivial HL7 transformations will involve hundreds of lines of Java code, therefore are a good candidates for migration. The adapter part of the HL7 solution is provided by the SOA Suite for healthcare integration. This makes the method domain-specific.

Discussion in this article addresses a subset of technologies available in the Java CAPS and in the SOA Suite for healthcare integration. Specifically, the Java Collaboration Definitions supported in Java CAPS 5.x and in Java CAPS 6/Repository, and the SOA Suite for healthcare integration (part of the SOA Suite 11g R1 PS5).

There is no discussion pertaining to JBI-based technologies or Java CAPS BPEL-based technologies. There is no discussion about other ways in which Java logic can be deployed as part of an Oracle SOA Suite solution.

The HL7 eWay and JCD based Java CAPS solution will be ported to the Oracle SOA Suite for healthcare integration and Java-based environment. HL7 Adapters will be replaced with the HL7 endpoints provided by the Oracle SOA Suite for healthcare integration infrastructure. What minimal routing is used will be provided by the Mediator component via JMS Queues. Transformation logic will be ported to the Java POJO (Plain Old Java Object) and will be embedded in the endpoint as a document callout.

This article walks through the process of “extracting” JCD source and related archives from Java CAPS, developing a stand-alone Java application which uses the JCD source, encapsulating JCD source in a SOA Suite for healthcare integration callout and finally reproducing Java CAPS HL7 solution functionality in an equivalent SOA Suite for healthcare integration solution.

Prerequisites and Assumptions

It is assumed that the reader is thoroughly familiar with Java CAPS and somewhat familiar with HL7 support in Java CAPS. This knowledge is assumed.

It is assumed that the reader has access to the JCD source and the built EAR file. If not, that a Java CAPS 6/Repository installation with appropriate libraries is available for building the HL7 project.

It is assumed that the reader has the SOA Suite for healthcare integration installation with the requisite software.

It is also assumed that the reader has at least a modest familiarity with the SOA Suite for healthcare integration, perhaps as a result of following a series of articles on the topic I published on my blog: <http://blogs.czapski.id.au/tag/soa-suite-for-healthcare-integration>.

This is reasonably advanced material so other implicit assumptions may have crept in.

The project export of the Java CAPS 6/Repository project discussed in this article, containing the Java CAPS Environment I used in the example, is available at http://blogs.czapski.id.au/wp-content/uploads/2010/09/HL7Transformer_jcaps6_project_export.zip.

I am not in a position to provide the built EAR file so you will need a Java CAPS 6.x environment to import this project and build your own EAR file, or you will need to have a Java CAPS 5 environment and re-create the project from scratch, perhaps using the JCD source available in the archive at <http://blogs.czapski.id.au/wp-content/uploads/2010/09/jcdHL702Transformer.zip>.

Migration Process

At the end of this article we will have migrated a Java CAPS 6/Repository-based HL7 transformation project to the Oracle SOA Suite for healthcare integration, using the Java Callout to preserve HL7 mapping and transformation rules and using the Oracle SOA Suite for healthcare integration to replicate the HL7 eWay functionality. To add structure to the discussion let's outline what steps will be followed to accomplish the objective. The specifics of the steps will follow.

1. Develop, build and test the Java CAPS 6/Repository HL7 solution (this will have been done by the time we get around to looking at migration)
2. Analyse components of the solution to determine if JCDs are good candidates for migration to a Java POJO, hence to a Java Callout (here I assume that the answer is yes and that the JCD to migrate has been identified – in fact I provide the JCD and use it in the process)
3. Obtain and unpack the Enterprise Archive (EAR file) containing the compiled JCD and related JAR files
4. Obtain the JCD Java source
5. Create a Java Project using Oracle JDeveloper
6. Add JARs to the Java project
7. Create, build and exercise the Stand-alone Java application
8. Create a Java Callout project
9. Identify and copy to Java Callout project all Java CAPS JARs needed by the JCD code which will be migrated
10. Create the Java Class that implements the Callout and migrate JCD code to it
11. Create a JAR Profile for the Java Callout
12. Build the Callout JAR
13. Copy all JARs to the directory in the WebLogic classpath and restart WebLogic Server
14. Develop a pass-through SOA Suite Mediator project to pass messages for the inbound JMS Queue to the outbound JMS Queue and deploy
15. Configure SOA Suite for healthcare integration endpoints for inbound and outbound messaging
16. Test/Exercise the solution

There are quite a few steps but then there would be a similar number of steps if we were to enumerate steps involved in developing an equivalent Java CAPS HL7 solution. It is also worth mentioning that most of these steps a simple and short in duration.

Anatomy of a Java CAPS HL7 Solution

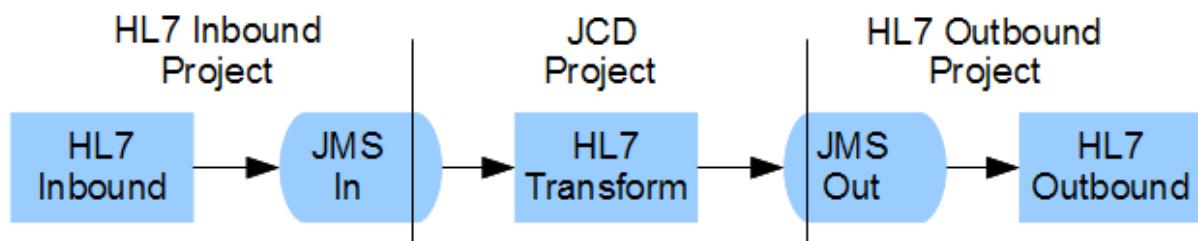
No Java Collaboration Definition stands alone. Each is triggered to process a message and each, in turn, may trigger other components, typically causing delivery of a message to/through one or more outbound connectors/adapters. Functionality provided by the set of Java CAPS connectors/adapters (in Java CAPS 5 and Java CAPS 6/Repository called eWays or eWay Adapters) overlaps to a significant degree with the functionality offered by Oracle and Oracle-certified third-party adapters. Having said that I must immediately point out that to my knowledge no Java CAPS adapters/eWays are either certified or supported for use with the Oracle SOA Suite. This means that the adapter-specific code in JCDs will have to be rewritten or remove as part of the migration effort. This also means that JCDs in which adapter code is a significant part of the JCD may not be good candidates for migration since the effort involved in rewriting adapter-specific code will likely exceed the saving arising from migrating the transformation and mapping code. Good candidates are JCDs

involved in transforming standards-based message structures, such as HL7, X12, EDIFACT, and similar. This is one of the reasons I chose HL7 as the messaging standard for the example in this article.

A HL7 processing solution in Java CAPS will typically receive HL7 v2 Delimited messages through the HL7 eWay, transform them in some way, and potentially send them on to HL7 receivers through a HL7 eWay. Standard HL7 processing, acknowledgements, message header validation, sequence number processing, are handled by pre-built Java CAPS projects, which are available as part of the installation and must be imported and potentially modified for use in a solution. The inbound project, prjHL7Inbound, receives HL7 messages and deposits them in a JMS Queue for a downstream solution to process further. The outbound project, prjHL7Outbound, receives HL7 messages from a JMS Queue and sends them on to the receivers. The site specific transformations and message processing happens in one or more components, the initial of which receives messages from the JMS Queue to which the HL7 Inbound sent them, and the final of which ultimately deposits messages in a JMS Queue for the HL7 outbound to send. The complexity involved in transformational of messages, access to various enterprise resources and message manipulation will vary from solution to solution.

Analysing HL7 JCD Solution

The schematic below shows major components involved in a typical Java CAPS HL7 solution described above.



Drawing 1: Simplest JCD-based HL7 Solution

If we construct a Java CAPS solution in such a way that the HL7 Inbound, the HL7 Transform and the HL7 Outbound are implemented as separate Java CAPS projects we might get a project hierarchy like the one shown below.

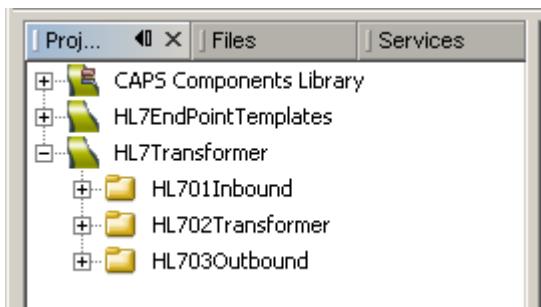


Illustration 1: HL7 Transformer Project Heirarchy

The HL7 Inbound Connectivity Map for the HL701Inbound, which is derived from prjHL7Inbound, is shown below.

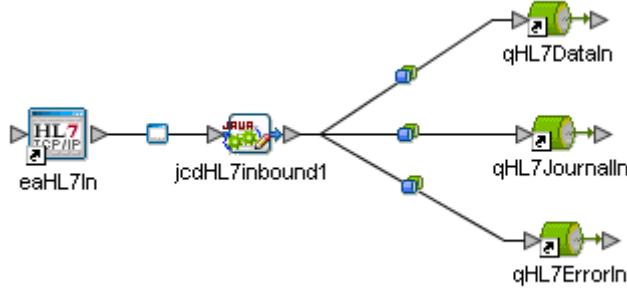


Illustration 2: HL7 Inbound Connectivity Map

The HL7 eWay receives messages and deposits them in the JMS Queue called qHL7DataIn. The JCD itself is the unmodified JCD imported with the project prjHL7Inbound. It handles all HL7-related communication functionality including ACKs.

The Connectivity Map for the HL703Outbound is shown below.

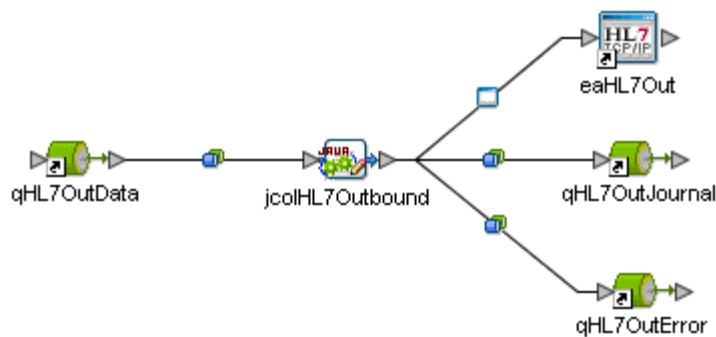


Illustration 3: HL7 Outbound Connectivity Map

The HL7 eWay sends messages it reads from the JMS Queue called qHL7OutData. The JCD itself is the unmodified JCD imported with the project prjHL7Outbound. It handles all HL7-related communication functionality including ACKs.

The connectivity map for the HL702Transformer project is simplicity itself and requires no elaboration.



Illustration 4: HL7 Transformer Connectivity Map

The collaboration receives a message from qHL7DataIn, transforms it in some manner and deposits the resulting message in qHL7OutData.

To anticipate what will follow let's say upfront that the HL701Inbound and the HL703Outbound projects will be replaced, in their entirety, by the Oracle SOA Suite for healthcare integration infrastructure with correctly configured endpoints, therefore there will be no further discussion of these projects. The HL702Transformer project's JCD, jcdHL702Transformer, will be migrated to the Java Callout to be invoked by the SOA Suite for healthcare integration inbound endpoint, therefore it will be analysed in detail.

```

package HL7TransformerHL702Transformer;

public class jcdHL702Transformer
{
    public com.stc.codegen.logger.Logger logger;
    public com.stc.codegen.alerter.Alerter alerter;
    public com.stc.codegen.util.CollaborationContext collabContext;
    public com.stc.codegen.util.TypeConverter typeConverter;

    public void receive
        ( com.stc.connectors.jms.Message input
        , com.stc.SeeBeyond.OTD_Library.HL7.X_2_3.HL7_23_ADT_A01.ADT_A01 vA01_23
        , com.stc.SeeBeyond.OTD_Library.HL7.X_2_4.HL7_24_ADT_A04.ADT_A04 vA04_24
        , com.stc.connectors.jms.JMS vJMSOut )
    throws Throwable
    {
        vA01_23.unmarshalFromString( input.getTextMessage() );

        /*
         * HL7 v2.3 to HL7 v2.4 transformation rules here
         ...
        */

        String sA04Out = vA04_24.marshalToString();

        com.stc.connectors.jms.Message vJMSMsg = vJMSOut.createTextMessage();
        vJMSMsg.setTextMessage( sA04Out );
        vJMSOut.sendText( sA04Out );
    }
}

```

Text 1: Abbreviated jcdHL702Transformer JCD

Analysing HL7 processing JCD

Let's now analyse what the JCD does and how it goes about doing it.

The structure of the JCD, omitting the actual HL7 transformation rules for the moment, is shown below.

This JCD is a Plain Old Java Object (POJO). It is invoked by the Java CAPS-generated wrapper Stateless Session Bean, which sets up all the adapter connectivity infrastructure and object instances, and invokes the JCD with objects instantiated and input structure (in this case the JMS OTD object) populated.

It is, hopefully, obvious that the JCD extracts the HL7 v2 Delimited message from the JMS input object and unmarshals it into the HL7 v2.3 ADT A01 structure. Once the manipulation, omitted in Text 1, is completed, the HL7 v2.4 ADT A04 is marshalled to String and send as a Text Message.

OTDs input (Message object), output (JMS object), vA01_23 (ADT_A01 object) and vA04_24 (ADT_A04 object) are instantiated on entry into this JCD. The 300+ lines on Java, omitted in Text 1, copy data from various fields on the vA01_23 structure to appropriate fields in the vA04_24 structure.

Analysing Enterprise Archive

Building this project using eDesigner/NetBeans produces an Enterprise Archive (EAR) file which contains all the runtime artefacts necessary to execute the solution. Typically the EAR file is stored

in a file system hierarchy parts of which are named according to the names of the projects and deployment profiles. Java CAPS 6/Repository would deposit the EAR file in a directory hierarchy like that shown below.

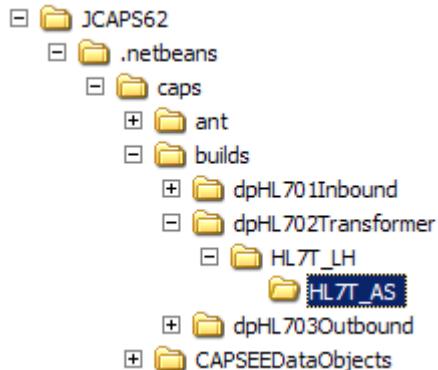


Illustration 5: HL702Transformer EAR directory

The outer directory, dpHL702Transformer, is named after the Deployment Profile name in the project. The names of the inner two directories correspond to the names of the corresponding objects in the Java CAPS Environment – HL7T_LH (Logical Host) and HL7T_AS (Application Server/Integration Server). This may be slightly different in Java CAPS 5.x. For comparison the Java CAPS Environment is reproduced below.

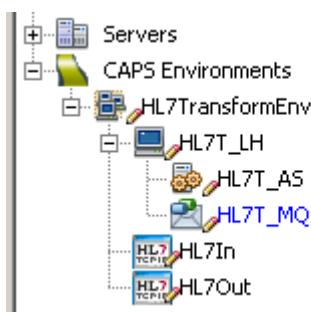


Illustration 6: Java CAPS Environment

Inspection of the content of the ./builds/dpHL702Transformer/HL7T_LH/HL7T_AS reveals the EAR file, dpHL702Transformer.ear.

Address		C:\JCAPS62\.netbeans\caps\builds\dpHL702Transformer\HL7T_LH\HL7T_AS			
Folders		Name	Size	Type	Date Modified
Desktop	My Documents	dpHL702Transformer.ear	12,612 KB	EAR File	19/09/2010 9:02 PM

Illustration 7: EAR file, dpHL702Transformer.ear

Using 7-zip, jar, or another tools which opens JAR archives, inspect the content of this file. In my case, and if you use and build the same project as I did you will see the same, the following files are present in the archive:

```

CMHL702Transformer_jcdHL702Transformer1.jar
CMHL702Transformer_jcdHL702Transformer1_507672842.jar
CMHL702Transformer_jcdHL702Transformer1_507672842.rar
com-stc-configuration.jar
com-stc-dtapi.jar
com-stc-einsightintegrationengineapi.jar
com-stc-JMSOTD.jar
  
```

com-stc-log4j.jar
com-stc-otd-udlimpl.jar
com-stc-util.jar
com-sun-org-apache-commons-jxpath.jar
com.stccodegen.alerterapi.jar
com.stccodegen.alerterimpl.jar
com.stccodegen.loggerapi.jar
com.stccodegen.loggerimpl.jar
com.stccodegen.utilapi.jar
com.stccodegen.utilimpl.jar
com.stccodegenapi.jar
com.stccodegenbeans.jar
com.stccodegenmetadataimpl.jar
com.stccodegenrtimpl.jar
com.stc.icu4j.jar
com.stc.jcecodegenimpl.jar
com.stc.jmscodegenimpl.jar
com.stc.jmsjca.core.jar
com.stc.jmsjca.rasunone.jar
com.stc.jmsmx.core.jar
com.stc.jmsmx.sjsmq.jar
com.stc.otd.fwrunapi.jar
com.stc.otdcodegenimpl.jar
com.stc.util.encodingconverter.jar
commons-beanutils-1.6.jar
commons-logging-1.1.jar
concurrent-1.3.1.jar
em_config.jar
META-INF/
META-INF/application.xml
META-INF/MANIFEST.MF
META-INF/sun-application.xml
qHL7DataIn_CMHL702Transformer_jcdHL702Tr2000592805.jar
qHL7DataIn_CMHL702Transformer_jcdHL702Tr2000592805.rar
runtime_properties.jar
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_ACC.jar
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_ADT_A01.jar
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_AL1.jar
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_DB1.jar
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_DG1.jar
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_DRG.jar
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_EVN.jar
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_GT1.jar
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_IN1.jar
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_IN2.jar
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_IN3.jar
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_MSH.jar
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_NK1.jar
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_OBX.jar
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_PD1.jar
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_PID.jar
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_PR1.jar
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_PV1.jar
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_PV2.jar
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_ROL.jar
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_UB1.jar
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_UB2.jar
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_ACC.jar
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_ADT_A04.jar
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_AL1.jar
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_DB1.jar
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_DG1.jar
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_DRG.jar
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_EVN.jar
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_GT1.jar
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_IN1.jar
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_IN2.jar
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_IN3.jar
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_MSH.jar
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_NK1.jar
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_OBX.jar

```
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_PD1.jar  
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_PDA.jar  
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_PID.jar  
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_PR1.jar  
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_PV1.jar  
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_PV2.jar  
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_ROL.jar  
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_UB1.jar  
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_UB2.jar  
StartUpConnector.jar  
StartUpConnector.rar  
xerces-2.8.0.jar
```

Broadly, there are 4 kinds of files in the archive. The project-specific code-generated files:

```
CMHL702Transformer_jcdHL702Transformer1.jar  
CMHL702Transformer_jcdHL702Transformer1_507672842.jar  
CMHL702Transformer_jcdHL702Transformer1_507672842.rar  
qHL7DataIn_CMHL702Transformer_jcdHL702Tr2000592805.jar  
qHL7DataIn_CMHL702Transformer_jcdHL702Tr2000592805.rar
```

The utility archives, provided by either Sun or third-parties:

```
commons-beanutils-1.6.jar  
commons-logging-1.1.jar  
concurrent-1.3.1.jar  
xerces-2.8.0.jar
```

The Sun/SeBeyond/STC Java CAPS-specific proprietary utility archives:

```
com-stc-configuration.jar  
com-stc-dtapi.jar  
com-stc-einsightintegrationengineapi.jar  
com-stc-JMSOTD.jar  
com-stc-log4j.jar  
com-stc-otd-udlimpl.jar  
com-stc-util.jar  
com-sun-org-apache-commons-jxpath.jar  
com.stccodegen.alerterapi.jar  
com.stccodegen.alerterimpl.jar  
com.stccodegen.loggerapi.jar  
com.stccodegen.loggerimpl.jar  
com.stccodegen.utilapi.jar  
com.stccodegen.utilimpl.jar  
com.stccodegenapi.jar  
com.stc_codegenbeans.jar  
com.stc_codegenmetadataimpl.jar  
com.stc_codegenrtimpl.jar  
com.stc.icu4j.jar  
com.stc.jcecodegenimpl.jar  
com.stc.jmscodegenimpl.jar  
com.stc.jmsjca.core.jar  
com.stc.jmsjca.rasunone.jar  
com.stc.jmsmx.core.jar  
com.stc.jmsmx.sjsmq.jar  
com.stc.otd.frunapi.jar  
com.stc.otdcodegenimpl.jar  
com.stc.util.encodingconverter.jar  
em_config.jar  
StartUpConnector.jar  
StartUpConnector.rar  
runtime_properties.jar
```

The HL7 OTD message structure-specific archives will vary in number and composition depending on the HL7 OTD libraries that the particular project uses.

```
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_ACC.jar
```

SeeBeyond_OTD_Library_HL7_2.3_HL7_23_ADT_A01.jar
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_AL1.jar
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_DB1.jar
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_DG1.jar
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_DRG.jar
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_EVN.jar
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_GT1.jar
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_IN1.jar
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_IN2.jar
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_IN3.jar
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_MSH.jar
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_NK1.jar
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_OBX.jar
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_PD1.jar
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_PID.jar
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_PR1.jar
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_PV1.jar
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_PV2.jar
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_ROL.jar
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_UB1.jar
SeeBeyond_OTD_Library_HL7_2.3_HL7_23_UB2.jar
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_ACC.jar
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_ADT_A04.jar
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_AL1.jar
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_DB1.jar
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_DG1.jar
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_DRG.jar
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_EVN.jar
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_GT1.jar
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_IN1.jar
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_IN2.jar
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_IN3.jar
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_MSH.jar
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_NK1.jar
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_OBX.jar
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_PD1.jar
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_PDA.jar
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_PID.jar
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_PR1.jar
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_PV1.jar
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_PV2.jar
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_ROL.jar
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_UB1.jar
SeeBeyond_OTD_Library_HL7_2.4_HL7_24_UB2.jar

People familiar with HL7 will instantly recognise parts of archive names like PID, UB2 and so on, which correspond to HL7 v2.x segment names.

We will need selected JARs from the EAR for use in the stand-alone Java application and the migrated Spring Component. There are some obvious candidates, the SeeBeyond_OTD_Library_HL7*.jar are obvious for HL7 support. Others are not so obvious and may vary from project to project. I happen to know that I also need com.stc.otd.fwrunapi.jar and com-stc-otd-udlimpl.jar. You may need to test using the standalone Java class we will develop later, to identify all dependencies.

Building and exercising a Stand-alone Java Application

To determine what is needed at runtime, and to clean up the JCD code before we get around to working with the Java Callout, we can attempt to create a stand-alone Java application that will implement the transformation. If this task cannot be accomplished than it is unlikely that the migration effort will succeed. This section walks through the process.

Let's create a JDeveloper "Generic Application" application, let's say HL7TransformApp, and a Java Project within it, let's call it HL7Transform using Java technology.

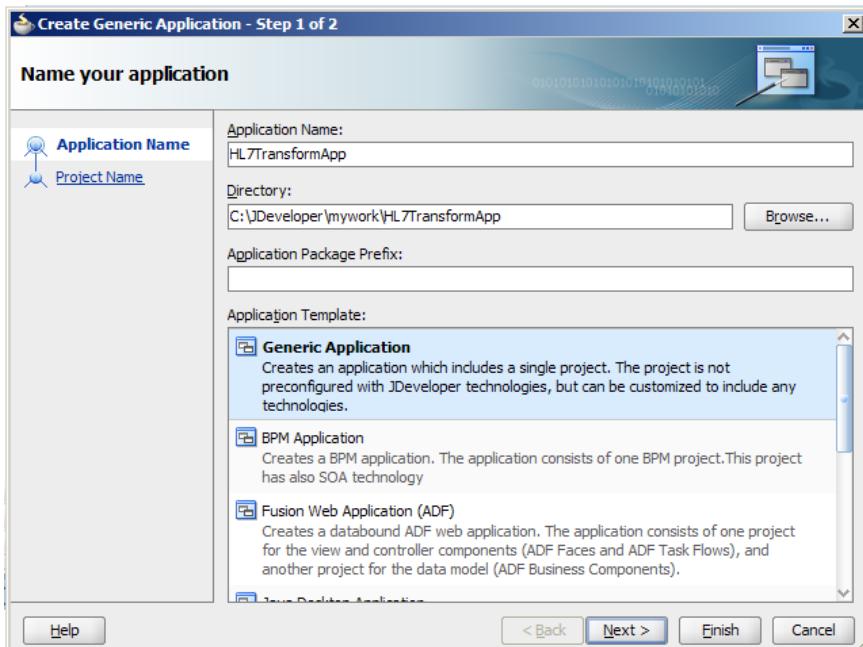


Illustration 8: Create a Generic Application

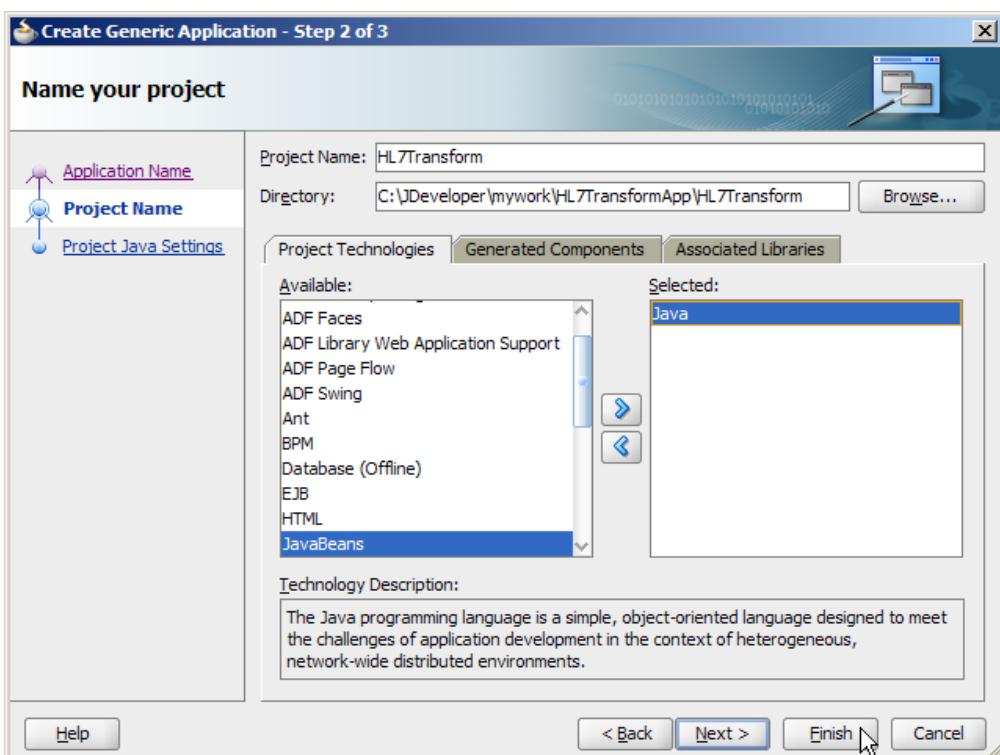


Illustration 9: Create a Java Project

Specify “hl7transform” as the default package name and “Finish”.

This will create a directory hierarchy that looks similar to that shown in the illustration.

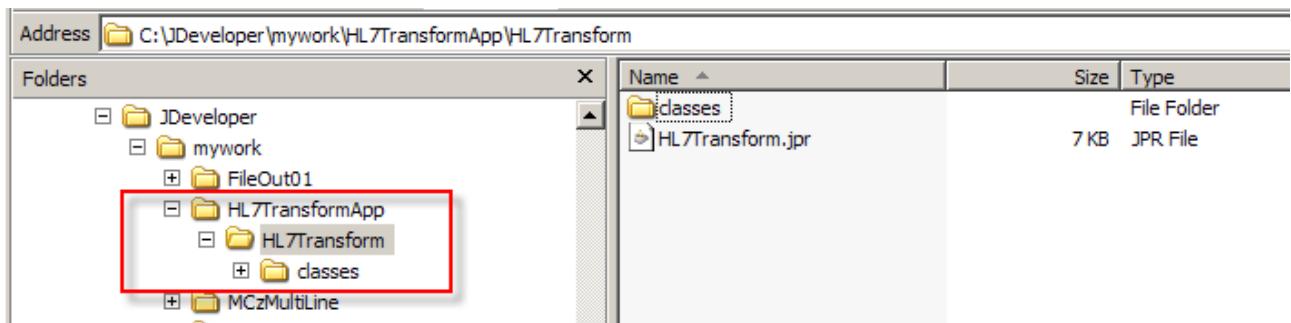


Illustration 10: Empty Java project directory hierarchy

To add JARs we need to support the stand-alone version of the JCD we need to create a lib directory. Let's do that in such a way that the lib directory appears under the HL7Transform directory.

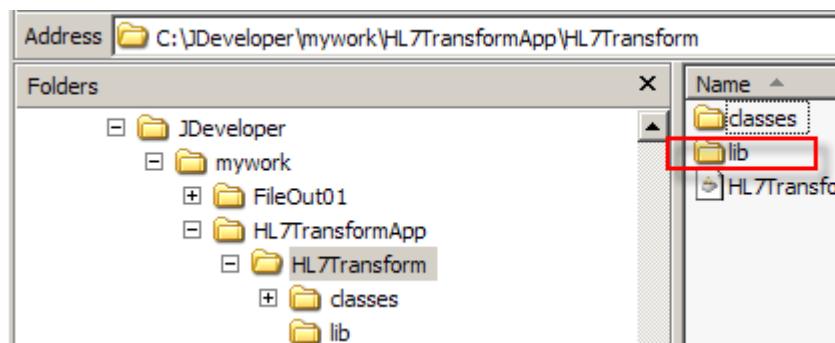


Illustration 11: New lib directory

Let's now copy the SeeBeyond_OTD_HL7_Library*.jar, the com.stc.otd.fwrunapi.jar and the com-stc-otd-ud1impl.jar JARs to this lib directory from wherever we extracted the EAR file to.

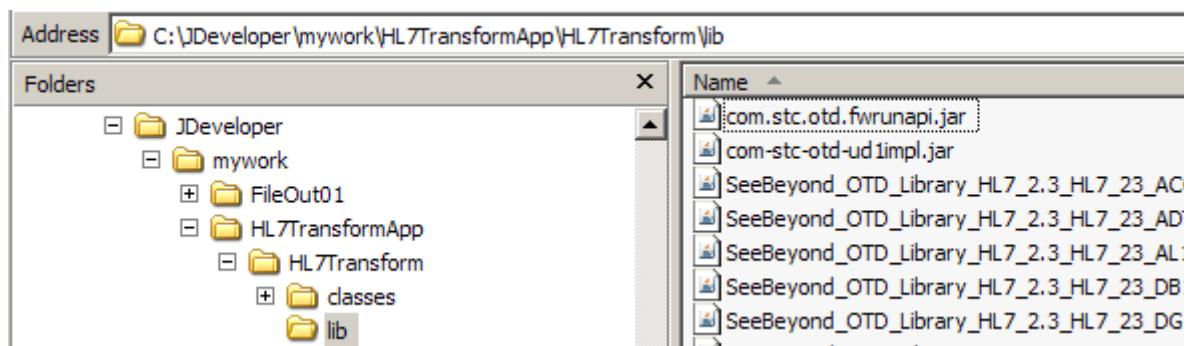


Illustration 12: Populated lib directory

Back in JDeveloper, right-click on the name of the project and choose Project Properties. Select “Libraries and Classpath”, click “Add JAR/Directory”, navigate to HL7TransformApp/HL7Transform/lib and select all JARs you copied to that lib directory. Click Select.

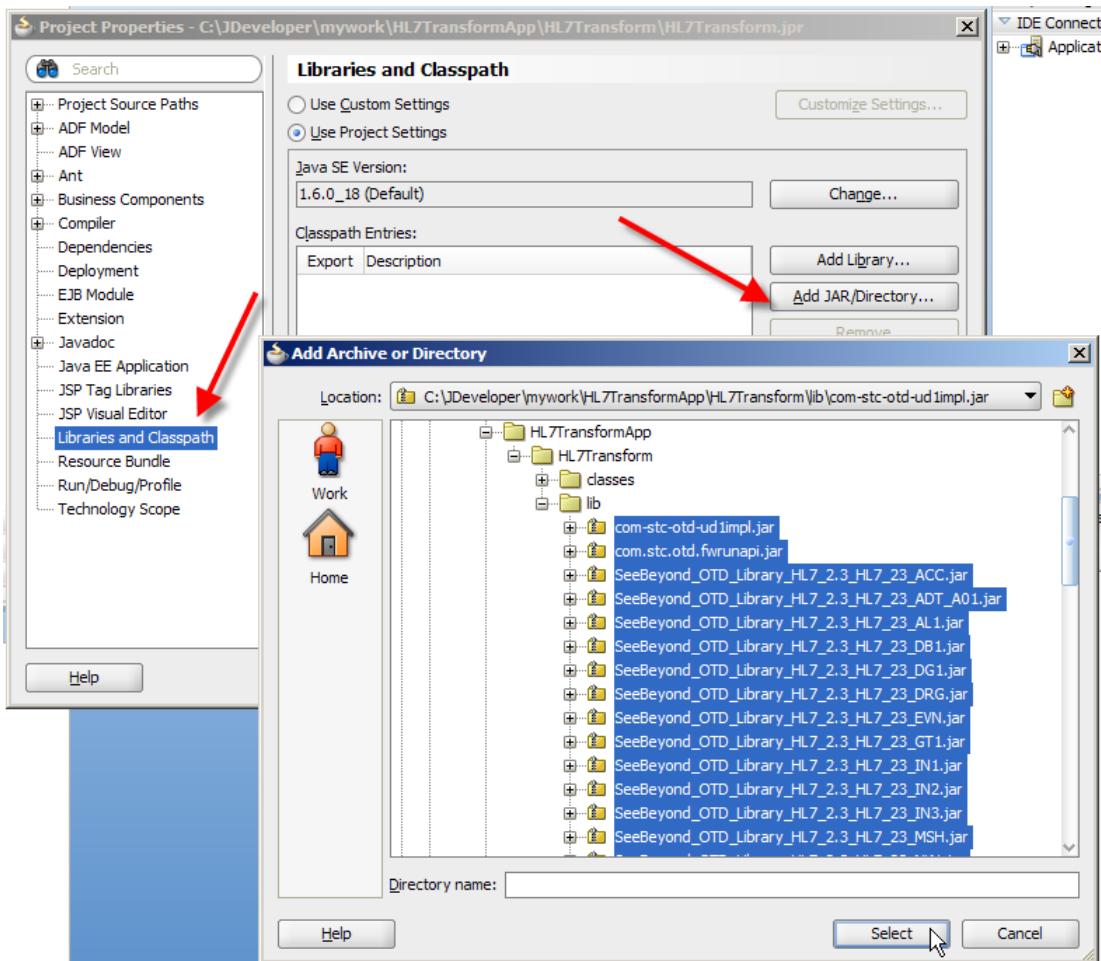


Illustration 13: Select all JARs added to the project

Click OK to dismiss the dialogue box.

Right-click on the name of the project and choose New → General → Java → Java Class then click OK.

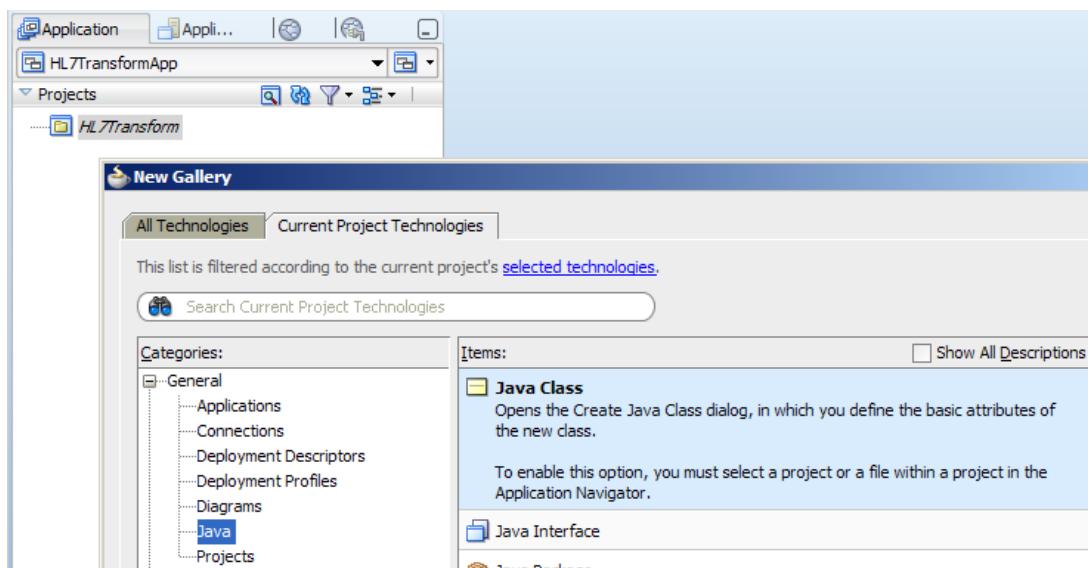


Illustration 14: Create new Java Class

Name this new class HL7Transform and click OK.

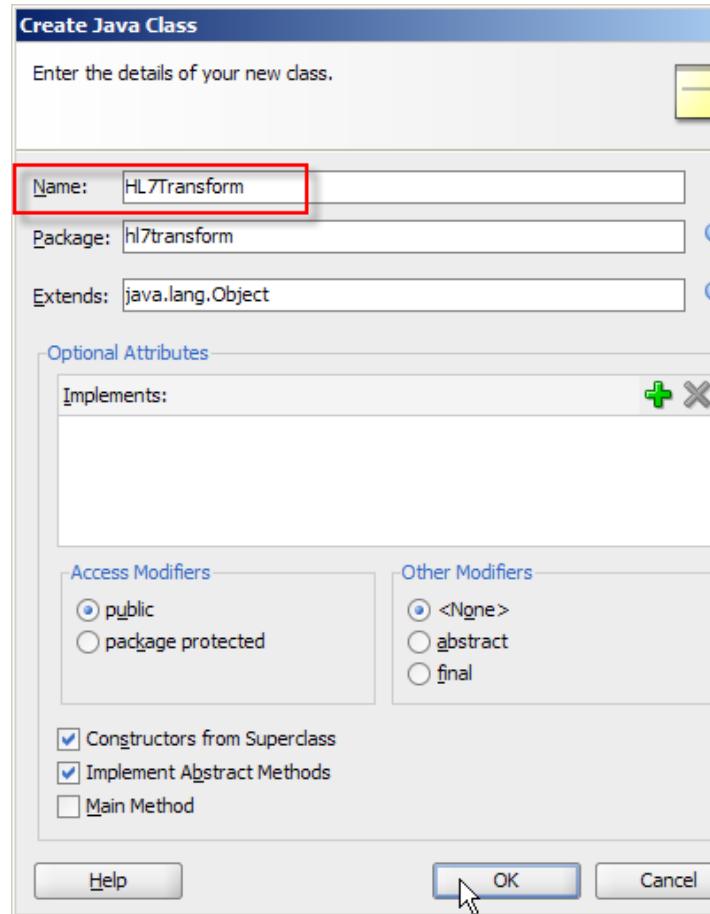


Illustration 15: Name new Java Class

The following skeleton will be created.

```
package hl7transform;

public class HL7Transform {
    public HL7Transform() {
        super();
    }
}
```

Let's copy the entire receive method from the JCD and paste it into the new class source following the default constructor. The figure below abbreviates the transformation code. Note the problem areas, highlighted in the figure.

```

package hl7transform;

public class HL7Transform {
    public HL7Transform() {
        super();
    }

    public void receive
        ( com.stc.connectors.jms.Message input
        , com.stc.SeeBeyond.OTD_Library.HL7.X_2_3.HL7_23_ADT_A01.ADT_A01 vA01_23
        , com.stc.SeeBeyond.OTD_Library.HL7.X_2_4.HL7_24_ADT_A04.ADT_A04 vA04_24
        , com.stc.connectors.jms.JMS vJMSOut )
        throws Throwable
    {
        vA01_23.unmarshalFromString( input.getTextMessage() );

        /*
         * HL7 v2.3 to HL7 v2.4 transformation rules here
         */
        ...

        String sA04Out = vA04_24.marshalToString();

        com.stc.connectors.jms.Message vJMSMsg = vJMSOut.createTextMessage();
        vJMSMsg.setTextMessage( sA04Out );
        vJMSOut.sendText( sA04Out );
    }
}

```

Illustration 16: JCD "receive" method source pasted into JDeveloper Java Class

We need to remove offending lines, which refer to the JMS adapter. We will have to populate the vA01_23 variable before invoking the “receive” method and will have to do something sensible with the content of the vA04_24 variable outside the “receive” method as well, including initial creation of these variables.

The resulting Java source is shown below.

```

package hl7transform;

public class HL7Transform {
    public HL7Transform() {
        super();
    }

    public void receive
        ( com.stc.SeeBeyond.OTD_Library.HL7.X_2_3.HL7_23_ADT_A01.ADT_A01 vA01_23
        , com.stc.SeeBeyond.OTD_Library.HL7.X_2_4.HL7_24_ADT_A04.ADT_A04 vA04_24)
        throws Throwable
    {
        /*
         * HL7 v2.3 to HL7 v2.4 transformation rules here
         *
        */
    }
}

```

Illustration 17: Java source after removal of JMS Adapter dependencies

As you undoubtedly realise I could have written the method signature differently. To maintain continuity and make it easier to follow I am keeping it as close as possible to the original JCD “receive” method signature.

Note that so far I omitted the transformation code completely since the objective is to transcribe it verbatim to the new class. By doing this I am concentrating on essential modifications that must be done to make this Java application work outside the Java CAPS environment.

Now we need to add code to create instances of the ADT_A01 and ADT_A04 classes and populate the instance of the ADT_A01 class with a HL7 message body so that our “receive” method has something to work on. Let's create a new method HL7Transform, which accepts a String, presumably containing the HL7 v2.3 ADT A01 message, and which returns a String containing the HL7 v2.4 ADT A04 message.

```

public String HL7Transform(String sA01In) throws java.io.IOException, Throwable
{
    com.stc.SeeBeyond.OTD_Library.HL7.X_2_3.HL7_23_ADT_A01.ADT_A01 vA01_23 =
        new com.stc.SeeBeyond.OTD_Library.HL7.X_2_3.HL7_23_ADT_A01.ADT_A01();

    com.stc.SeeBeyond.OTD_Library.HL7.X_2_4.HL7_24_ADT_A04.ADT_A04 vA04_24 =
        new com.stc.SeeBeyond.OTD_Library.HL7.X_2_4.HL7_24_ADT_A04.ADT_A04();

    vA01_23.unmarshalFromString(sA01In);

    receive(vA01_23, vA04_24);

    String sA04Out = vA04_24.marshalToString();

    return sA04Out;
}

```

The “receive” method, with JMS dependencies removed, is similar to what the JCD wrapper bean

would have called in Java CAPS.

I will now add the transformation code I kept out so far, back to the receive method, so it is ready to be invoked and do its work.

The original JCD is available in the companion archive at <http://blogs.czapski.id.au/wp-content/uploads/2010/09/jcdHL702Transformer.zip>. The entire Java class as it stands, with the mapping code, is also reproduced at the end of this document.

Let's now create a "driver" class that will invoke the HL7Transformer with a HL7 message in a byte array and will display the transformed message received from the HL7Transformer.

Right-click on the name of the package, choose New → Java Class.

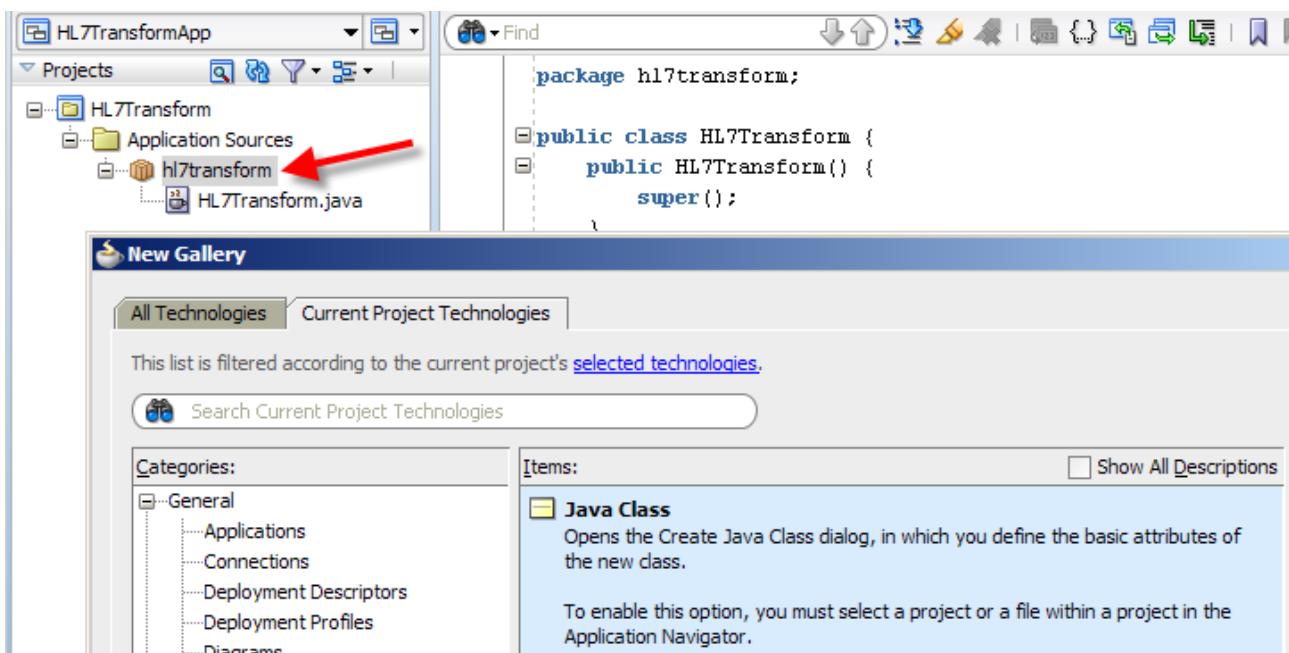


Illustration 18: Create a new Java class wizard

Name this new class HL7TransformerDriver, check Main Method checkbox and click OK.

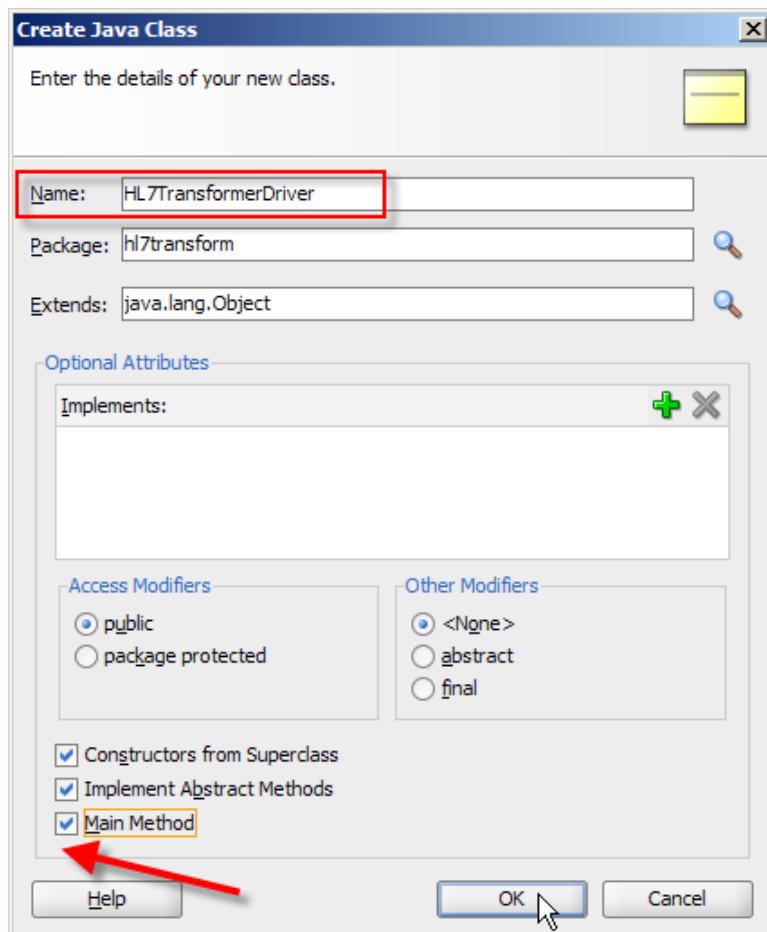


Illustration 19: Complete class skeleton creation

The following skeleton appears in the JDeveloper window.

```

package hl7transform;

public class HL7TransformerDriver {
    public HL7TransformerDriver() {
        super();
    }

    public static void main(String[] args) {
        HL7TransformerDriver hL7TransformerDriver = new HL7TransformerDriver();
    }
}

```

Illustration 20: Skeleton of the driver class

To eliminate complexities involved in acquiring a HL7 message from somewhere we will hardcode a sample message as a String constant in the driver class. Add the following code before the main method.

```

private static String sA01In = """
+ "MSH|^~\&|SystemA|HosA|PI|MDM|2008090801529||ADT^A01|200809080|P|2.3.1|||AL|NE\r"
+ "EVN|A01|2008090801529|||JavaCAPS6^^^^^^^USERS\r"
+ "PID|1||A000010^^^HosA^MR^HosA||Kessel^Abigail||19460101123045|M|||A Street^^Sydney^2000^AU\r"
+ "PV1|1|I||I|||FUL^Fulde^Gordian^^^^^^^^^MAIN|||EMR|||||||V2008090801529^^^VISIT\r"
;

```

Add the following code to the body of the main method:

```
System.out.println("\nA01:\n" + sA01In);

HL7Transform hl7t = new HL7Transform();
String sA04Out = "";

try {
    sA04Out = hl7t.HL7Transform(sA01In);
} catch (IOException e) {
    e.printStackTrace();
} catch (Throwable e) {
    e.printStackTrace();
}
System.out.println("\nA04:\n" + sA04Out);
```

The result will look like in the following figure.

```
1 package hl7transform;
2
3 import java.io.IOException;
4
5 public class HL7TransformDriver {
6
7     private static String sA01In = """
8         + "MSH|^~\>|SystemA|HosA|PI|MDM|2008090801529||ADT^A01|200809080|P|2.3.1|||AL|ME\r"
9         + "EVN|A01|2008090801529|||JavaCAPS6^^^^^^^USERS\r"
10        + "PID|1||A000010^^^HosA^MR^Hosa||Kessel^Abigail||19460101123045|M||A Street^^Sydney^2000^AU\r"
11        + "PV1|1|I||I||FUL^Fulde^Gordian^^^^^^^^^MAIN|||EMR|||||||V2008090801529^^^VISIT\r"
12    ;
13
14
15     public HL7TransformDriver() {
16         super();
17     }
18
19     public static void main(String[] args) {
20         HL7TransformDriver hl7TransformDriver = new HL7TransformDriver();
21
22         System.out.println("\nA01:\n" + sA01In);
23
24         HL7Transform hl7t = new HL7Transform();
25         String sA04Out = "";
26
27         try {
28             sA04Out = hl7t.HL7Transform(sA01In);
29         } catch (IOException e) {
30             e.printStackTrace();
31         } catch (Throwable e) {
32             e.printStackTrace();
33         }
34         System.out.println("\nA04:\n" + sA04Out);
35
36     }
37 }
```

Illustration 21: Driver class

Run this class, by right-clicking the name of the class and choosing Run.

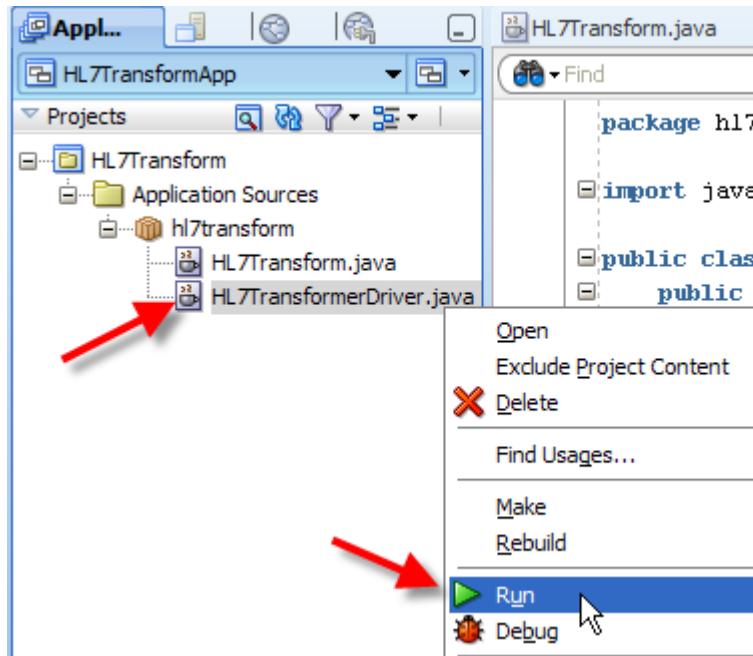


Illustration 22: Run the driver class

The output windows should display the A01 message and the A04 message which the transformer produced.

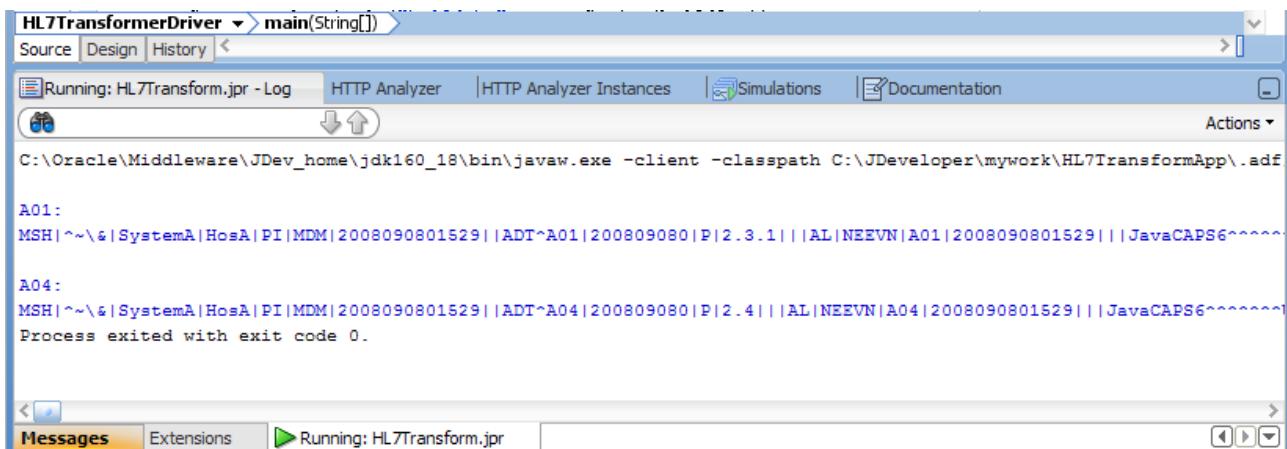


Illustration 23: Execution trace

This works fine as a stand-alone Java application.

Dependency Issues

We now know that we have all the necessary JARs identified and included in the project and that we have a POJO class, which performs transformation.

In reality this may not be as simple as that. The JCD was triggered by a JMS message and sent a message to JMS. The former happened at the beginning of the transformation code and the latter at the end. Knowing that all the rest of the collaboration maps between the input structure and the output structure meant that I did not have to spend the time walking through the code to identify and eliminate or port any non-transformation code that may be there in a more complex / different collaboration. Any obvious issues will be identified by JDeveloper as soon as we paste the JCD's

receive method into the new project, which we did early in the piece. The new class will not be able to be compiled until the issues are resolved.

Note that the JCD has access to logger, alerter, collabContext and typeConverter functionality. The logger is used reasonably frequently at development time. Some people use alerter to send explicit alerts to the runtime environment in Java CAPS, including stopping an inbound HL7 collaboration to prevent the system from accepting new incoming messages. I used collabContext on occasion, typically to identify the collaboration instance if I reused collaboration code. The typeConverter functionality is occasionally useful as well.

JCDs that make use of this functionality will have to be reviewed to see if it is critical to the transformation, how it is used and how it can be eliminated or modified if necessary.

As at October 2010, Oracle developed a series of classes and interfaces, packaged in a JAR called jcaps_interfaces.jar, to facilitate porting JCDs which make use of logger, alerter, collabContext and typeConverter. I have seen and used some of these classes but I am not aware, at this time, how a customer would go about getting hold of it. I got the collabContext and logger to work. I know that the alerter functionality is stubbed out and I was unable to make the typeConverter to work using the JAR I had access to. It being over 2 years since I last looked at this I don't know whether this archive is available and on what basis if it is.

The long and the short is that if the JCD uses this kind of functionality, or uses adapter functionality (for example to access a database), or uses some other functionality provided by other libraries, the JCD code will have to be reviewed and any issues identified and addressed. There is no prescription for this kind of work. It will vary in size and complexity with the JCD to be reviewed. There may be a point where work involved in addressing dependencies may exceed the benefit to be had from porting JCD code for a particular JCD.

Being able to externalise JCD code to a stand-alone Java class, as we have done in this section, will allow us to pick up and address these kinds of issues early, before we get into the complexity of the SOA Composite development.

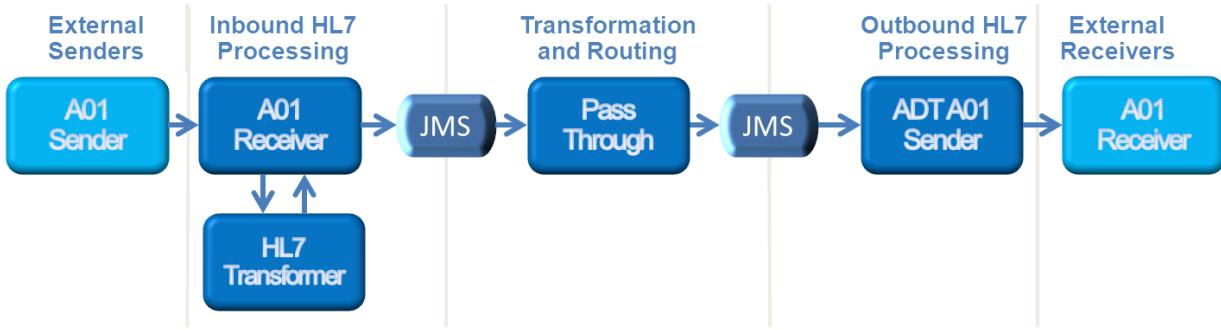
Basic SOA Suite for healthcare integration Solution

The Java CAPS HL7 solution, as discussed before, consists of the HL7 inbound and HL7 outbound projects, which take care of the HL7-compliant communication, and the HL7 transformation project. The schematic below illustrates the key components of this solution.



Drawing 2: Java CAPS HL7 Solution

The “equivalent” SOA Suite solution will look very similar, as shown in the following figure, except the HL7 Transform JCD replacement will move to between the HL7 Inbound and the JMS In Queue and the “HL7 Transformer” will be replaced by a pass-through component..



Drawing 3: SOA Suite for healthcare integration "Equivalent" reusing JCD code

The PassThrough component is a Mediator Project which receives whatever opaque payload is deposited in the JMS Queue, adds a JMS User-defined Property which will identify the target outbound endpoint, and dumps the whole payload into the outbound JMS Queue.

Stage I: Implement End-to-End HL7 Passthrough with JMS

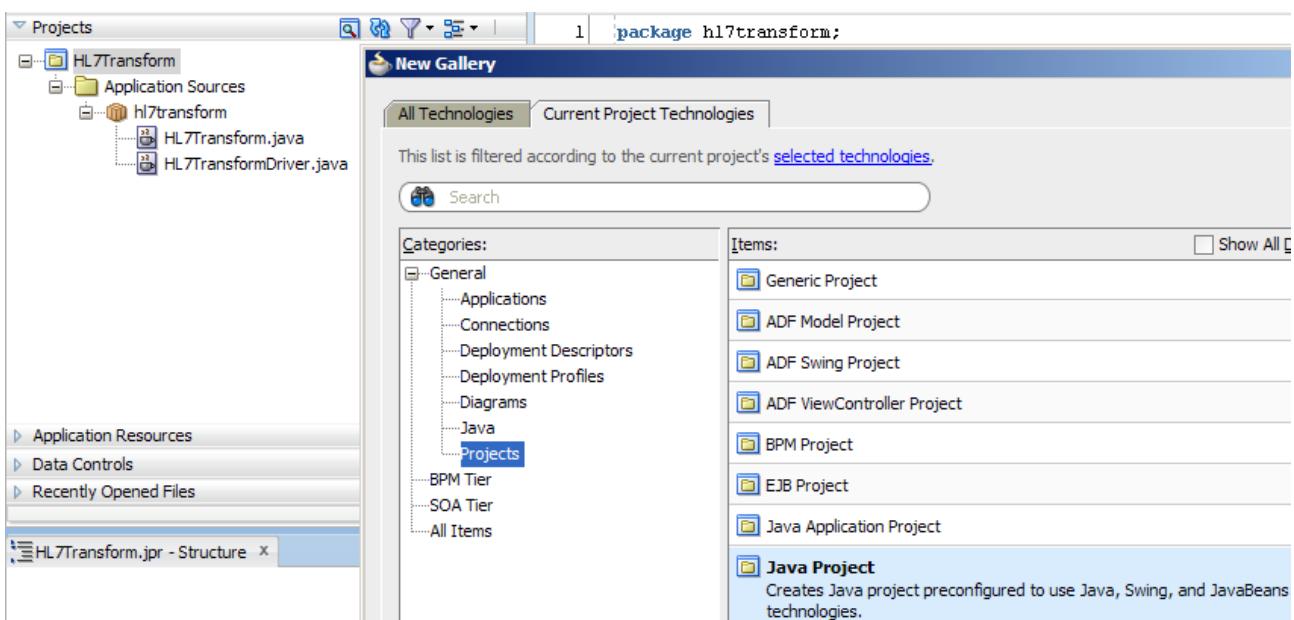
Before we get into the complexities of using Callouts, let's build and exercise the end-to-end HL7 v2 processing solution which uses JMS Queues as staging areas. This project is described in the blog article, “SOA Suite for healthcare integration Series – HL7 v2 solution using JMS “the Java CAPS way””, at <http://blogs.czapski.id.au/2013/02/soa-suite-for-healthcare-integration-series-hl7-v2-solution-using-jms-the-java-caps-way>.

Implement this project before resuming this article here.

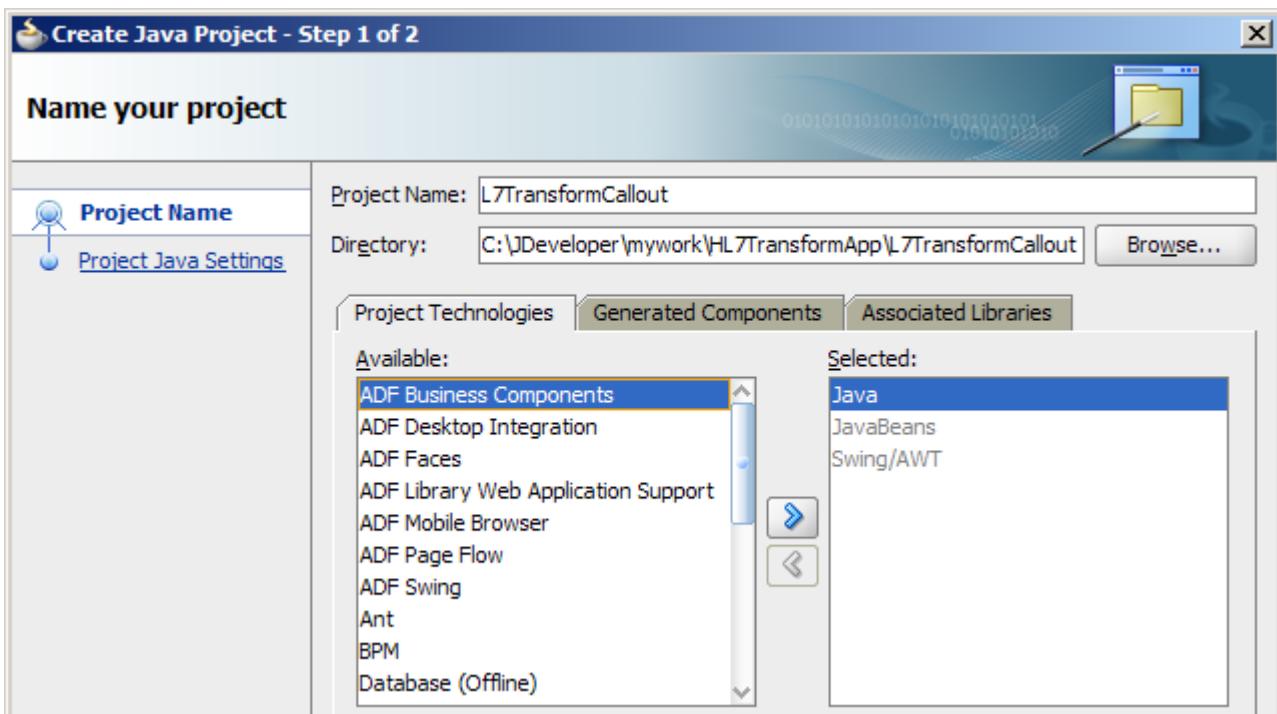
Stage II: Develop Java Callout with JCD code

Now we will create a Java Callout to host our JCD code, which we will reuse from the stand-alone Java application we created earlier. Later we will add this component to the inbound endpoint configuration.

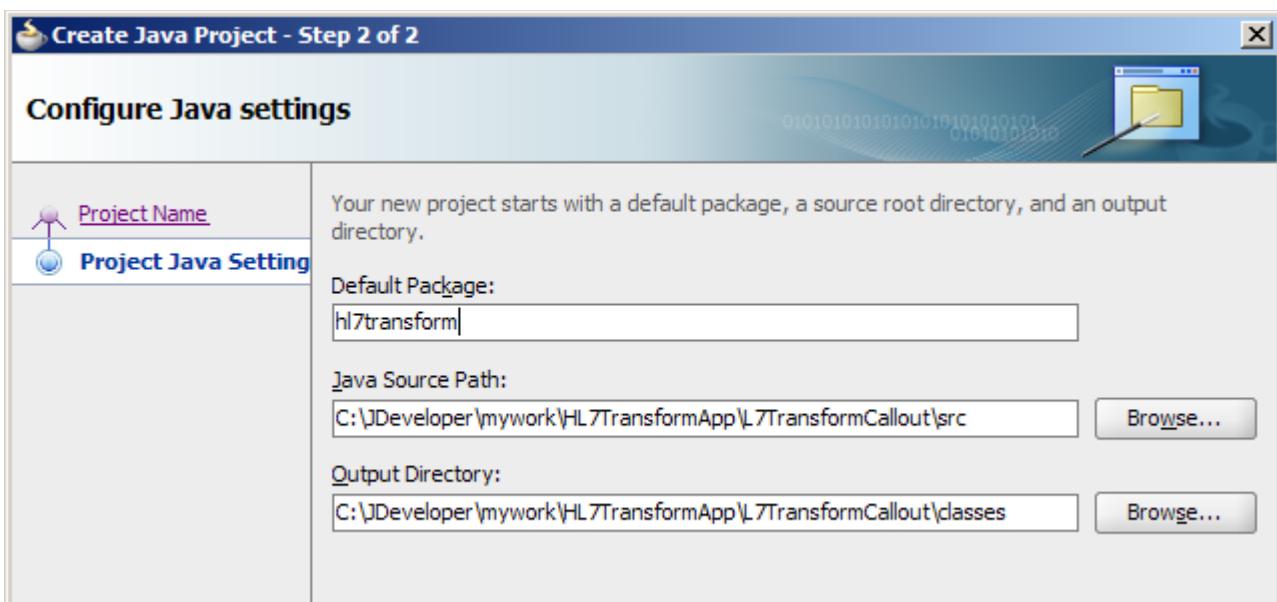
Right-click the name of the project HL7Transform. Choose “New”. Choose “Project” → “Java Project”.



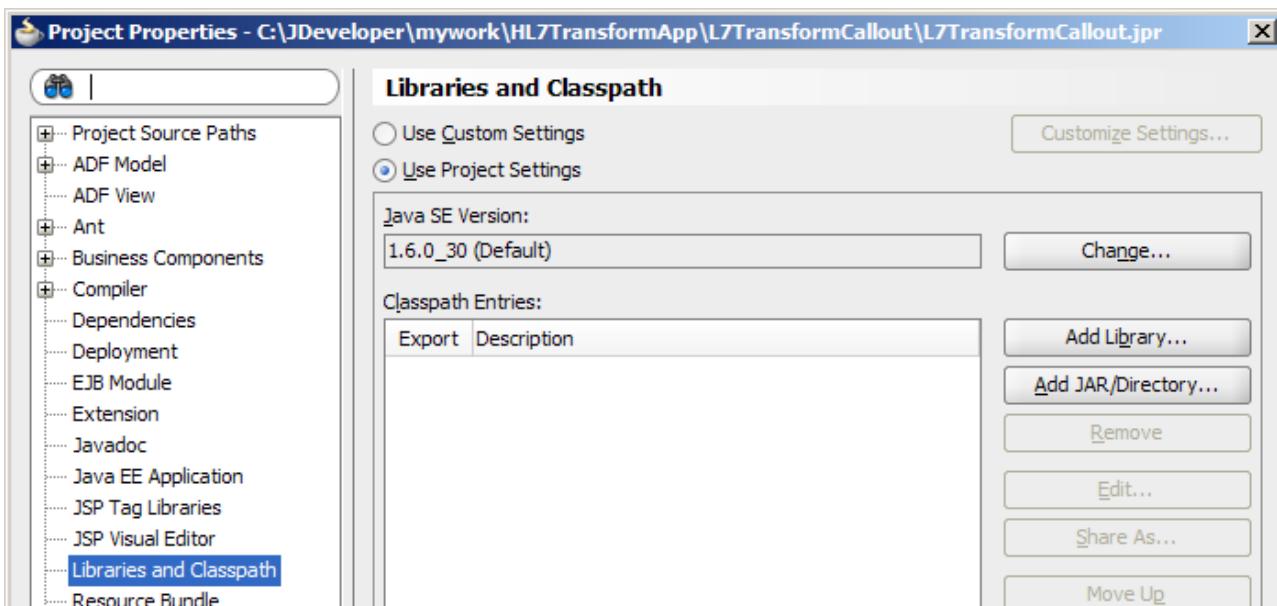
Name the project “L7TransformCallout”.



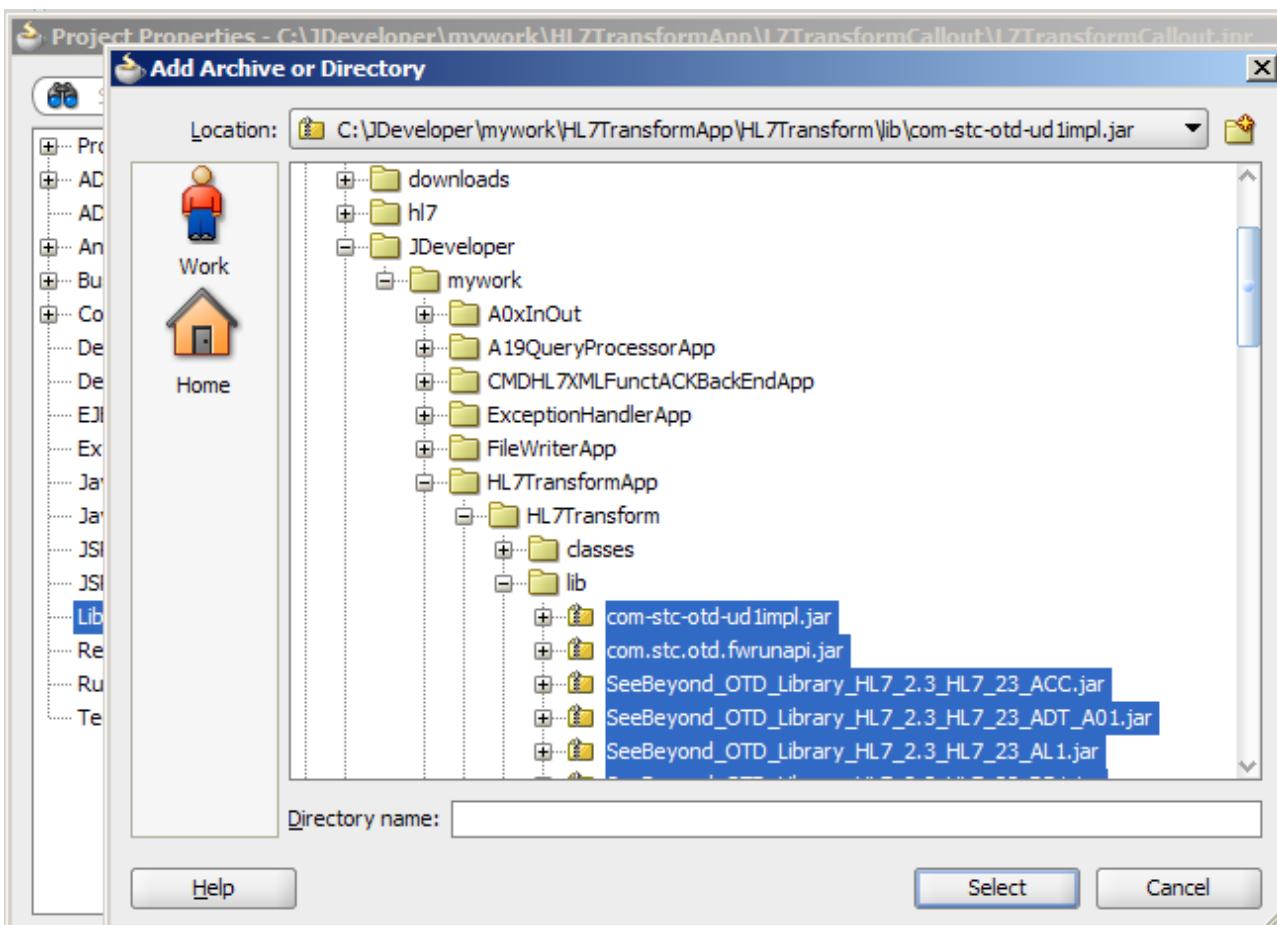
Change default package name to “hl7transform” and click “Finish”



Right-click the name of the new project, “HL7TransformCallout”, choose “Project Properties”, choose “Libraries and Classpath” and click “Add JAR/Directory...”



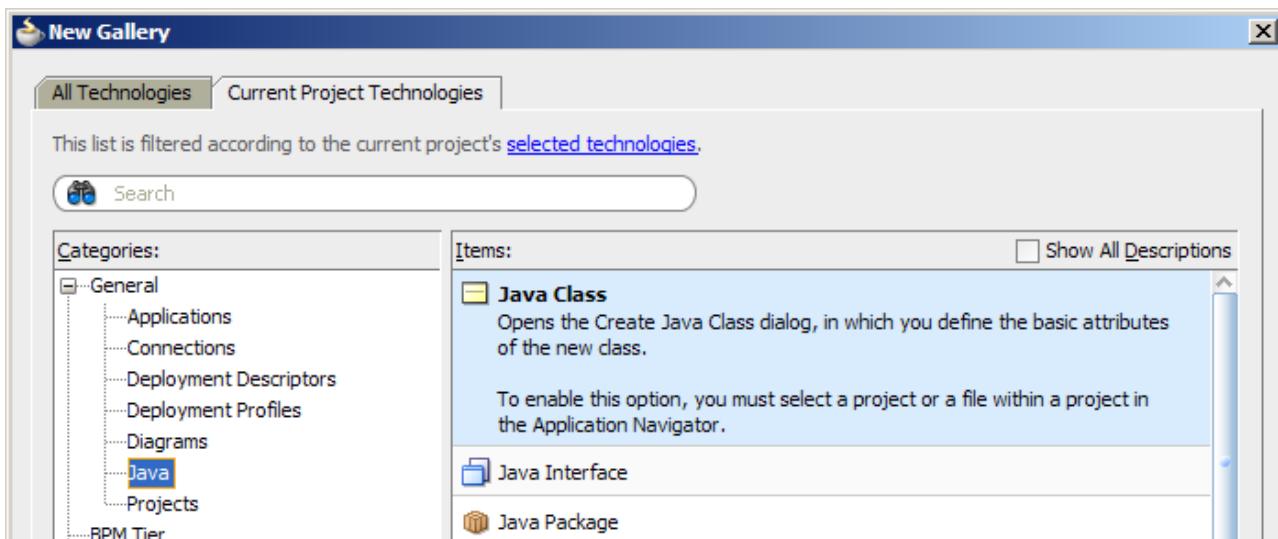
Select all JARs under HL7TransformApp/HL7Transform/lib and click “Select”



Click the “Add JAR/Directory...” again, navigate to a file system directory which contains the jar b2b.jar (in {WebLogic root}/ Oracle_SOAI/soa/modules/oracle.soa.b2b_11.1.1) and select it to get it added to the list.

Click “OK” to dismiss the dialogue box.

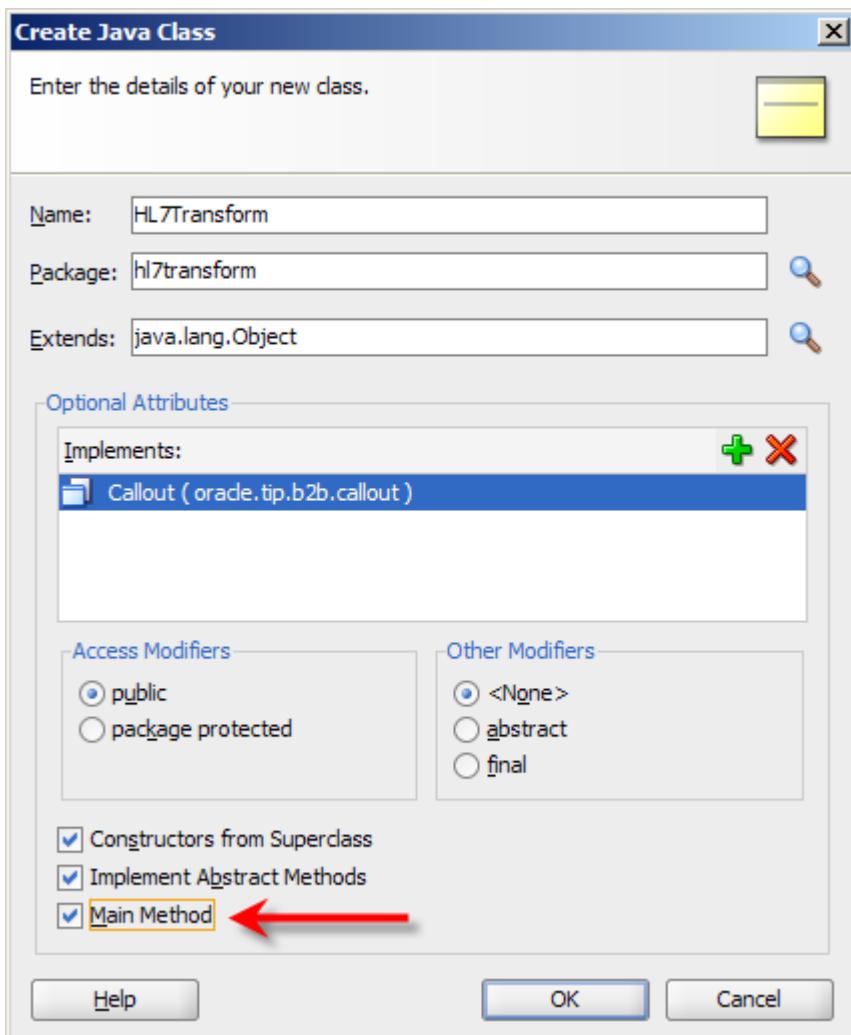
Right-click the name of the new project, choose “New...”, choose “Java”, choose “Java Class” and click “OK”



Name the new class “HL7Transform”, click on the “+” sign alongside the “Implements” label, choose Callout interface and click “OK”

The screenshot shows the 'Create Java Class' dialog and the 'Class and Package Browser'. In the 'Create Java Class' dialog, the 'Name' is 'HL7Transform', 'Package' is 'hl7transform', and 'Extends' is 'java.lang.Object'. In the 'Implements' field, 'Callout' is selected. A red arrow points to the 'Callout' entry in the 'Matching Classes and Packages' list of the 'Class and Package Browser'. The 'Class and Package Browser' also shows other matching classes like 'Callout (oracle.tip.b2b.message)'.

Check the “Main Method” checkbox and click “OK” to complete this task.



The following source code will have been generated:

```

package hl7transform;

import java.util.List;

import oracle.tip.b2b.callout.Callout;
import oracle.tip.b2b.callout.CalloutContext;
import oracle.tip.b2b.callout.exception.CalloutDomainException;
import oracle.tip.b2b.callout.exception.CalloutSystemException;

public class HL7Transform implements Callout {
    public HL7Transform() {
        super();
    }

    public static void main(String[] args) {
        HL7Transform hL7Transform = new HL7Transform();
    }

    public void execute(CalloutContext calloutContext, List list,
                       List list2) throws CalloutDomainException,
                                         CalloutSystemException {
    }
}

```

Add the following code as the implementation of the “execute” method:

```
try {
```

```

        CalloutMessage cmIn = (CalloutMessage)list.get(0);
        String cmStr = cmIn.getBodyAsString();
        System.out.println("\n====>>\n" +
                           cmStr);

        ADT_A01 vA01_23;
        vA01_23 =
            new
com.stc.SeeBeyond.OTD_Library.HL7.X_2_3.HL7_23_ADT_A01.ADT_A01();

        ADT_A04 vA04_24;
        vA04_24 =
            new
com.stc.SeeBeyond.OTD_Library.HL7.X_2_4.HL7_24_ADT_A04.ADT_A04();

        vA01_23.unmarshalFromString(cmStr);

        receive(vA01_23, vA04_24);

        String cmStrOut = vA04_24.marshalToString();

        System.out.println("\n====>>\n" +
                           cmStrOut);

        CalloutMessage cmOut = new CalloutMessage(cmStrOut);
        list2.add(cmOut);
    } catch (Exception e) {
        e.printStackTrace();
        throw new CalloutDomainException(e);
    } catch (Throwable e) {
        e.printStackTrace();
    }
}

```

Note the invocation of the “receive” method and recall this method from the “HL7Transformer” stand-alone Java application project which we developed and exercised earlier.

Copy the receive method code from that project and paste it after the “execute” method in the “HL7Transformer” class we are developing now.

The source, omitting details of the “receive” method for brevity, will look like the following:

```

package hl7transform;

import com.stc.SeeBeyond.OTD_Library.HL7.X_2_3.HL7_23_ADT_A01.ADT_A01;

import com.stc.SeeBeyond.OTD_Library.HL7.X_2_4.HL7_24_ADT_A04.ADT_A04;

import java.util.List;

import oracle.tip.b2b.callout.Callout;
import oracle.tip.b2b.callout.CalloutContext;
import oracle.tip.b2b.callout.CalloutMessage;
import oracle.tip.b2b.callout.exception.CalloutDomainException;
import oracle.tip.b2b.callout.exception.CalloutSystemException;

public class HL7Transform implements Callout {
    public HL7Transform() {
        super();
    }

    public static void main(String[] args) {
        HL7Transform hL7Transform = new HL7Transform();
    }
}

```

```

public void execute(CalloutContext calloutContext, List list,
                   List list2) throws CalloutDomainException,
                   CalloutSystemException {

    try {
        CalloutMessage cmIn = (CalloutMessage)list.get(0);
        String cmStr = cmIn.getBodyAsString();
        System.out.println("\n====>>\n" +
                           cmStr);

        ADT_A01 vA01_23;
        vA01_23 =
            new
com.stc.SeeBeyond.OTD_Library.HL7.X_2_3.HL7_23_ADT_A01.ADT_A01();

        ADT_A04 vA04_24;
        vA04_24 =
            new
com.stc.SeeBeyond.OTD_Library.HL7.X_2_4.HL7_24_ADT_A04.ADT_A04();

        vA01_23.unmarshalFromString(cmStr);

        receive(vA01_23, vA04_24);

        String cmStrOut = vA04_24.marshalToString();

        System.out.println("\n====>>\n" +
                           cmStrOut);

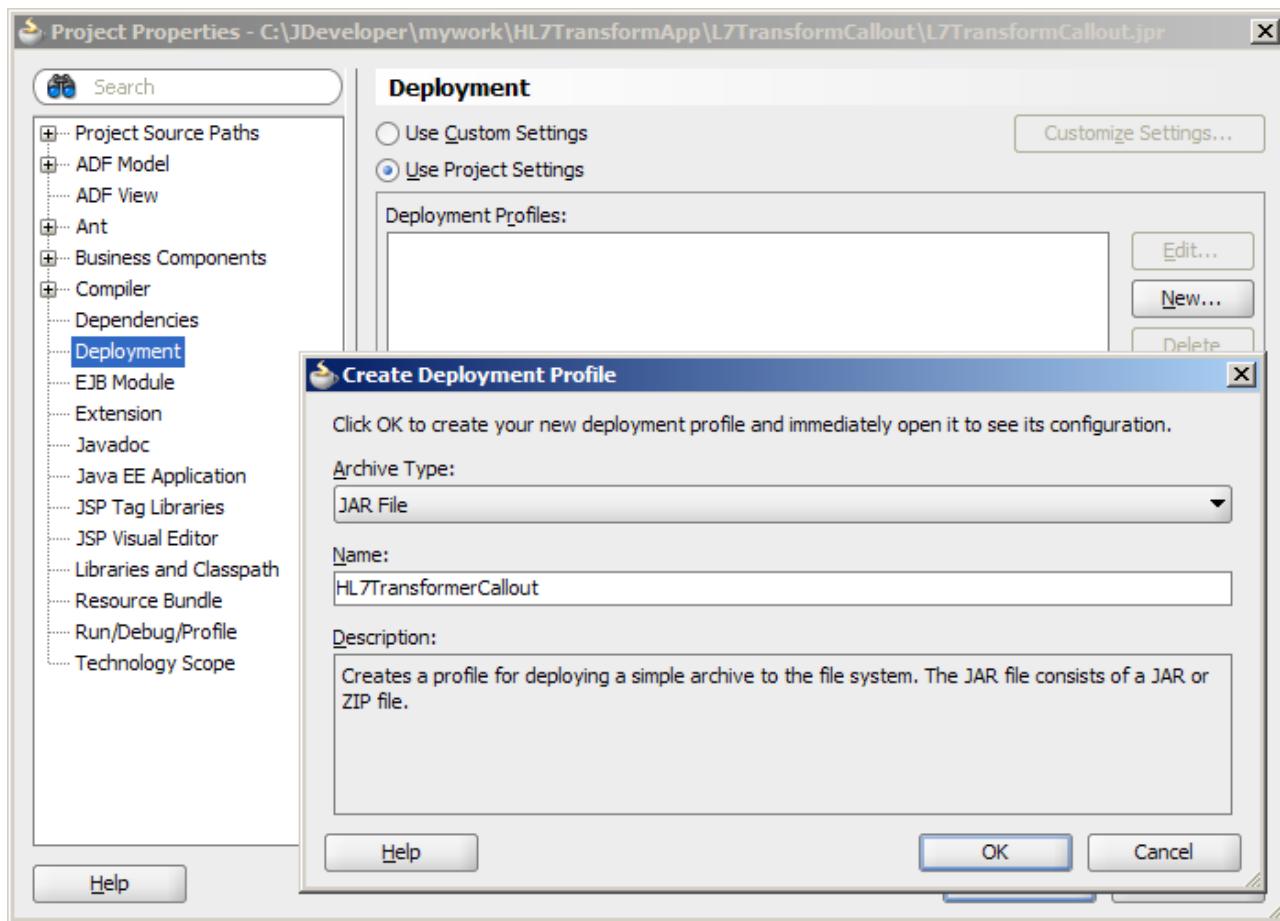
        CalloutMessage cmOut = new CalloutMessage(cmStrOut);
        list2.add(cmOut);
    } catch (Exception e) {
        e.printStackTrace();
        throw new CalloutDomainException(e);
    } catch (Throwable e) {
        e.printStackTrace();
    }
}

public void receive
(
com.stc.SeeBeyond.OTD_Library.HL7.X_2_3.HL7_23_ADT_A01.ADT_A01 vA01_23
,
com.stc.SeeBeyond.OTD_Library.HL7.X_2_4.HL7_24_ADT_A04.ADT_A04 vA04_24
)
throws Throwable
{
;
.
.
.
;
}
}

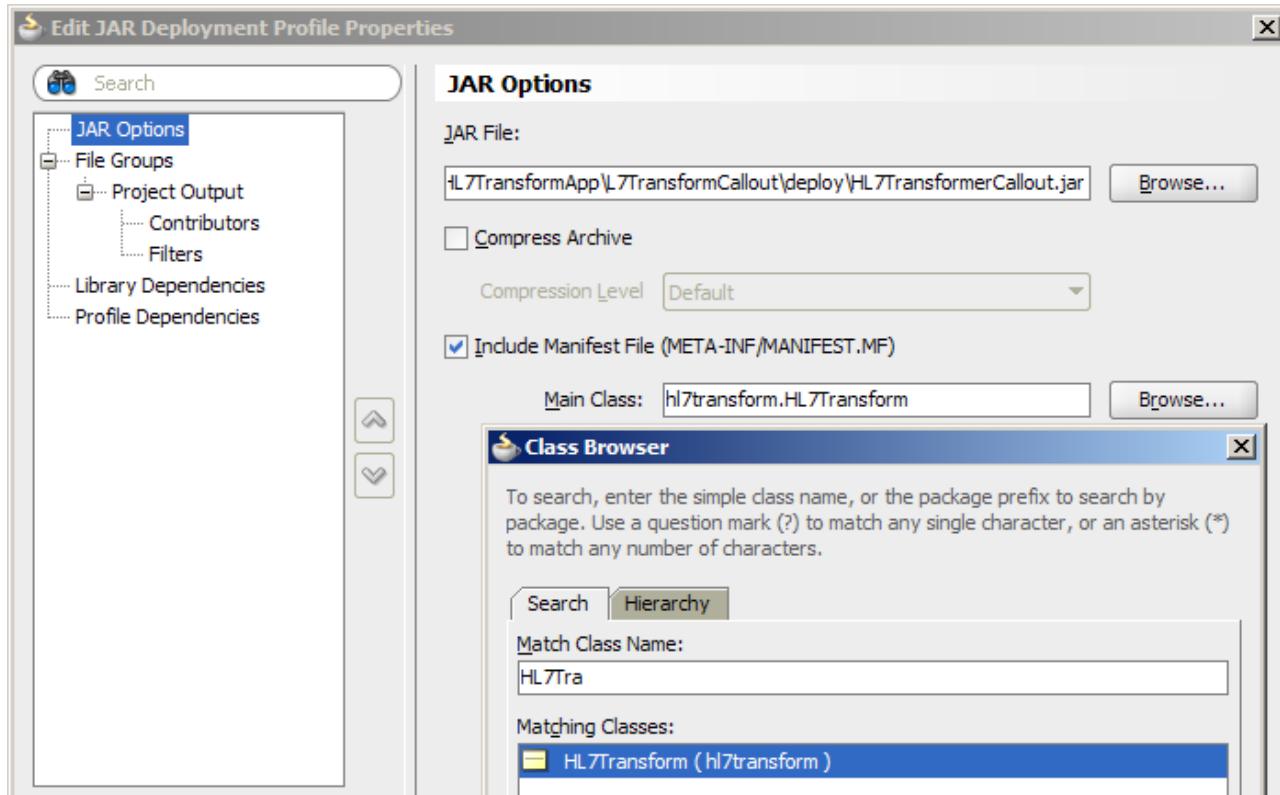
```

We need to package this class in a JAR so we can reference it in the callout definition.

Right-click the name of the project, “HL7TransformerCallout”, choose “Project Properties”, choose “Deployment”, leave archive type as default JAR, name the JAR “HL7TransformerCallout” and click “OK”



Browse for “Main Class”, choose “HL7Transform” and click “OK”



Click “OK” and “OK” to dismiss the wizard.

Right-click the name of the project, “HL7TransformerCallout”, choose “Deploy” and “deploy” the

project (build and generate the JAR file). Remember where the JAR file was written for later (for me C:\JDeveloper\mywork\HL7TransformApp\L7TransformCallout\deploy).

Stage III: Create and Populate File System Directories

We have the callout JAR and the JARs it depends on. We must place them in the directories where the SOA Suite for healthcare integration design time and runtime infrastructure can find them.

The callout archive, “HL7TransformerCallout.jar“, must be in two places, the callouts directory configured in the SOA Suite for healthcare integration Admin Console and a directory in the WebLogic domain’s classpath.

Archives which HL7TransformerCallout.jar depends on must be in a directory in the WebLogic domain’s classpath.

Let’s create a file system directory, “callouts” somewhere in the file systems and copy the JAR, “HL7TransformerCallout.jar“ to it. I created one as C:\oracle\CallOuts (on Windows capitalisation does not matter).

Let’s copy this JAR and all the HL7-related and subsidiary JARs from the HL7Transformer project’s lib directory to {WebLogic root}/user_projects/domains/{WebLogic Domain Name}\lib.

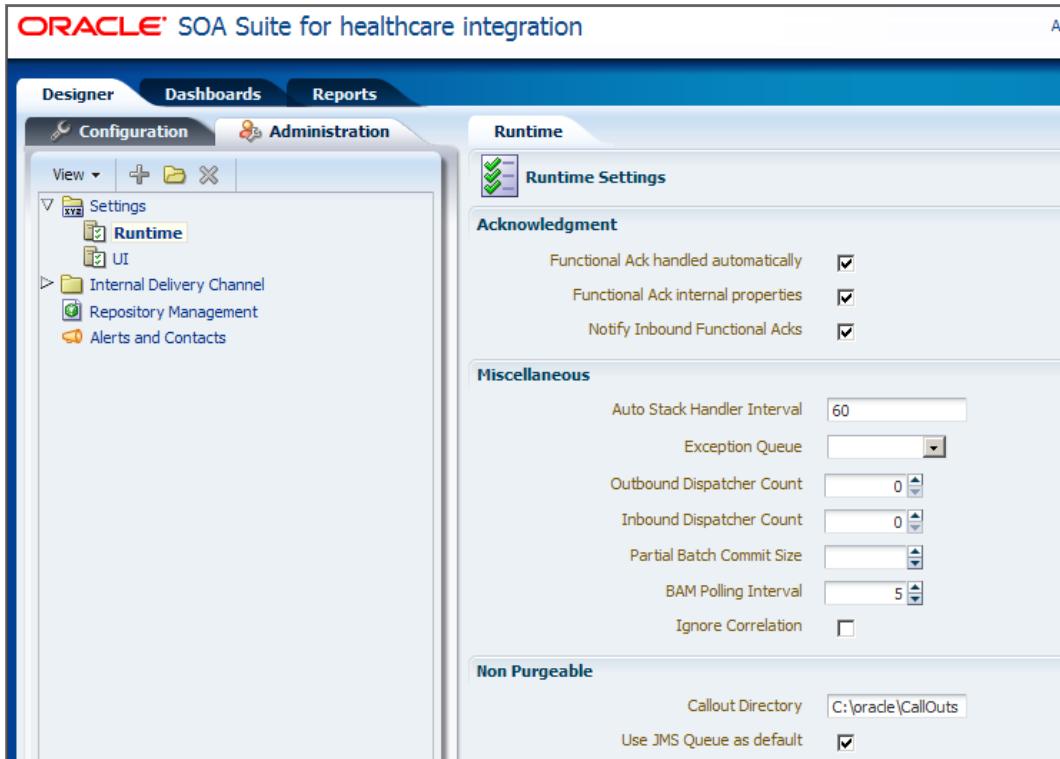
Stage IV: Restart WebLogic Server

To allow the JARs to be “discovered” we need to restart the WebLogic server. Let’s do that now.

Stage V: Set the location of callouts directory

By default the SOA Suite for healthcare integration has the callouts directory configured as “/MyCallous”, or some such. This is clearly incorrect for our purposes so let’s provide a correct directory path.

Start the SOA Suite for healthcare integration Admin Console, navigate to Designer→Administration→Settings→Runtime and set the Callouts Directory property as required (for me C:\oracle\CallOuts)



Log out of the SOA Suite for healthcare integration Admin Console

Stage VI: Configure the callout using the callout JAR

There is a bug in the current release of the SOA Suite for healthcare integration which prevents the designer from automatically recognising the main class in the callout archive. To work around this issue we will use the SOA Suite B2B Admin Console to define the callout. The outcome will be the same as it would have been had the SOA Suite for healthcare integration Admin Console worked as expected.

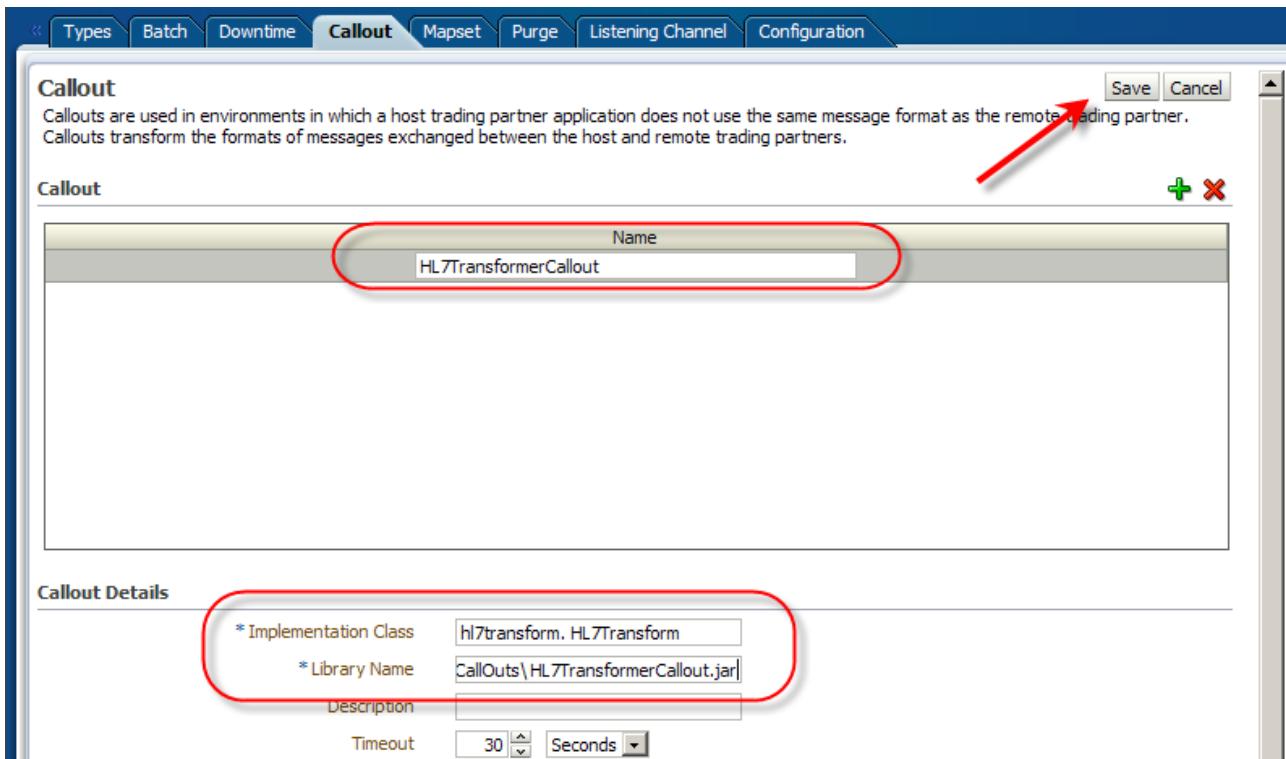
Start the SOA Suite B2B Admin Console, for me <http://localhost:7001/b2b>, log in as weblogic user, navigate through Administration→Callouts and click the “+” sign to add a new callout

Name the callout “HL7TransformerCallout”

Provide the following details then click “Save”:

* Implementation Class: hl7transform.HL7Transform

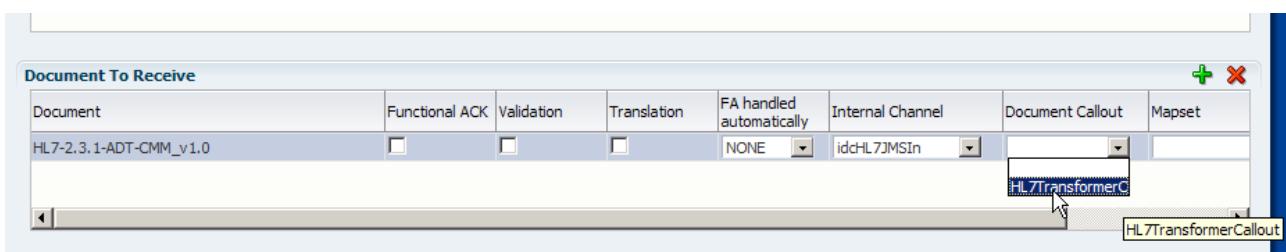
* Library Name: C:\oracle\CallOuts\HL7TransformerCallout.jar



Log out of the SOA Suite B2B Admin Console.

Stage VII: Configure the callout in the inbound Endpoint

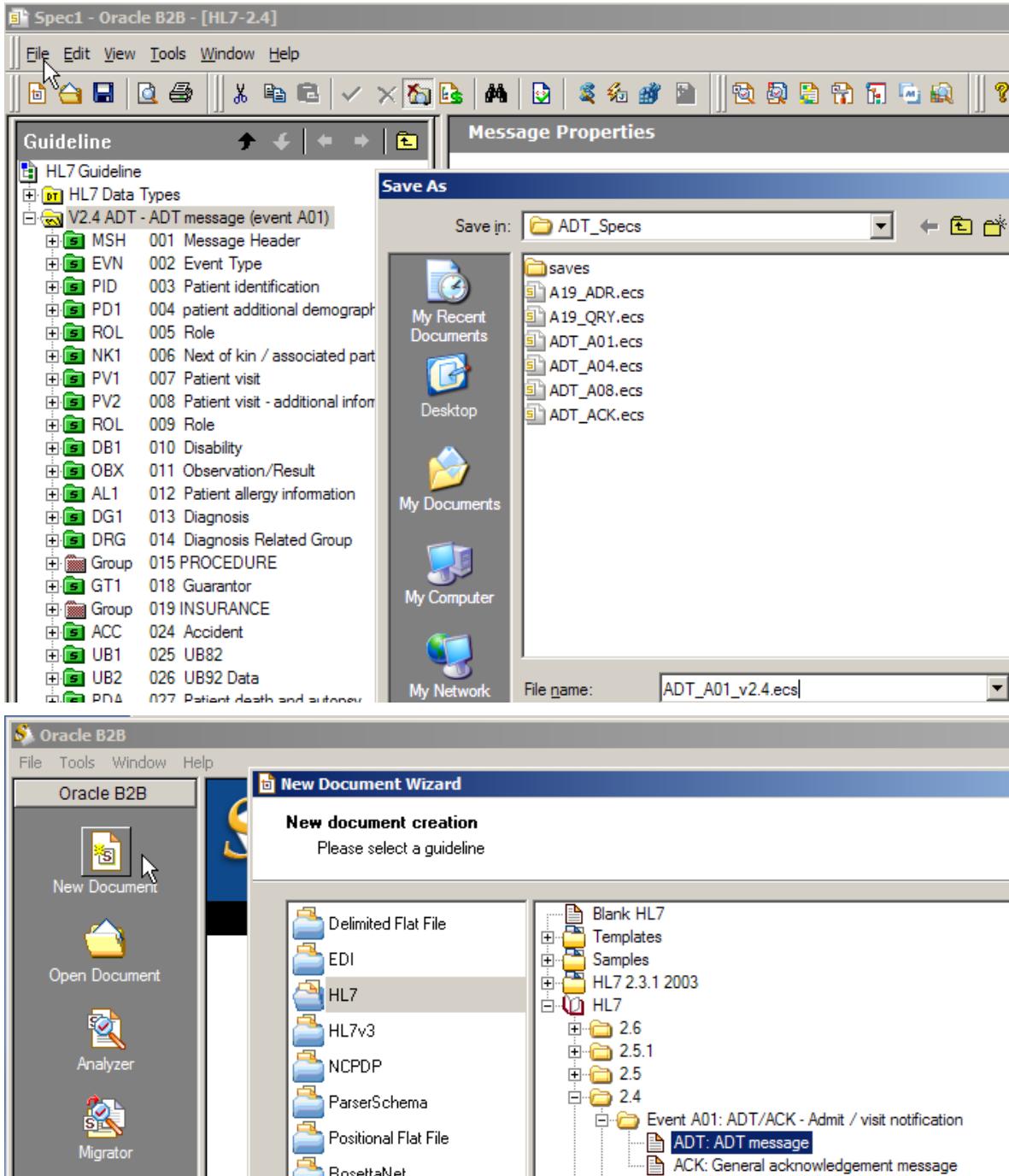
Log into the SOA Suite for healthcare integration, navigate to “Designer”→“Configuration”→“Endpoints”, the HL7JMSIn endpoint configuration, from the callouts dropdown in the “Document to Receive” section choose “HL7TransformerCallout” from the “Document Callout” dropdown and click “Apply”



Stage VIII: Add HL7 v2.4 ADT A01 Document

Use the Oracle B2B Document Editor to create the HL7 2.4 ADT A01 document

ADT_A01_v2.4.ecs file and the ADT_A01_v2.4.xsd file for the HL7 v2.4 ADT A01 structure. The method is discussed in detail in the blog article “SOA Suite for healthcare integration Series – Creating a Canonical HL7 v2 Message Model”, at <http://blogs.czapski.id.au/2012/09/soa-suite-for-healthcare-integration-series-creating-a-canonical-hl7-v2-message-model> and will not be repeated here.



Use the SOA Suite for healthcare integration Admin Console to add the document to the document hierarchy under HL7 → 2.4 → ADT node.

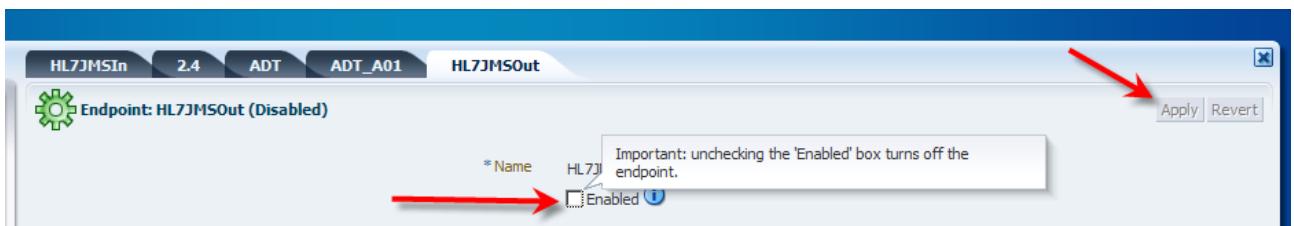


Stage IX: Make HL7JMSOut endpoint use the HL7 v2.4 ADT A01

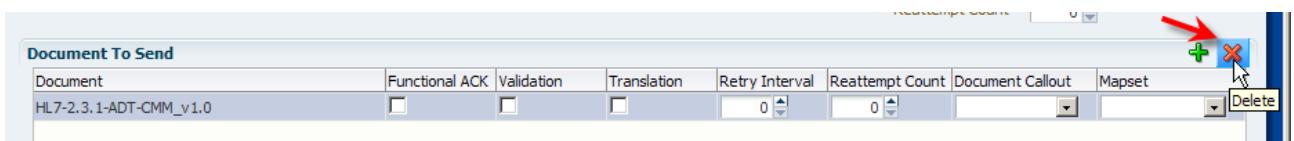
The HL7JMSOut endpoint is configured to send HL7 v2.3.1 ADT A01 document. We need to reconfigure it so that it uses the HL7 v2.4 ADT A01 document, which we just created, and to activate the endpoint with the change.

Use the SOA Suite for healthcare integration Admin Console, navigate to “Designer” → “Configuration” → “Endpoints”, open HKL7JMSOut configuration

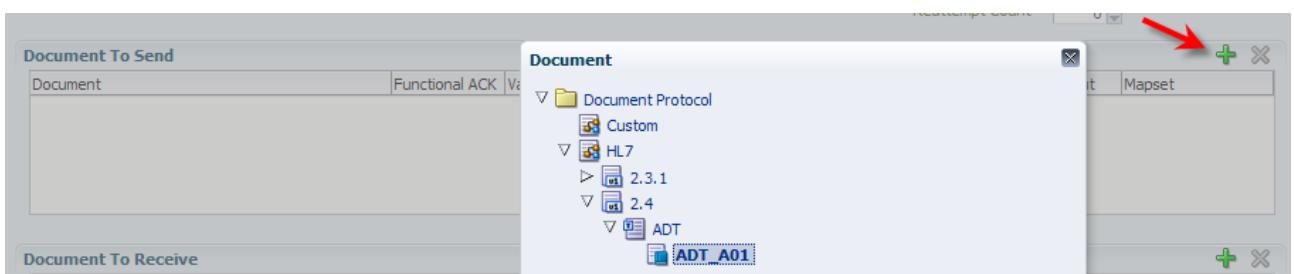
Uncheck the “Enabled” checkbox and “Apply”



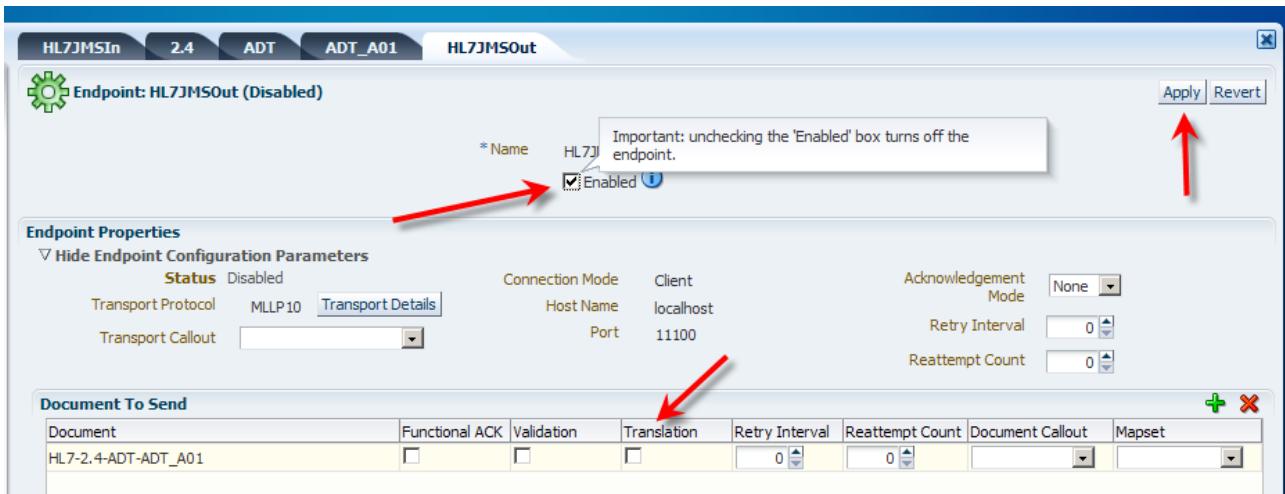
Delete the “HL7-2.3.1-ADT-CMM_v1.0” document from the “Documents to Send” section



Add “HL7”→”2.4”→”ADT”→”ADT_A01” to the “Documents to Send” section

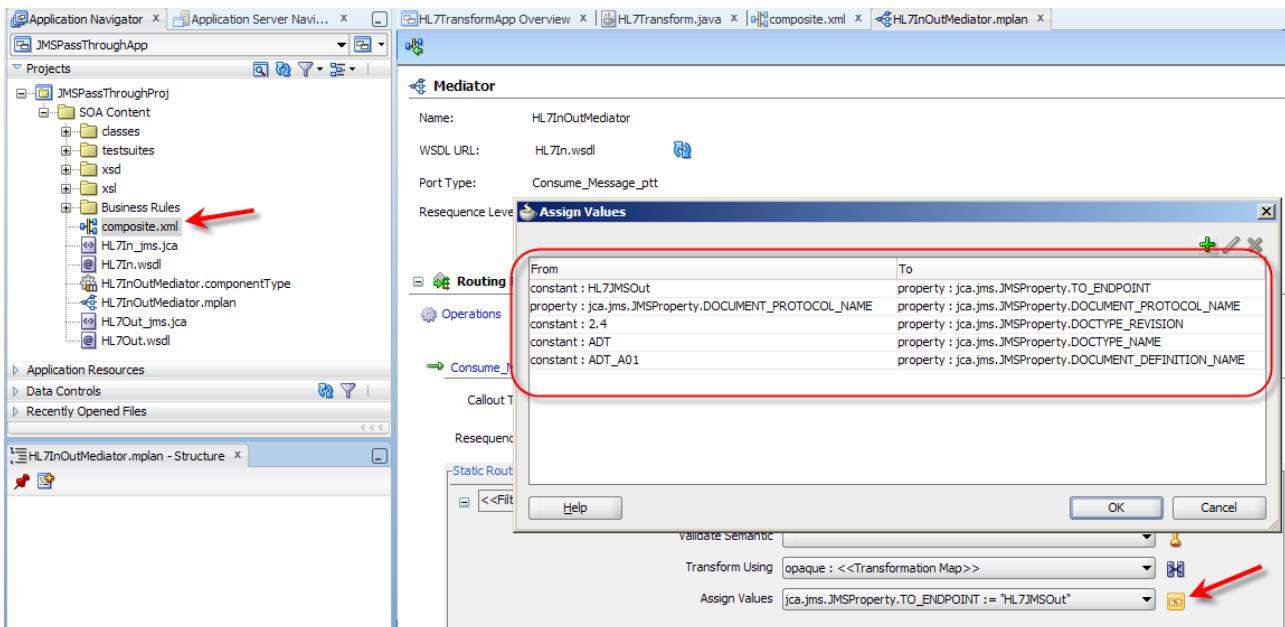


Uncheck the “Translation” checkbox, check the “Enable” checkbox and click “Apply” to activate the endpoint



Stage X: Modify JMSPassThrough project to support HL7 v2.4 ADT A01

The JMSPassThrough project passes along HL7 2.3.1 ADT CMM_v1.0 as parameters the endpoint is to use to locate the document to be used. These are not correct as we now have a document which uses a HL7 2.4 ADT A01. We need to modify the properties and re-deploy the application.



Stage XI: Exercise the solution

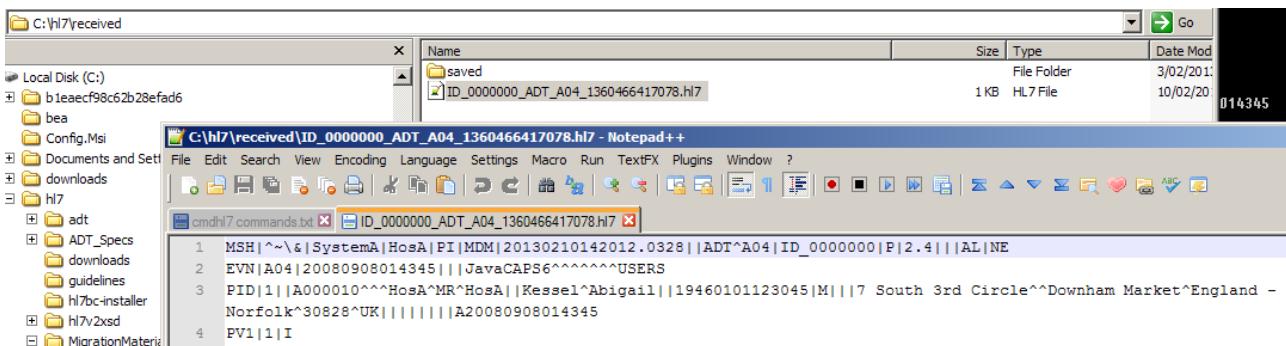
Start the CMDHL7 Listener in a command window with the command like the following, all on a single line:

```
java -Djava.util.logging.config.file=c:\tools\cmdhl7\logging_info.properties
-jar c:\tools\CMDHL7\CMSHL7Listener_v0.7.jar -p 11100 -s C:\hl7\received
```

Submit a HL7 v2.3.1 message with a command in a command window like the following,, all on a single line:

```
java -Djava.util.logging.config.file=c:\tools\cmdhl7\logging_info.properties
-jar c:\tools\CMDHL7\CMSHL7Sender_v0.7.jar -c ID -z -d \r\r\n -f
c:\hl7\adt\sources\ADT_A01_output_5099.hl7 -h localhost -p 11000 -r 1 -n 1
```

Examine output in the output directory (in my case -s C:\hl7\received).



Summary

Discussion in this article addressed a subset of technologies available in the Java CAPS and in the SOA Suite for healthcare integration. Specifically, the Java Collaboration Definitions supported in Java CAPS 5.x and in Java CAPS 6/Repository, and the Java Callout supported in the SOA Suite for healthcare integration. Both use the Java programming language and related runtime environment to implement processing logic.

The HL7 eWays and JCD based Java CAPS solutions were replaced by the Oracle SOA Suite healthcare integration endpoints.

The JCD code and class libraries were first externalised and tested as a standalone Java class.

The Java Callout project was then developed and elaborated to host the JCD code. This project was tested to ensure the port worked.

Finally, the Java Callout was added to the inbound HL7 endpoint to perform an on-the-fly transformation, which was originally performed by the “HL7 Transformer CD” to replicate the functionality of the end-to-end Java CAPS HL7 project with which we started.

It is clear that certain classes of Java CAPS JCDs can be good candidates for porting to Java Callouts to protect investment in development of transformation rules.

Appendix - HL702Transformer JCD Source

This code is available in a ZIP archive at <http://blogs.czapski.id.au/wp-content/uploads/2010/09/jcdHL702Transformer.zip>.

```
package HL7TransformerHL702Transformer;
public class jcdHL702Transformer
{
    public com.stccodegen.logger.Logger logger;
    public com.stccodegen.alerter.Alerter alerter;
    public com.stccodegen.util.CollaborationContext collabContext;
    public com.stccodegen.util.TypeConverter typeConverter;

    public void receive
        ( com.stc.connectors.jms.Message input
        , com.stc.SeeBeyond.OTD_Library.HL7.X_2_3.HL7_23_ADT_A01.ADT_A01 vA01_23
        , com.stc.SeeBeyond.OTD_Library.HL7.X_2_4.HL7_24_ADT_A04.ADT_A04 vA04_24
        , com.stc.connectors.jms.JMS vJMSOut )
        throws Throwable
    {
        vA01_23.unmarshalFromString( input.getTextMessage() );
        ;
        vA04_24.getMSH().setMSH_segment_ID( vA01_23.getMSH().getMSH_segment_ID() );
        vA04_24.getMSH().setMsh1FieldSeparator( vA01_23.getMSH().getMsh1FieldSeparator() );
        vA04_24.getMSH().setMsh2EncodingCharacters( vA01_23.getMSH().getMsh2EncodingCharacters() );
        if (vA01_23.getMSH().hasMsh3SendingApplication()) {
            if (vA01_23.getMSH().getMsh3SendingApplication().hasHD()) {
                if (vA01_23.getMSH().getMsh3SendingApplication().getHD().hasN342NamespaceId()) {
                    vA04_24.getMSH().getMsh3SendingApplication().getHD().setN342NamespaceId(
vA01_23.getMSH().getMsh3SendingApplication().getHD().getN342NamespaceId() );
                }
            }
        }
        if (vA01_23.getMSH().hasMsh4SendingFacility()) {
            if (vA01_23.getMSH().getMsh4SendingFacility().hasHD()) {
                if (vA01_23.getMSH().getMsh4SendingFacility().getHD().hasN342NamespaceId()) {
                    vA04_24.getMSH().getMsh4SendingFacility().getHD().setN342NamespaceId(
vA01_23.getMSH().getMsh4SendingFacility().getHD().getN342NamespaceId() );
                }
            }
        }
        if (vA01_23.getMSH().hasMsh5ReceivingApplication()) {
            if (vA01_23.getMSH().getMsh5ReceivingApplication().hasHD()) {
                if (vA01_23.getMSH().getMsh5ReceivingApplication().getHD().hasN342NamespaceId()) {
                    vA04_24.getMSH().getMsh5ReceivingApplication().getHD().setN342NamespaceId(
vA01_23.getMSH().getMsh5ReceivingApplication().getHD().getN342NamespaceId() );
                }
            }
        }
        if (vA01_23.getMSH().hasMsh6ReceivingFacility()) {
            if (vA01_23.getMSH().getMsh6ReceivingFacility().hasHD()) {
                if (vA01_23.getMSH().getMsh6ReceivingFacility().getHD().hasN342NamespaceId()) {
                    vA04_24.getMSH().getMsh6ReceivingFacility().getHD().setN342NamespaceId(

```

```

vA01_23.getMSH().getMsh6ReceivingFacility().getHD().getN342NamespaceId() );
}

}

if (vA01_23.getMSH().hasMsh7DateTimeOfMessage()) {
    if (vA01_23.getMSH().getMsh7DateTimeOfMessage().hasTS()) {
        if (vA01_23.getMSH().getMsh7DateTimeOfMessage().getTS().hasN439TimeOfAnEvent()) {
            vA04_24.getMSH().getMsh7DateTimeOfMessage().getTS().setN439TimeOfAnEvent(
vA01_23.getMSH().getMsh7DateTimeOfMessage().getTS().getN439TimeOfAnEvent() );
        }
    }
}

if (vA01_23.getMSH().hasMsh8Security()) {
    vA04_24.getMSH().setMsh8Security( vA01_23.getMSH().getMsh8Security() );
}

if (vA01_23.getMSH().getMsh9MessageType().hasCM_MSG()) {
    if (vA01_23.getMSH().getMsh9MessageType().getCM_MSG().hasN223MessageType()) {
        vA04_24.getMSH().getMsh9MessageType().getMSG().setN223MessageType(
vA01_23.getMSH().getMsh9MessageType().getCM_MSG().getN223MessageType() );
    }
}

vA04_24.getMSH().getMsh9MessageType().getMSG().setN2TriggerEvent( "A04" );
vA04_24.getMSH().setMsh10MessageControlId( vA01_23.getMSH().getMsh10MessageControlId() );
if (vA01_23.getMSH().getMsh11ProcessingId().hasPT()) {
    if (vA01_23.getMSH().getMsh11ProcessingId().getPT().hasN231ProcessingId()) {
        vA04_24.getMSH().getMsh11ProcessingId().getPT().setN231ProcessingId(
vA01_23.getMSH().getMsh11ProcessingId().getPT().getN231ProcessingId() );
    }
}

vA04_24.getMSH().getMsh12VersionId().getVID().setN362VersionId( "2.4" );
if (vA01_23.getMSH().hasMsh13SequenceNumber()) {
    vA04_24.getMSH().setMsh13SequenceNumber( vA01_23.getMSH().getMsh13SequenceNumber() );
}

if (vA01_23.getMSH().hasMsh14ContinuationPointer()) {
    vA04_24.getMSH().setMsh14ContinuationPointer( vA01_23.getMSH().getMsh14ContinuationPointer() );
}

if (vA01_23.getMSH().hasMsh15AcceptAcknowledgementType()) {
    vA04_24.getMSH().setMsh15AcceptAcknowledgmentType(
vA01_23.getMSH().getMsh15AcceptAcknowledgementType() );
}

if (vA01_23.getMSH().hasMsh16ApplicationAcknowledgementType()) {
    vA04_24.getMSH().setMsh16ApplicationAcknowledgmentType(
vA01_23.getMSH().getMsh16ApplicationAcknowledgementType() );
}

if (vA01_23.getMSH().hasMsh17CountryCode()) {
    vA04_24.getMSH().setMsh17CountryCode( vA01_23.getMSH().getMsh17CountryCode() );
}

if (vA01_23.getMSH().hasMsh18CharacterSet()) {
    vA04_24.getMSH().setMsh18CharacterSet( 0, vA01_23.getMSH().getMsh18CharacterSet() );
}

;

vA04_24.getEVN().setEVN_segment_ID( vA01_23.getEVN().getEVN_segment_ID() );
vA04_24.getEVN().setEvn1EventTypeCode( "A04" );
if (vA01_23.getEVN().hasEvn4EventReasonCode()) {

```

```

    vA04_24.getEVN().setEvn4EventReasonCode( vA01_23.getEVN().getEvn4EventReasonCode() );
}

if (vA01_23.getEVN().hasEvn2RecordedDateTime()) {
    if (vA01_23.getEVN().getEvn2RecordedDateTime().hasTS()) {
        if (vA01_23.getEVN().getEvn2RecordedDateTime().getTS().hasN439TimeOfAnEvent()) {
            vA04_24.getEVN().getEvn2RecordedDateTime().getTS().setN439TimeOfAnEvent(
                vA01_23.getEVN().getEvn2RecordedDateTime().getTS().getN439TimeOfAnEvent() );
        }
    }
}

if (vA01_23.getEVN().hasEvn3DateTimePlannedEvent()) {
    if (vA01_23.getEVN().getEvn3DateTimePlannedEvent().hasTS()) {
        if (vA01_23.getEVN().getEvn3DateTimePlannedEvent().getTS().hasN439TimeOfAnEvent()) {
            vA04_24.getEVN().getEvn3DateTimePlannedEvent().getTS().setN439TimeOfAnEvent(
                vA01_23.getEVN().getEvn3DateTimePlannedEvent().getTS().getN439TimeOfAnEvent() );
        }
    }
}

if (vA01_23.getEVN().hasEvn5OperatorId()) {
    if (vA01_23.getEVN().getEvn5OperatorId().hasCN()) {
        if (vA01_23.getEVN().getEvn5OperatorId().getCN().hasN272SourceTable()) {
            for (int i1 = 0; i1 < 1; i1 += 1) {
                vA04_24.getEVN().getEvn5OperatorId( i1 ).getXCN().setN272SourceTable(
                    vA01_23.getEVN().getEvn5OperatorId().getCN().getN272SourceTable() );
            }
        }
        if (vA01_23.getEVN().getEvn5OperatorId().getCN().hasN292IdNumberSt()) {
            vA04_24.getEVN().getEvn5OperatorId( 0 ).getXCN().setN292IdNumberSt(
                vA01_23.getEVN().getEvn5OperatorId().getCN().getN292IdNumberSt() );
        }
    }
}

if (vA01_23.getEVN().hasEvn6EventOccured()) {
    if (vA01_23.getEVN().getEvn6EventOccured().hasTS()) {
        if (vA01_23.getEVN().getEvn6EventOccured().getTS().hasN439TimeOfAnEvent()) {
            vA04_24.getEVN().getEvn6EventOccurred().getTS().setN439TimeOfAnEvent(
                vA01_23.getEVN().getEvn6EventOccured().getTS().getN439TimeOfAnEvent() );
        }
    }
}

vA04_24.setPID().setPID_segment_ID( vA01_23.getPID().getPID_segment_ID() );
if (vA01_23.getPID().hasPid1SetIdPatientId()) {
    vA04_24.setPID().setPid1SetIdPid( vA01_23.getPID().getPid1SetIdPatientId() );
}

if (vA01_23.getPID().hasPid2PatientIdExternalId()) {
    if (vA01_23.getPID().getPid2PatientIdExternalId().hasCX()) {
        if (vA01_23.getPID().getPid2PatientIdExternalId().getCX().hasN297Id()) {
            vA04_24.setPID().getPid2PatientId().getCX().setN297Id(
                vA01_23.getPID().getPid2PatientIdExternalId().getCX().getN297Id() );
        }
        if (vA01_23.getPID().getPid2PatientIdExternalId().getCX().hasN281AssigningAuthority()) {
            if (vA01_23.getPID().getPid2PatientIdExternalId().getCX().getN281AssigningAuthority().hasHD())
{
                if
(vA01_23.getPID().getPid2PatientIdExternalId().getCX().getN281AssigningAuthority().getHD().hasN342NamespaceId())

```

```

vA04_24.getPID().getPid2PatientId().getCX().getN281AssigningAuthority().getHD().setN342NamespaceId(
vA01_23.getPID().getPid2PatientIdExternalId().getCX().getN281AssigningAuthority().getHD().getN342NamespaceId() );
}
}

}

if (vA01_23.getPID().getPid2PatientIdExternalId().getCX().hasN252IdentifierTypeCode()) {
    vA04_24.getPID().getPid2PatientId().getCX().setN252IdentifierTypeCodeId(
vA01_23.getPID().getPid2PatientIdExternalId().getCX().getN252IdentifierTypeCode() );
}

if (vA01_23.getPID().getPid2PatientIdExternalId().getCX().hasN237AssigningFacility()) {
    if (vA01_23.getPID().getPid2PatientIdExternalId().getCX().getN237AssigningFacility().hasHD())
{
    if
(vA01_23.getPID().getPid2PatientIdExternalId().getCX().getN237AssigningFacility().getHD().hasN342NamespaceId()) {

vA04_24.getPID().getPid2PatientId().getCX().getN237AssigningFacility().getHD().setN342NamespaceId(
vA01_23.getPID().getPid2PatientIdExternalId().getCX().getN237AssigningFacility().getHD().getN342NamespaceId() );
}

}

}

}

for (int il = 0; il < vA01_23.getPID().countPid3PatientIdInternalId(); il += 1) {
    if (vA01_23.getPID().getPid3PatientIdInternalId( il ).hasCX()) {
        if (vA01_23.getPID().getPid3PatientIdInternalId( il ).getCX().hasN297Id()) {
            vA04_24.getPID().getPid3PatientIdentifierList( il ).getCX().setN297Id(
vA01_23.getPID().getPid3PatientIdInternalId( il ).getCX().getN297Id() );
        }

        if (vA01_23.getPID().getPid3PatientIdInternalId( il ).getCX().hasN281AssigningAuthority()) {
            if (vA01_23.getPID().getPid3PatientIdInternalId( il
).getCX().getN281AssigningAuthority().hasHD()) {
                if (vA01_23.getPID().getPid3PatientIdInternalId( il
).getCX().getN281AssigningAuthority().getHD().hasN342NamespaceId()) {
                    vA04_24.getPID().getPid3PatientIdentifierList( il
).getCX().getN281AssigningAuthority().getHD().setN342NamespaceId( vA01_23.getPID().getPid3PatientIdInternalId( il
).getCX().getN281AssigningAuthority().getHD().getN342NamespaceId() );
                }

            }

        }

        if (vA01_23.getPID().getPid3PatientIdInternalId( il ).getCX().hasN252IdentifierTypeCode()) {
            vA04_24.getPID().getPid3PatientIdentifierList( il ).getCX().setN252IdentifierTypeCodeId(
vA01_23.getPID().getPid3PatientIdInternalId( il ).getCX().getN252IdentifierTypeCode() );
        }

        if (vA01_23.getPID().getPid3PatientIdInternalId( il ).getCX().hasN237AssigningFacility()) {
            if (vA01_23.getPID().getPid3PatientIdInternalId( il
).getCX().getN237AssigningFacility().hasHD()) {
                if (vA01_23.getPID().getPid3PatientIdInternalId( il
).getCX().getN237AssigningFacility().getHD().hasN342NamespaceId()) {
                    vA04_24.getPID().getPid3PatientIdentifierList( il
).getCX().getN237AssigningFacility().getHD().setN342NamespaceId( vA01_23.getPID().getPid3PatientIdInternalId( il
).getCX().getN237AssigningFacility().getHD().getN342NamespaceId() );
                }

            }

        }

    }

}

}

if (vA01_23.getPID().getPid5PatientName().hasXPN()) {
    if (vA01_23.getPID().getPid5PatientName().getXPN().hasN201FamilyName()) {
        for (int il1 = 0; il1 < 1; il1 += 1) {

```

```

        vA04_24.getPID().getPid5PatientName( il ).getXPN().getN201FamilyName().getFN().setN386Surname(
vA01_23.getPID().getPid5PatientName().getXPN().getN201FamilyName() );
    }

}

if (vA01_23.getPID().getPid5PatientName().getXPN().hasN22GivenName()) {
    for (int il = 0; il < 1; il += 1) {

        vA04_24.getPID().getPid5PatientName( il ).getXPN().setN22GivenName(
vA01_23.getPID().getPid5PatientName().getXPN().getN22GivenName() );
    }
}

if (vA01_23.getPID().getPid5PatientName().getXPN().hasN23MiddleInitialOrName()) {
    for (int il = 0; il < 1; il += 1) {

        vA04_24.getPID().getPid5PatientName( il
).getXPN().setN23SecondAndFurtherGivenNamesOrInitialsThereof(
vA01_23.getPID().getPid5PatientName().getXPN().getN23MiddleInitialOrName() );
    }
}

if (vA01_23.getPID().getPid5PatientName().getXPN().hasN273SuffixEGJrOrIiii()) {
    for (int il = 0; il < 1; il += 1) {

        vA04_24.getPID().getPid5PatientName( il ).getXPN().setN273SuffixEGJrOrIiii(
vA01_23.getPID().getPid5PatientName().getXPN().getN273SuffixEGJrOrIiii() );
    }
}

if (vA01_23.getPID().getPid5PatientName().getXPN().hasN235PrefixEGDr()) {
    for (int il = 0; il < 1; il += 1) {

        vA04_24.getPID().getPid5PatientName( il ).getXPN().setN235PrefixEGDr(
vA01_23.getPID().getPid5PatientName().getXPN().getN235PrefixEGDr() );
    }
}

if (vA01_23.getPID().getPid5PatientName().getXPN().hasN203DegreeEGMd()) {
    for (int il = 0; il < 1; il += 1) {

        vA04_24.getPID().getPid5PatientName( il ).getXPN().setN203DegreeEGMd(
vA01_23.getPID().getPid5PatientName().getXPN().getN203DegreeEGMd() );
    }
}

if (vA01_23.getPID().hasPid7DateOfBirth()) {
    if (vA01_23.getPID().getPid7DateOfBirth().hasTS()) {
        if (vA01_23.getPID().getPid7DateOfBirth().getTS().hasN439TimeOfAnEvent()) {
            vA04_24.getPID().getPid7DateTimeOfBirth().getTS().setN439TimeOfAnEvent(
vA01_23.getPID().getPid7DateOfBirth().getTS().getN439TimeOfAnEvent() );
        }
    }
}

if (vA01_23.getPID().hasPid8Sex()) {
    vA04_24.getPID().setPid8AdministrativeSex( vA01_23.getPID().getPid8Sex() );
}

if (vA01_23.getPID().hasPid11PatientAddress()) {
    for (int il = 0; il < vA01_23.getPID().countPid11PatientAddress(); il += 1) {
        if (vA01_23.getPID().getPid11PatientAddress( il ).hasXAD()) {
            if (vA01_23.getPID().getPid11PatientAddress( il ).getXAD().hasN27StreetAddress()) {
                vA04_24.getPID().getPid11PatientAddress( il
).getXAD().getN403StreetAddressSad().getSAD().setN398StreetOrMailingAddress(
vA01_23.getPID().getPid11PatientAddress( il ).getXAD().getN27StreetAddress() );
            }
        }
        if (vA01_23.getPID().getPid11PatientAddress( il ).getXAD().hasN28OtherDesignation()) {
            vA04_24.getPID().getPid11PatientAddress( il ).getXAD().setN28OtherDesignation(

```

```

vA01_23.getPID().getPid11PatientAddress( i1 ).getXAD().getN28OtherDesignation() );
}

if (vA01_23.getPID().getPid11PatientAddress( i1 ).getXAD().hasN29City()) {
    vA04_24.getPID().getPid11PatientAddress( i1 ).getXAD().setN29City(
vA01_23.getPID().getPid11PatientAddress( i1 ).getXAD().getN29City() );
}

if (vA01_23.getPID().getPid11PatientAddress( i1 ).getXAD().hasN30StateOrProvince()) {
    vA04_24.getPID().getPid11PatientAddress( i1 ).getXAD().setN30StateOrProvince(
vA01_23.getPID().getPid11PatientAddress( i1 ).getXAD().getN30StateOrProvince() );
}

if (vA01_23.getPID().getPid11PatientAddress( i1 ).getXAD().hasN31ZipOrPostalCode()) {
    vA04_24.getPID().getPid11PatientAddress( i1 ).getXAD().setN31ZipOrPostalCode(
vA01_23.getPID().getPid11PatientAddress( i1 ).getXAD().getN31ZipOrPostalCode() );
}

if (vA01_23.getPID().getPid11PatientAddress( i1 ).getXAD().hasN32Country()) {
    vA04_24.getPID().getPid11PatientAddress( i1 ).getXAD().setN32Country(
vA01_23.getPID().getPid11PatientAddress( i1 ).getXAD().getN32Country() );
}

if (vA01_23.getPID().getPid11PatientAddress( i1 ).getXAD().hasN202AddressType()) {
    vA04_24.getPID().getPid11PatientAddress( i1 ).getXAD().setN202AddressType(
vA01_23.getPID().getPid11PatientAddress( i1 ).getXAD().getN202AddressType() );
}

if (vA01_23.getPID().getPid11PatientAddress( i1 ).getXAD().hasN34OtherGeographicDesignation())
{
    vA04_24.getPID().getPid11PatientAddress( i1 ).getXAD().setN34OtherGeographicDesignation(
vA01_23.getPID().getPid11PatientAddress( i1 ).getXAD().getN34OtherGeographicDesignation() );
}

if (vA01_23.getPID().getPid11PatientAddress( i1 ).getXAD().hasN330CountyParishCode()) {
    vA04_24.getPID().getPid11PatientAddress( i1 ).getXAD().setN330CountyParishCode(
vA01_23.getPID().getPid11PatientAddress( i1 ).getXAD().getN330CountyParishCode() );
}

if (vA01_23.getPID().getPid11PatientAddress( i1 ).getXAD().hasN266CensusTract()) {
    vA04_24.getPID().getPid11PatientAddress( i1 ).getXAD().setN266CensusTract(
vA01_23.getPID().getPid11PatientAddress( i1 ).getXAD().getN266CensusTract() );
}

if (vA01_23.getPID().getPid11PatientAddress( i1
).getXAD().hasXAD_sun_unexpected_subcomponent()) {
    for (int i2 = 0; i2 < vA01_23.getPID().getPid11PatientAddress( i1
).getXAD().countXAD_sun_unexpected_subcomponent(); i2 += 1) {
        vA04_24.getPID().getPid11PatientAddress( i1
).getXAD().setN365AddressRepresentationCode( vA01_23.getPID().getPid11PatientAddress( i1
).getXAD().getXAD_sun_unexpected_subcomponent( i2 ) );
    }
}
}

if (vA01_23.getPID().hasPid16MaritalStatus()) {
    for (int i1 = 0; i1 < vA01_23.getPID().countPid16MaritalStatus(); i1 += 1) {
        vA04_24.getPID().getPid16MaritalStatus().getCE_0002().setN391IdentifierSt(
            vA01_23.getPID().getPid16MaritalStatus( i1 ) );
    }
}

if (vA01_23.getPID().hasPid18PatientAccountNumber()) {
    if (vA01_23.getPID().getPid18PatientAccountNumber().hasCX()) {
        if (vA01_23.getPID().getPid18PatientAccountNumber().getCX().hasN297Id()) {
            vA04_24.getPID().getPid18PatientAccountNumber().getCX().setN297Id(
                vA01_23.getPID().getPid18PatientAccountNumber().getCX().getN297Id() );
        }
    }
}

```

```

        }

    }

    if (vA01_23.getPID().hasPid19SsnNumberPatient()) {
        vA04_24.setPID().setPid19SsnNumberPatient( vA01_23.getPID().getPid19SsnNumberPatient() );
    }

;

vA04_24.getPV1().setPV1_segment_ID( vA01_23.getPV1().getPV1_segment_ID() );
if (vA01_23.getPV1().hasPv11SetIdPatientVisit()) {
    vA04_24.getPV1().setPv11SetIdPv1( vA01_23.getPV1().getPv11SetIdPatientVisit() );
}

vA04_24.getPV1().setPv12PatientClass( vA01_23.getPV1().getPv12PatientClass() );
;

;

;

String sA04Out = vA04_24.marshallToString();
;

;

com.stc.connectors.jms.Message vJMSMsg = vJMSOut.createTextMessage();
vJMSMsg.setTextMessage( sA04Out );
vJMSOut.sendText( sA04Out );
}
}

```