

HL7 JCD to Spring Component Migration

michal@czapski.id.au, October 2010, Rev 1.3

Table of Contents

Introduction.....	1
Prerequisites and Assumptions.....	2
Migration Process	2
Anatomy of a Java CAPS HL7 Solution.....	3
Analysing HL7 JCD Solution	4
Analysing HL7 processing JCD.....	5
Analysing Enterprise Archive	6
Building and exercising a Stand-alone Java Application.....	10
Dependency Issues	22
Basic SOA Suite Solution with Spring Component.....	22
Stage I: Basic File to File Mediator project	23
Stage II: Adding Spring Component with JCD code	34
SOA Suite Solution with B2B and Spring Component.....	47
Configure Inbound HL7 Partnership Agreement	48
Configure Outbound HL7 Partnership Agreement	49
Add B2B HL7 Support to the Mediator Solution	52
Summary	62
Appendix - HL702Transformer JCD Source	64

Introduction

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. 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 migrating to the SOA Suite Spring Component as means of preserving the code and the effort invested in developing it. This does not make the method domain-specific. On the contrary, the method is applicable to all other domains where JCDs with significant transformation and mapping rules content are used.

Discussion in this article addresses a subset of technologies available in the Java CAPS and in the SOA Suite. Specifically, the Java Collaboration Definitions supported in Java CAPS 5.x and in Java CAPS 6/Repository, and the Spring Component supported in the SOA Suite 11g R1 PS2. Both use the Java programming language and related runtime environment to implement processing logic. 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 a Oracle SOA Suite solution.

The HL7 eWay and JCD based Java CAPS solution will be ported to the Oracle SOA Suite 11g B2B and Mediator-based environment. HL7 Adapters will be replaced with the Oracle “Healthcare Adapters”, provided by the SOA Suite B2B HL7 support infrastructure. Routing will be provided by the Mediator component and transformation logic will be ported to the Spring Component.

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 Spring component and finally reproducing Java CAPS HL7 solution functionality in an equivalent SOA Suite 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, 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 11g R1 PS2 installation with the requisite software, perhaps as a result of following instructions documented in the article “Installing Oracle SOA Suite 11g for HL7 Exploration” at <http://blogs.czapski.id.au/2010/06/installing-oracle-soa-suite-11g-for-hl7-exploration>.

It is also assumed that the reader has at least a modest familiarity with the SOA Suite B2B HL7 support, perhaps as a result of following a series of articles on the topic I published on my blog: <http://blogs.czapski.id.au/?s=soa+suite+hl7>.

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 6.2 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>.

It is assumed that the Spring technology is available in JDeveloper. If it is not, as will become obvious when we try to create a Spring Context and cannot, it is necessary to update JDeveloper. Pull down the Help menu, choose check for updates, locate Spring technology update and add it. I strongly suggest that you back up your entire SOA Suite installation before you do this.

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 11g, using the Spring Component to preserve HL7 mapping and transformation rules and using the Oracle SOA Suite B2B HL7 infrastructure 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 Spring Components (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 containing the compiled JCD and related JAR files
4. Obtain the JCD Java source
5. Create a Java Project
6. Add JARs to the Java project
7. Create, build and exercise the Stand-alone Java application
8. Create a SOA Suite Mediator project
9. Identify and copy to SOA Suite project Java CAPS JARs needed by the JCD code which will be migrated
10. Create a Java Interface corresponding to the Java Class which will contain the JCD code
11. Create the Java Class that implements the interface and migrate JCD code to it
12. Create Spring Context that exposes the Java class
13. Create a SOA Composite and a Mediator that uses the Spring Component
14. Deploy the SOA project
15. Configure B2B trading partnership agreements for inbound and outbound messaging
16. Test/Exercise the solution

There are quite a few steps but then there would be an equal or 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 are simple and short in duration.

Anatomy of a Java CAPS HL7 Solution

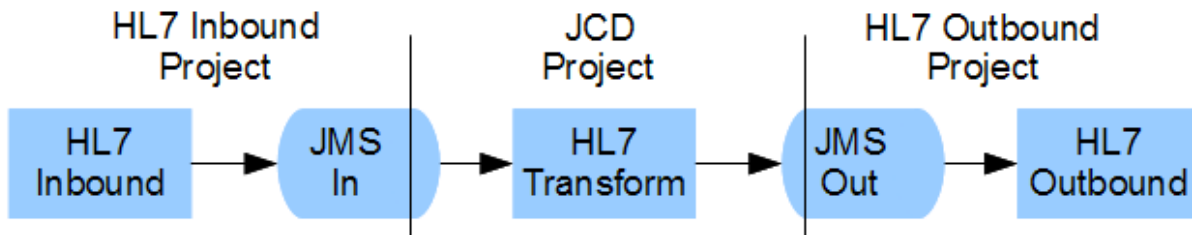
No Java Collaboration Definition stands alone. Each is triggered to process a message and each, in turn, may trigger other components, typically causing delivery of a message to/through one or more outbound connectors/adapters. Functionality provided by the set of Java CAPS connectors/adapters (in Java 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 11g R1 PS2. This means that the adapter-specific code in JCDs will have to be rewritten or removed 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.

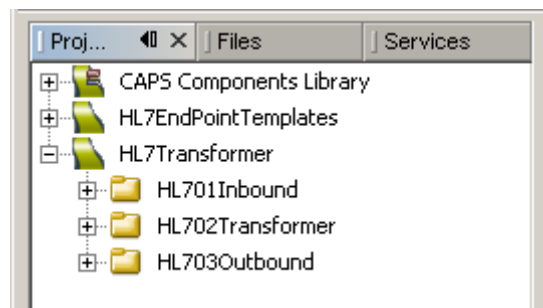


Illustration 1: HL7 Transformer Project Hierarchy

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

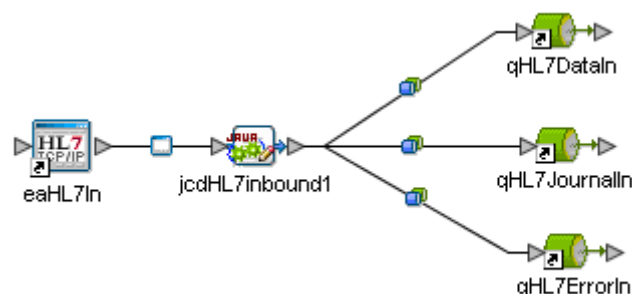


Illustration 2: HL7 Inbound Connectivity Map

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.

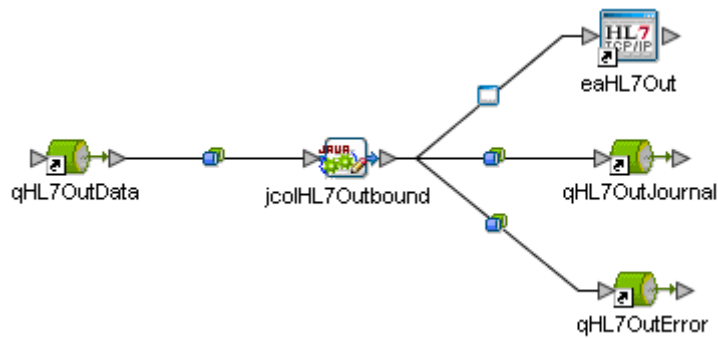


Illustration 3: HL7 Outbound Connectivity Map

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.

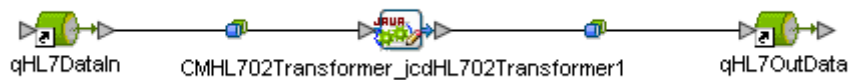


Illustration 4: HL7 Transformer Connectivity Map

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 B2B HL7 infrastructure with correctly configured Trading Partners and Trading Partnership Agreements, therefore there will be no further discussion of these projects. The HL702Transformer project's JCD, jcdHL702Transformer, will be migrated to the Oracle SOA Suite Spring Component therefore it will be analysed in detail.

Analyzing 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.

```

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

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.

Analyzing 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.

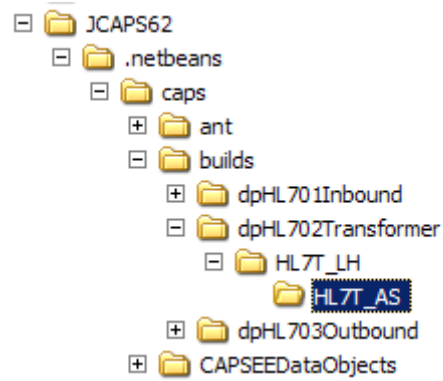


Illustration 5: HL702Transformer EAR directory

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.

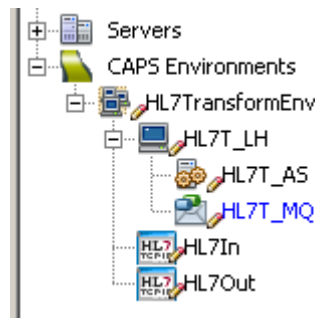


Illustration 6: Java CAPS Environment

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

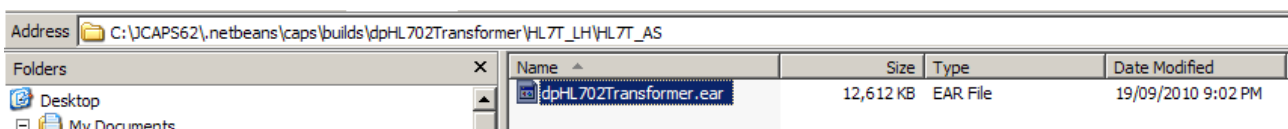


Illustration 7: EAR file, dpHL702Transformer.ear

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.stc.codegen.alerterapi.jar
com.stc.codegen.alerterimpl.jar
com.stc.codegen.loggerapi.jar
com.stc.codegen.loggerimpl.jar
com.stc.codegen.utilapi.jar
com.stc.codegen.utilimpl.jar
com.stc.codegenapi.jar
com.stc.codegenmbeans.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.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.stc.codegen.alerterapi.jar
com.stc.codegen.alerterimpl.jar
com.stc.codegen.loggerapi.jar
com.stc.codegen.loggerimpl.jar
com.stc.codegen.utilapi.jar
com.stc.codegen.utilimpl.jar
com.stc.codegenapi.jar
com.stc.codegenmbeans.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.fwrunapi.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`.

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 Spring Component, we can attempt to create a stand-alone Java application that will implement the transformation. If this task cannot be accomplished it is unlikely that the migration effort will succeed. This section walks through the process.

Let's create a JDeveloper application, let's say `HL7TransformApp`, and a Java Project within it, let's call it `HL7Transform`.

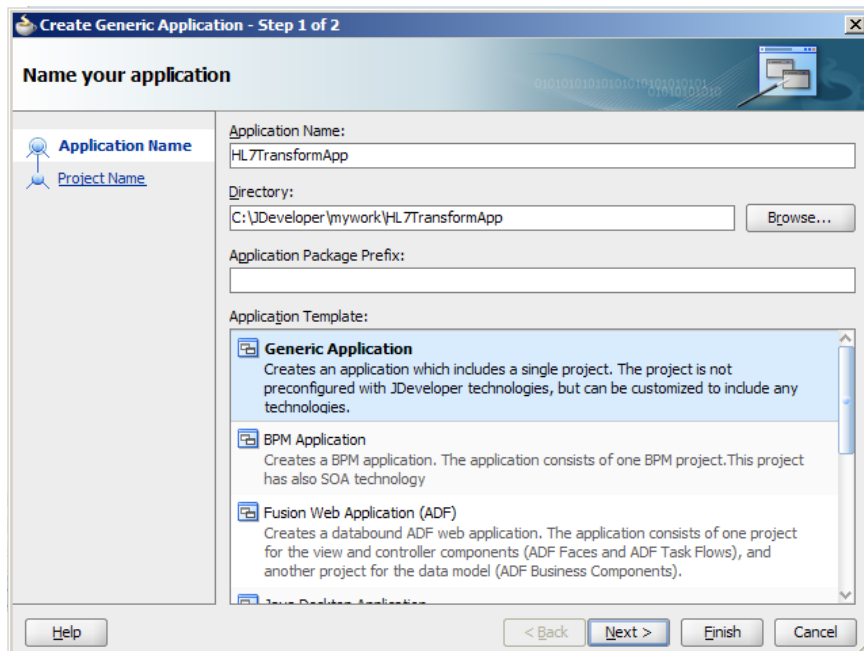


Illustration 8: Create a Generic Application

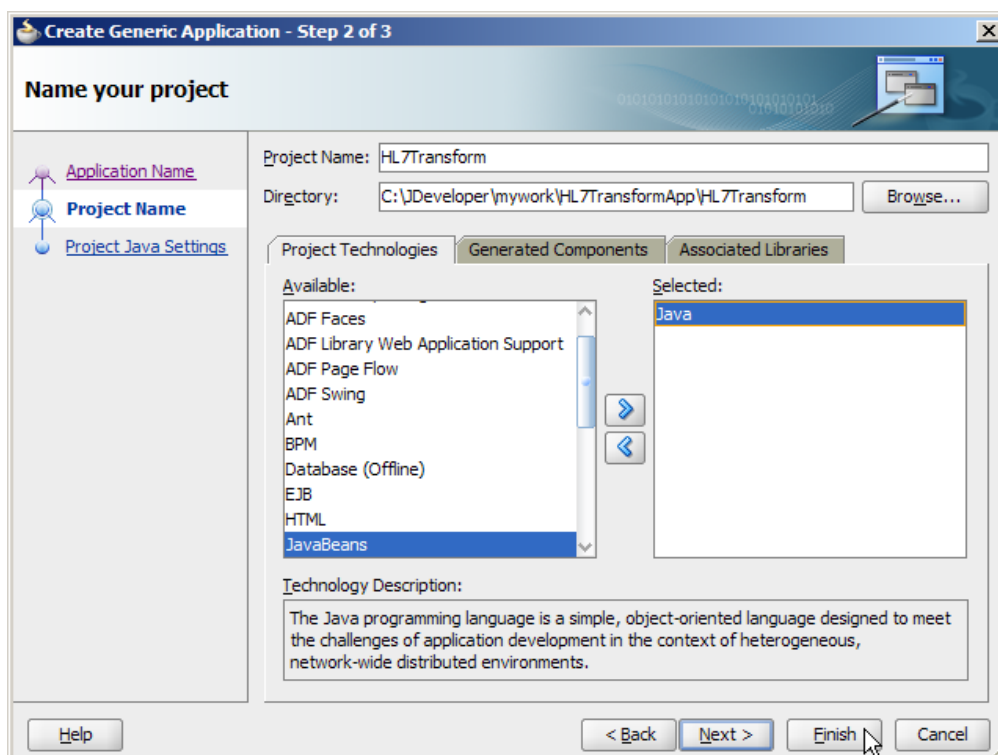


Illustration 9: Create a Java Project

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

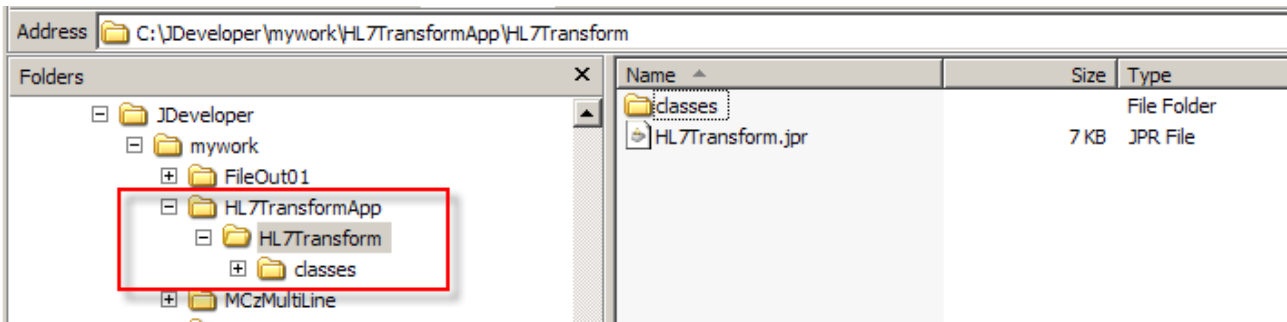


Illustration 10: Empty Java project directory hierarchy

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.

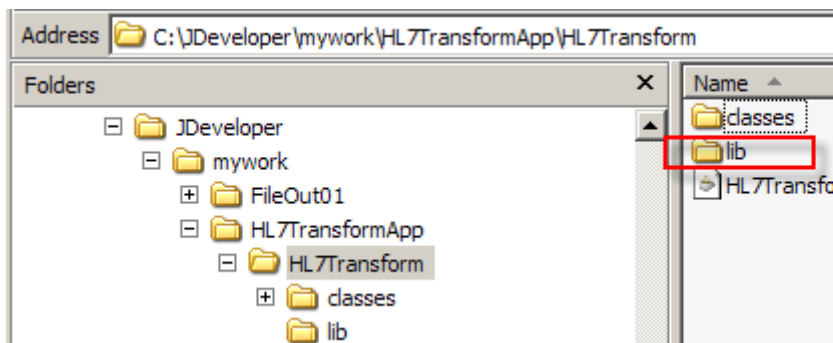


Illustration 11: New lib directory

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.

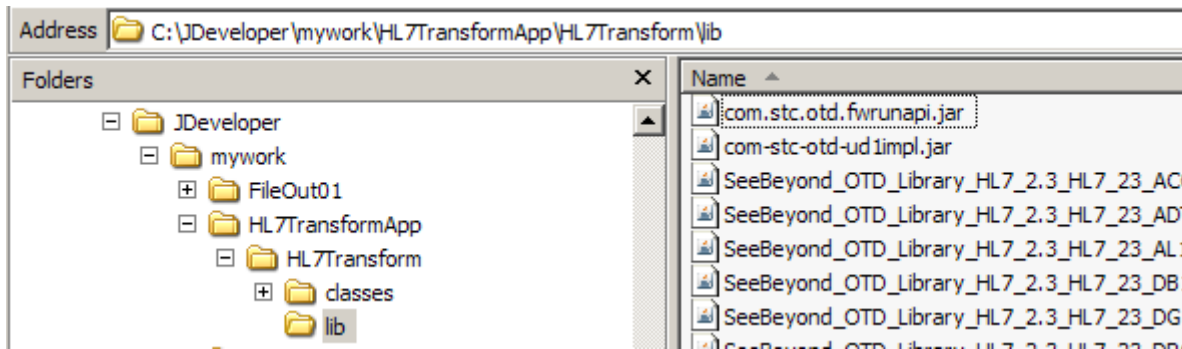


Illustration 12: Populated lib directory

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.

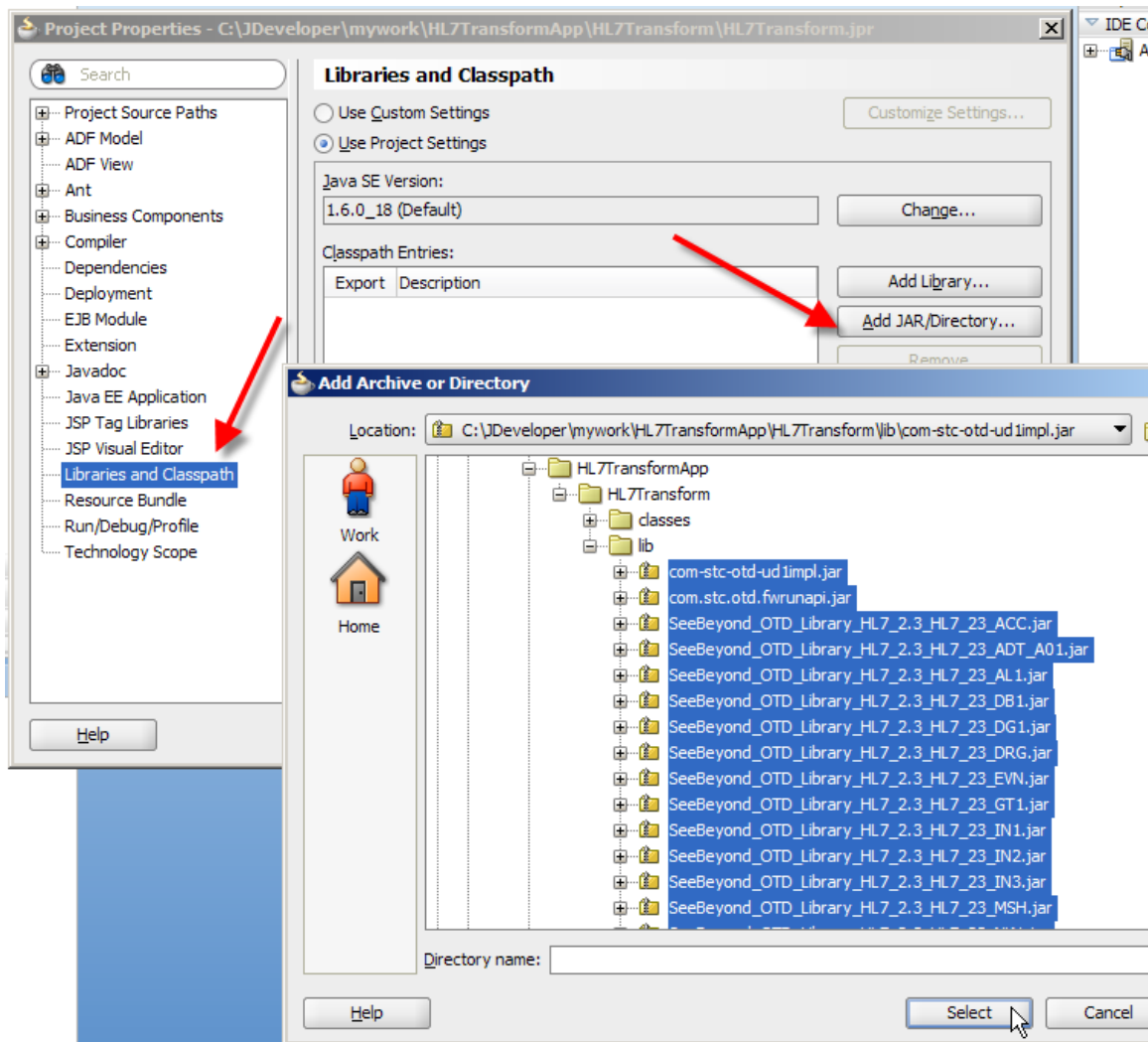


Illustration 13: Select all JARs added to the project

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.

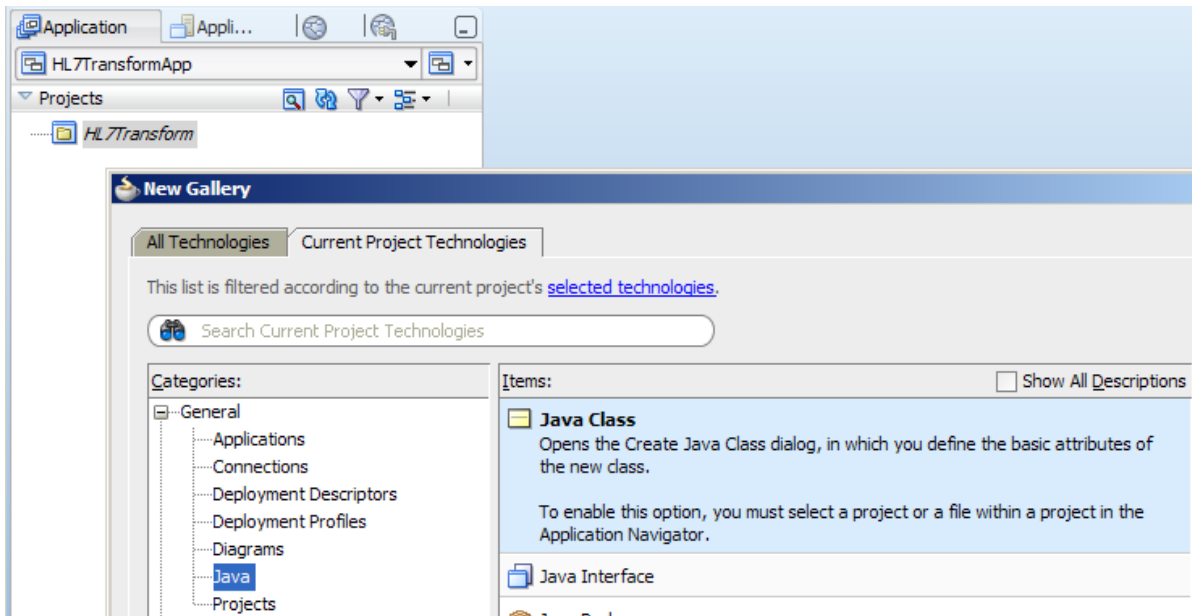


Illustration 14: Create new Java Class

Name this new class HL7Transform and click OK.

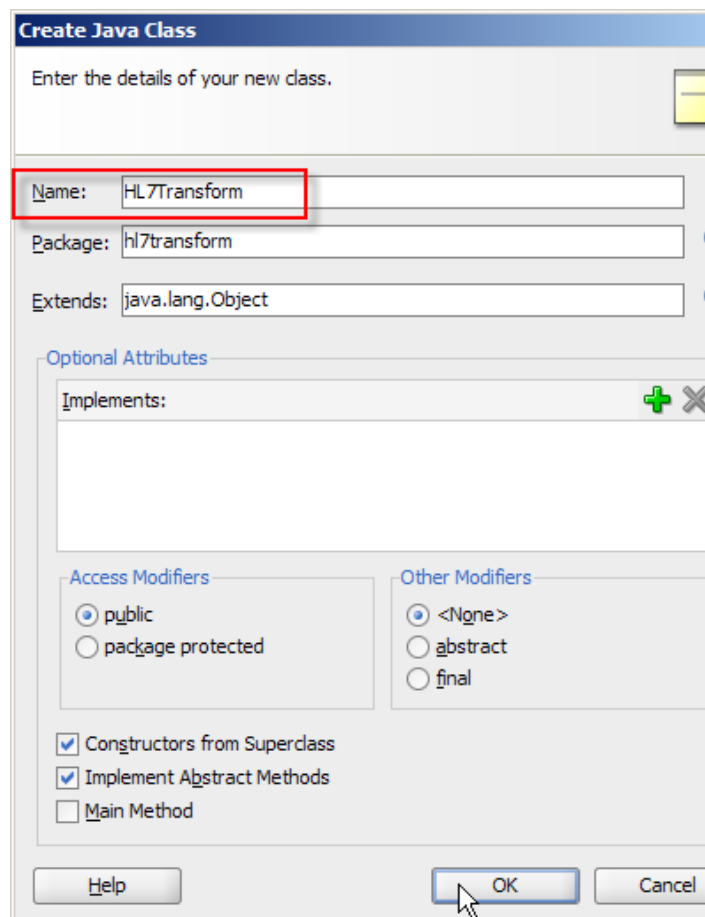


Illustration 15: Name new Java Class

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 vA
        , com.stc.SeeBeyond.OTD_Library.HL7.X_2_4.HL7_24_ADT_A04.ADT_A04 vA
        , com.stc.connectors.jms.JMS vJMSOut )
        throws Throwable
    {
        vA01_23.unmarshalFromMessage( 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 byte array, presumably containing the HL7 v2.3 ADT A01 message, and which returns a byte array containing the HL7 v2.4 ADT A04 message.

```

public byte[] HL7Transform(byte[] baA01In) 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.unmarshalFromBytes (baA01In);

    receive (vA01_23, vA04_24);

    byte[] baA04Out = vA04_24.marshalToBytes ();

    return baA04Out;
}

```

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.

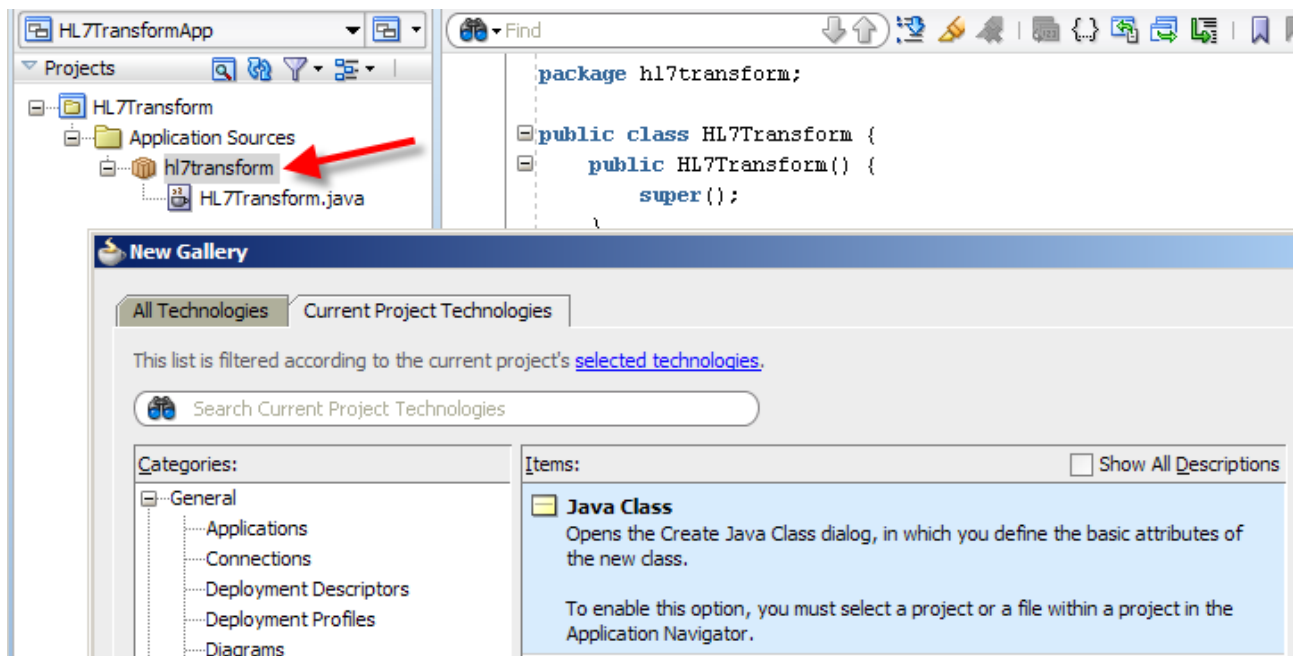


Illustration 18: Create a new Java class wizard

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

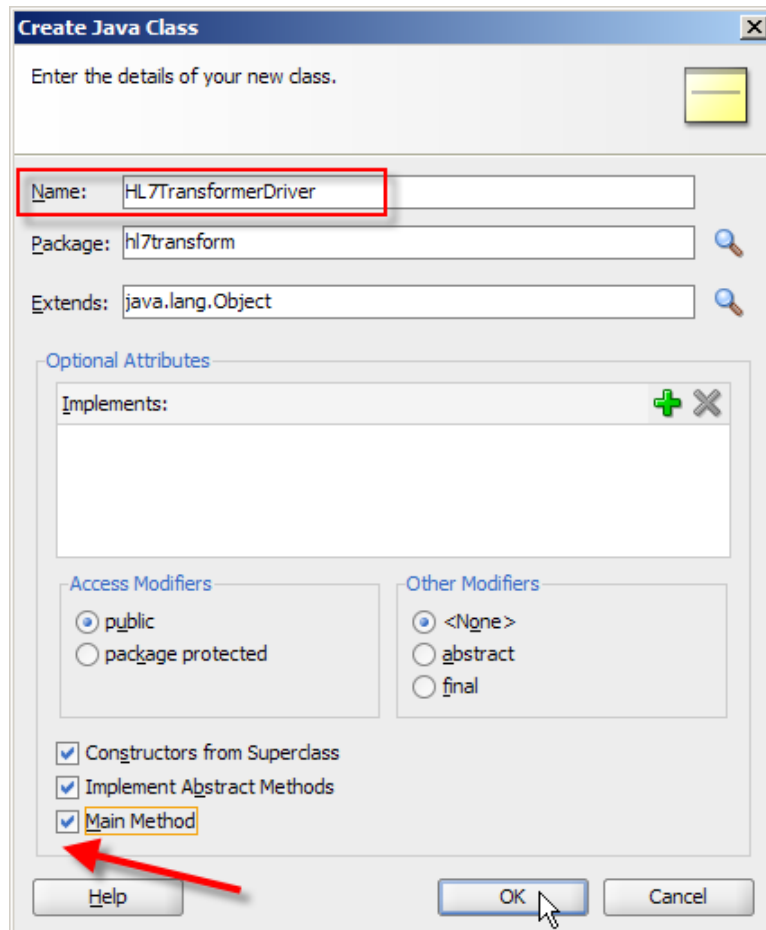


Illustration 19: Complete class skeleton creation

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 byte[] constant in the driver class. Add the following code before the main method.

```

private static byte[] baA01In = ("
+ "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")
.getBytes();

```

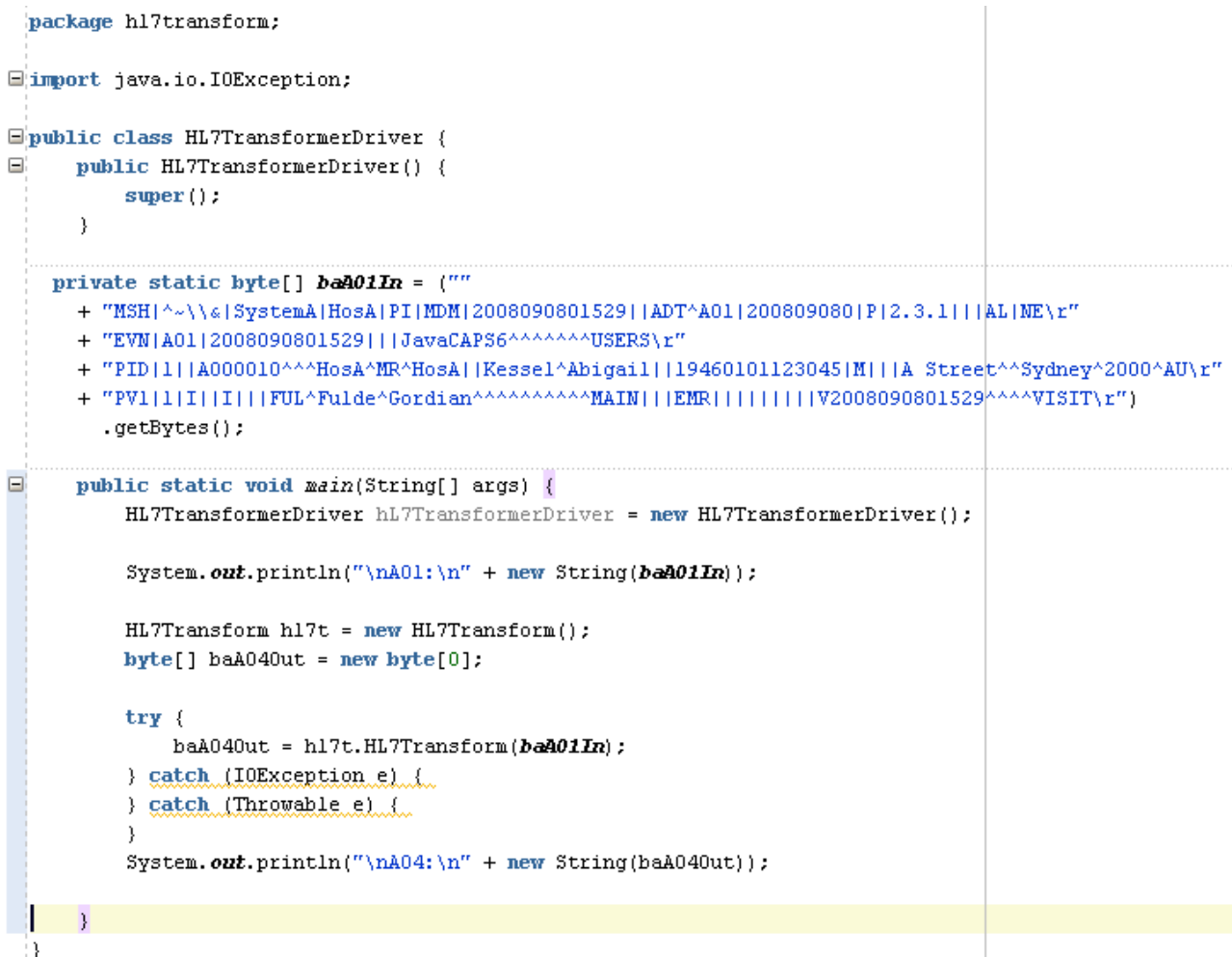
Add the following code to the body of the main method:

```
System.out.println("\nA01:\n" + new String(baA01In));

HL7Transform hl7t = new HL7Transform();
byte[] baA04Out = new byte[0];

try {
    baA04Out = hl7t.HL7Transform(baA01In);
} catch (IOException e) {
} catch (Throwable e) {
}
System.out.println("\nA04:\n" + new String(baA04Out));
```

The result will look like in the following figure.

The image shows a screenshot of an IDE with the code for the HL7TransformerDriver class. The code is as follows:

```
package hl7transform;

import java.io.IOException;

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

    private static byte[] baA01In = {""
+ "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"
+ "PVL|1|I||I|||FUL^Fulde^Gordian^^^^^^^MAIN|||EMR|||||||V2008090801529^^^^^VISIT\r"}
        .getBytes();

    public static void main(String[] args) {
        HL7TransformerDriver hl7TransformerDriver = new HL7TransformerDriver();

        System.out.println("\nA01:\n" + new String(baA01In));

        HL7Transform hl7t = new HL7Transform();
        byte[] baA04Out = new byte[0];

        try {
            baA04Out = hl7t.HL7Transform(baA01In);
        } catch (IOException e) {
        } catch (Throwable e) {
        }
        System.out.println("\nA04:\n" + new String(baA04Out));
    }
}
```

Illustration 21: Driver class

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

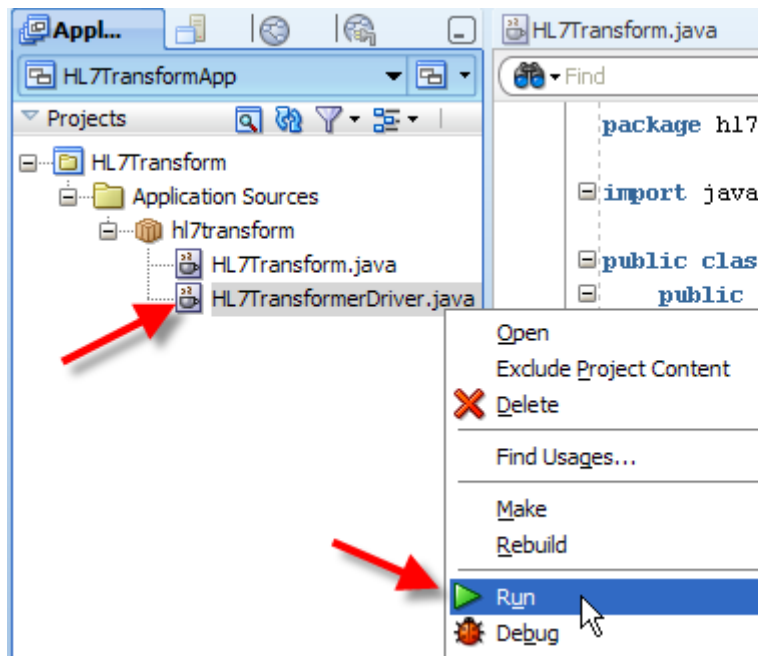


Illustration 22: Run the driver class

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

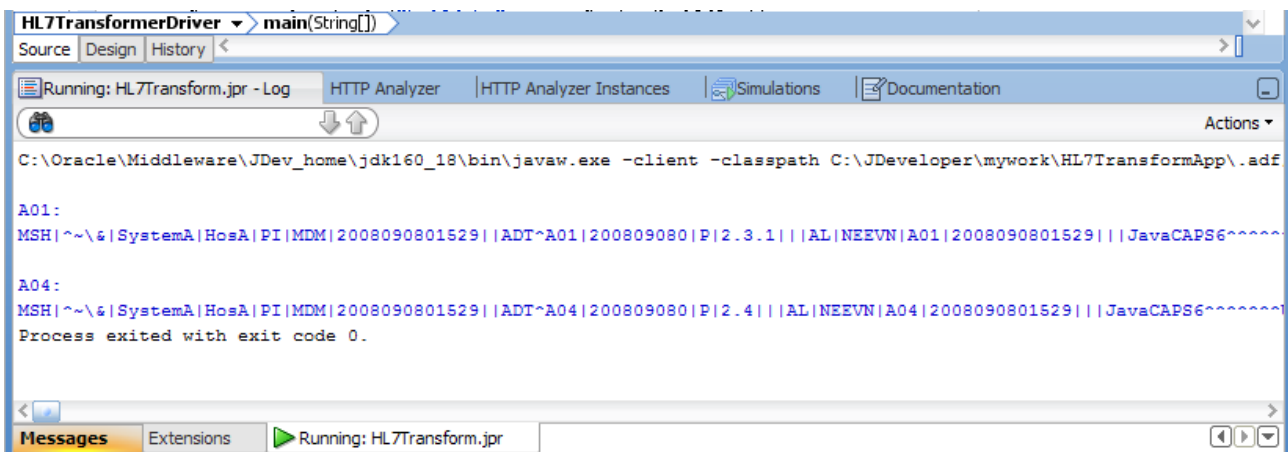


Illustration 23: Execution trace

This works fine as a stand-alone Java application. To make it usable as a Spring component we need to roll the HL7Transform and receive methods into one. We simply take the body of the receive method and paste it over the top of the receive method invocation, deleting the invocation and the remaining receive method signature.

The original skeleton looks like this:

```
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
    {

        vA04_24.getMSH().setMSH_segment_ID( vA01_23.getMSH().getMSH_segment_ID() );
        vA04_24.getMSH().setMsh1FieldSeparator( vA01_23.getMSH().getMsh1FieldSeparator() );
        ...
        vA04_24.getPv1().setPv12PatientClass( vA01_23.getPv1().getPv12PatientClass() );
        ;

    }

    public byte[] HL7Transform(byte[] baA01In) 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.unmarshalFromBytes( baA01In );

        receive( vA01_23, vA04_24 );

        byte[] baA04Out = vA04_24.marshalToBytes();

        return baA04Out;
    }
}

```

The reworked skeleton will look like this:

```

package hl7transform;

/* delete these lines
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
    {

    }
*/

    public byte[] HL7Transform(byte[] baA01In) 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.unmarshalFromBytes( baA01In );

        receive( vA01_23, vA04_24 );

        vA04_24.getMSH().setMSH_segment_ID( vA01_23.getMSH().getMSH_segment_ID() );
        vA04_24.getMSH().setMsh1FieldSeparator( vA01_23.getMSH().getMsh1FieldSeparator() );
        ...
        vA04_24.getPv1().setPv12PatientClass( vA01_23.getPv1().getPv12PatientClass() );
        ;

        byte[] baA04Out = vA04_24.marshalToBytes();
    }
}

```

```
    return baA04Out;
}
}
```

Build and run the project to make sure it still works.

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 later 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.

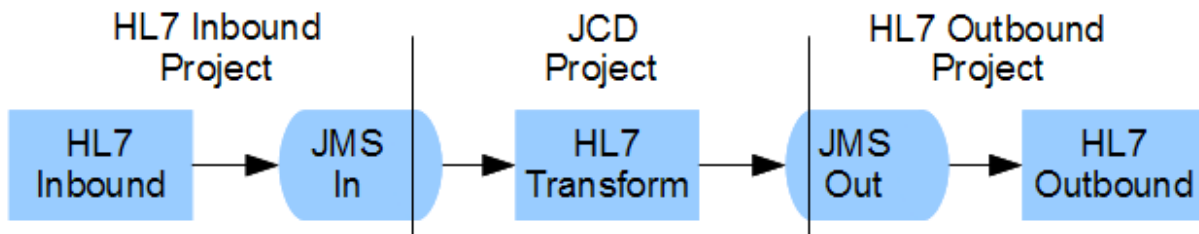
Oracle developed a series of classes and interfaces, packaged in a JAR called `jcaps_interfaces.jar`, to facilitate porting JCDs which make use of logger, alerter, collabContext and typeConverter. I have seen and used some of these classes but I am not aware, at this time, how a customer would go about 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.

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.

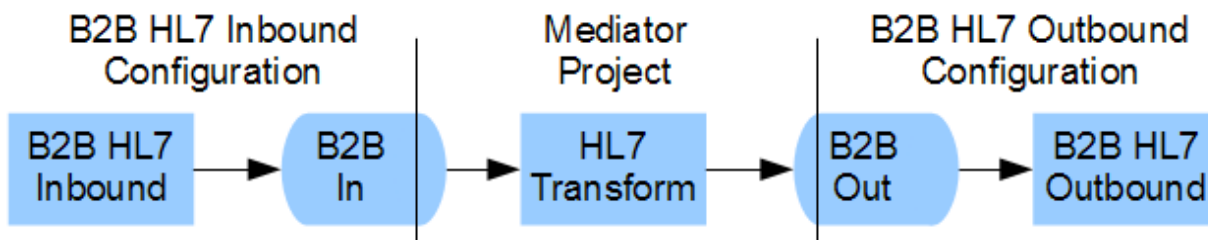
Basic SOA Suite Solution with Spring Component

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.



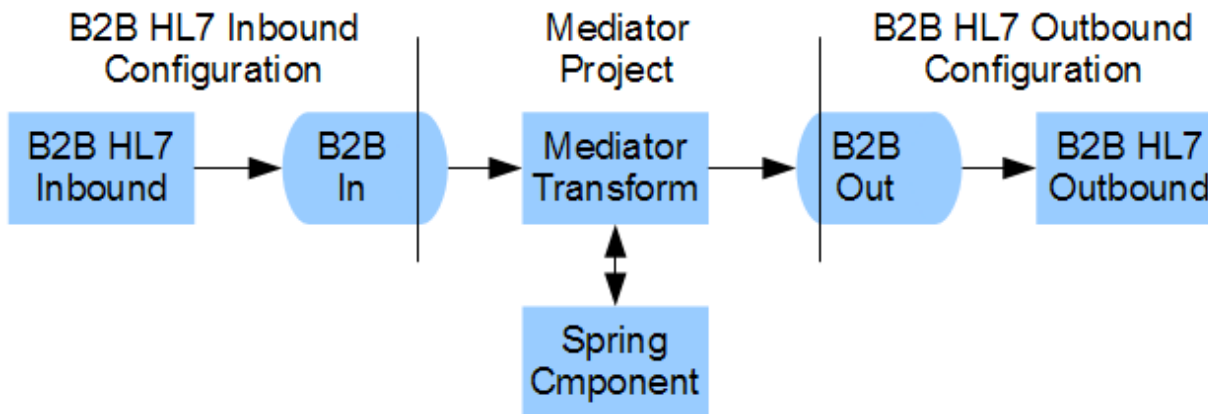
Drawing 2: Java CAPS HL7 Solution

The equivalent SOA Suite solution, which will reuse the JCD transformation code, will look very similar, as shown in the following figure.



Drawing 3: SOA Suite "Equivalent" reusing JCD transformation code

Expanding slightly, to more realistically represent the components involved in porting JCD code, we see the following:

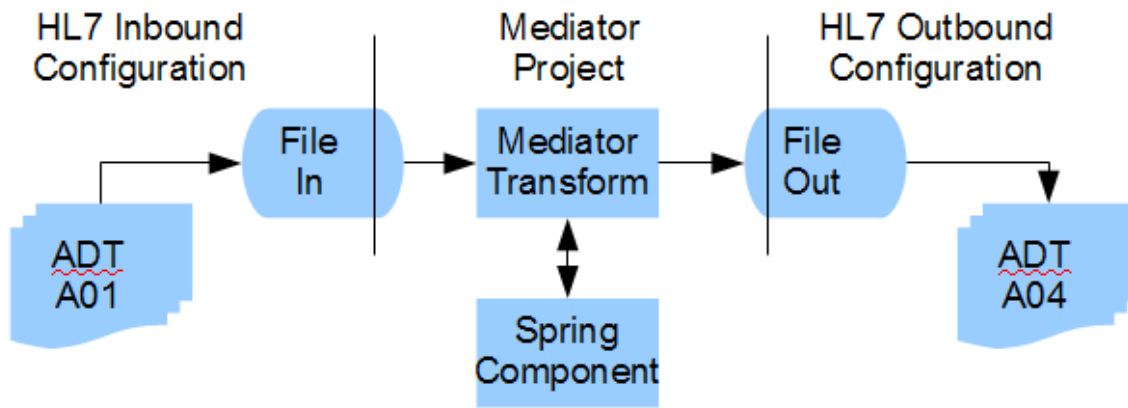


Drawing 4: Expanded Mediator Project

The Mediator Project consists of an inbound adapter, which receives HL7 v2 Delimited messages, a Mediator XSL transform, which invokes the Spring Component and passes transformed messages to the outbound adapter. The boundary between the Mediator Project and the inbound and outbound adapters is crossed by HL7 v2 Delimited messages. For the purpose of porting the JCD code to the Spring Component, and testing the port, we can use basic File Adapters as both the inbound and the outbound adapters. This will make this section of the article easier to follow and will again concentrate on the essentials.

Stage I: Basic File to File Mediator project

The solution to be built in this section is represented in Drawing 5.



Drawing 5: Simplified solution

Let's proceed to configure the inbound and outbound adapters, assuming the HL7 files to process will come from `c:\hl7\adt\data` and transformed messages will be written to files in `c:\hl7\received`.

In JDeveloper's HL7TransformApp application create a new SOA Project called HL7TranformFF.

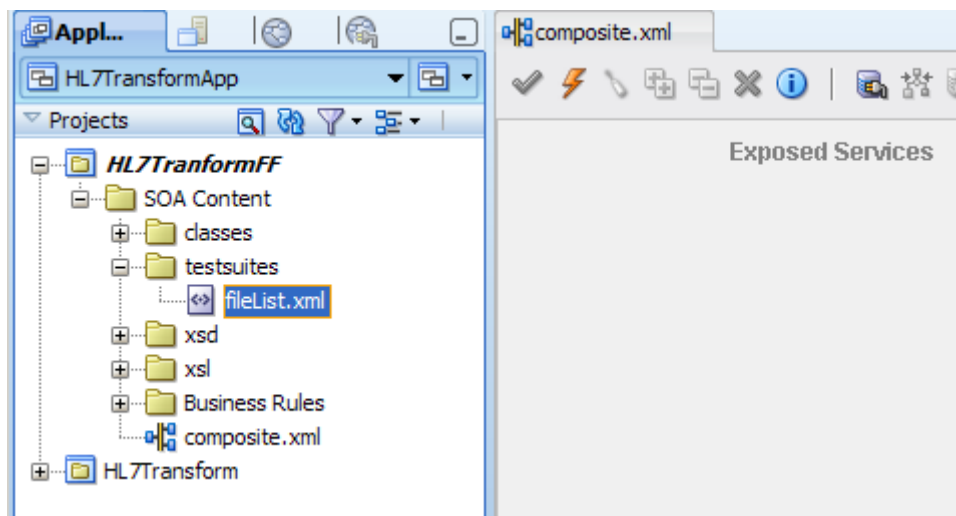


Illustration 24: New SOA Project HL7TranformFF

Drag a File Adapter from the SOA Palette to the Composite canvas and drop it in the Exposed Services swim line.

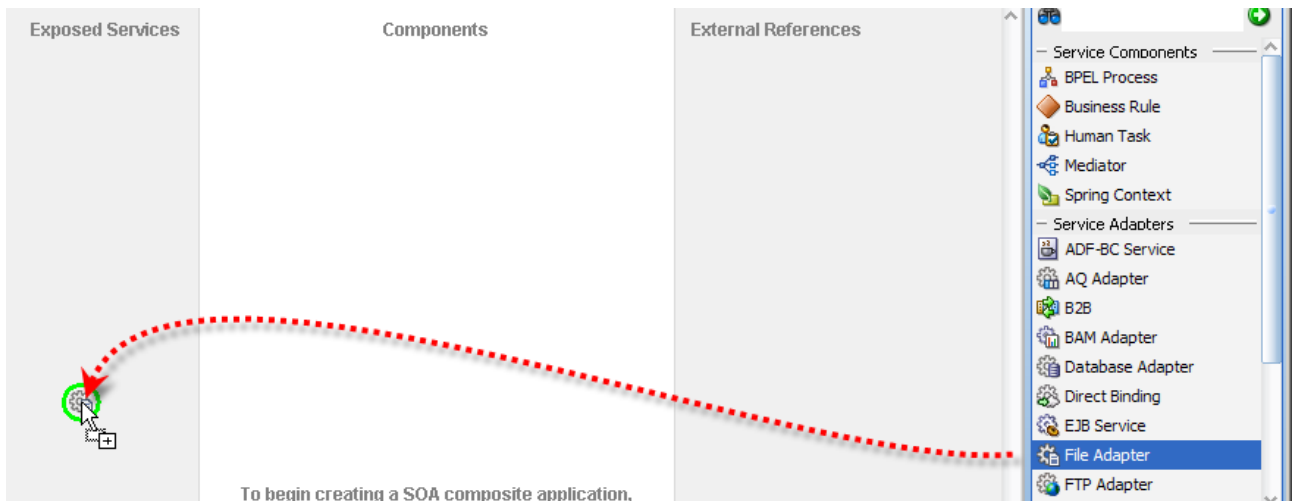


Illustration 25: Add inbound File Adapter to the composite

Configure:

Service Name	HL7TransformFFFileIn
Interface	Define from operations and schema (defined later) – leave default
Operation Type	Read File
Directory for Incoming Files	c:\hl7\adt\data (or some other directory which <u>does not</u> contain HL7 files)
Process files recursively	Unchecked
Include files with name Pattern	AT_A01_output_1.hl7
Polling Frequency	5 seconds
Native format translation is not required (Schema is Opaque)	Checked – is is critical to check this checkbox – HL7 v2 Delimited messages are not XML and must not be processed by the infrastructure before they are handed over to the Spring Component for transformation

Table 1 Inbound File Adapter Configuration

Drag a File Adapter from the SOA Palette to the Composite canvas and drop it in the External References swim line.

Configure:

Service Name	HL7TrasformFFFFileOut
Interface	Define from operations and schema (defined later) – leave default
Operation Type	Write File
Directory for Outgoing Files	C:\hl7\received

File Naming Convention	ADT_A04_outbound_%SEQ%.hl7
Native format translation is not required (Schema is Opaque)	Checked – is is critical to check this checkbox – HL7 v2 Delimited messages are not XML and must not be processed by the infrastructure after they are produced by the Spring Component transformation

Table 2 Outbound File Adapter Configuration

At this stage the composite canvas looks like that shown below.



Illustration 26: Composite canvas with inbound and outbound file adapters

Before we get into the Spring Component work let's add a simple mediator XSL transform, which copies input to output, deploy and test the project. This will ensure our file adapters are configured correctly and the project works so far.

Drag Mediator from the SOA Palette to the canvas and drop it in the Components swim line. Name it HL7TransformFFMediator.

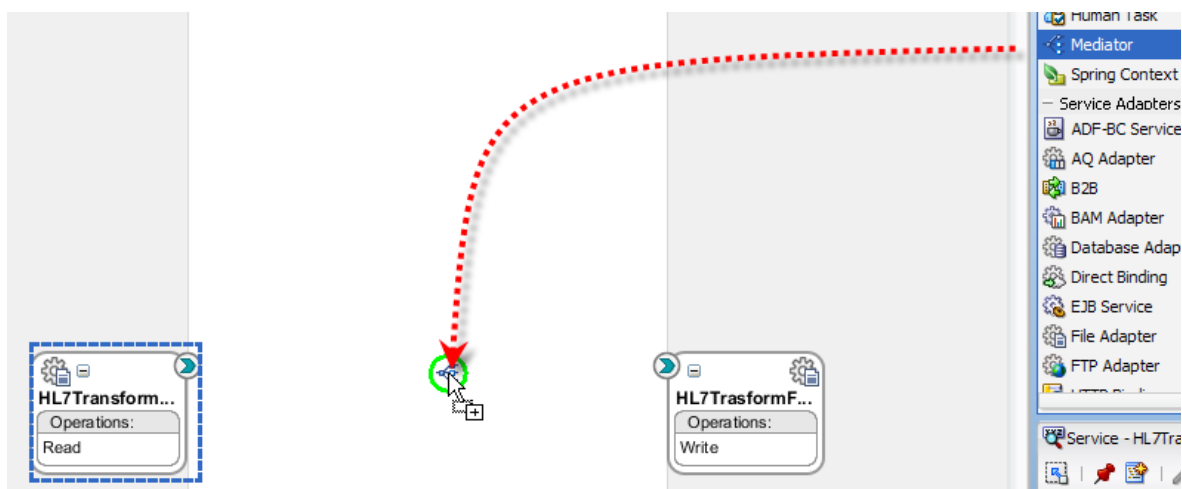


Illustration 27: Add Mediator component

Connect the inbound file adapter to the Mediator.

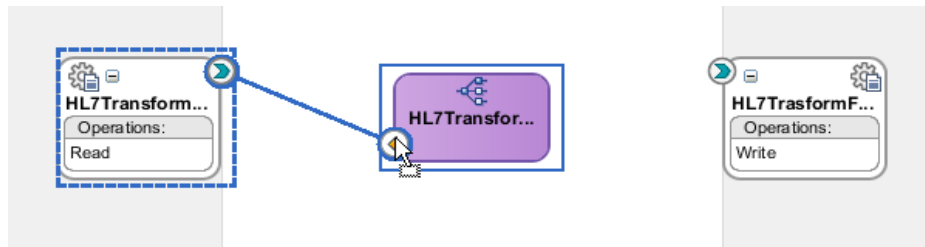


Illustration 28: Connect inbound file adapter to the mediator component

Connect the Mediator component to the outbound file adapter.

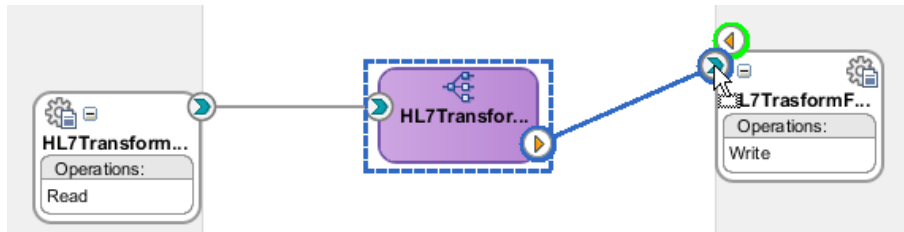


Illustration 29: Connect mediator component to the outbound file adapter

Double-click the Mediator component to open the mplan.

Click the Create New Mapper File button.

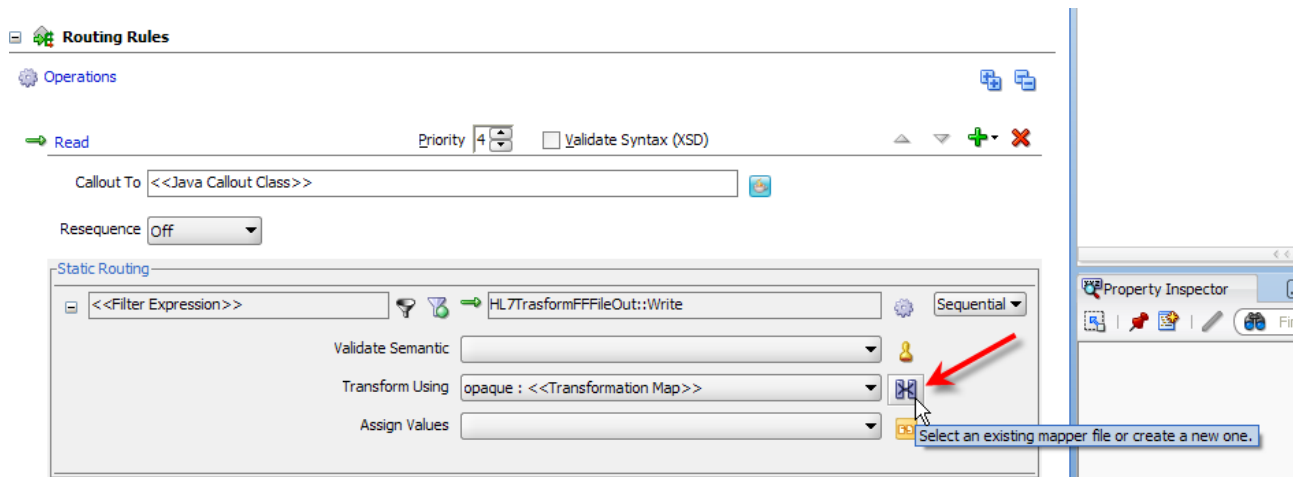


Illustration 30: Create a new mapper file

Check the “Create New Mapper File” radio button and click OK.

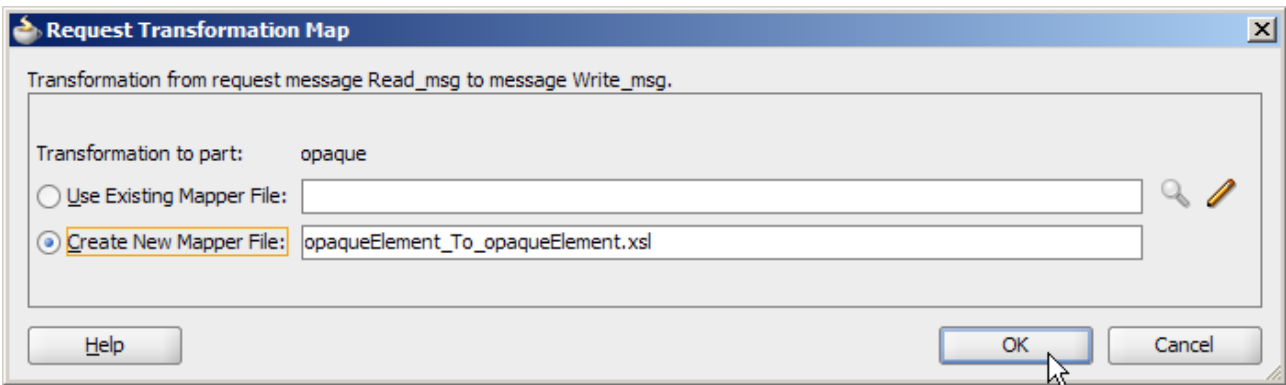


Illustration 31: Choose to create a new mapper file

Connect the two `opaque:opaqueElement` nodes by dragging from the left to the right. Select one of the nodes and note the datatype: `base64Binary`.

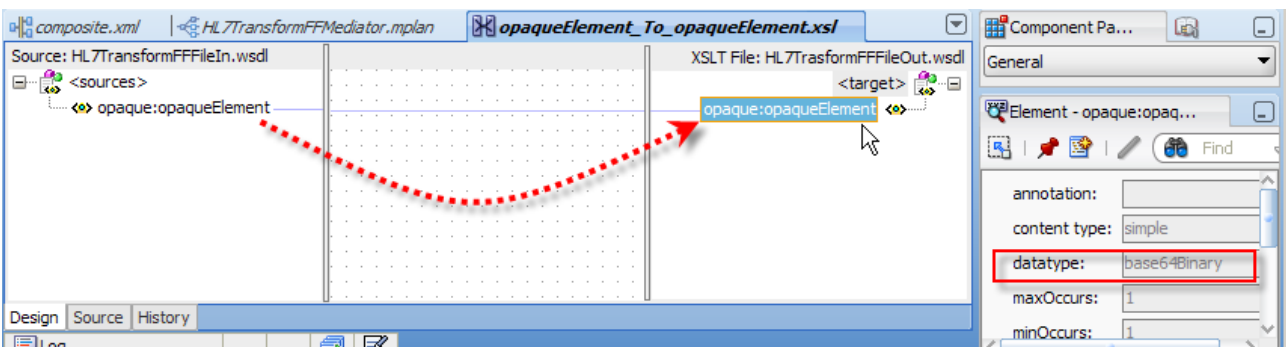


Illustration 32: Complete mapping and note the datatype

The datatype, `base64Binary`, will become significant when we start working with the Spring Component in a short while.

Save and close the XSL. Save and close the mplan. Save the Composite.

Right-click the name of the project and choose `Deploy` → `HL7TransformFF...`

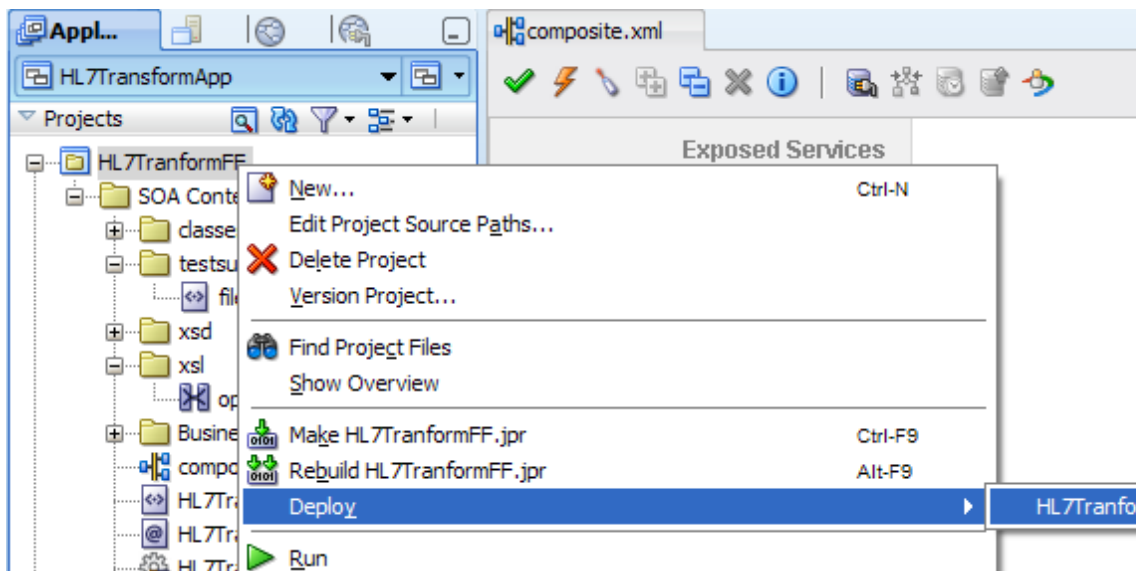


Illustration 33: Start the deployment wizard

Accept the default: Deploy to application server.

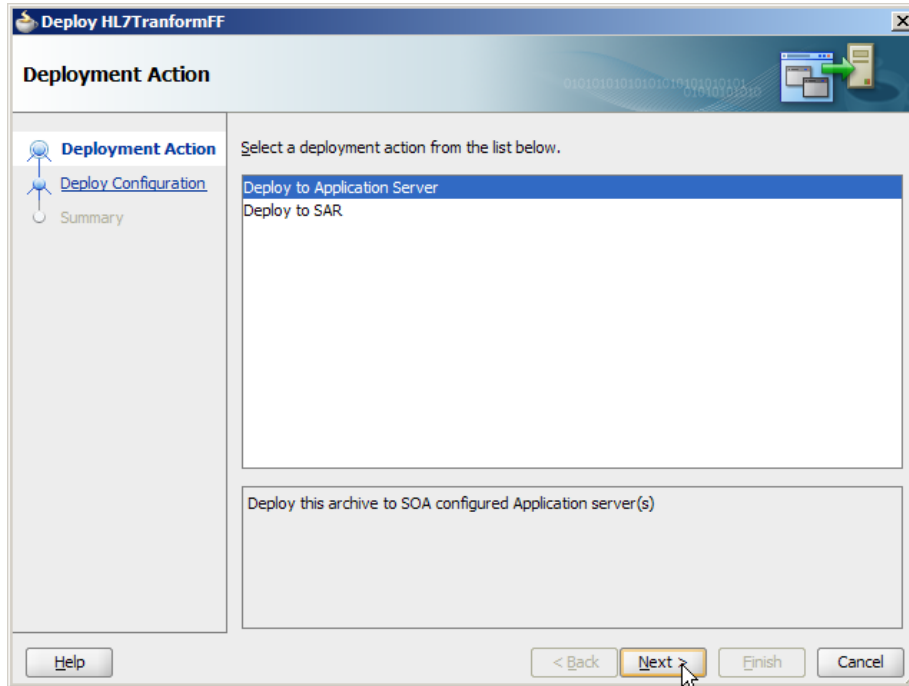


Illustration 34: Accept default deployment action

Check "Overwrite any existing composites with the same revision ID" and click Next.

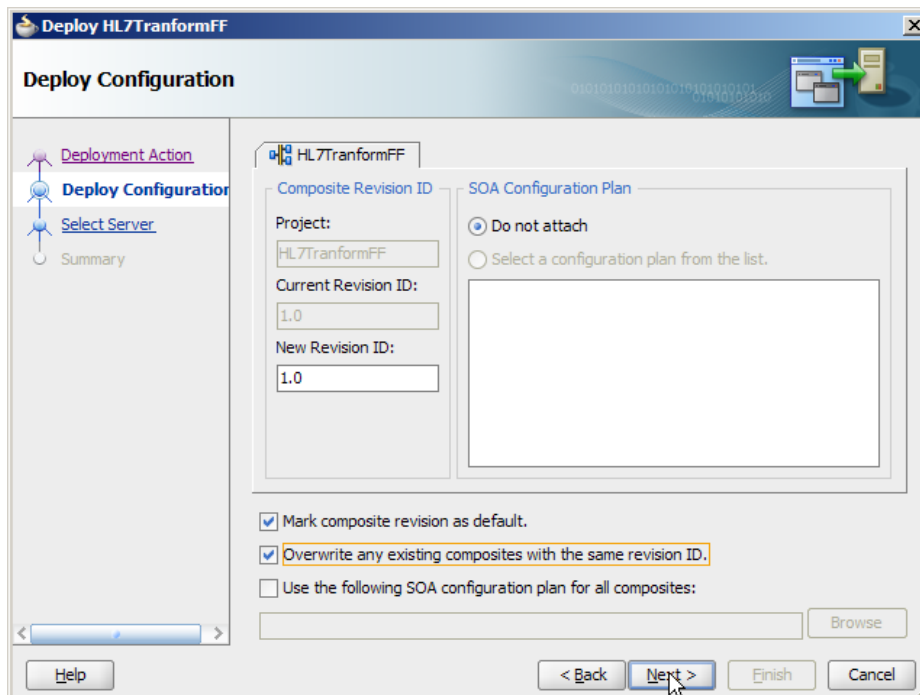


Illustration 35: Overwrite existing revision

Choose the correct application server connection and click Finish.

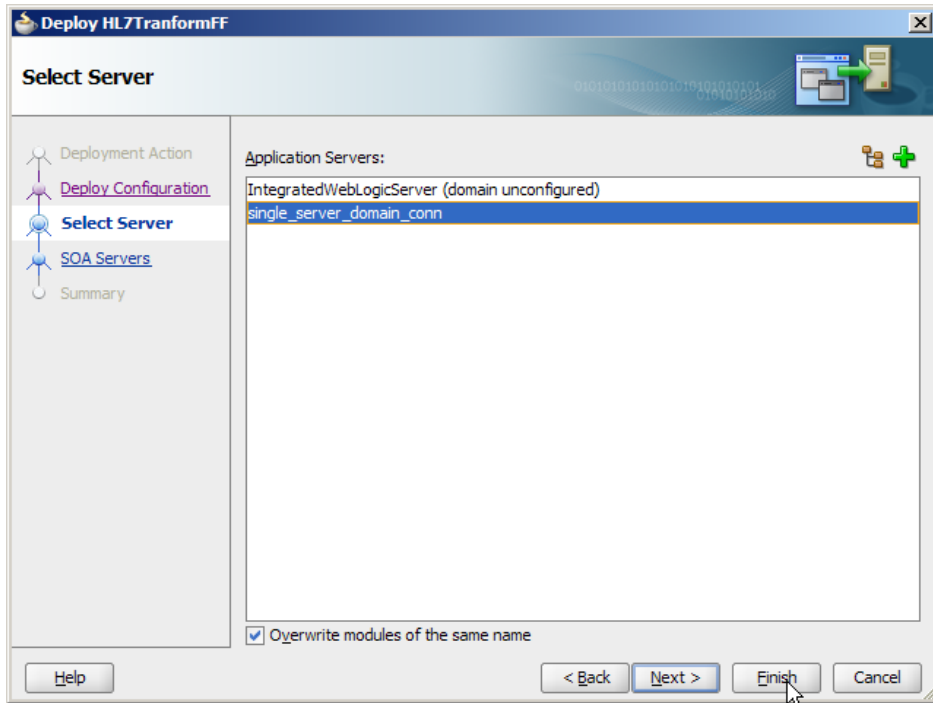


Illustration 36: Choose appropriate application server connection

In due course the log window will show deployment completion.

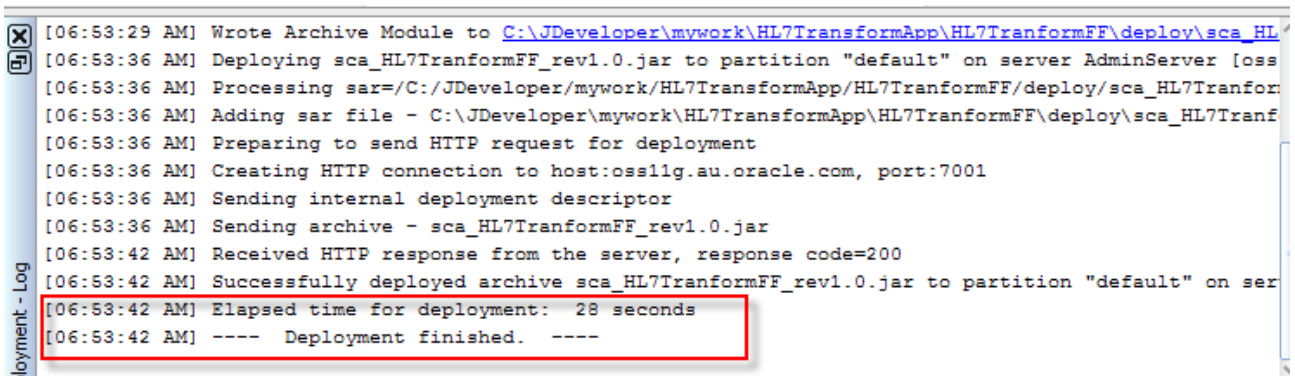


Illustration 37: Deployment feedback

We are about to exercise this solution. If you are interested in runtime feedback from the Mediator then use the Enterprise Manager to set selected logging categories to TRACE:32.

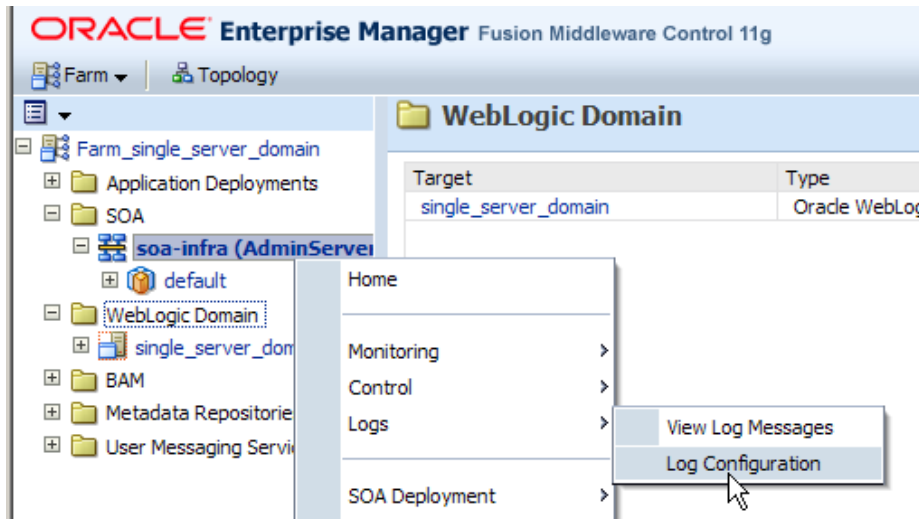


Illustration 38: Getting to the log configuration

Click Log Levels Tab, expand oracle.soa, expand oracel.soa.mediator.

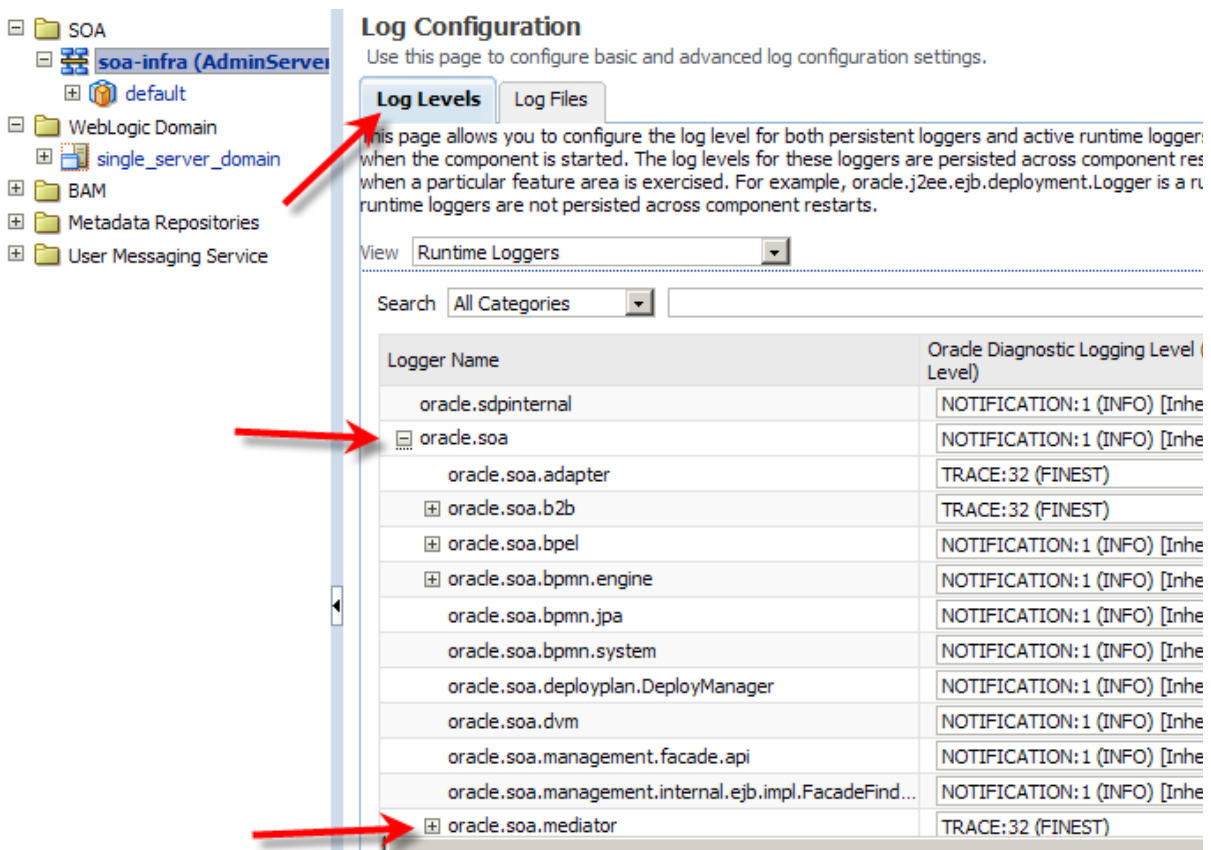


Illustration 39: Locating logging categories to change

Enable TRACE:32 logging for oracle.soa.adapter by pulling down the dropdown and choosing the correct logging level.

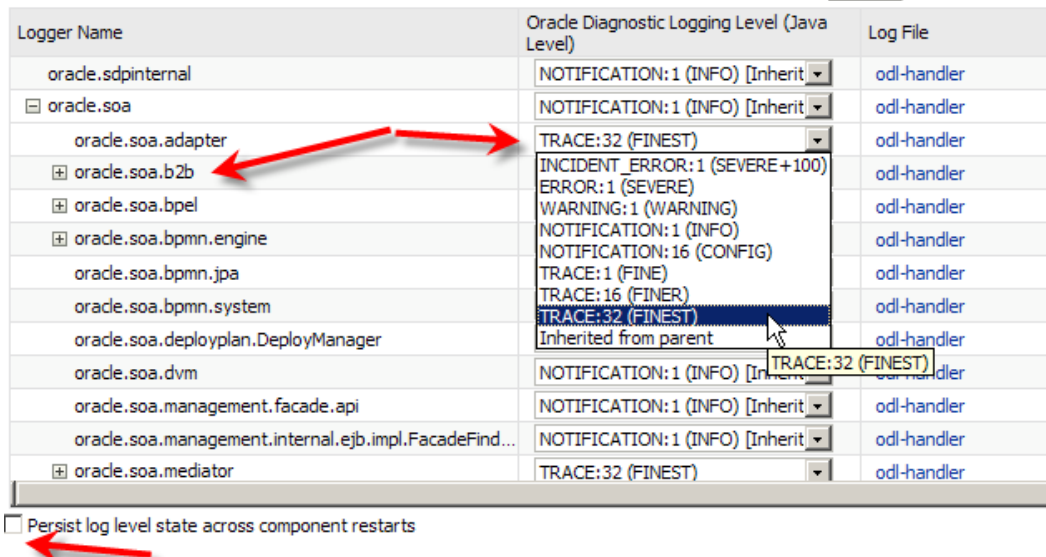


Illustration 40: Enable TRACE:32 logging for oracle.soa.adapter

Do this also for oracle.soa.mediator. Experiment with including and excluding logging subcategories until you see what you need to see and no more (this is an iterative process).

When you have all levels configured as desired check the “Persist log level state ...” checkbox and click Apply.

Make sure that your output directory is empty (for me c:\hl7\received).

Copy ADT_A01_output_1.hl7 from the source directory to the directory being polled by the inbound file adapter (for me c:\hl7\adt\data). Within 5 seconds the file will be picked up and its content copied to a file in the output directory.

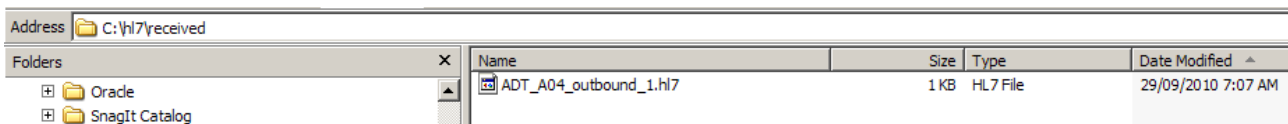


Illustration 41: Output file was produced

The file content was “translated” and written to the output.

If you increased logging levels for the oracle.soa.adapter and oracle.soa.mediator categories you will be able to see “interesting” feedback in the AdminServer-diagnostic.log. For example:

And more

```
[2010-09-29T07:07:32.550+10:00] [AdminServer] [TRACE] [] [oracle.soa.adapter] [tid:
weblogic.work.j2ee.J2EEWorkManager$WorkWithListener@5e4698f6] [userId: weblogic] [ecid:
0000IhQ7nWE2zGWjLxmJOA1CcZM00001Fm,1:29241] [SRC_CLASS:
oracle.integration.platform.blocks.adapter.fw.log.LogManagerImpl] [APP: soa-infra] [dcid:
02d1ff4621073810:-598e0369:12b59a0c1f8:-8000-00000000000001ca] [SRC_METHOD: log] File Adapter
HL7TranformFF Poller enqueueing file for processing :C:\hl7\adt\data\ADT_A01_output_1.hl7

[2010-09-29T07:07:32.566+10:00] [AdminServer] [TRACE] [] [oracle.soa.adapter] [tid:
weblogic.work.j2ee.J2EEWorkManager$WorkWithListener@1e7b5db4] [userId: weblogic] [ecid:
0000IhQ7nWE2zGWjLxmJOA1CcZM00001Fm,1:29243] [SRC_CLASS:
oracle.integration.platform.blocks.adapter.fw.log.LogManagerImpl] [APP: soa-infra] [dcid:
02d1ff4621073810:-598e0369:12b59a0c1f8:-8000-00000000000001ca] [SRC_METHOD: log] File Adapter
HL7TranformFF File : C:\hl7\adt\data\ADT_A01_output_1.hl7 is ready to be processed.

...

[2010-09-29T07:07:32.660+10:00] [AdminServer] [TRACE] [] [oracle.soa.adapter] [tid:
weblogic.work.j2ee.J2EEWorkManager$WorkWithListener@1e7b5db4] [userId: weblogic] [ecid:
0000IhQ7nWE2zGWjLxmJOA1CcZM00001Fm,1:29243] [SRC_CLASS:
oracle.integration.platform.blocks.adapter.fw.log.LogManagerImpl] [APP: soa-infra] [dcid:
```



```

02d1ff4621073810:-598e0369:12b59a0c1f8:-8000-00000000000001ca] [SRC_METHOD: log] File Adapter
HL7TranformFF Sending message to Adapter Framework for posting to BPEL engine: {[
  file=C:\hl7\adt\data\ADT_A01_output_1.hl7
}]
...
[2010-09-29T07:07:33.081+10:00] [AdminServer] [TRACE] [] [oracle.soa.mediator.serviceEngine] [tid:
weblogic.work.j2ee.J2EEWorkManager$WorkWithListener@1e7b5db4] [userId: weblogic] [ecid:
0000IhQAzZ32zGwJLxmJOA1CcZM00001OR,0] [SRC_CLASS:
oracle.tip.mediator.serviceEngine.MediatorServiceEngine] [APP: soa-infra] [dcid: 02d1ff4621073810:-
598e0369:12b59a0c1f8:-8000-00000000000001ca] [SRC_METHOD: printMessage] method=post key=opaque
payload [[
  <opaqueElement
xmlns="http://xmlns.oracle.com/pcbpel/adapter/opaque/">TVNIff5+XCZ8U31zdGvtQXxIb3NBfFBJfE1ETXwyMDA4M
DkwODAxNTI5fHxBRFReQTaxfDAwMDAw
MF9DVExJRF8yMDA4MDkwODAxNTI5fFB8Mi4zLjF8fHxBTHxORQ1FVv58QTaxfDIwMDgwOTA4MDE1
Mj18fHxKYXZhQ0FQUzZeXl5eXl5eVvNFU1MNUE1EfdF8fEEwMDAwMTBeXl5Ib3NBXk1SXkhvc0F8
fEt1c3NlbF5BYmlnYWlsfHwXOTQ2MDEwMTEyMzA0NjY3V0aCAzcmQgQ2lyY2x1Xl5E
b3duaGFTIE1hcmtldF5FbmdsYW5kIC0gTm9yZm9sa14zMDgyOF5VS3x8fHx8fHx8QTlwMDgwOTA4
MDE1Mj18fHxKYXZhQ0FQUzZeXl5eXl5eVvNFU1MNUE1EfdF8fEEwMDAwMTBeXl5Ib3NBXk1BSU58fHxFTVJ8
fHx8fHx8fHxWMjAwODAxNTI5DQ0K</opaqueElement>
]]
...
[2010-09-29T07:07:33.753+10:00] [AdminServer] [TRACE] []
[oracle.soa.mediator.service.transformation] [tid:
weblogic.work.j2ee.J2EEWorkManager$WorkWithListener@1e7b5db4] [userId: weblogic] [ecid:
0000IhQAzZ32zGwJLxmJOA1CcZM00001OR,0] [SRC_CLASS:
oracle.tip.mediator.service.transformation.MediatorTransformationHandler] [APP: soa-infra]
[composite_name: HL7TranformFF] [component_name: HL7TransformFFMediator] [component_instance_id:
69647490CB4411DFBFA6C5760DD892B3] [dcid: 02d1ff4621073810:-598e0369:12b59a0c1f8:-8000-
00000000000001ca] [SRC_METHOD: log] [composite_instance_id: 160001] Source message payload :[[
  <opaqueElement
xmlns="http://xmlns.oracle.com/pcbpel/adapter/opaque/">TVNIff5+XCZ8U31zdGvtQXxIb3NBfFBJfE1ETXwyMDA4M
DkwODAxNTI5fHxBRFReQTaxfDAwMDAw
MF9DVExJRF8yMDA4MDkwODAxNTI5fFB8Mi4zLjF8fHxBTHxORQ1FVv58QTaxfDIwMDgwOTA4MDE1
Mj18fHxKYXZhQ0FQUzZeXl5eXl5eVvNFU1MNUE1EfdF8fEEwMDAwMTBeXl5Ib3NBXk1SXkhvc0F8
fEt1c3NlbF5BYmlnYWlsfHwXOTQ2MDEwMTEyMzA0NjY3V0aCAzcmQgQ2lyY2x1Xl5E
b3duaGFTIE1hcmtldF5FbmdsYW5kIC0gTm9yZm9sa14zMDgyOF5VS3x8fHx8fHx8QTlwMDgwOTA4
MDE1Mj18fHxKYXZhQ0FQUzZeXl5eXl5eVvNFU1MNUE1EfdF8fEEwMDAwMTBeXl5Ib3NBXk1BSU58fHxFTVJ8
fHx8fHx8fHxWMjAwODAxNTI5DQ0K</opaqueElement>
]]
...
[2010-09-29T07:07:33.769+10:00] [AdminServer] [TRACE] []
[oracle.soa.mediator.service.transformation] [tid:
weblogic.work.j2ee.J2EEWorkManager$WorkWithListener@1e7b5db4] [userId: weblogic] [ecid:
0000IhQAzZ32zGwJLxmJOA1CcZM00001OR,0] [SRC_CLASS:
oracle.tip.mediator.service.transformation.XSLTransformer] [APP: soa-infra] [composite_name:
HL7TranformFF] [component_name: HL7TransformFFMediator] [component_instance_id:
69647490CB4411DFBFA6C5760DD892B3] [dcid: 02d1ff4621073810:-598e0369:12b59a0c1f8:-8000-
00000000000001ca] [SRC_METHOD: getXSLTTransformer] [composite_instance_id: 160001] Creating XSL
Template for xsl :xsl/opaqueElement_To_opaqueElement.xsl
...
[2010-09-29T07:07:34.441+10:00] [AdminServer] [TRACE] [] [oracle.soa.adapter] [tid:
weblogic.work.j2ee.J2EEWorkManager$WorkWithListener@1e7b5db4] [userId: weblogic] [ecid:
0000IhQAzZ32zGwJLxmJOA1CcZM00001OR,0] [SRC_CLASS:
oracle.integration.platform.blocks.adapter.fw.LogManagerImpl] [APP: soa-infra] [dcid:
02d1ff4621073810:-598e0369:12b59a0c1f8:-8000-00000000000001ca] [SRC_METHOD: log] File Adapter
HL7TranformFF Successfully sent message : C:\hl7\adt\data\ADT_A01_output_1.hl7 to Adapter Framework.
...

```

Note that the file adapter encodes the content of the file to Bas64 encoding, before embedding it in a XML message and handing it to the Mediator component. This is the result of configuring the inbound file adapter with “Native format translation is not required (Schema is Opaque)” selected. Our basic XSL transform copies input to output, both base64Binary datatypes. The outbound

adapter decodes the Base64 encoded data and writes it to the file.

Stage II: Adding Spring Component with JCD code

Now we will create a Spring Component to host out JCD code, which we will reuse from the stand-alone Java application we created earlier. We will add this component to the Mediator solution and will exercise the project.

Right-click the name of the project, HL7TransformFF. Choose New. Click the “All Technologies” Tab. Select Java and Java Interface, and then click OK.

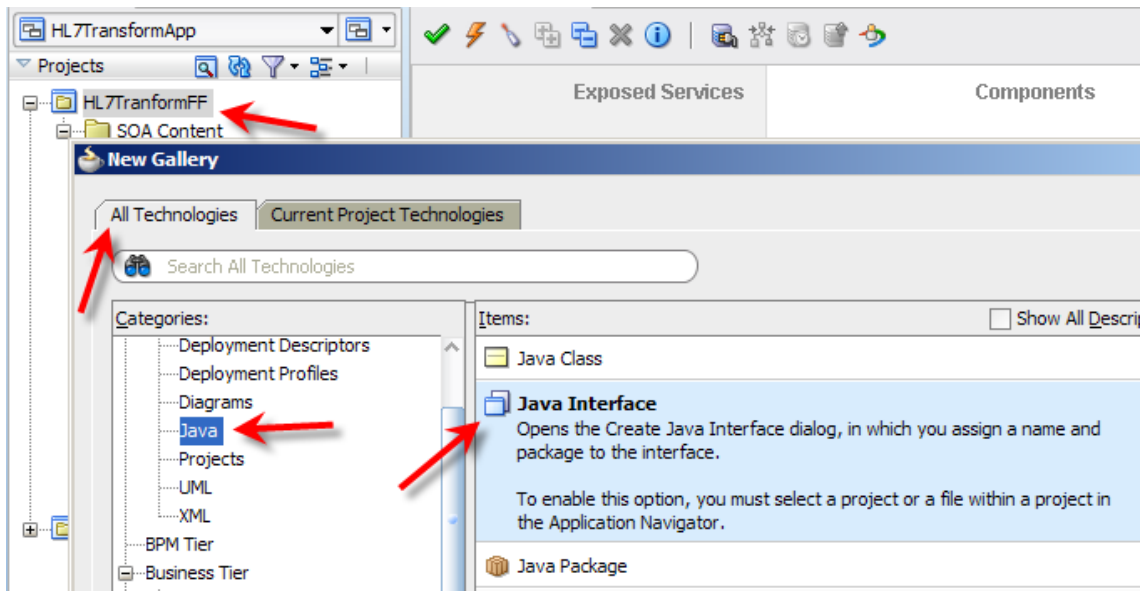


Illustration 42: Start the New Java Interface wizard

Name the interface HL7TransformInterface, change package name to hl7transform and click OK. Changing package name will eliminate the need for change of package name when we copy the class source from the HL7Transform project.

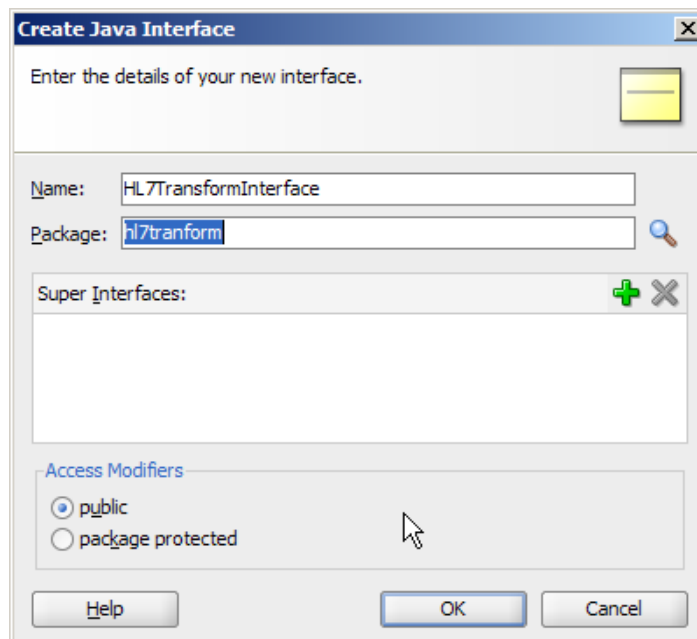


Illustration 43: Name the interface and package

Choose to save the artefacts to SCA-INF/src directory and click OK.

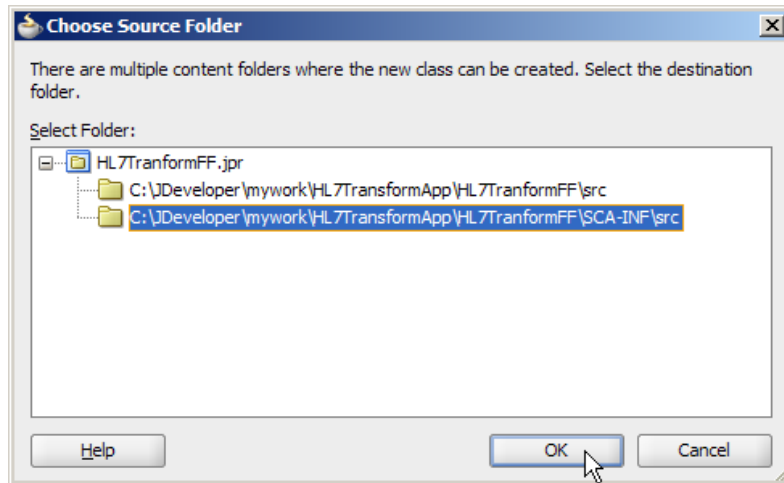


Illustration 44: Choose the target directory

Java interface skeleton will be generated.

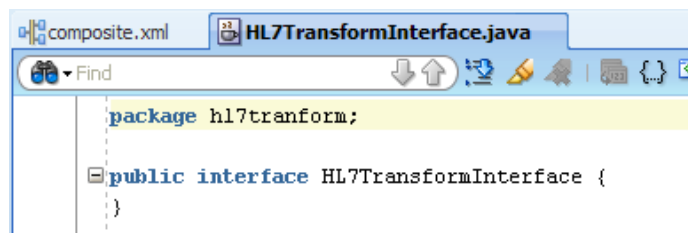


Illustration 45: Interface skeleton

Switch to the OS directory explorer, locate the directory in which the Java CAPS JAR files used in the HL7Transformer stand-alone Java application are located, and copy them to the SCA-INF/lib director of the HL7TransformFF project.

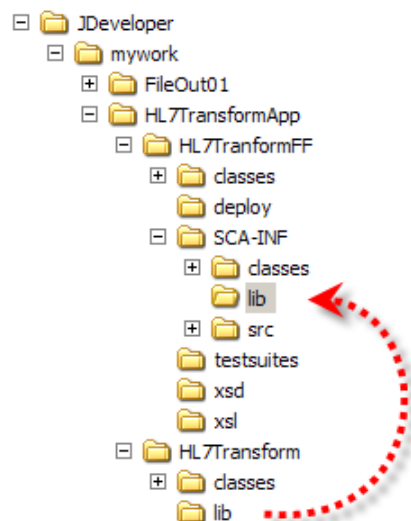


Illustration 46: Lib directories from and to

Switch back to JDeveloper. Right-click the name of the project, HL7TransformerFF, choose project properties and add the JARs to the project “Libraries and Classpath”.

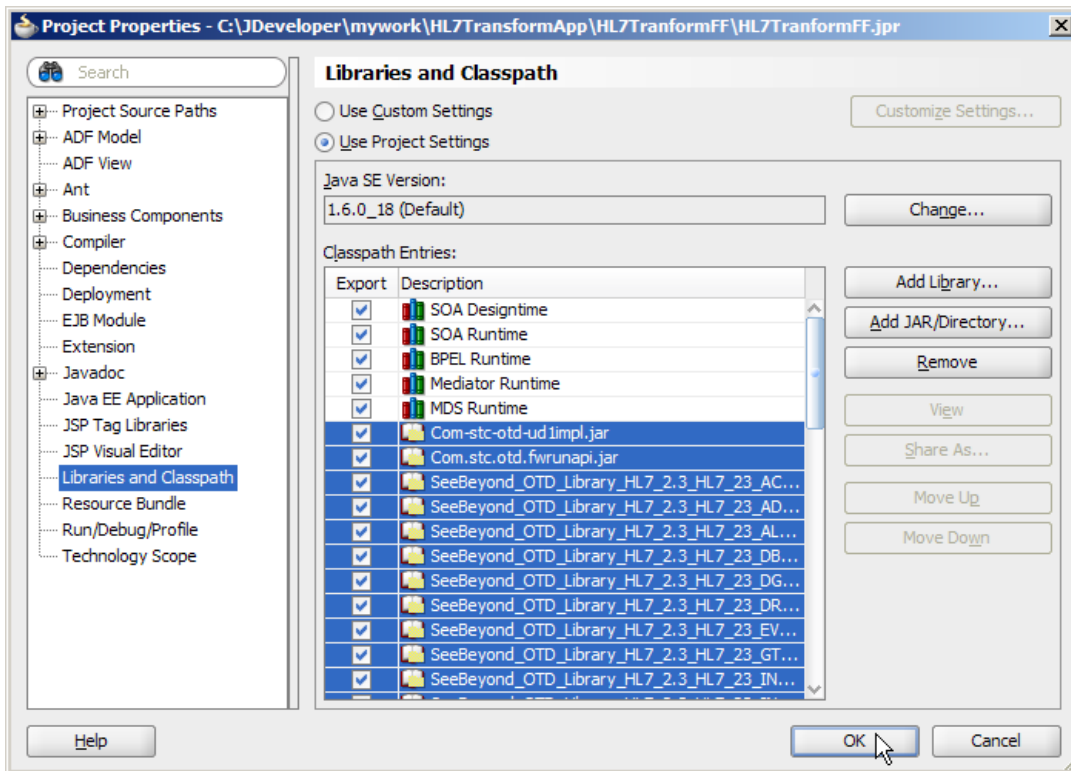


Illustration 47: Add libraries

Open the Java class, HL7Transform.java, created in earlier in project HL7Transform. Locate the method HL7Transform. Copy the method signature:

```
public byte[] HL7Transform(byte[] baA01In) throws java.io.IOException, Throwable {
```

and paste it into the new interface as shown below. Take care to replace the trailing curly brace with a semicolon.

```
package hl7transform;

public interface HL7TransformInterface {
    public byte[] HL7Transform(byte[] baA01In) throws java.io.IOException, Throwable;
}
```

Illustration 48: Define the interface

Right-click on the package name in the HL7TransformFF project's Application Sources folder and choose New → Java → Java Class. Name the new class HL7TransformImpl and select the HL7TransformInterface as the interface to implement.

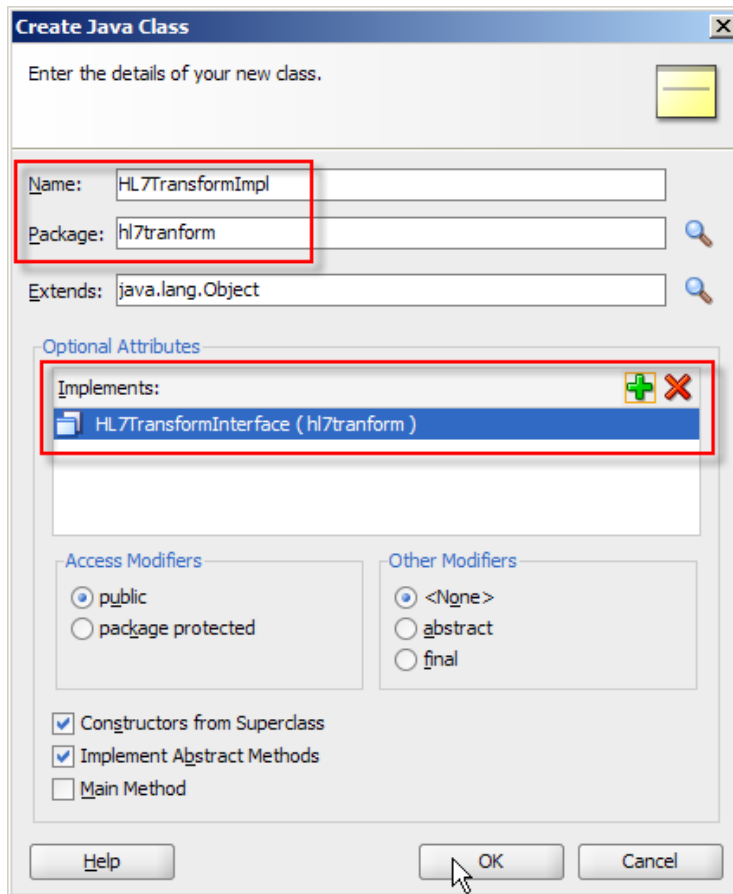


Illustration 49: New class implement the interface defined earlier

Copy the body of the class from HL7Tranform/HL7Transform.java and paste it into this new class replacing its body completely. Remove constructor. The final sources, with implementation code omitted for brevity, will have this structure:

```
package hl7transform;
import java.io.IOException;
public class HL7TransformImpl implements HL7TransformInterface {
    public byte[] HL7Transform(byte[] baA01In) throws java.io.IOException,
        Throwable {
        ...
        return baA04Out;
    }
}
```

“Make” the class to ensure it is error-free.

Let's now create a Spring Context.

If the Spring technology is not available in your JDeveloper installation you will need to add it before proceeding, as mentioned in Prerequisites and Assumptions. I will proceed on the basis of this matter having been taken care of.

Right-click the name of the project, HL7TransformFF, choose New → Business Tier → Spring → Spring Bean Configuration.

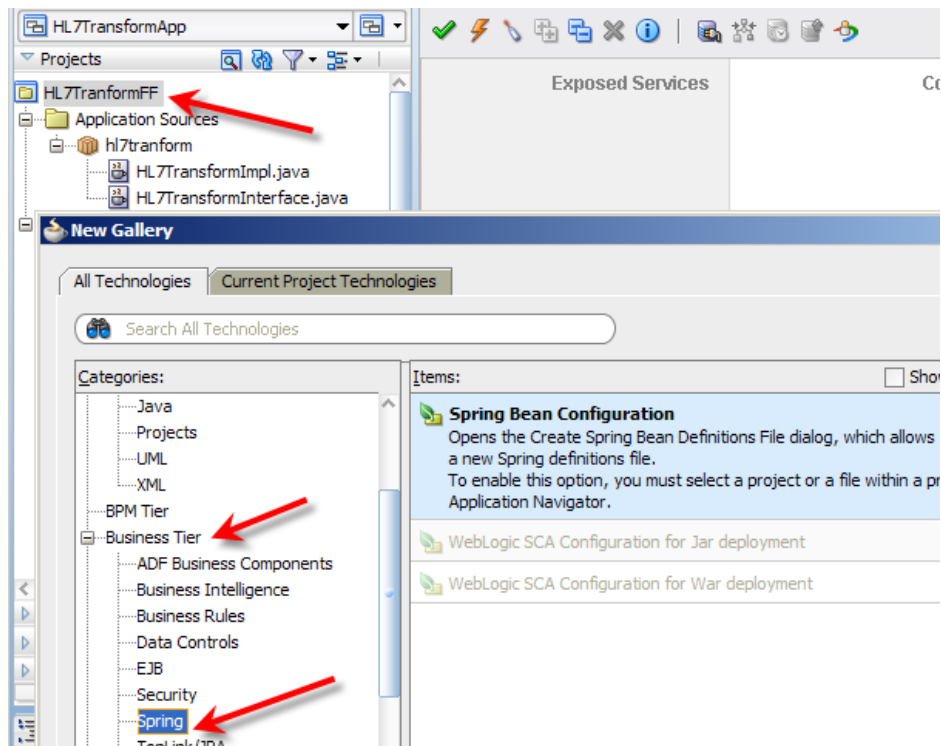


Illustration 50: Start the Spring Bean Configuration wizard

Change file name to HL7Transform-beans.xml and directory to where the composite.xml lives.

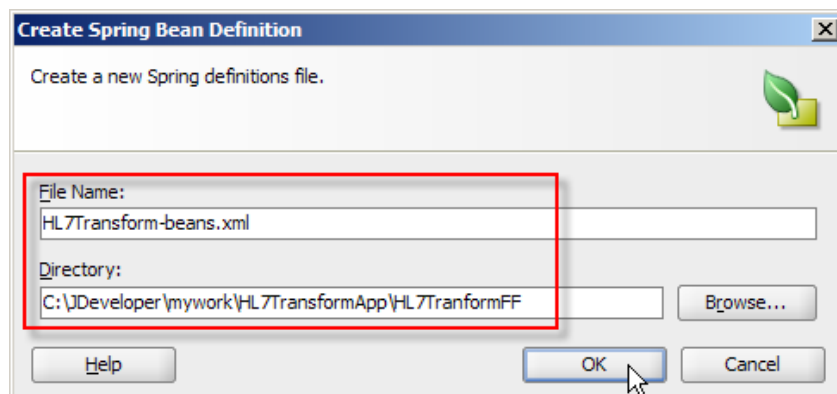


Illustration 51: Name the configuration file

Drag a bean component from the components palette onto the configuration source in the place indicated.

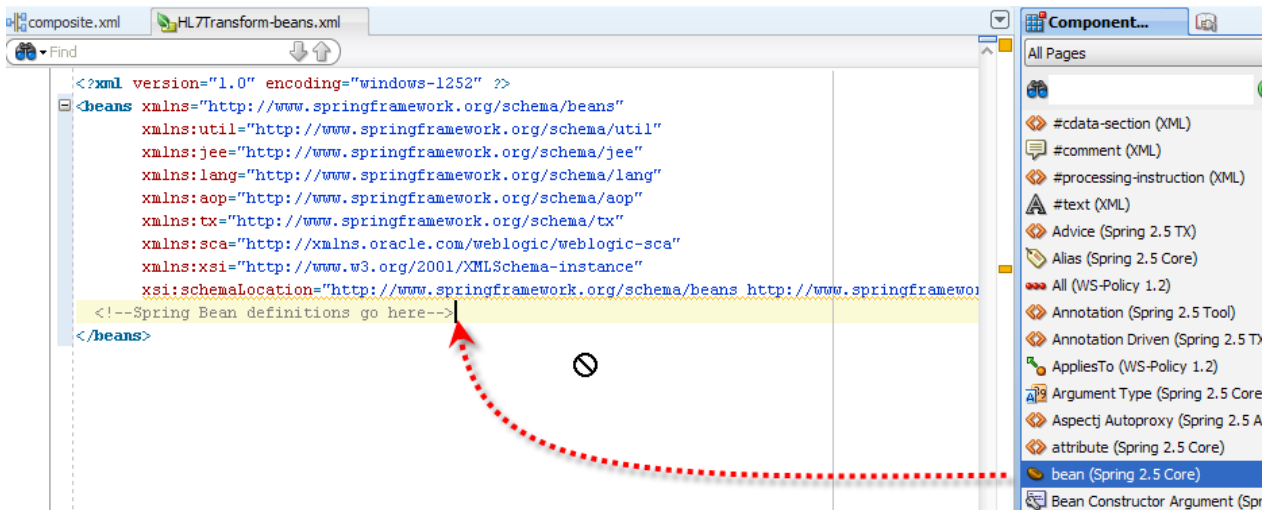


Illustration 52: Add a bean definition to the configuration

Place the cursor inside the <bean/> tag to show bean properties in the property inspector. Set bean name property to HL7TransformBean and class to hl7transform.HL7TransformImpl.

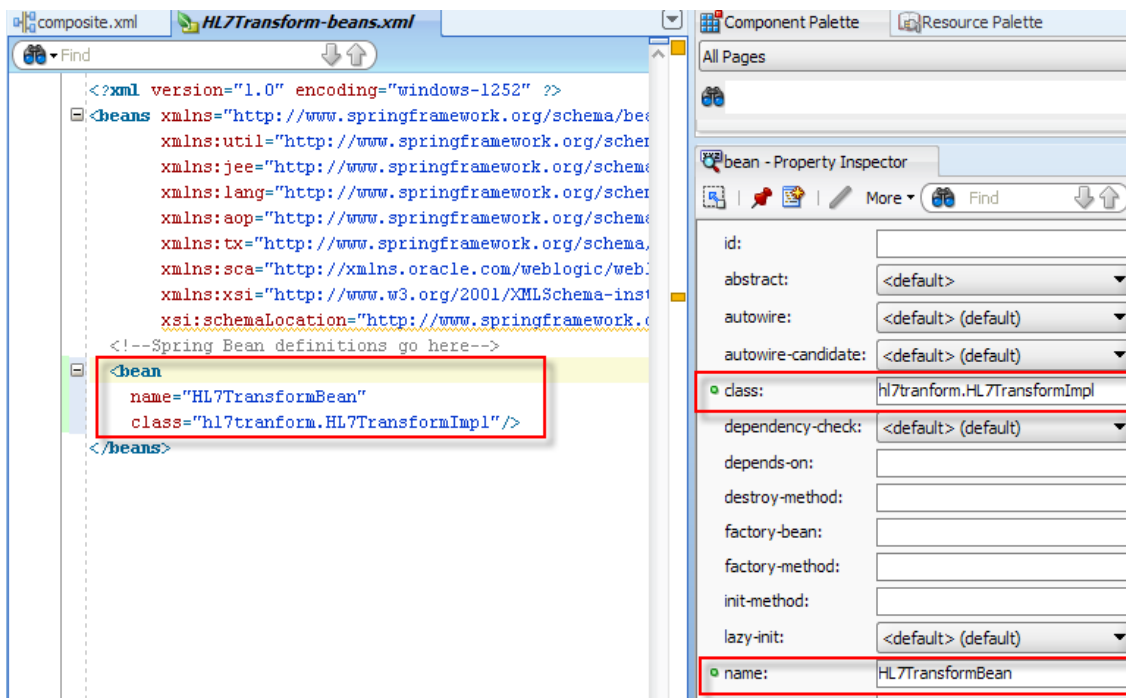


Illustration 53: Set bean properties

Now drag the SCA Service component (WebLogic SCA) to the canvas immediately after the closing ">" markup of the bean tag as shown.

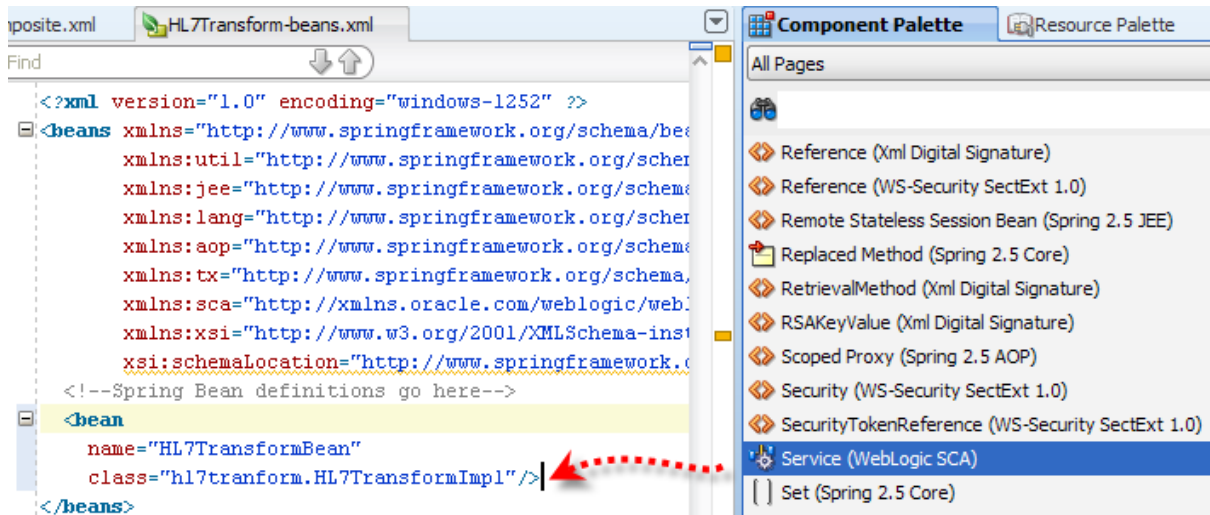


Illustration 54: Add SCA Service configuration

Set properties of the service to name: HL7TransformService, target: HL7TransformBean and type: hl7transform.HL7TransformImpl.

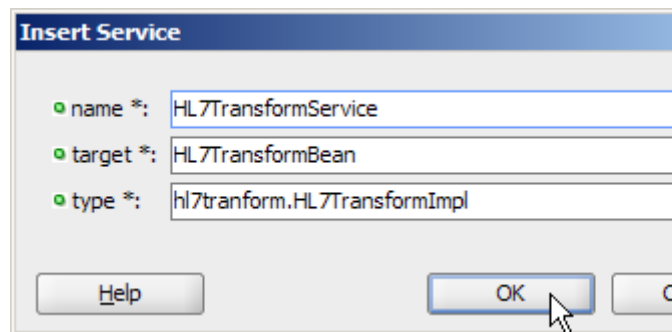


Illustration 55: Configure SCA Service

The target property value of the SCA Service must be the same as the name property of the Bean. The configured bean markup is shown below.



Illustration 56: Spring component configuration

Switch back to composite canvas and drag the Spring Context component from the component palette to the canvas, dropping it in the Components swim line.

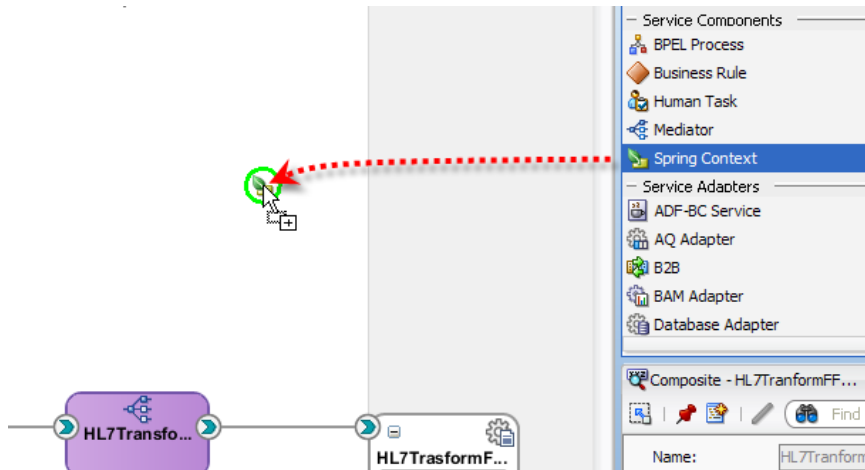


Illustration 57: Add Spring Context to the canvas

Name the spring context HL7TransformSpring and locate the existing context we created before.

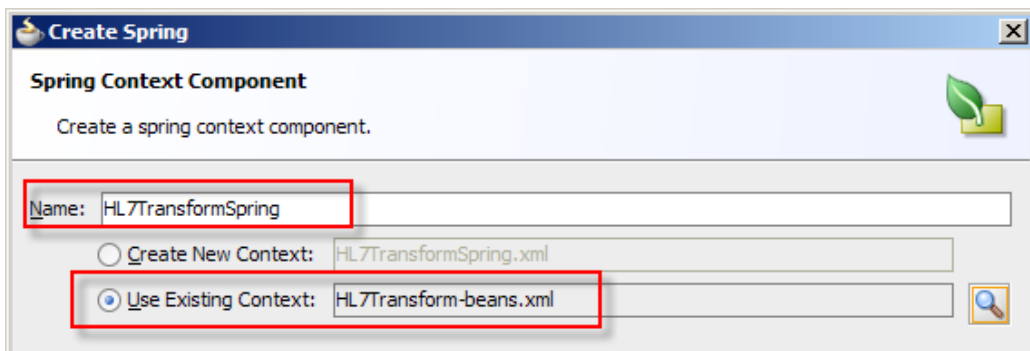


Illustration 58: Name the component and locate existing context

Select and delete connectors between the adapters and the mediator component.

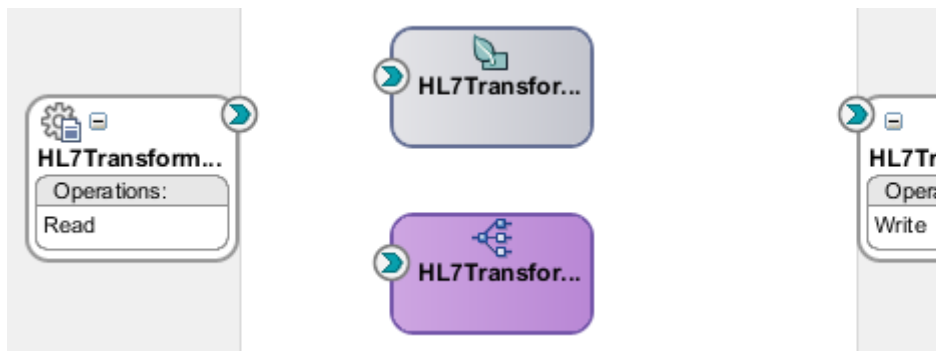


Illustration 59: After connectors are deleted

Connect the HL7TransformMediator component to the HL7TransformSpring component.

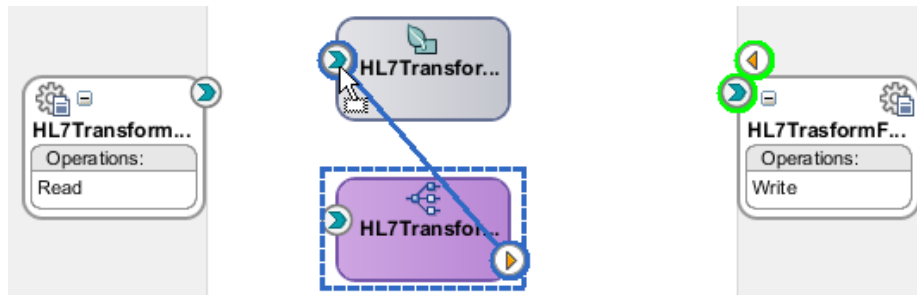
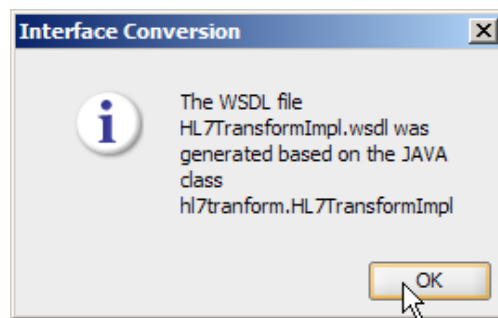


Illustration 60: Connect mediator and spring components

You should see a feedback dialogue saying that a WSDL was generated.



Illustration

61: WSDL was generated

Connect the inbound adapter to the Mediator component.

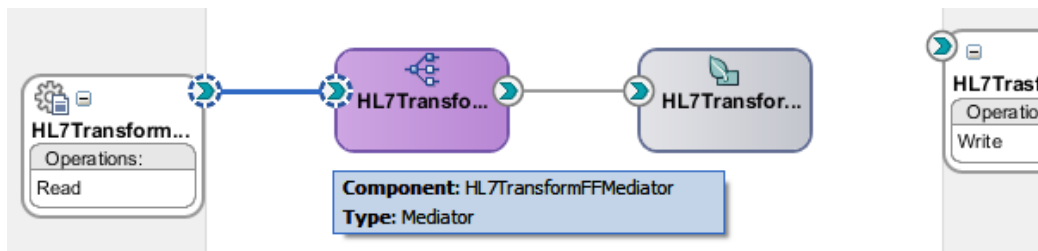


Illustration 62: Connect inbound adapter to the mediator

Double-click the mediator component to open its mplan.

Select, by clicking at it, single static routing rules with no targets. Delete by clicking the cross button.

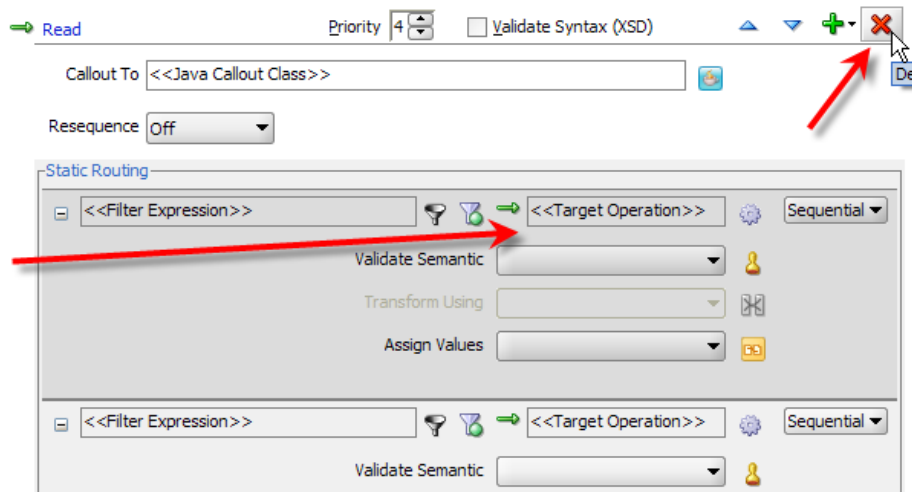


Illustration 63: Delete target-less static routing rules

Repeat for all such rules until only the single request/reply rule with a target remains.

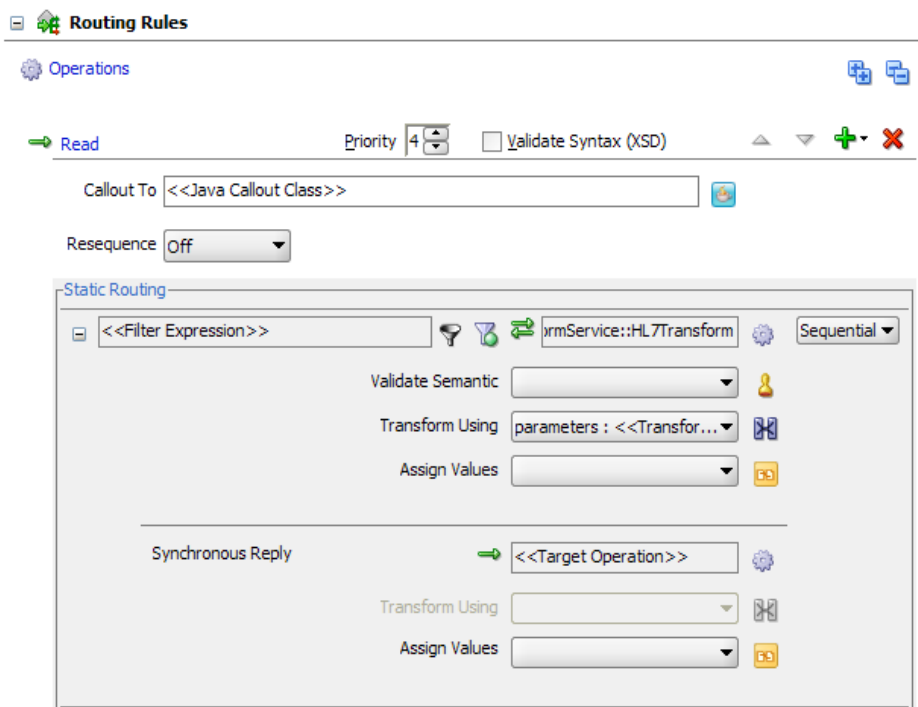


Illustration 64: Request/Reply rule invoking the Spring component

Click on the mapper button on the request part of the rule to create a mapper file.

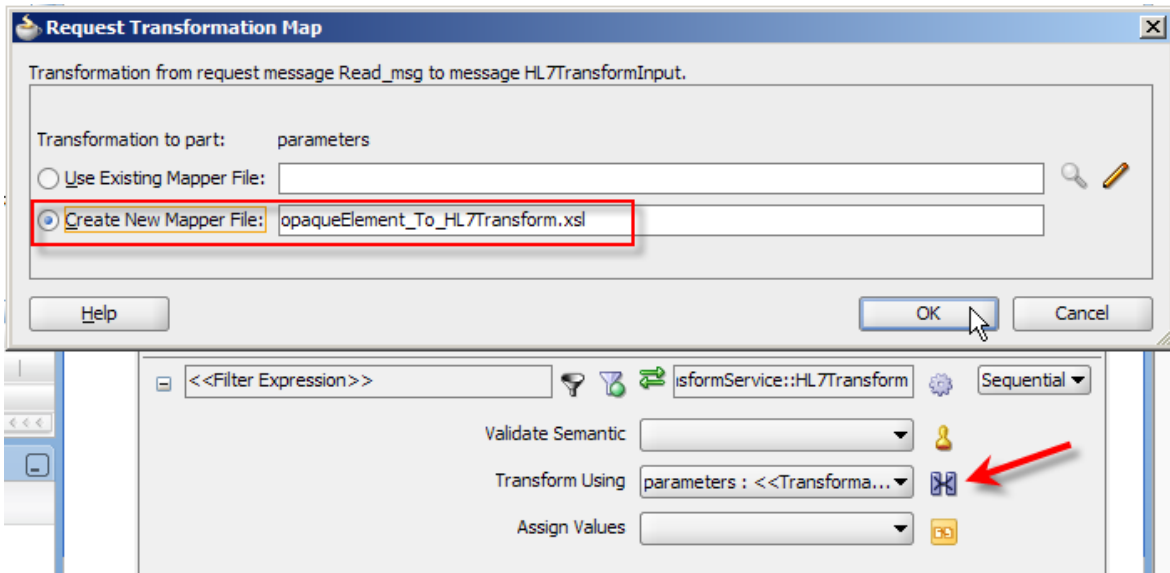


Illustration 65: Craete a mapper file for the request

Map the `opaque:opaqueElement`, of `base64Binary` datatype, to `HL7Transform` → `arg0` of the `HL7Transform` class, which is also a `base64Binary` datatype. Note that the underlying infrastructure transparently encodes and decodes data as needed, in this case, that the byte array which the Java method is given is not base64 encoded.

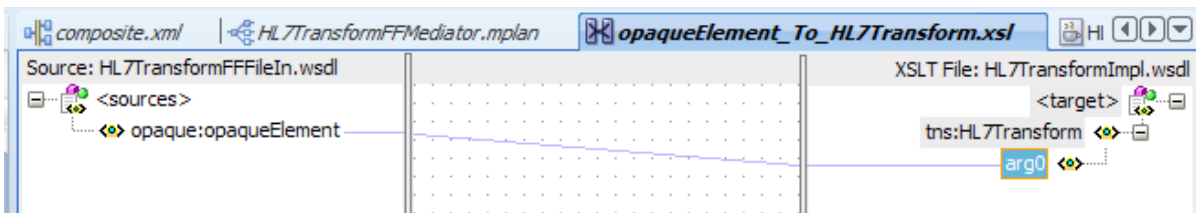


Illustration 66: Map input to output

This assignment could have also been done using Assign Values functionality rather than transform functionality.

Save and close the mapper file. The request is mapped.

Let's configure the target service so that the result value from the Spring component is handed over to the next component in the composite.

Click the Target Service button.

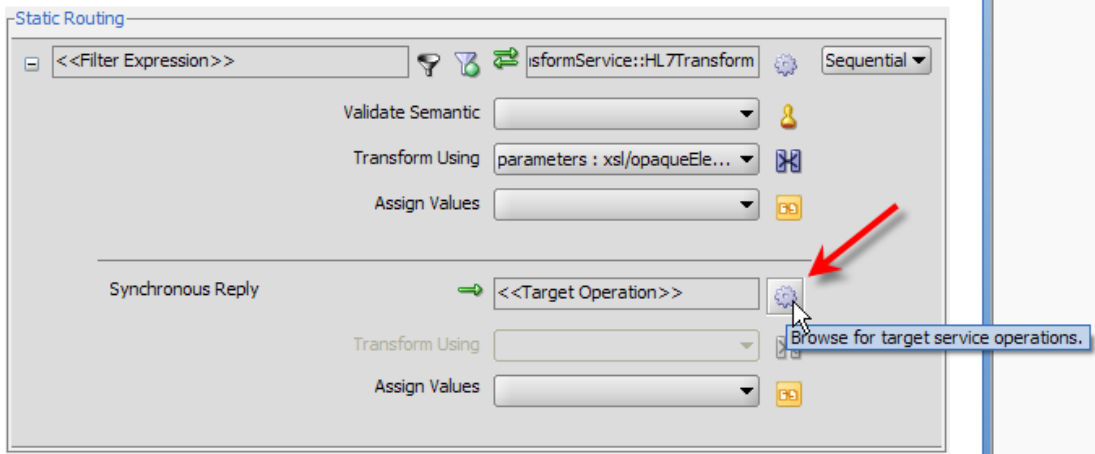


Illustration 67: Click the Target Service button

Click the Service button then locate the HL7TransformFFFileOut service, select the Write operation of the HL7TransformFFFileOut service and click OK.

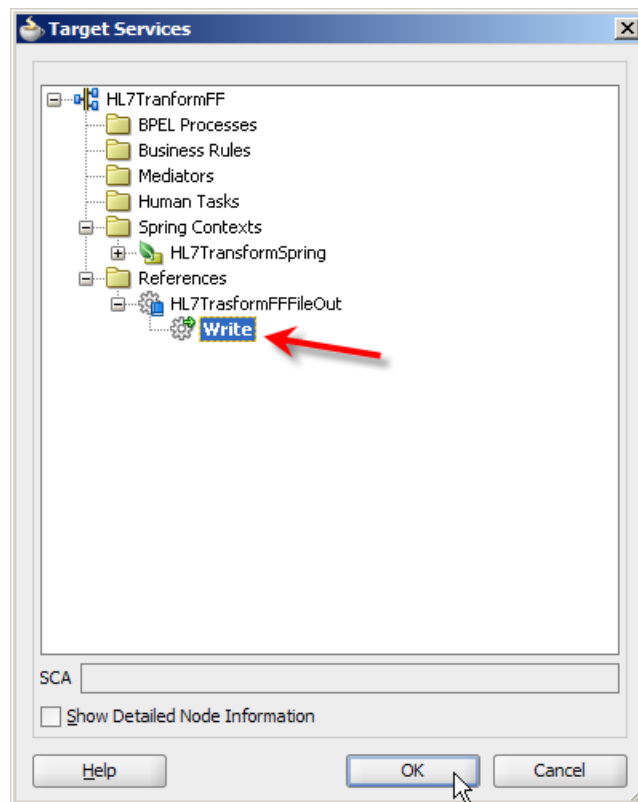


Illustration 68: Select target service operation

Now create a new mapper file.

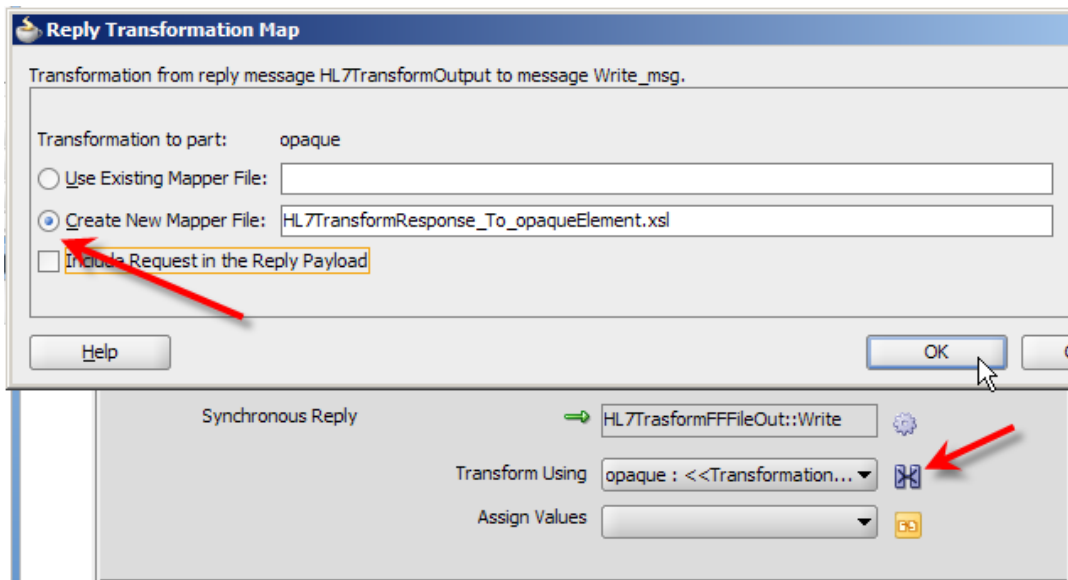


Illustration 69: Create mapper file for the reply

Map HL7TransportResponse → Return to the opaque:opaqueElement of the FileOut adapter.

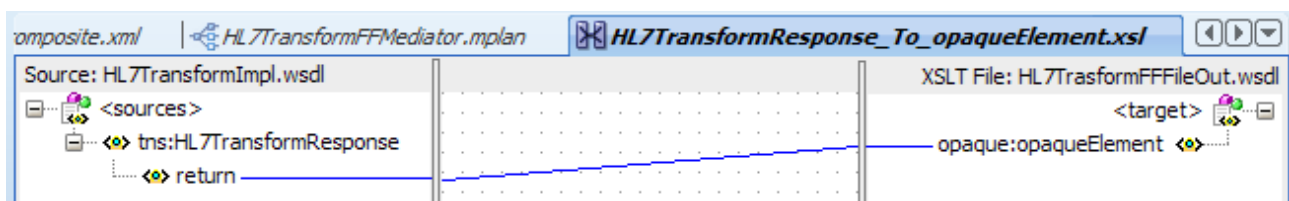


Illustration 70: Map return to reply

Close the mapper file.

The composite will now look like that shown below.

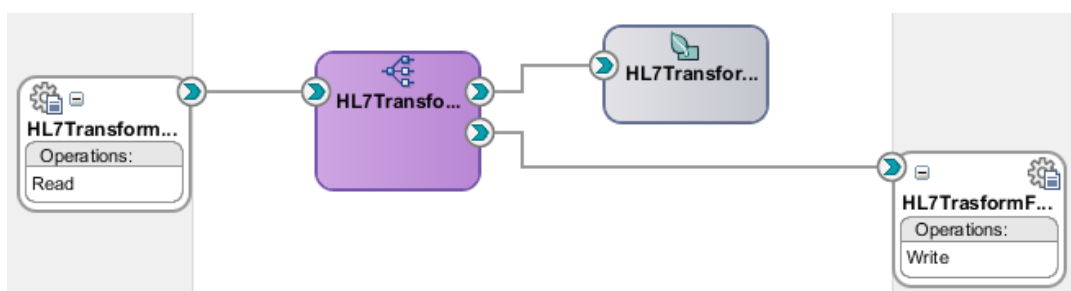


Illustration 71: Finished composite

Deploy the project and submit the test file.

If all went well the input file containing a single HL7 v2.3.1 ADT A01 message should have gotten transformed into a file containing a single HL7 v2.4 ADT A04 message.

```

1 MSH|^~\&|SystemA|HosA|PI|MDM|2008090801529||ADT^A01|000000_CTLID_2008090801529|P|2.3.1||AL|NE
2 EVN|A01|2008090801529|||JavaCAPS6^^^^^^^USERS
3 PID|1||A000010^^^HosA^MR^HosA||Kessel^Abigail||19460101123045|M|||7 South 3rd Circle^^Downham
Market^England - Norfolk^30828^UK|||||||A2008090801529
4 PV1|1|I||I|||FUL^Fulde^Gordian^^^^^^^^^^^MAIN|||EMR|||||||V2008090801529^^^^^VISIT|||||||
|||||||2008090801529
5

```

Illustration 72: Input ADT A01

```

1 MSH|^~\&|SystemA|HosA|PI|MDM|2008090801529||ADT^A04|000000_CTLID_2008090801529|P|2.4||AL|NE
2 EVN|A04|2008090801529|||JavaCAPS6^^^^^^^USERS
3 PID|1||A000010^^^HosA^MR^HosA||Kessel^Abigail||19460101123045|M|||7 South 3rd Circle^^Downham
Market^England - Norfolk^30828^UK|||||||A2008090801529
4 PV1|1|I
5

```

Illustration 73: Output ADT A04

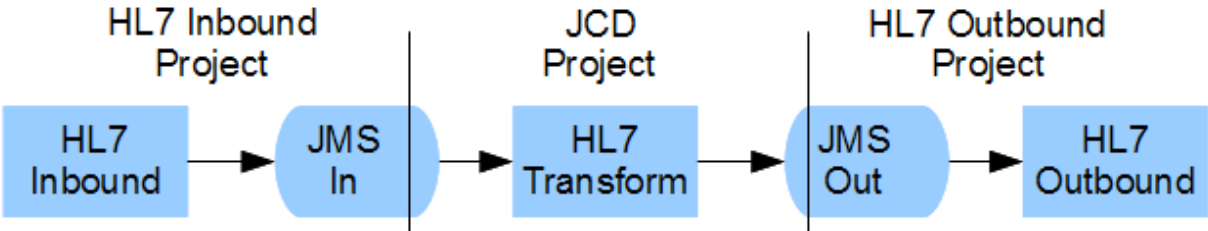
When working with this sort of projects I noted that sometimes the deployment process does not succeed in undeploying the running version of the composite. In such a case the composite must be undeployed explicitly through the Enterprise Manager or the JDeveloper. Once undeployed, the new incarnation can be deployed again.

To summarise, we took the transformation code and utility JARs from the Java CAPS HL7 JCD-based solution and re-implemented the transformation code in a SOA Suite Spring Component, simply pasting the code into a skeleton Java class. We exercise this solution by feeding it a HL7 v2.3.1 ADT A01 message through a file and writing transformed message to a file.

SOA Suite Solution with B2B and Spring Component

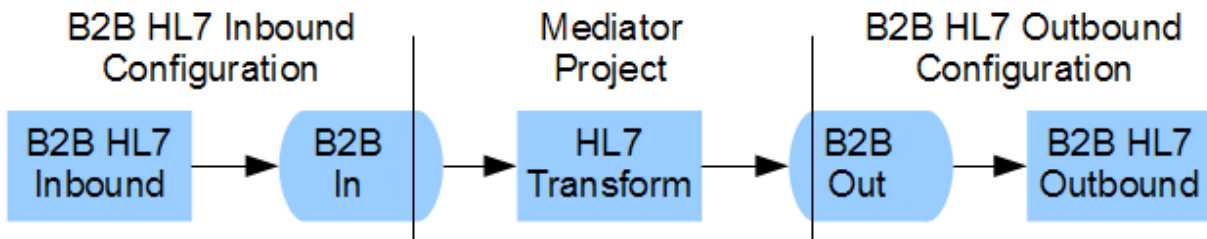
At this point we will add HL7 communication facilities on the inbound and outbound side of the transformation to replicate what the complete Java CAPS solution does.

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 key components of this solution.



Drawing 6: Java CAPS HL7 Solution

The equivalent SOA Suite solution, which will reuse the JCD transformation code, will look very similar, as shown in the following figure.



Drawing 7: SOA Suite "Equivalent" reusing JCD transformaiton code

Rather than using pre-built HL7 inbound and outbound projects, the SOA Suite-based solution uses the SOA Suite B2B HL7 functionality, with appropriately configured “trading partners” and “trading partnership agreements”. I discuss configuration of inbound and outbound B2B HL7 partnership agreements in various articles at <http://blogs.czapski.id.au/?s=oracle+hl7>.

Configure Inbound HL7 Partnership Agreement

The B2B HL7 Inbound trading partnership agreement is used to configure the infrastructure to receive and acknowledge HL7 messages. The article “Oracle SOA Suite 11g HL7 Inbound Example” at <http://blogs.czapski.id.au/2010/06/oracle-soa-suite-11g-hl7-inbound-example>, discusses configuration of the inbound HL7 partnership agreement and a Mediator solution that receives messages from it. Please review the article and refer to it as needed. For this article I will assume that you have done so and can perform the necessary configuration work based on the brief discussion given here. I will be referring to specific section on the “Oracle SOA Suite 11g HL7 Inbound Example” by page number.

Page 1, Prepare HL7 Data: Get the archive which contains HL7 A01 messages and extract them as suggested. You should already have done that to test the solutions we have been working with so far.

Page 2, Obtain and Explore the HL7 Browser: Get the browser and get familiar with it.

Page 7, Extract HL7 Message Structure: Follow the steps to export a HL7 ADT A01 message structure and save it as ADT_A01.ecs and ADT_A01.xsd.

Page 12, Configure B2B Partnership: Follow instructions through to page 14 until the document is defined in B2B Administration → Document Tab.

If you don't have a partner for the local side then follow instructions on page 14 and rename MyCompany to HL7LocalReceiver partner. If you already have a local partner then use that name.

If needs be, add identifiers for HL7 Message Application ID of PI and HL7 Message Facility ID of MDM, as instructed.

Continue working through page 14 and add HL7 v2.3.1 ADT A01 document to the list of documents that the local trading partner is willing to deal with, if not already there.

Follow instructions from page 15 through page 20 to set up the HL7RemoteSender partner, if you don't already have one, or to confirm that it is set up as indicated, if you do.

Make sure that the Translate checkbox is not checked in the trading partner agreement. If it is the HL7 message will get translated to XML and the project will fail at runtime because the Spring component is expecting a HL7 v2.x delimited message, not XML.

Stop at the end of the section. Do not continue with next section, “Develop Writer Solution”.

Configure Outbound HL7 Partnership Agreement

The B2B HL7 Outbound trading partnership agreement is used to configure the infrastructure to send HL7 messages to external parties and process ACKs.

As you realise we will be sending out HL7 v2.4 ADT A04 messages, which the Mediator and Spring composite produce using the JCD Java transformation code. To support HL7 v2.4 ADT A04 messages we need to create and export appropriate B2B Specs.. Follow instructions in section “Extract HL7 Message Structure” on pages 7 through 11 of the “Oracle SOA Suite 11g HL7 Inbound Example” at <http://blogs.czapski.id.au/2010/06/oracle-soa-suite-11g-hl7-inbound-example>, choosing HL7 v 2.4 and ADT A04 instead of the version and event message discussed in the article.

I assume that by now we have the v24_ADТ_A04.ecs and v24_ADТ_A04.xsd in c:\hl7\ADТ_Specs.

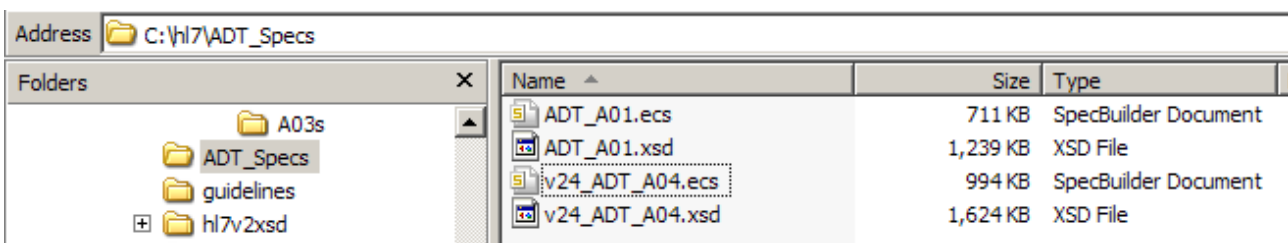


Illustration 74: A01 and A04 guidelines

Start the B2B Web Console and add the document hierarchy and definition. Administration → Document Tab, pages 12-14, choosing HL7 v 2.4 and ADT A04 instead of the version and event message discussed in the article.

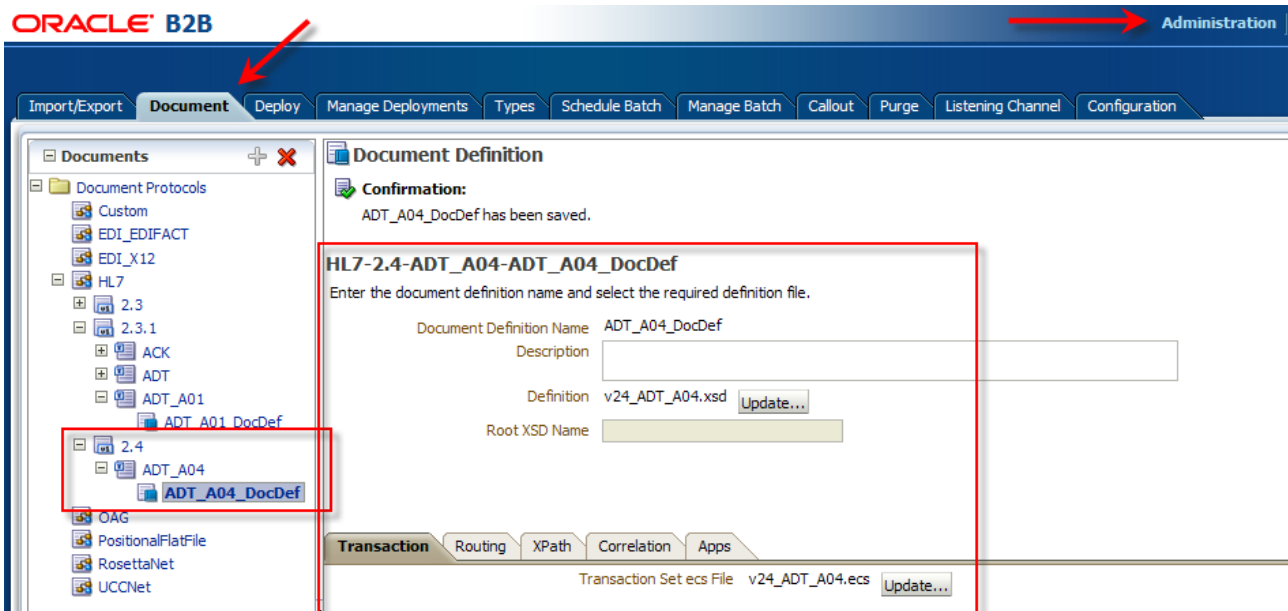


Illustration 75: ADT A04 added to the global B2B configuration

Now we need to add the ACK document type. Follow the same procedure in B2B Web Console but rather than using an externalised ACK document, which we did not prepare, we will use the default ACK document. Create a document type ACK under Protocol Version 2.4. Create a document type definition ACK_DocDef, using pre-populated XSD and ECS files.

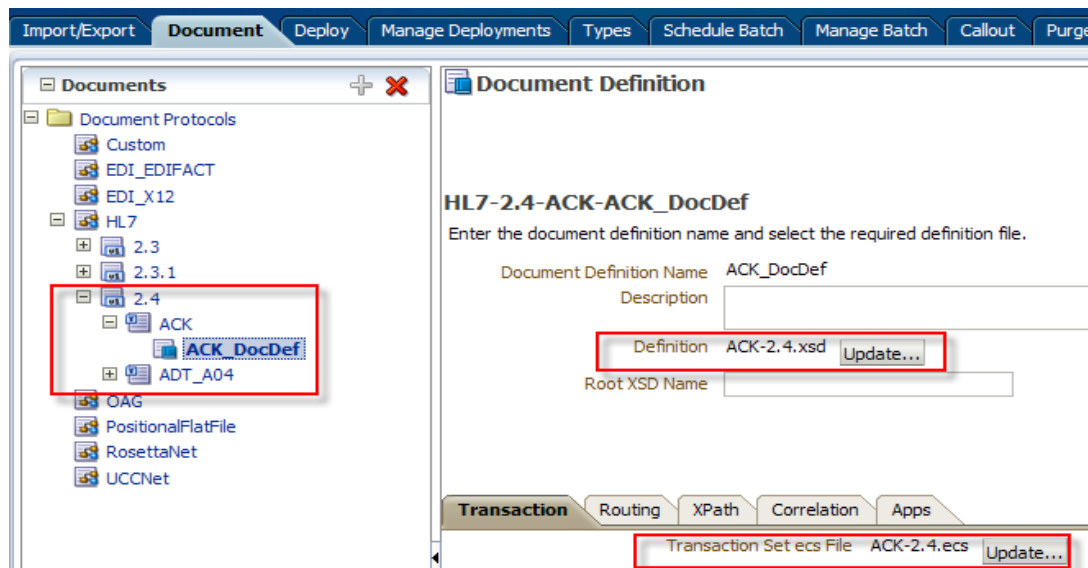


Illustration 76: Add ACK document

Now the Self partner knows about both version 2.4 documents.

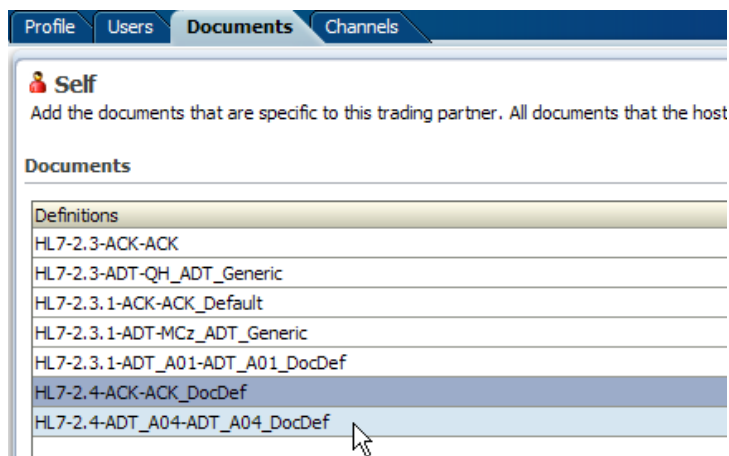


Illustration 77: Local partner knows about new documents

The article “Oracle SOA Suite 11g HL7 Outbound Example” at <http://blogs.czapski.id.au/2010/06/oracle-soa-suite-11g-hl7-outbound-example>, discusses configuration of the outbound HL7 partnership agreement and a Mediator solution that sends messages to it. Please review the article and refer to it as needed. For this article I will assume you have done so and can perform the necessary configuration work based on the brief discussion given here. I will be referring to specific section on the “Oracle SOA Suite 11g HL7 Outbound Example” by page number, varying names and settings as necessary, in the discussion that follows.

Page 3, Configure B2B Partnership: Skip all the way to the bottom of page 3 and proceed from there, remembering that we are dealing with HL7 v2.4 ADT A04 messages rather than HL7 v2.3.1 ADT A01 messages. Your local trading partner may also be called something other than LocalHL7Receiver. In my screenshots in this article it is called Self, for example. Adjust as necessary. Assuming you have a local trading partner configured, whatever its name, as you should if you followed this article so far, let's proceed to page 5 to add HL7 2.4 document types to the RemoteHL7Sender partner, which we will have if we configured it in the previous section.

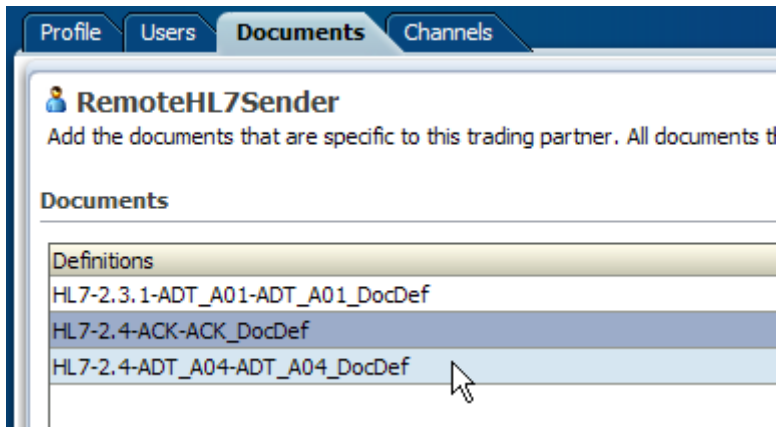


Illustration 78: RemoteHL7Sender knows about ADT A04 and ACK

Proceed to add an outbound channel, page 5.

Proceed to add a Trading Partnership Agreement, HL7ReceiverOuboundTPA, as discussed on page 6, making sure to choose ADT_A04_DocType document from HL7 2.4 ADT_A04 node tree.

Continue configuring as discussed on page 7. Make sure to uncheck both the Translate and the Functional Ack checkboxes. We don't want to do the former and we cannot do the later because of the former. Once done, save and deploy.

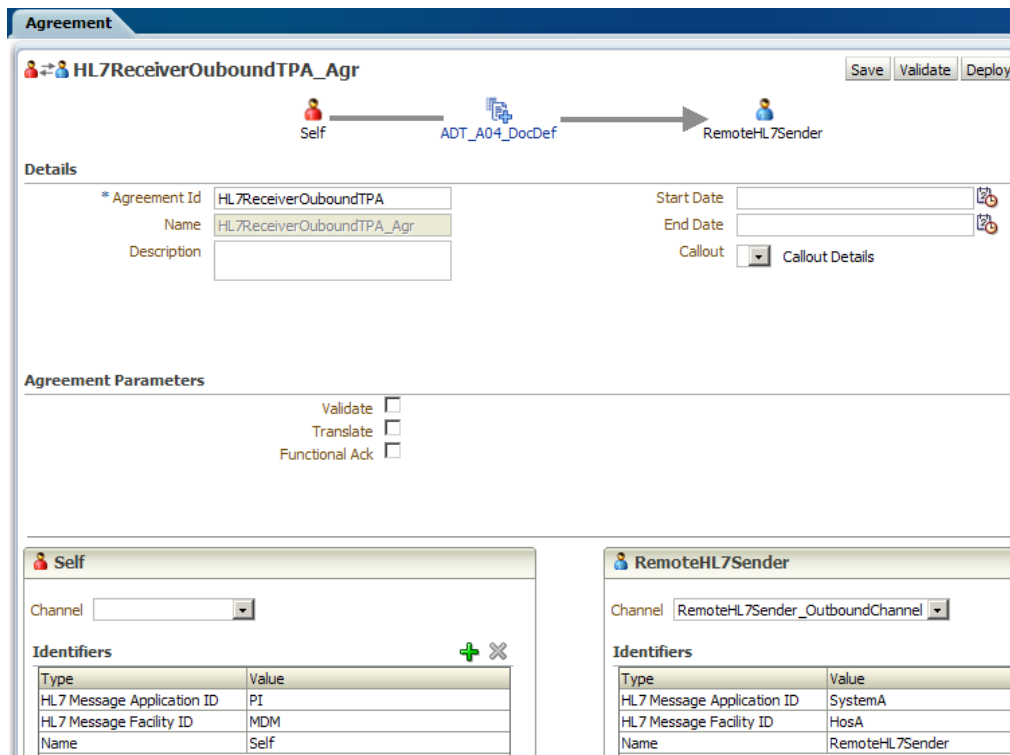


Illustration 79: Outbound partnership agreement

Stop half way through page 10 and do not continue to develop File Reader Solution.

Add a new Trading Partnership Agreement, HL7ReceiverOuboundACK_TPA, to handle receipt of the ACK form the remote party.

Agreement

HL7ReceiverOuboundACK_TPA_Ack Save Validate Deploy

Self ← ACK_DocDef → RemoteHL7Sender

Details

* Agreement Id: HL7ReceiverOuboundACK_TPA
 Name: HL7ReceiverOuboundACK_TPA_Ack
 Description:

Start Date:
 End Date:
 Callout: Callout Details

Agreement Parameters

Self

Channel:

Identifiers

Type	Value
HL7 Message Application ID	PI
HL7 Message Facility ID	MDM
Name	Self

RemoteHL7Sender

Channel: RemoteHL7Sender_OutboundChannel

Identifiers

Type	Value
HL7 Message Application ID	SystemA
HL7 Message Facility ID	HosA
Name	RemoteHL7Sender

Illustration 80: TPA for handling ACKs

Add B2B HL7 Support to the Mediator Solution

B2B HL7 trading partnership agreements for both the inbound and the outbound sides are ready. Let's now modify the Mediator solution to use these channels to receive and send HL7 messages.

We will add new adapters and a simple mediator, which will reuse the Spring Component, to the same composite which we developed so far.

Switch to JDeveloper, locate the project HL7TransformFF and open the composite. Drag the B2B Service Adapter from the Component Palette to the Exposed Services swim line.

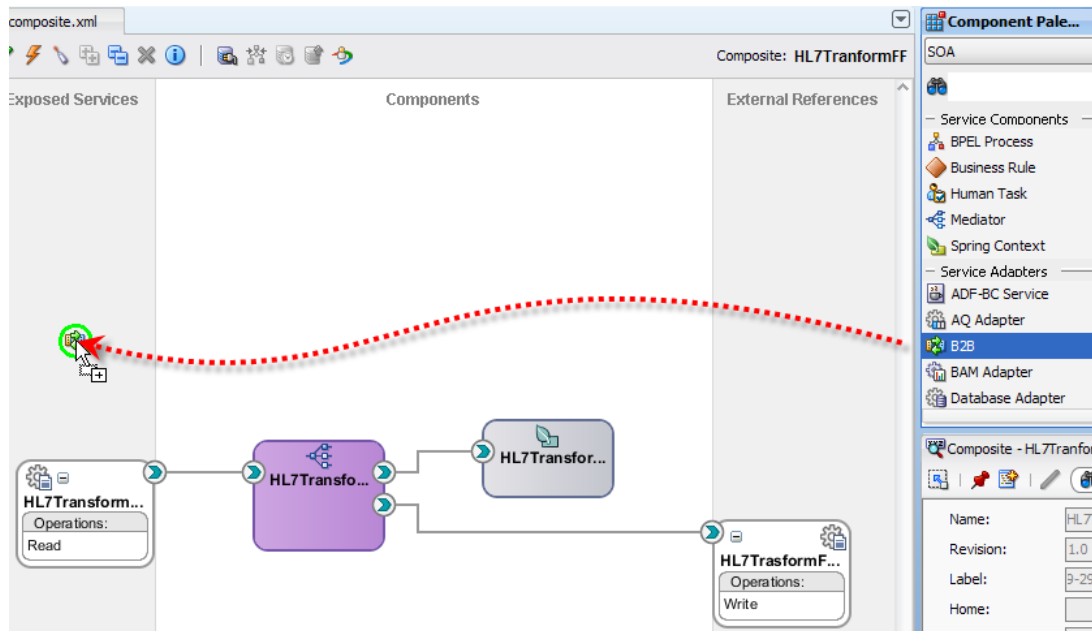


Illustration 81: Add B2B Service Adapter to the canvas

Configure as follows:

Service Name	HL7TransformFFB2BIn
Select the B2B Integration Type	Default
AppServe Connection	Choose the connection and Test B2B
Operation	Receive
Document Definition Handling	Advanced: Check the Opaque radio button
Document Definition	HL7 → 2.3.1 → ADT_A01 → ADT_A01_DocDef

Drag the B2B Service Adapter to the External References swim line and configure as follows

Service Name	HL7TransformFFB2BOut
B2B Integration Type	Default
AppServe Connection	Choose the connection and Test B2B
Operation	Send
Document Definition Handling	Advanced: Check the Opaque radio button
Document Definition	HL7 → 2.4 → ADT_A04 → ADT_A04_DocDef

Drag a Mediator component onto the components swim line and name it HL7TransformFFB2BMediator.

Connect the inbound B2B adapter to the new Mediator (HL7TransformFFB2BMediator). Connect the new Mediator to the Spring Component on the canvas.

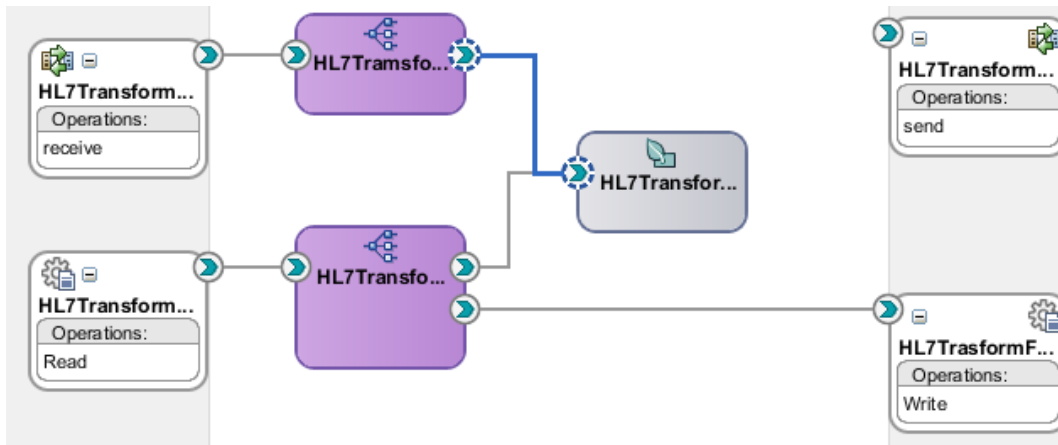


Illustration 82: Connct mediator to the inbound B2B adapter and the Spring Component

Allow the wizard to replace the WSDL (not that it should matter).

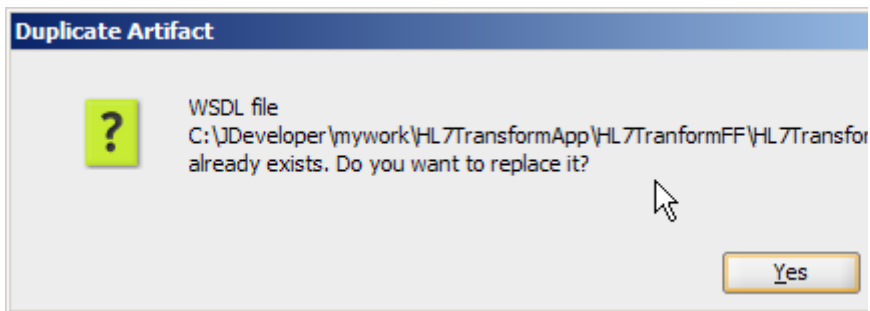


Illustration 83: Allow WSDL replacement

Double-click the new Mediator to configure its mapping files. Create a new mapping file for the request mapping.

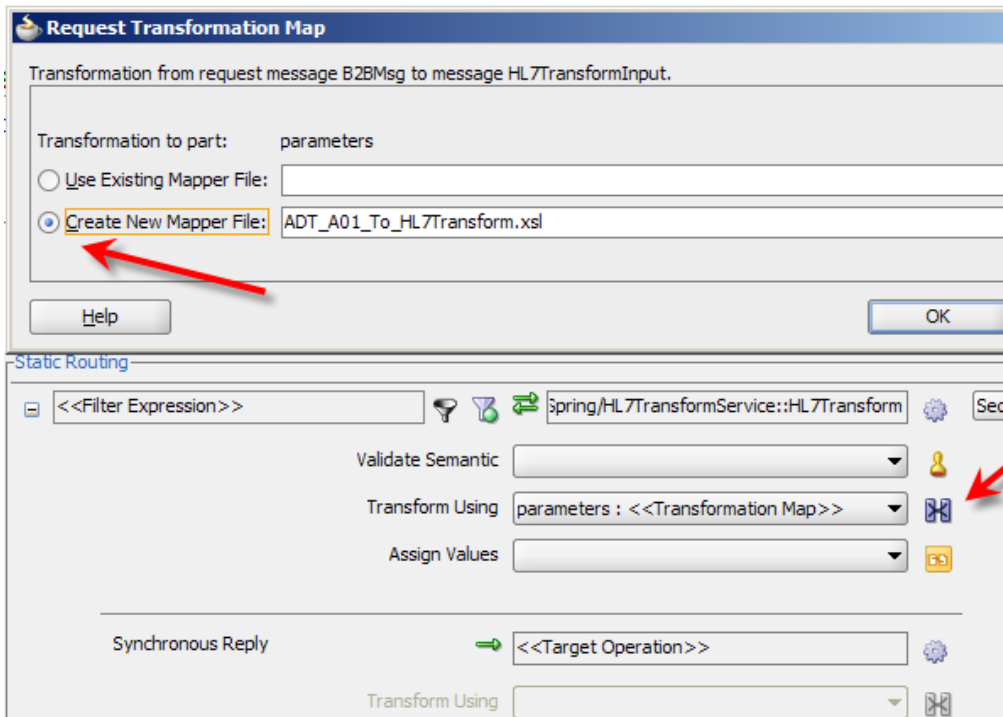


Illustration 84: Create mapping file for the request

As before, copy input to output to pass request data to the HL7Transform Spring Component.

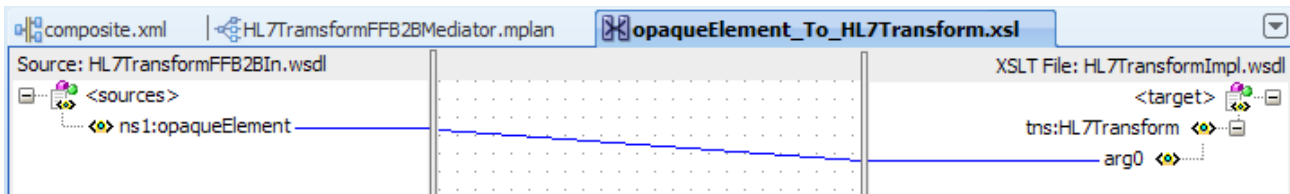


Illustration 85: Copy input to output for request side

Save and close the mapping file.

Click on the Target Service button in the Reply part, click the Service button and choose the B2BOut service Send operation.

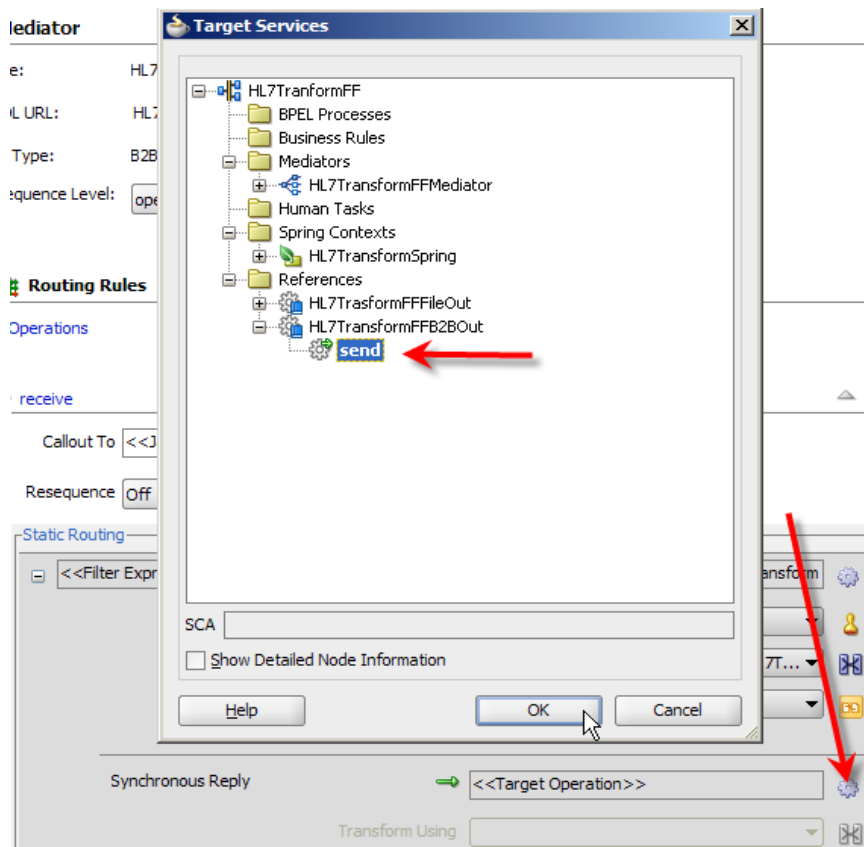


Illustration 86: choose Service Target

Create a new mapper file for the response and copy input to output, as we did before.

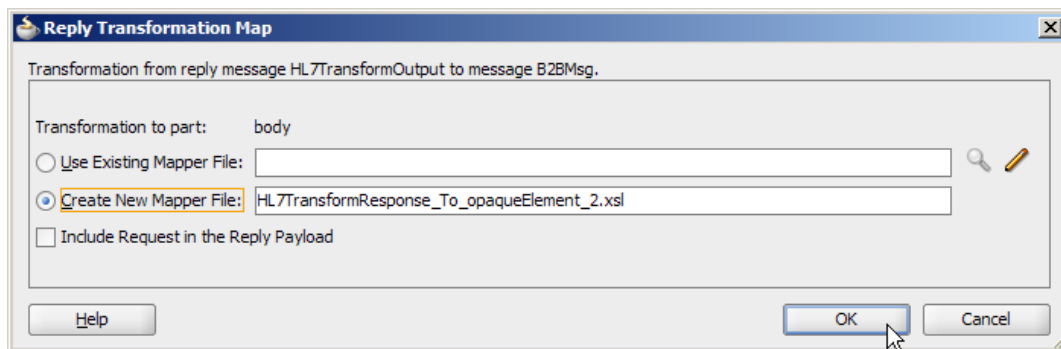


Illustration 87: Create a new mapper file

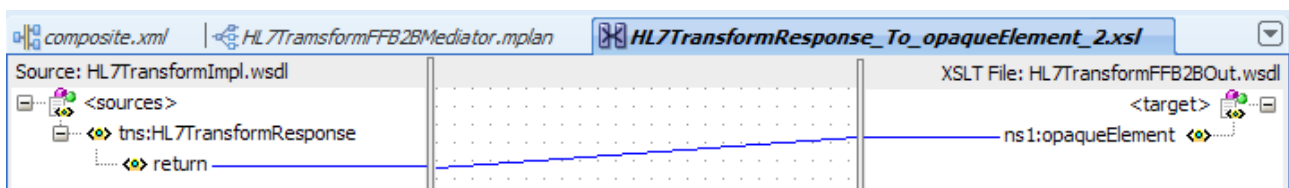


Illustration 88: Copy input to output

The outbound Oracle SOA Suite B2B component must pass key pieces of information to the B2B runtime to allow it to identify the partnership agreement to use for sending the message. In addition to creating a mapping file that will copy the result of the transformation to the input of the outbound B2B adapter we must add the following Assign Values assignments:

Expression	Property
oraext:generate-guid() - use expression builder	b2b.messageId
"Name"	b2b.fromTradingPartnerIdType
"Name"	b2b.toTradingPartnerIdType
"RemoteHL7Sender"	b2b.toTradingPartnerId
"Self" (or whatever the local partner name is)	b2b.fromTradingPartnerId
"ADT_A04_DocDef"	b2b.documentDefinitionName
"ADT_A04"	b2b.documentTypeName
"HL7"	b2b.documentProtocolName
"2.4"	b2b.documentProtocolVersion
"1"	b2b.messageType

Follow the steps on pages 24 through 26 in the article “Oracle SOA Suite 11g HL7 Inbound Example” at <http://blogs.czapski.id.au/2010/06/oracle-soa-suite-11g-hl7-inbound-example>, to complete this task.

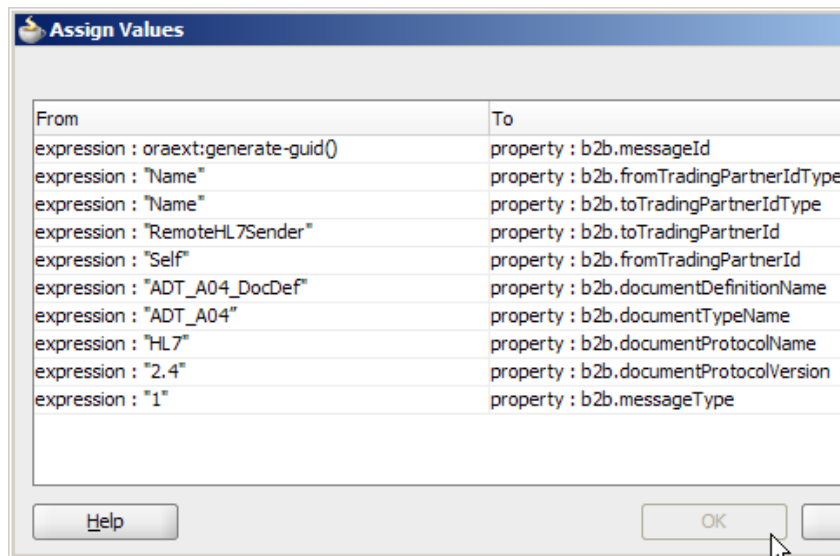


Illustration 89: B2B Exchange Properties

The final composite is shown below.

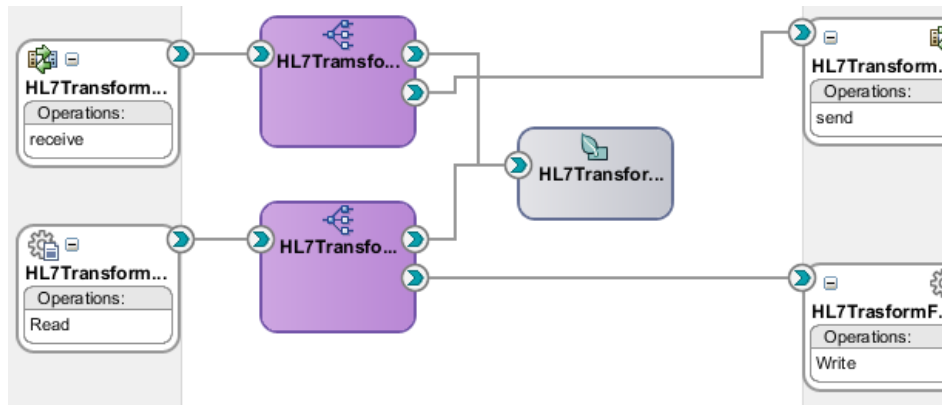


Illustration 90: Final composite

This composite, with two inbound adapters invoking a single service, implements what is referred to in EAI as the Service Activator Pattern.

Deploy this composite.

To verify that file handing functionality still works submit the test file and observe the outcome.

To exercise the solution using HL7 externals we can use the HL7 Browser, discussed in the article “Oracle SOA Suite 11g HL7 Inbound Example” at <http://blogs.czapski.id.au/2010/06/oracle-soa-suite-11g-hl7-inbound-example>, pages 2 through 6.

Start the HL7 Browser, load the ADT_A01_output_1.hl7 file, start the network utility with the receiver configured to listen on port 12122 and the sender configured to send to localhost on port 12121. Connect both and send a message.

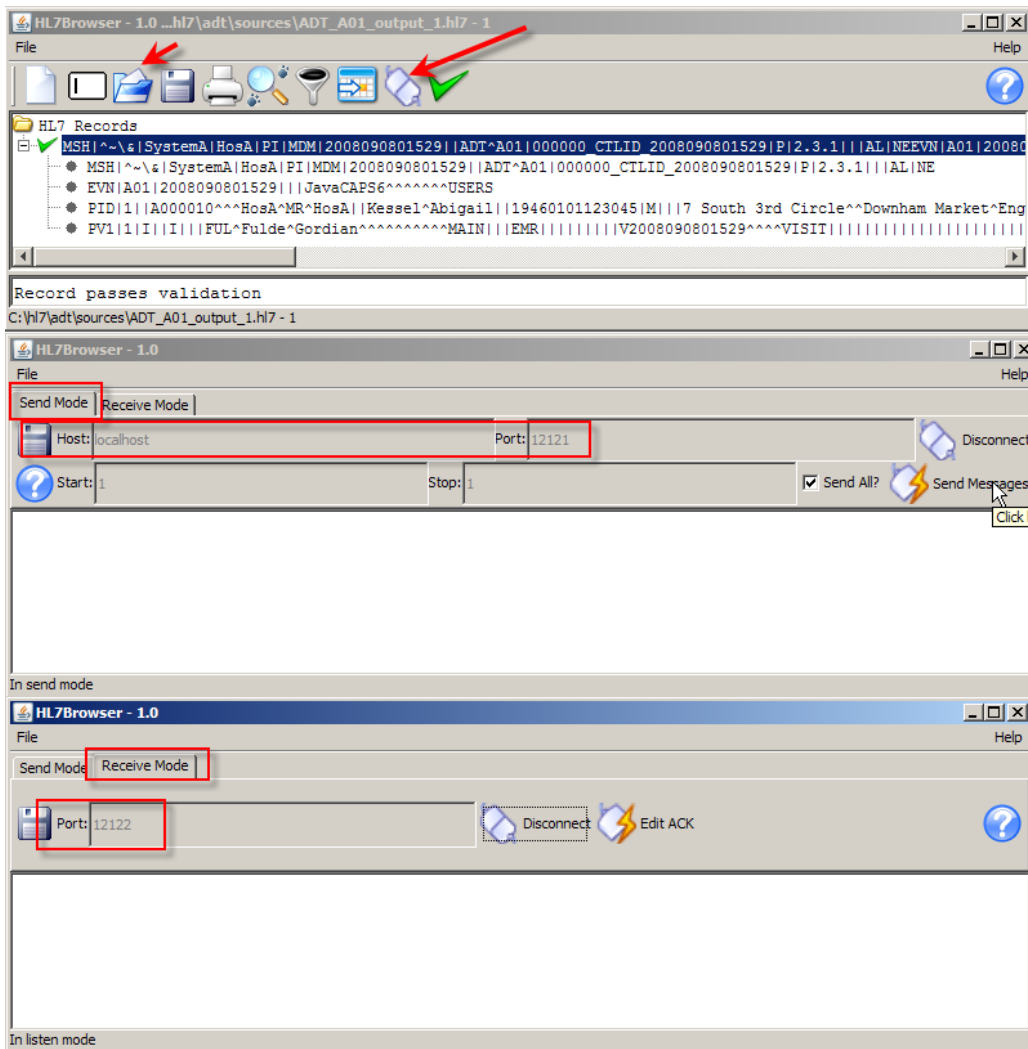


Illustration 91:

HL7 Browser configured to send and receive

Click the Send Messages button and observe the outcome.

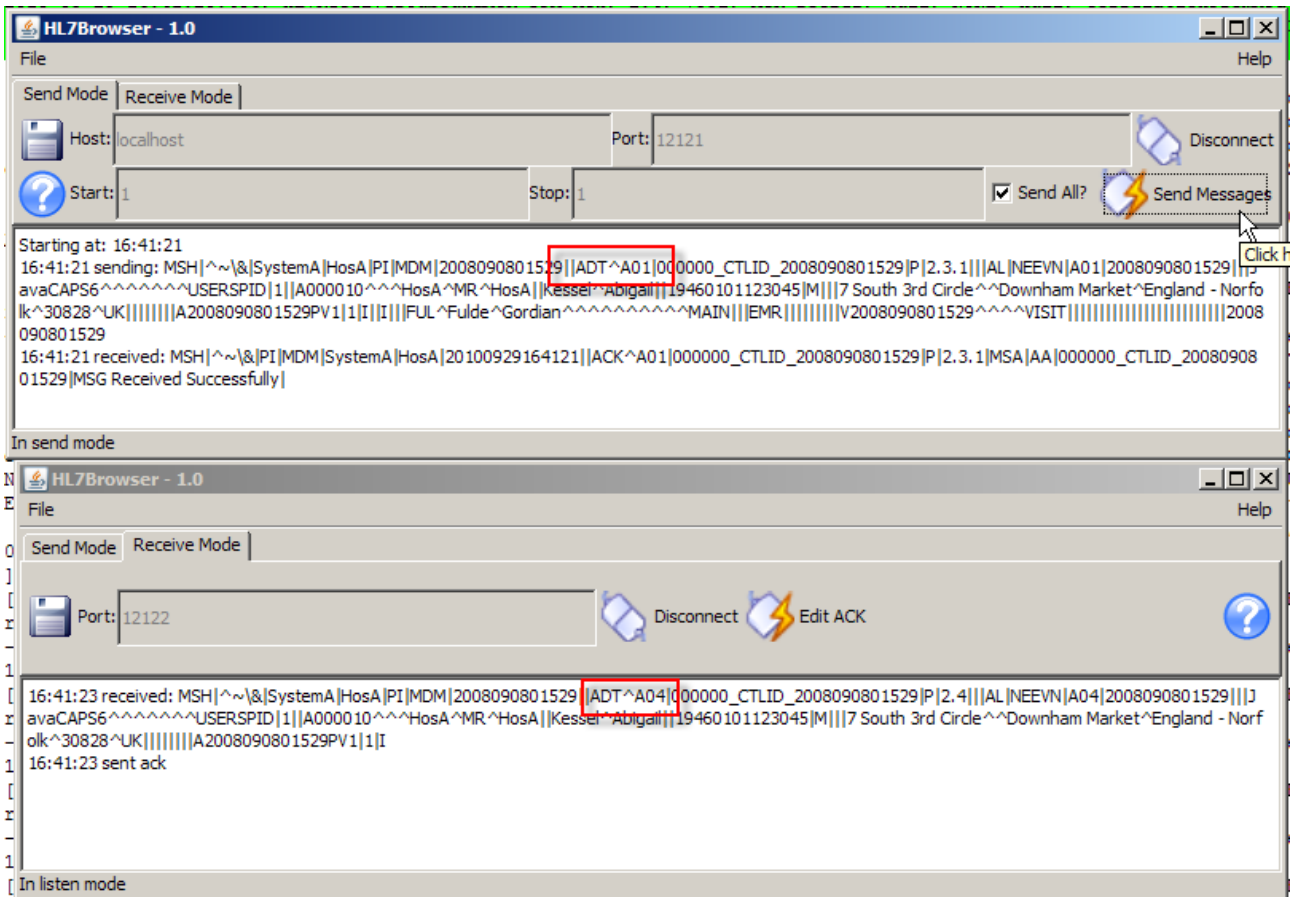


Illustration 92: ADT A01 Sent and ADT A04 Received

B2B Web Console Reports Tab will show message exchange.

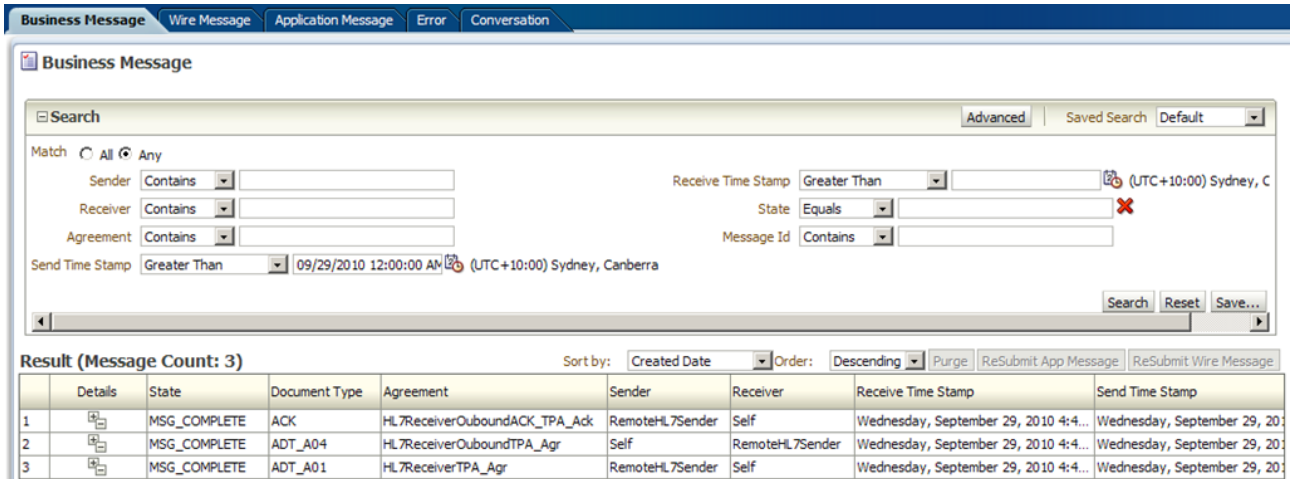


Illustration 93: Business Messages exchanged between partners

An A01 was received with a pre-set ACK sent and not shown in the list. The SOA Composite processed the message, including using the Spring Component to execute the JCD transformation logic. This is not shown in the B2B Console. Finally, an A04 message was sent and the ACK was received.

Start the Enterprise Manager and expand the SOA → soa-infra node. Click on Instances Tab and on the most recent instance id.

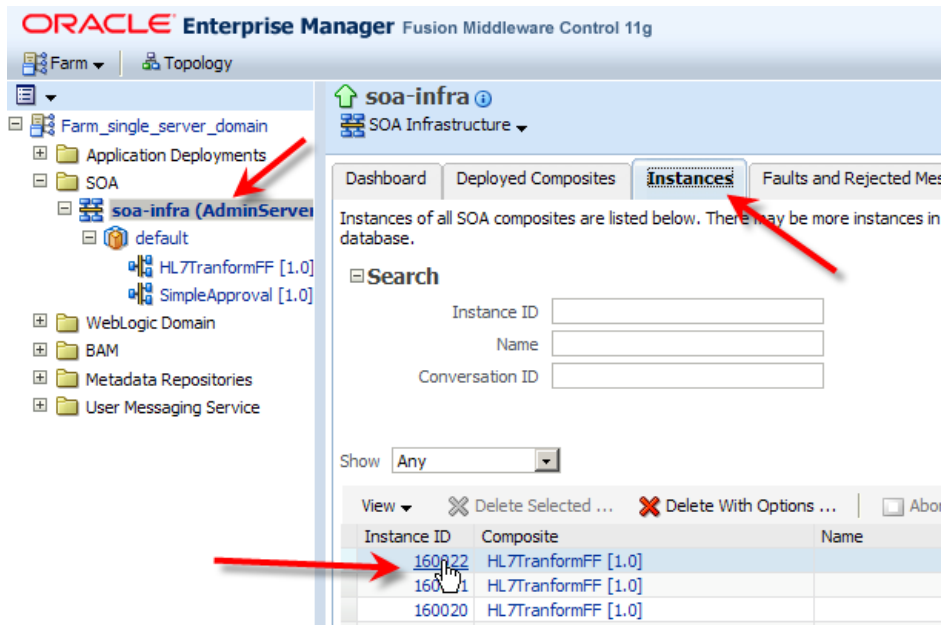


Illustration 94: Inspect instance

Execution flow trace is shown. Click on the Mediator link to see details of this instance's execution, including messages processed through the solution.

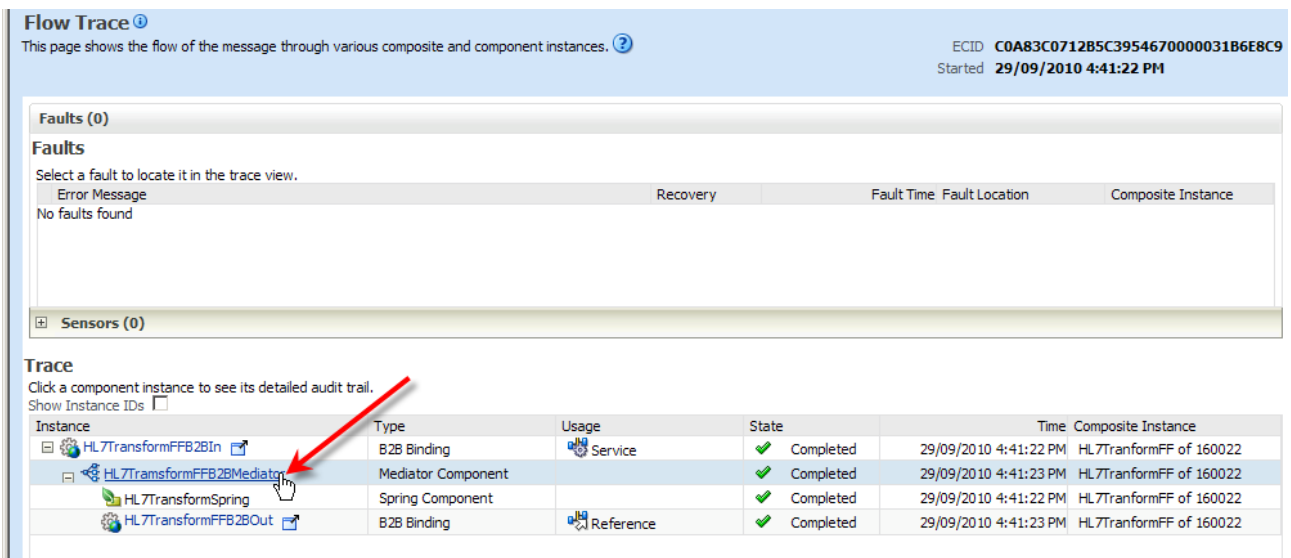


Illustration 95: Execution flow trace

Flow Trace > Instance of HL7TransformFFB2BMediator Data Refreshed 29/09/2010 4:55:21 PM

Instance Details of HL7TransformFFB2BMediator Instance ID **mediator:93137B10CB9411DFBFA6C5760DD892**
Started **29/09/2010 4:41:22 PM**

This page shows Mediator component instance details. Current Audit Level: development ? View Raw X

Audit Trail Faults

Expand the payload nodes to view the details of the instance audit trail.

```

<payload>
  <message>
    <properties>
      <property name="b2b.documentProtocolVersion" value="2.3.1"/>
      <property name="b2b.documentProtocolName" value="HL7"/>
      <property name="b2b.toTradingPartnerIdType" value="Name"/>
      <property name="b2b.documentDefinitionName" value="ADT_A01_DocDef"/>
      <property name="b2b.messageId" value="C0A83C0712B5C3954180000031B6E8C2-1"/>
      <property name="b2b.fromTradingPartnerIdType" value="MLLP ID"/>
      <property name="b2b.messageType" value="1"/>
      <property name="tracking.compositeInstanceId" value="160022"/>
      <property name="b2b.toTradingPartnerId" value="Self"/>
      <property name="tracking.ecid" value="C0A83C0712B5C3954670000031B6E8C9"/>
      <property name="b2b.fromTradingPartnerId" value="127.0.0.1"/>
      <property name="tracking.conversationId" value="C0A83C0712B5C3954670000031B6E8C8"/>
      <property name="b2b.documentTypeName" value="ADT_A01"/>
    </properties>
    <parts>
      <part name="body">
        <opaqueElement>TVNIfF5+XCZ8U3IzdGVtQXxIb3NBfFBjFE1ETXwyMDA4MDkwODAxNTI5fHxBRFRReQTAxMDAwMDAw MF9DVEExJRF8yMDA4M
        </part>
      </parts>
    </message>
  </payload>
  29/09/2010 4:41:22 PM Transformed message part "parameters" using "xsl/opaqueElement_To_HL7Transform.xsl"
  <payload>
  29/09/2010 4:41:23 PM Invoked 2-way operation "HL7Transform" on target service "HL7TransformSpring.HL7TransformService"
  <payload>
  29/09/2010 4:41:23 PM Received response from operation "HL7Transform" of service "HL7TransformSpring.HL7TransformService"
  29/09/2010 4:41:23 PM Transformed message part "body" using "xsl/HL7TransformResponse_To_opaqueElement_2.xsl"
  <payload>
  <message>
    <properties>
      <property name="tracking.compositeInstanceId" value="160022"/>
      <property name="tracking.ecid" value="C0A83C0712B5C3954670000031B6E8C9"/>
      <property name="tracking.conversationId" value="C0A83C0712B5C3954670000031B6E8C8"/>
    </properties>
    <parts>
      <part name="body">
        <ns1:opaqueElement>TVNIfF5+XCZ8U3IzdGVtQXxIb3NBfFBjFE1ETXwyMDA4MDkwODAxNTI5fHxBRFRReQTA0fDAwMDAwMF9DVEExJRF8yMD
        </part>
      </parts>
    </message>
  </payload>

```

Illustration 96: Instance details of HL7TransformFFB2BMediator

Explore.

Summary

The example developed in this article came from the healthcare domain and used the HL7 OTDs (Object Type Definitions). HL7 JCDs and JCDs used for mapping other complex messaging standards are good candidates for migrating to the SOA Suite Spring Component as means of preserving the transformation code and the effort invested in developing it. The method is applicable to all other domains where JCDs with significant transformation and mapping rules content are used.

Discussion in this article addressed a subset of technologies available in the Java CAPS and in the SOA Suite. Specifically, the Java Collaboration Definitions supported in Java CAPS 5.x and in Java CAPS 6/Repository, and the Spring Component supported in the SOA Suite 11g R1 PS2. Both use the Java programming language and related runtime environment to implement processing logic.

The HL7 eWay and JCD based Java CAPS solution was ported to the Oracle SOA Suite 11g B2B and Mediator-based solution.

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

The Mediator project was then developed and elaborated to host the JCD code in a Spring Component. This project was tested to ensure the port worked.

Finally, B2B HL7 inbound and outbound were added 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 Spring Components 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.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() );
        ;
        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().getEvn6EventOccured().getTS().setN439TimeOfAnEvent(
vA01_23.getEVN().getEvn6EventOccured().getTS().getN439TimeOfAnEvent() );
            }
        }
    }
    ;
    vA04_24.getPID().setPID_segment_ID( vA01_23.getPID().getPID_segment_ID() );
    if (vA01_23.getPID().hasPid1SetIdPatientId()) {
        vA04_24.getPID().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.getPID().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 i1 = 0; i1 < vA01_23.getPID().countPid3PatientIdInternalId(); i1 += 1) {
        if (vA01_23.getPID().getPid3PatientIdInternalId( i1 ).hasCX()) {
            if (vA01_23.getPID().getPid3PatientIdInternalId( i1 ).getCX().hasN297Id()) {
                vA04_24.getPID().getPid3PatientIdentifierList( i1 ).getCX().setN297Id(
vA01_23.getPID().getPid3PatientIdInternalId( i1 ).getCX().getN297Id() );
            }
            if (vA01_23.getPID().getPid3PatientIdInternalId( i1 ).getCX().hasN281AssigningAuthority()) {
                if (vA01_23.getPID().getPid3PatientIdInternalId( i1
).getCX().getN281AssigningAuthority().hasHD()) {
                    if (vA01_23.getPID().getPid3PatientIdInternalId( i1
).getCX().getN281AssigningAuthority().getHD().hasN342NamespaceId()) {
                        vA04_24.getPID().getPid3PatientIdentifierList( i1
).getCX().getN281AssigningAuthority().getHD().setN342NamespaceId( vA01_23.getPID().getPid3PatientIdInternalId( i1
).getCX().getN281AssigningAuthority().getHD().getN342NamespaceId() );
                    }
                }
            }
            if (vA01_23.getPID().getPid3PatientIdInternalId( i1 ).getCX().hasN252IdentifierTypeCode()) {
                vA04_24.getPID().getPid3PatientIdentifierList( i1 ).getCX().setN252IdentifierTypeCodeId(
vA01_23.getPID().getPid3PatientIdInternalId( i1 ).getCX().getN252IdentifierTypeCode() );
            }
            if (vA01_23.getPID().getPid3PatientIdInternalId( i1 ).getCX().hasN237AssigningFacility()) {
                if (vA01_23.getPID().getPid3PatientIdInternalId( i1
).getCX().getN237AssigningFacility().hasHD()) {
                    if (vA01_23.getPID().getPid3PatientIdInternalId( i1
).getCX().getN237AssigningFacility().getHD().hasN342NamespaceId()) {
                        vA04_24.getPID().getPid3PatientIdentifierList( i1
).getCX().getN237AssigningFacility().getHD().setN342NamespaceId( vA01_23.getPID().getPid3PatientIdInternalId( i1
).getCX().getN237AssigningFacility().getHD().getN342NamespaceId() );
                    }
                }
            }
        }
    }
    }
    if (vA01_23.getPID().getPid5PatientName().hasXPN()) {
        if (vA01_23.getPID().getPid5PatientName().getXPN().hasN201FamilyName()) {
            for (int i1 = 0; i1 < 1; i1 += 1) {

```

```

                vA04_24.getPID().getPid5PatientName( i1 ).getXPN().getN201FamilyName().getFN().setN386Surname(
vA01_23.getPID().getPid5PatientName().getXPN().getN201FamilyName() );
            }
        }
        if (vA01_23.getPID().getPid5PatientName().getXPN().hasN22GivenName()) {
            for (int i1 = 0; i1 < 1; i1 += 1) {
                vA04_24.getPID().getPid5PatientName( i1 ).getXPN().setN22GivenName(
vA01_23.getPID().getPid5PatientName().getXPN().getN22GivenName() );
            }
        }
        if (vA01_23.getPID().getPid5PatientName().getXPN().hasN23MiddleInitialOrName()) {
            for (int i1 = 0; i1 < 1; i1 += 1) {
                vA04_24.getPID().getPid5PatientName( i1
).getXPN().setN23SecondAndFurtherGivenNamesOrInitialsThereof(
vA01_23.getPID().getPid5PatientName().getXPN().getN23MiddleInitialOrName() );
            }
        }
        if (vA01_23.getPID().getPid5PatientName().getXPN().hasN273SuffixEGJrOrIii()) {
            for (int i1 = 0; i1 < 1; i1 += 1) {
                vA04_24.getPID().getPid5PatientName( i1 ).getXPN().setN273SuffixEGJrOrIii(
vA01_23.getPID().getPid5PatientName().getXPN().getN273SuffixEGJrOrIii() );
            }
        }
        if (vA01_23.getPID().getPid5PatientName().getXPN().hasN235PrefixEGDr()) {
            for (int i1 = 0; i1 < 1; i1 += 1) {
                vA04_24.getPID().getPid5PatientName( i1 ).getXPN().setN235PrefixEGDr(
vA01_23.getPID().getPid5PatientName().getXPN().getN235PrefixEGDr() );
            }
        }
        if (vA01_23.getPID().getPid5PatientName().getXPN().hasN203DegreeEGMd()) {
            for (int i1 = 0; i1 < 1; i1 += 1) {
                vA04_24.getPID().getPid5PatientName( i1 ).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().getPid7DateOfTimeOfBirth().getTS().setN439TimeOfAnEvent(
vA01_23.getPID().getPid7DateOfTimeOfBirth().getTS().getN439TimeOfAnEvent() );
            }
        }
    }
    if (vA01_23.getPID().hasPid8Sex()) {
        vA04_24.getPID().setPid8AdministrativeSex( vA01_23.getPID().getPid8Sex() );
    }
    if (vA01_23.getPID().hasPid11PatientAddress()) {
        for (int i1 = 0; i1 < vA01_23.getPID().countPid11PatientAddress(); i1 += 1) {
            if (vA01_23.getPID().getPid11PatientAddress( i1 ).hasXAD()) {
                if (vA01_23.getPID().getPid11PatientAddress( i1 ).getXAD().hasN27StreetAddress()) {
                    vA04_24.getPID().getPid11PatientAddress( i1
).getXAD().getN403StreetAddressSad().getSAD().setN398StreetOrMailingAddress(
vA01_23.getPID().getPid11PatientAddress( i1 ).getXAD().getN27StreetAddress() );
                }
                if (vA01_23.getPID().getPid11PatientAddress( i1 ).getXAD().hasN28OtherDesignation()) {
                    vA04_24.getPID().getPid11PatientAddress( i1 ).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.getPID().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 );
}
}

```