Oracle SOA Suite 11g R1 PS5

SOA Suite for healthcare integration Series

Domain Value Map (DVM) – On-the-fly Code Mapping

michael@czapski.id.au

January 2013

Table of Contents

Introduction 1
Solution Overview 2
Define Domain Value Map (DVM) 3
Add DVM to TransformerMediator 5
Start the "Receiving" "External Systems" 9
Send ADT messages 10
Modify runtime version of the DVM 11
Summary 13

Introduction

In this article the routing and transformation solution developed in the article entitled "SOA Suite for healthcare integration Series – Routing and Transformation using XSL Solution, will be extended to add an on-the-fly code mapping. This does not have anything to do with HL7 or HL7 processing, or indeed SOA Suite for healthcare integration infrastructure, but since the task is a pretty common one in the HL7 messaging world, it will be discussed and illustrated.

This article assumes that the reader has the SOA Suite for healthcare integration environment with all necessary components installed and ready to use. The Bill of Materials for such an environment and a discussion on where the components can be obtained is provided in the earlier article, "SOA Suite for healthcare integration Series - Overview of the Development Environment", to be found at http://blogs.czapski.id.au/2012/08/soa-suite-for-healthcare-integration-series-overview-of-the-development-environment.

This article assumes that the reader completed the solution discussed in the earlier article, "SOA Suite for healthcare integration Series – Routing and Transformation using XSL Solution", to be found at http://blogs.czapski.id.au/2012/12/soa-suite-for-healthcare-integration-series-routing-and-transformation-using-xsl-solution.

Solution Overview

Domain Value Map, the infrastructure functionality found, amongst others, in the Oracle XSL transformation technology implementation, allows one to define a "mapping table" consisting of a set of an original values and a substitution values, and a default (when an original value is not found), and to configure a XSL transformation to perform the substitution as transformation proceeds. In this case the "mapping table" is stored as a XML file, which can be transiently modified at runtime using a web-based UI. Other mechanisms, which use database tables, also exists, but will not be discussed here.

The solution components are depicted in Figure 1 Solution Components.

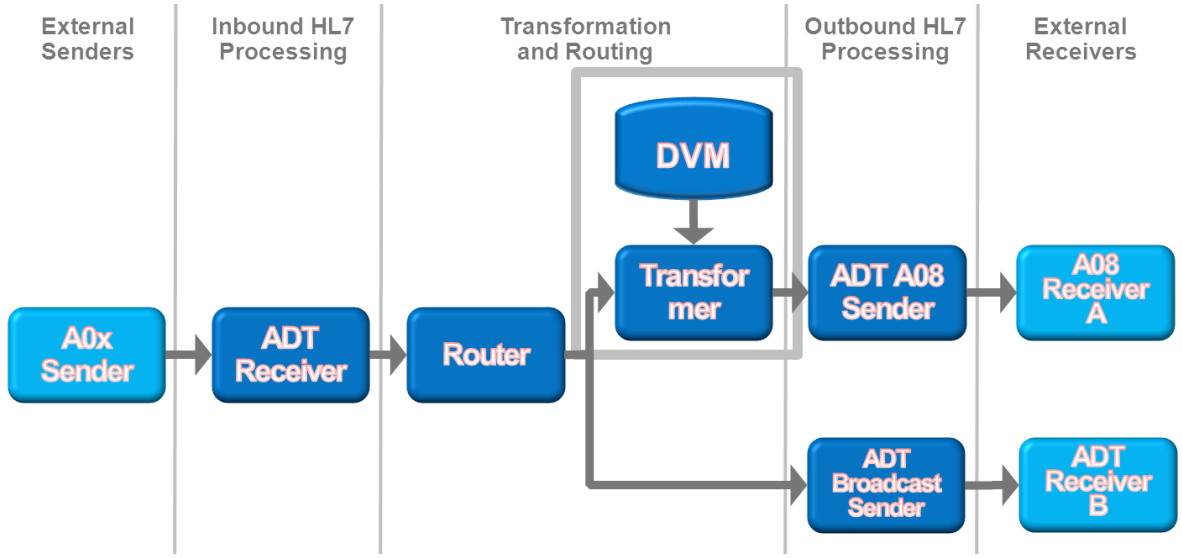


Figure 1 Solution Components

The diagram uses the convention which clearly separates the external systems, the SOA Suite for healthcare integration-specific components and generic SOA Suite components using the "swim-line" analogy.

Since the entire solution, with the exception of the on-the-fly mapping, already exists, we will only modify the "TransformerMediator" component.

As before, the solution will receive a HL7 v2 message, acknowledge it with an immediate acknowledgement and pass it onto the SOA Composite. The immediate acknowledgement will be sent as soon as the message is received and persisted, before it is processed in any way. The acknowledgement received from the HL7 listener will be discarded.

As the "TransformerMediator", which implements and XSL transformation, mapps the segments, fields, conponents and subcomponents of the incoming ADT A01 or A03 message to construct the outgoing ADT A08 message, it will translate / map the gender code, carried in the PID-8 field, inot its equivalent long version, for example "M" into "MALE".

We need to create the Domain Value Map (DVM) and add the DVM Lookup function in the appropriate place in the XSLT to accomplish the taks.

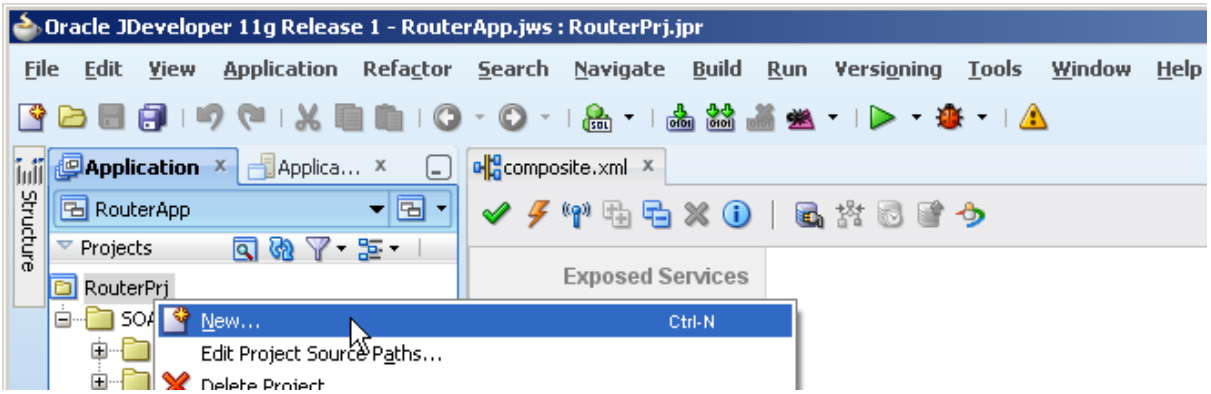
Define Domain Value Map (DVM)

In this section we will define the Domain Value Map, the mapping table containing original and substitute codes.

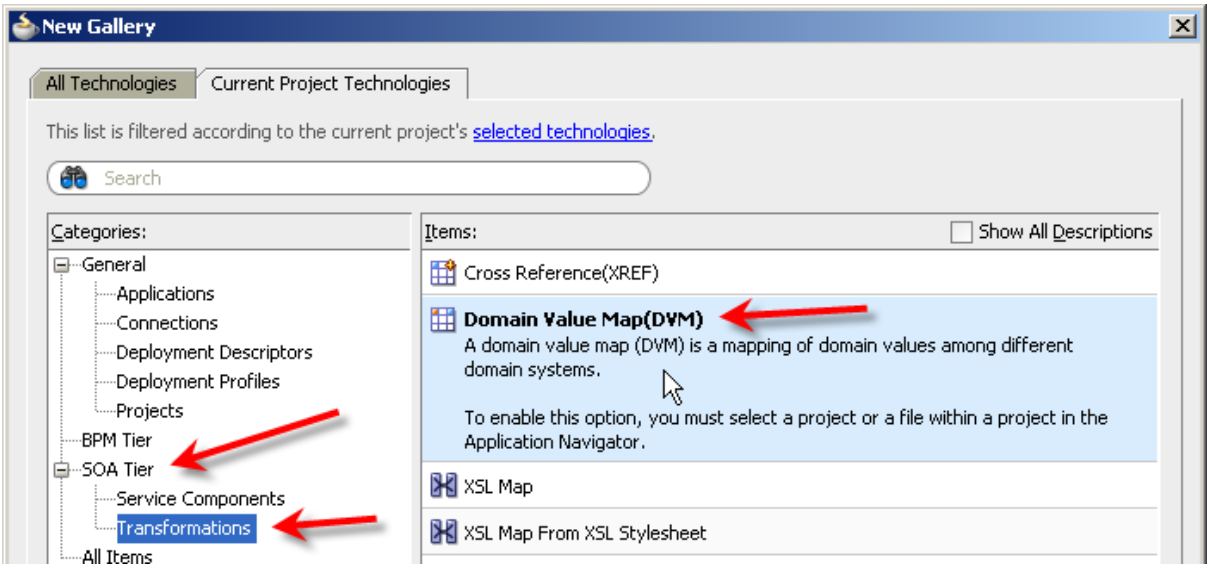
☐ Start JDeveloper Studio

☐ Make sure the "RouterApp" Application is open – open it is it is not

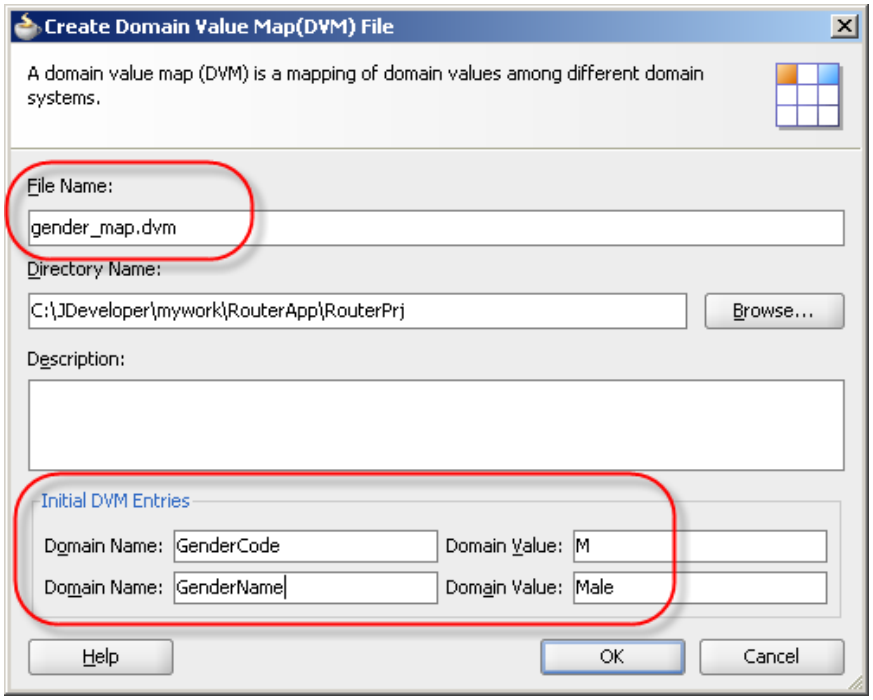
☐ Right-click the name of the project, "RouterPrj" and choose "New…"



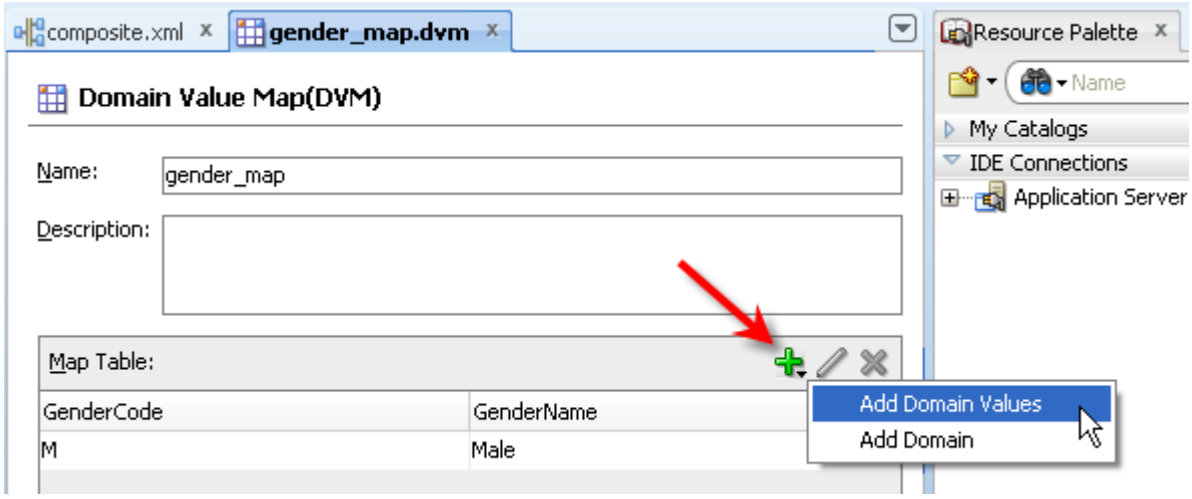
☐ Choose "SOA Tier" → "Transformations" → "Domain Value Map(DVM)"



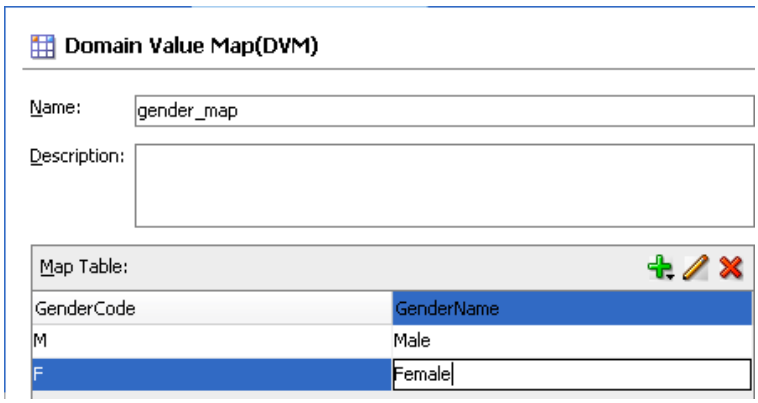
☐ Name the new DVM "gender_map.dvm", replace "domain1" with "Gender Code", "value1" with "M", "domain2" with "GenderName", "domain2" with "Male" and click "OK"



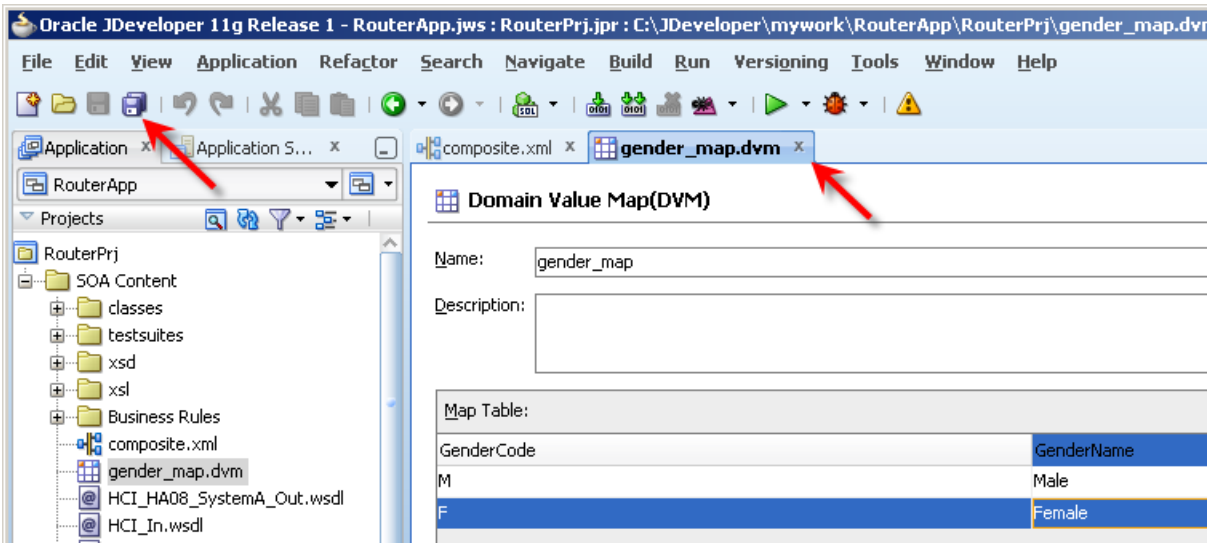
☐ Pull down the "Plus" button and choose "Add Domain Values"



☐ Enter "F" for "GenderCode" and and "Female" for "GenderName"



☐ Click "Save All" and close the domain value map



We are done. The domain value map is ready. Let's modify the "TransformerMediator" to perform the actual mapping.

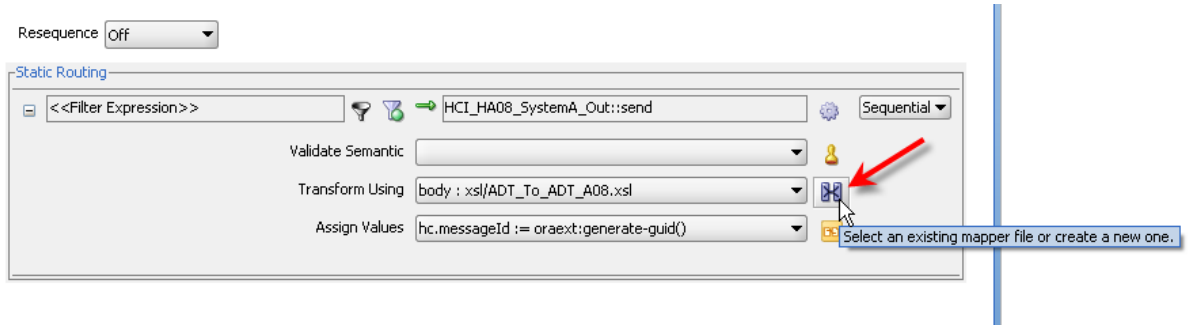
Add DVM to TransformerMediator

In this section we will modify the "TransformerMediator"'s XSLT-based mapping file to perrofmr code mapping using the domain value map we which created earlier.

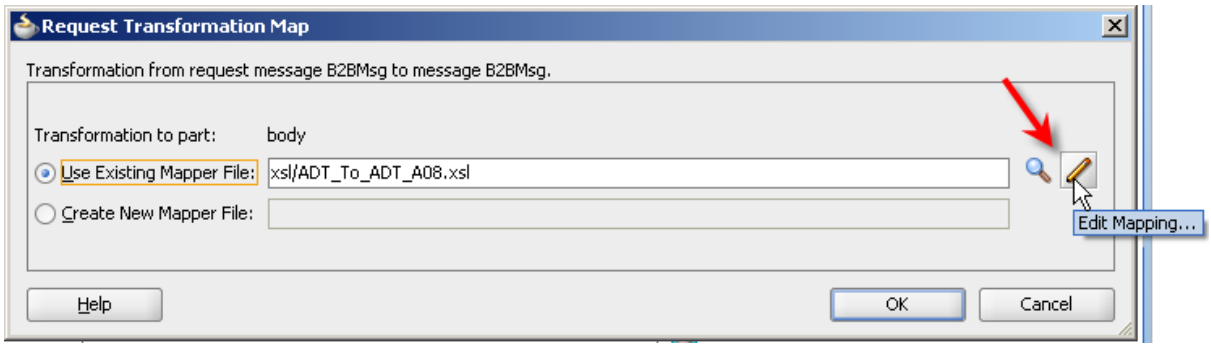
☐ Open the "composite.xml", if not already open

☐ Double-click the "TransformerMediator" component to open the "mplan" editor

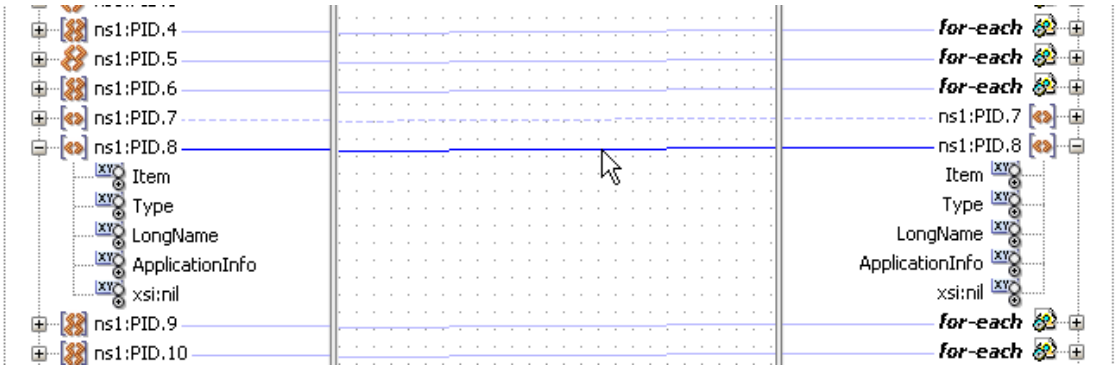
☐ Click the "Mapper File" button



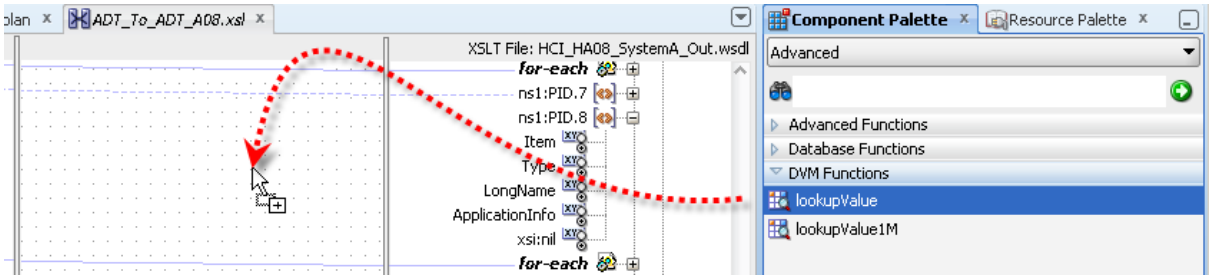
☐ Click the "Edit Mapping…" button



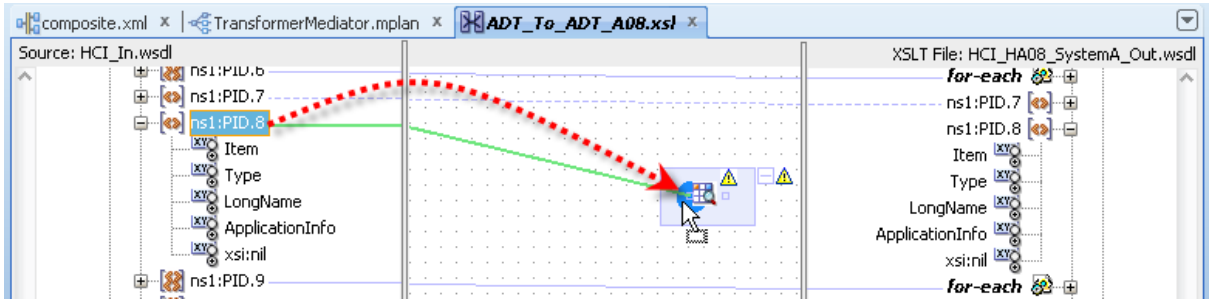
☐ Expand PID node trees on both sides by one level, expand PID-8 node trees on both sides, repeat for each of the mapping lines between PID-8 field and its attributes: click the line to select it and press the "Del" key to delete the mapping



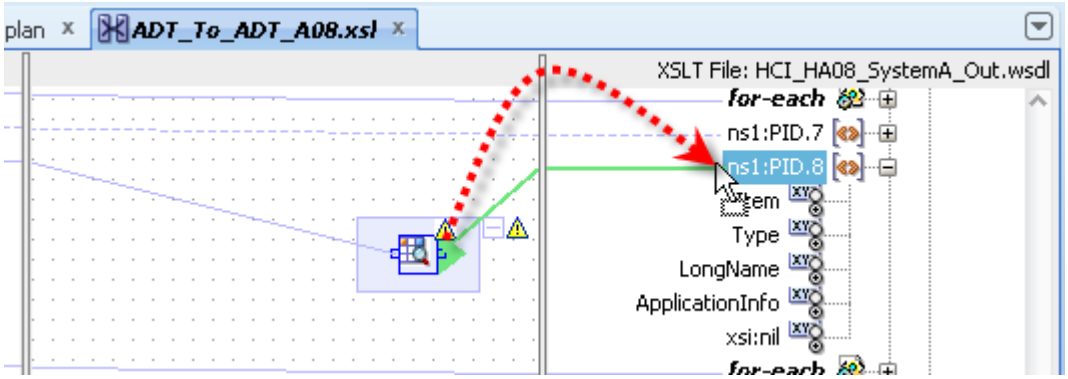
☐ Select "Advanced" from the "Component Palette", expand the "DVM Functions" and drag the "lookupValue": function onto the center swim line in the empty space alongside the PID-8 node



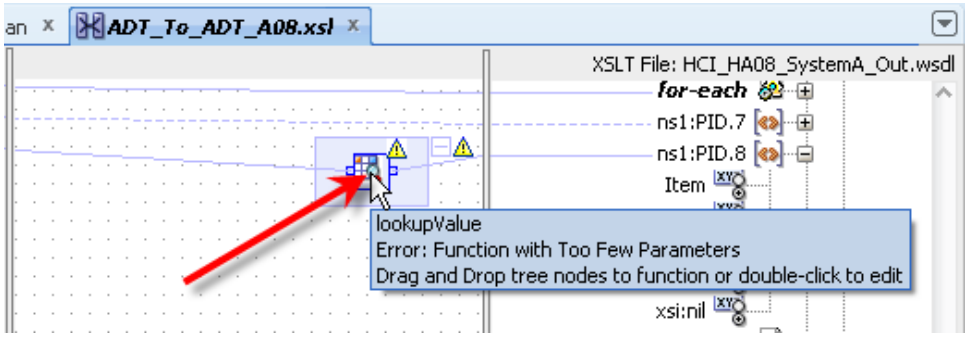
☐ Drag from the left-had side PID-8 node to the tiny square at the right of the function box to connect the node to the function, providing node value as the argument



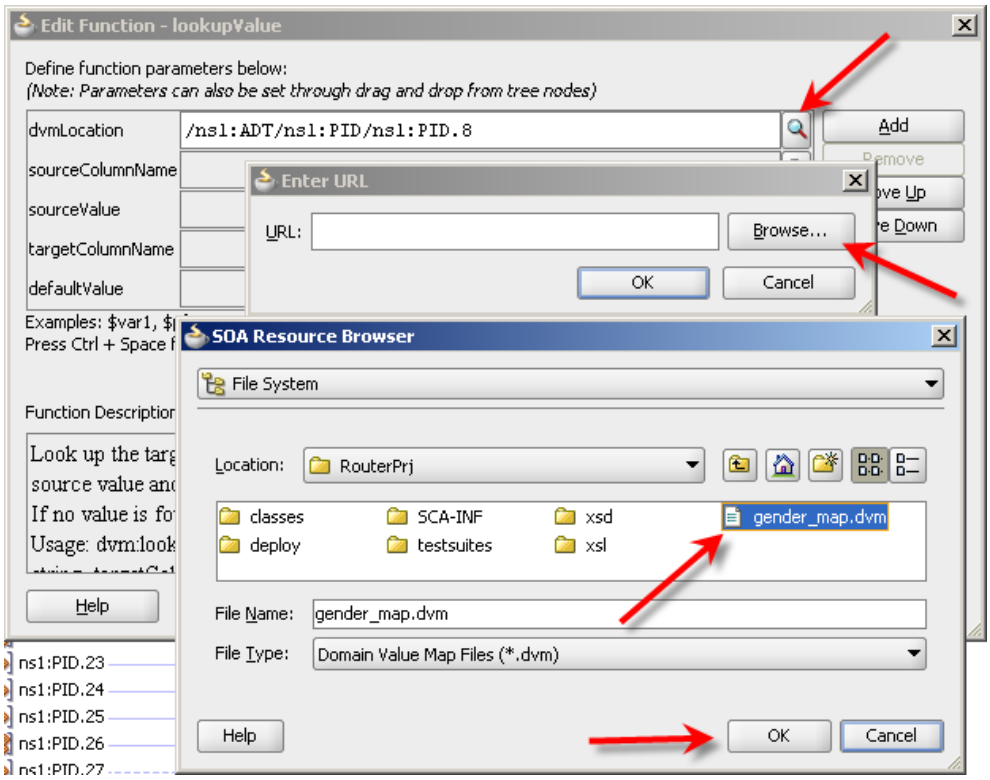
☐ Drag from the tiny square at the right og the function to the PID-8 node oin the right hand node tree to assign the function result to the PID-8 node



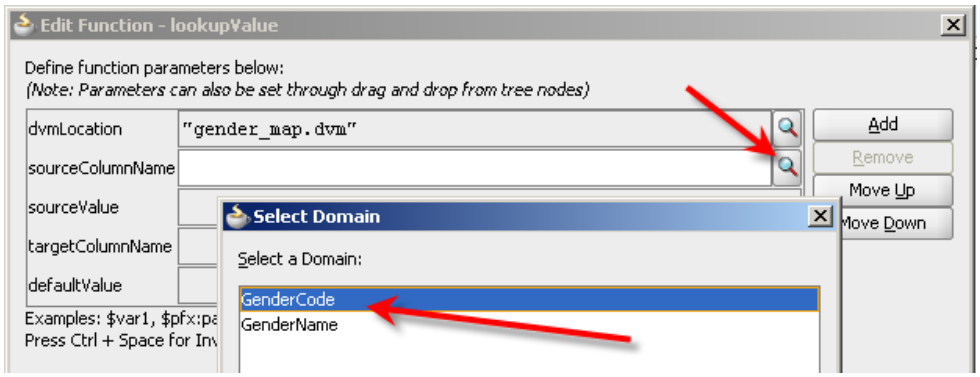
☐ Double-click the center of the function to start the configuration wizard



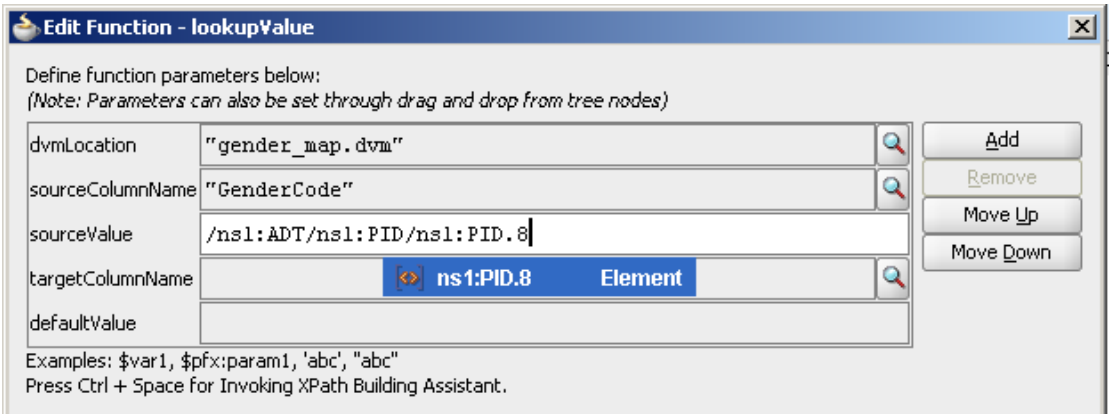
☐ Click on the "looking glass" button alongside the "dvmLocation" box, click the "Browse…" button, locate and select the "gender_map.dvm" then click "OK" and "OK" to complete DVM file selection



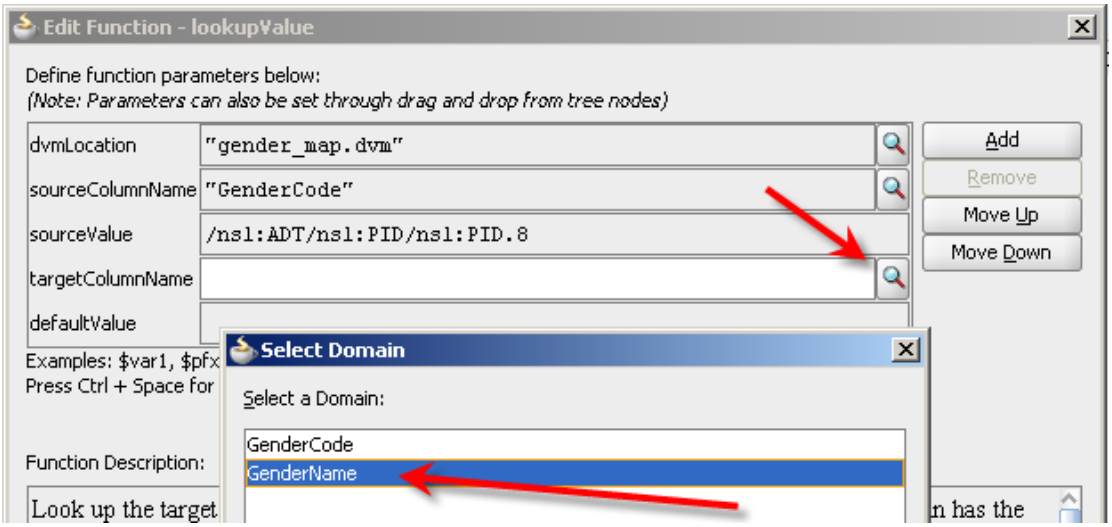
☐ Click the "magnifying glass" icon alongside the "sourceColumnName" box, choose "GenderCode" and click "OK" to complete setting the value



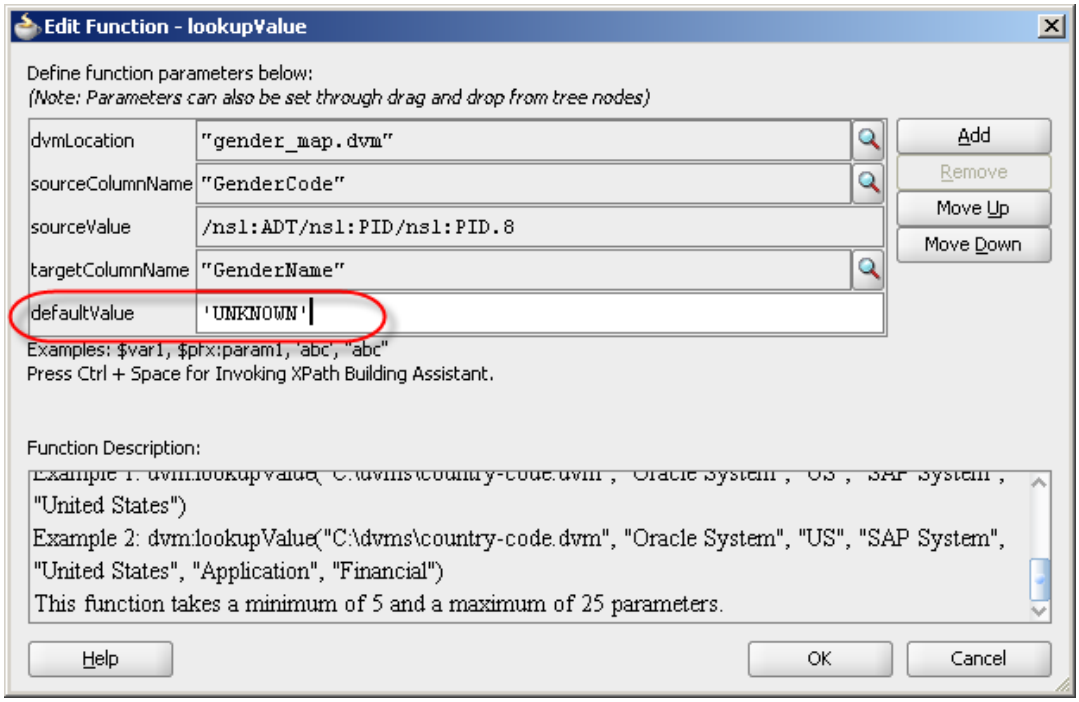
☐ Click the data entry box alongside the "sourceValue" label, press control+spacebar to obtain the value list, choose "ns1:ADT" and press enter to expand one level, and continute doing so until you have "/ns1:ADT/ns1:PID/ns1:PID.8" selected - make sure PID.8 is NOT followed by a "/"



☐ Click the "magnifying glass" button alongside the "targetColumnName" label, choose "GenderName" and click "OK" to complete selection



☐ Enter "'UNKNOWN'" into the text box alongside the "defaultValue" label and click "OK" to complete function configuration – this is the value which will be populated in the target node when the GenderCode is not found in the DVM



☐ "Save All" and deploy the project

Start the "Receiving" "External Systems"

We will use the CMDHL7Listener command line client to receive HL7 ADT messages look at the output in the output directory specified on the listener's command line – for me c:\hl7\received. The CMDHL7Listener will display trace of message exchange in the console window. The SOA Suite for healthcare integration will record message tracking information which we will look at a later stage.

Please note that in this solution the CMDHL7Listener returns an ACK as soon as it gets the message.

☐ Check that your configured output directory is empty

☐ To start the listener for the "BcastB_SystemA_Out" sender endpoint, in a command / terminal window execute the following command

```
java -jar c:\tools\CMDHL7\CMDHL7Listener_v0.7.jar -c ID_ -p 22200 -s
c:\hl7\received
```

☐ Inspect the CMDHL7Listener console output making sure the listener started and is listening on the appropriate port

```
08/12/2012 10:45:36 AM au.id.czapski.hl7.CMDHL7Listener main
INFO: Port: 22200
08/12/2012 10:45:36 AM au.id.czapski.hl7.CMDHL7Listener main
INFO: Store in: c:\hl7\received
08/12/2012 10:45:36 AM ca.uhn.log.HapiLogImpl info
INFO: au.id.czapski.hl7.SimpleACKApplication registered to handle *^*
messages
08/12/2012 10:45:36 AM ca.uhn.log.HapiLogImpl info
INFO: SimpleServer running on port 22200
```

☐ To start the listener for the "HA08_SystemA_Out" sender endpoint, in a command / terminal window execute the following command

```
java -jar c:\tools\CMDHL7\CMDHL7Listener_v0.7.jar -c ID_ -p 22100 -s
c:\hl7\received
```

Send ADT messages

As before, we will use the CMDHL7Sender command line client to read files containing a single HL7 ADT message and submit them to the ADT Receiver endpoint. We will then look at the console output produced by the CMDHL7Listener which we started earlier, then look at the output in our configured output directory – for me c:\hl7\received, and finally review message tracking information in the Healthcare Integration Console.

Please note that in this solution the receiver endpoint returns immediate ACK as soon as it gets the message. There may be a delay, most noticeable the first time one executes the processing flow after application server restart, between the receipt of the ACK and the time the message is written to a file in the file system.

☐ Check that your configured output directory is empty

☐ Locate the input file containing a single HL7 message - for me this will be C:\hl7\adt\sources\ADT_A01_output_1.hl7

The content of my file, where each segment starting with the 3 character segment ID in bold text is a single line up to the next 3 character segment ID, looks like this:

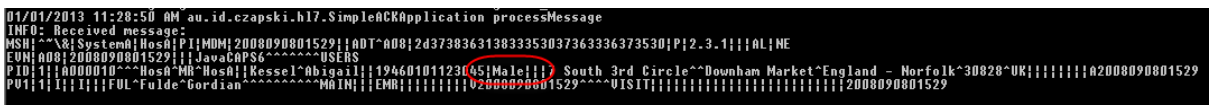
```
MSH|^~\&|SystemA|HosA|PI|MDM|2008090801529||ADT^A01|000000_CTLID_2008
090801529|P|2.3.1|||AL|NE
EVN|A01|2008090801529|||JavaCAPS6^^^^^^^USERS
PID|1||A000010^^^HosA^MR^HosA||Kessel^Abigail||19460101123045|M|||7
South 3rd Circle^^Downham Market^England -
Norfolk^30828^UK||||||||A2008090801529
PV1|1|I||I|||FUL^Fulde^Gordian^^^^^^^^^^MAIN|||EMR|||||||||V200809080
1529^^^^VISIT|||||||||||||||||||||||||2008090801529
```

☐ In a command / terminal window execute the following command

```
java -jar c:\tools\CMDHL7\CMDHL7Sender_v0.7.jar -a SystemA -b HosA -c ID_
-n 1 -d \r\r\n -p 22222 -h localhost -t 30000 -f
c:\hl7\adt\sources\ADT_A01_output_1.hl7
```

☐ Locate the output files in the received directory and inspect them to confirm that a) A01 or A03 and A08 have been written and b) that A01/A03 has the same content as the input file and the A08 has the same content as the input file except for the tree changes we made in the "TransformerMediator" component. Take particular note of the Gender value – which will be "Male"

The content of the A08 output file looks like this:



The content of the file named "383133323336303138313537343835 32_ADT_A08_1356512791863.hl7" is the same as the message which was sent except for the fields and components which we explicitly modified.

☐ Submit the ADT A03 file, `ADT_A03_output_1.hl7`, and inspect the output.

Our solution works to the extent of receiving HL7 v2.3.1 messages, and acknowledging them, transforming ADT messages to A08 for one stream of processing and writing them to files in the file system.

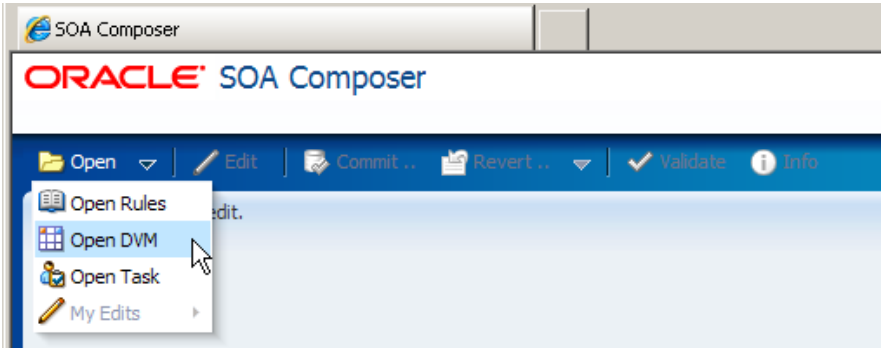
The Domain Value Map, which we implemented in this article, sees the gender codes, "M" and "F", translated into gender names, "Male" and "Female".

If you feel particularly keen, modify the input file to use a gender code other than "M" or "F", submit it and look for "UNKNOWN" in the gender field

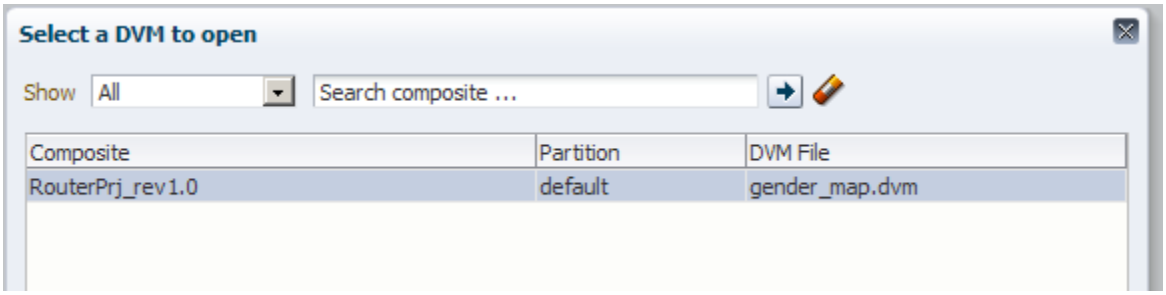
Modify runtime version of the DVM

In this section we will explore the mechanism for modifying the DVM at runtime

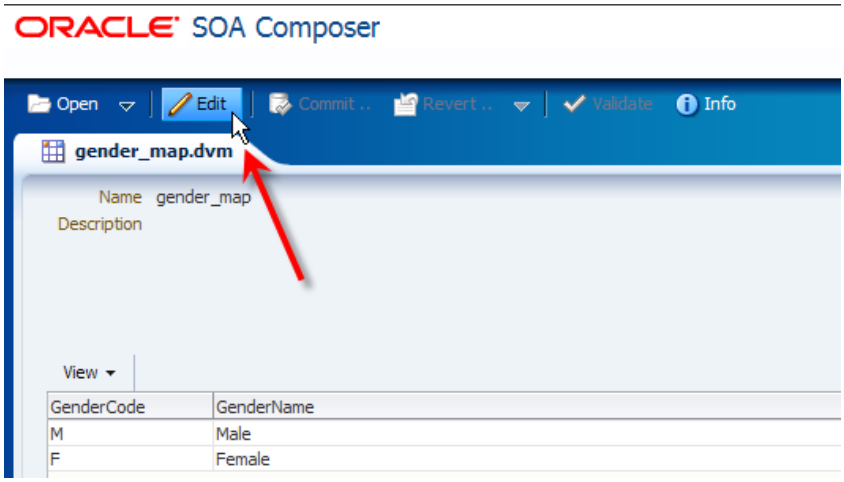
- ☐ Start the COS Composer – http://localhost:7001/soa/composer
- ☐ Log in with your credentials – mine are weblogic/welcome1
- ☐ Pulls down the "Open" menu and choose "Open DVM"



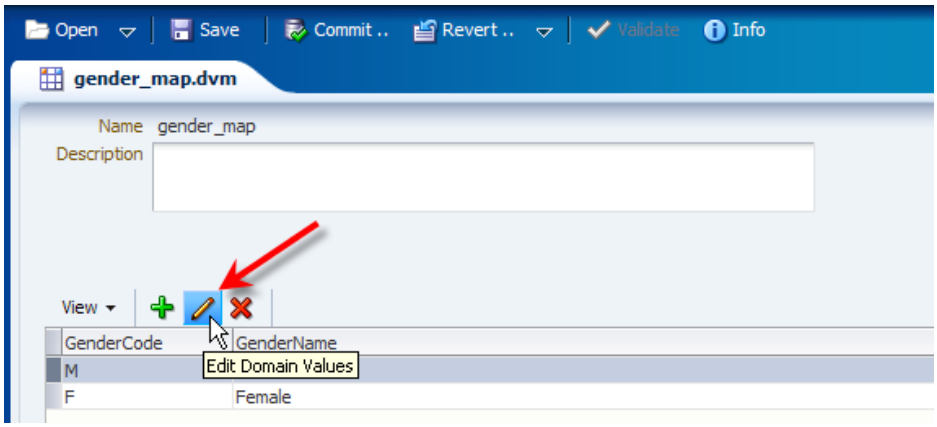
- ☐ Select the DVM and click "Open"



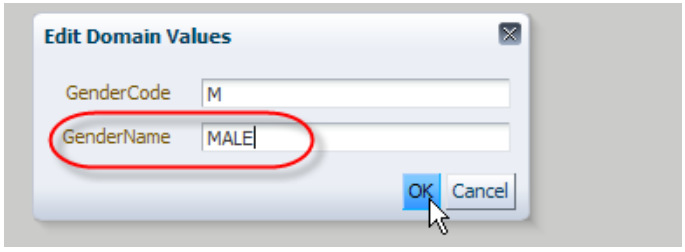
- ☐ Click "Edit" button



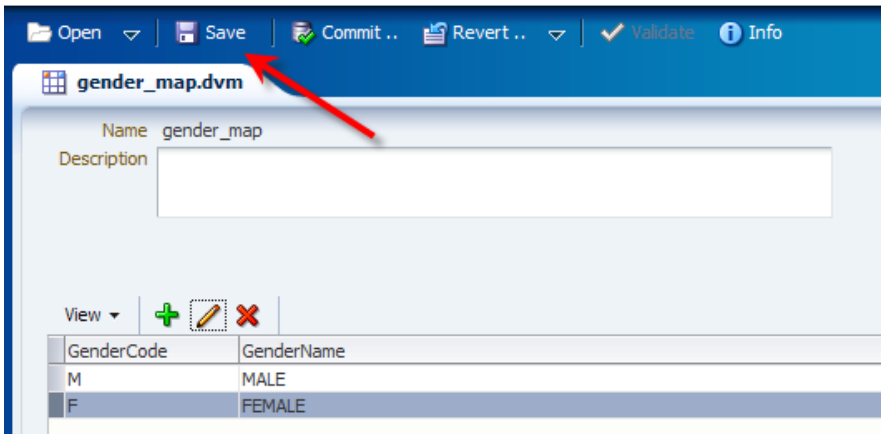
☐ Select the DVM entry and click "Edit"



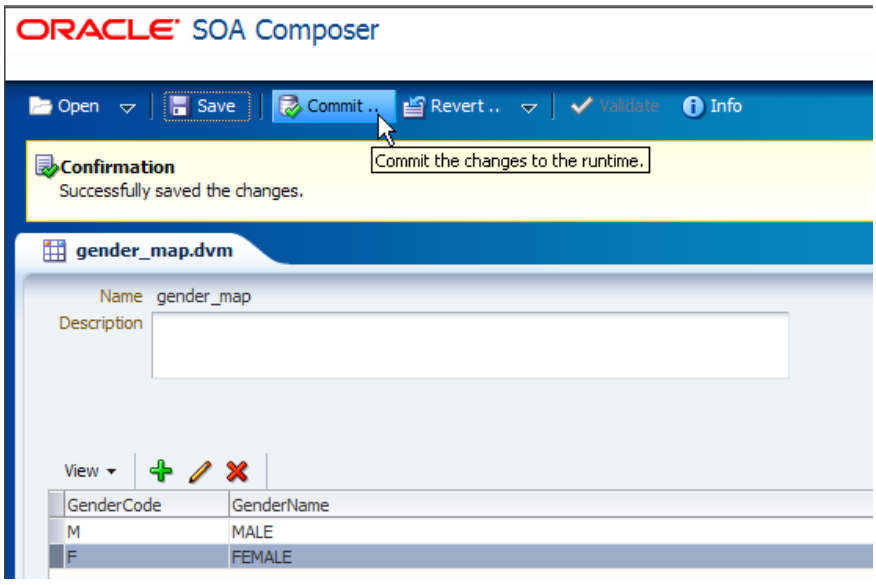
☐ Change the vale from "Male" to "MALE" and click "OK"



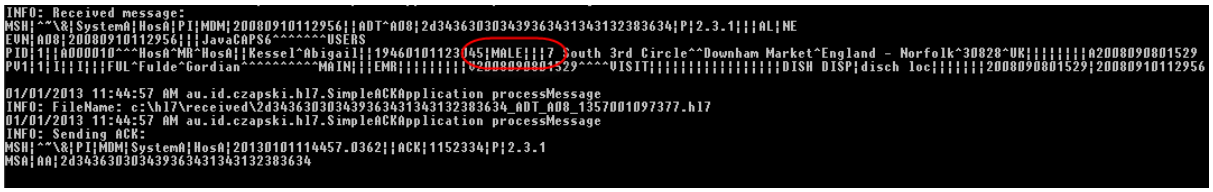
☐ Repeat the process changing "Female" to "FEMALE" and click the "Save" button to save the changes



☐ Click "Commit" button to commit the changes to the runtime, confirm commit and exit from the SOA Composer



☐ Submit a message and observe new outcome



Note that the changes will be lost when the application is re-deployed. To make the changes permanent one must edit the DVM in JDeveloper (or using a text editor in the file system) so that re-deployment of the application propagates the changes made to runtime through the SOA Composer

Summary

In this article the router developed in the previous article in the series was modified to perform an on-the-fly code mapping. The design-time and runtime mapping modificaoitn were also explored.