Java CAPS 6/JBI and OpenESB

HL7 Processor Demonstration

Michael.Czapski@sun.com

December, 2008

Table of Contents

Introduction

This Note walks the reader through development of a Java CAPS 6/JBI-based / OpenESB-based solution that addresses a Healthcare-related business problem. The Note elaborates on the healthcare background necessary to get a notion of what is being done and why, and provides detailed steps required to implement and exercise the solution to the business problem.

We will use the HL7 Binding Component, the File Binding Component, the JMS Binding Component, the SOAP/HTTP Binding Component, the BPEL 2.0 Service Engine, the JavaEE Service Engine, the HL7 Encoder and EJB-based Web Services in a JBI-based solution.

In the process we will create XML Schema Documents (XSDs), Web Services Description Language Documents (WSDLs), a BPEL 2.0 Business Process, an EJB-based “Implementation First” web service, an EJB- and WSDL-based “Interface First” web service, a bunch of Composite Applications, BPLE 2.0 mapping, BPEL 2.0-based Web Service orchestration, on-the-fly conversion of HL7 version 2.3.1 delimited messages to their XML equivalents. We will get a pretty good exposure to what OpenESB and Java CAPS 6/JBI components look like, how they work and how

they can be used to create real business solutions. Above all, we will develop and test a solution that is more sophisticated then the customary “Hello World” examples but not so complex as to take too long to build and become too hard to comprehend by a novice user.

The particular business problem and the particular solution came about because once upon a time there was intent to build a series of related OpenESB projects – HL7 Processor, MDM Processor and IEP Processor - that would:

- receive HL7 v2.x delimited messages
- convert HL7 v2.x messages to their equivalent XML format
- split message stream into ADT A01s, ADT A03s and other
- convert A01s to an abbreviated Custom Patient XML format
- convert A03s to an abbreviated Custom Discharge format
- send Custom Patients to a JMS Queue for processing by a MDM solution
- send Custom Discharges to a JMS Queue for processing by an IEM solution

- have the MDM process Custom Patients into a Master Patient Index
- have the IEP process Custom Discharges to flag excessive length of stay

The MDM Processor and the IEP Processor made it to the Sun CEC 2008 as demonstrations, with associated Tutorials by Tom Barrett, and demonstration recordings by me. The HL7 Processor did not make it. With the appearance of Java CAPS 6 Update 1 more JBI components made it into the officially supported Sun product. While the HL7 BC and the HL7 Encoder did not make it into this Update they will, eventually. Both components are already available from the OpenESB site and can be installed into the Java CAPS 6 Update 1 installation as unsupported components. This is what we will do for this Note.

> **Note** that I use Australian English spelling and my own spelling errors and speech mannerisms. You may find it strange since the former differs from the US English spelling and the latter may be not altogether English since I am not a native English speaker. Do the best you can with that – I do. ☺

Prerequisites

This Note assumes the use of the Java CAPS 6 Update 1. The OpenESB distribution from around December 2008/January 2009 should work as well but has not been tested. Feel free to do so and to add a comment to say how you went ☺

Materials Provided in the Archive

This Note should be accompanied by an archive containing supplementary material you can use to save yourself the trouble of downloading component and, above all else, containing test files with HL7 delimited records. These will be used later to test our projects.

The archive is named 00_HL7Processor_example_final.zip and contains the following objects (not that project directories, containing projects developed in the Note, are not recoursed into):

```
Listing archive: 00_HL7Processor_example_final.zip

   Date      Time   Name
------------------- ------------
2008-09-08 14:00:04 00_HL7\data\sources\ADT_A01_one_tx.dat
2008-09-08 21:59:22 00_HL7\data\sources\ADT_A03_one_tx.dat
2008-12-31 15:55:28 00_HL7\data\sources\ADT_A0x_output1.dat
2008-12-31 15:59:52 00_HL7\data\sources\ADT_A0x_output10.dat
2008-12-31 15:56:06 00_HL7\data\sources\ADT_A0x_output2.dat
2009-01-01 12:42:24 00_HL7\documents
2008-12-31 17:01:08 00_HL7\HL7Consumer_CA
2009-01-01 12:36:32 00_HL7\HL7Feeder_CA
2009-01-01 12:36:22 00_HL7\HL7Processor
2009-01-01 12:36:24 00_HL7\HL7Processor_CA
2008-12-30 10:37:12 00_HL7\prerequisites\com-eviware-soapui-netbeans-module-2.0.2.nbm
2008-12-27 12:12:14 00_HL7\prerequisites\com-sun-encoder-hl7.nbm
2008-12-28 14:23:50 00_HL7\prerequisites\encoderlib.jar
2008-12-27 12:07:28 00_HL7\prerequisites\hl7bc.jar
2008-12-27 18:31:02 00_HL7\prerequisites\hl7v2xsd.zip
2008-12-27 12:15:24 00_HL7\prerequisites\org-netbeans-modules-encoder-hl7-aip.nbm
2008-12-27 12:07:36 00_HL7\prerequisites\org-netbeans-modules-wsdlextensions-hl7.nbm
2009-01-01 12:36:22 00_HL7\WSSConvertDate
2009-01-01 12:36:24 00_HL7\WSSDateDiff
------------------- ------------
                    959 files
```

Note, in particular, that the prerequisite objects to be downloaded from the OpenESB site are already in the archive and that sample data is available in the data/sources directory.

Obtain prerequisite components

Java CAPS 6 Update 1 does not support HL7 Binding Component and the related HL7 Encoder. To make them available they need to be obtained from the OpenESB site and installed. OpenESB already includes these components so there is no need to obtain and install them separately if your environment is a recent OpenESB distribution.

Obtain and install the HL7 Binding Component, the HL7 Encoder Library and the HL7 version 2.x XML Schema Definition archive into your Java CAPS 6 Update 1 installation. All of these will be used in the development of the solution. How and why will be explained as we go along

The HL7 Binding Component is available at https://open-esb.dev.java.net/HL7BC.html. Download the hl7bc.jar and the org-netbeans-modules-wsdlextensions-hl7.nbm from that location to a convenient directory.

The HL7 Encoder NetBeans support is available at http://download.java.net/jbi/binaries/open-esb-full-install/nbm/latest/. Download from that location the HL7 Encoder NetBeans design-time modules to a convenient directory:

- com-sun-encoder-hl7.nbm
- org-netbeans-modules-encoder-hl7-aip.nbm

The HL7 Encoder Library is a part of the Encoder Library available at http://download.java.net/jbi/binaries/open-jbi-components/main/nightly/latest/ojc/. Download the encoderlib.jar to a convenient directory. While Java CAPS 6 includes the sun-encoder-library component it does not contain support for HL7. One must uninstall this shared library and replace it with the shared library that contains support for HL7. The installation / update process is discussed later.

The HL7 version 2.x XML Schema Documents can be downloaded from the HL7 Encoder page at the OpenESB site: http://wiki.open-esb.java.net/attach/HL7/hl7v2xsd.zip. Download the XML Schemas and unzip the archive to a convenient directory.

> **Note** that if you are modifying your supported Java CAPS 6 Update 1 environment by installing unsupported components it behoves you to back up the entire installation beforehand. If you get into trouble with your modified environment, Sun Support may be unwilling to spend the time helping you out unless you can reproduce the issue in a completely supported environment, so you may need to restore from the backup.

Install the HL7 Encoder

Update the Encoder Shared Library using the encoderlib.jar, downloaded earlier. Shut down the HL7 and the File Binding Components, if they are running.
Click on the Services Tab in NetBeans. Expand the Servers->GlassFish V2->JBI->Shared Libraries node, right-click it and choose Uninstall.

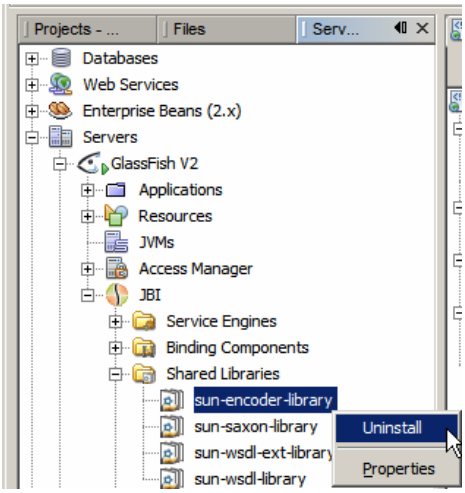


Figure 0-1 Uninstall Shared Libraries->sun-encoder-library

Right-click on the Shared Libraries node and choose Install …

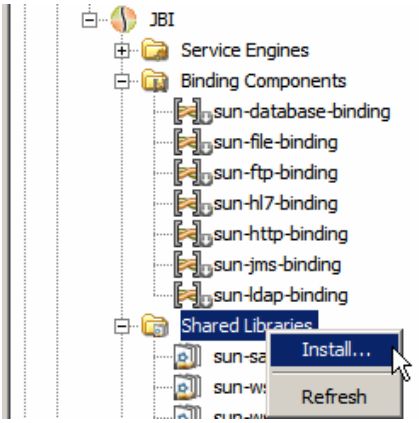


Figure 0-2 Start installation of the encoder library

Locate and choose the encoderlib.jar archive and click Install.

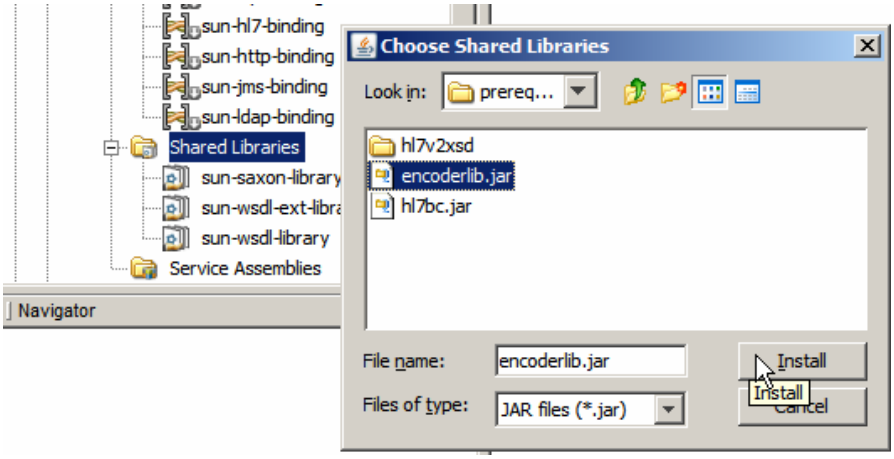


Figure 0-3 Complete installation of the encoder library.

To install the NetBeans Plugins, *.nbm, open the Plugins Manager from the Tools->Plugins menu.

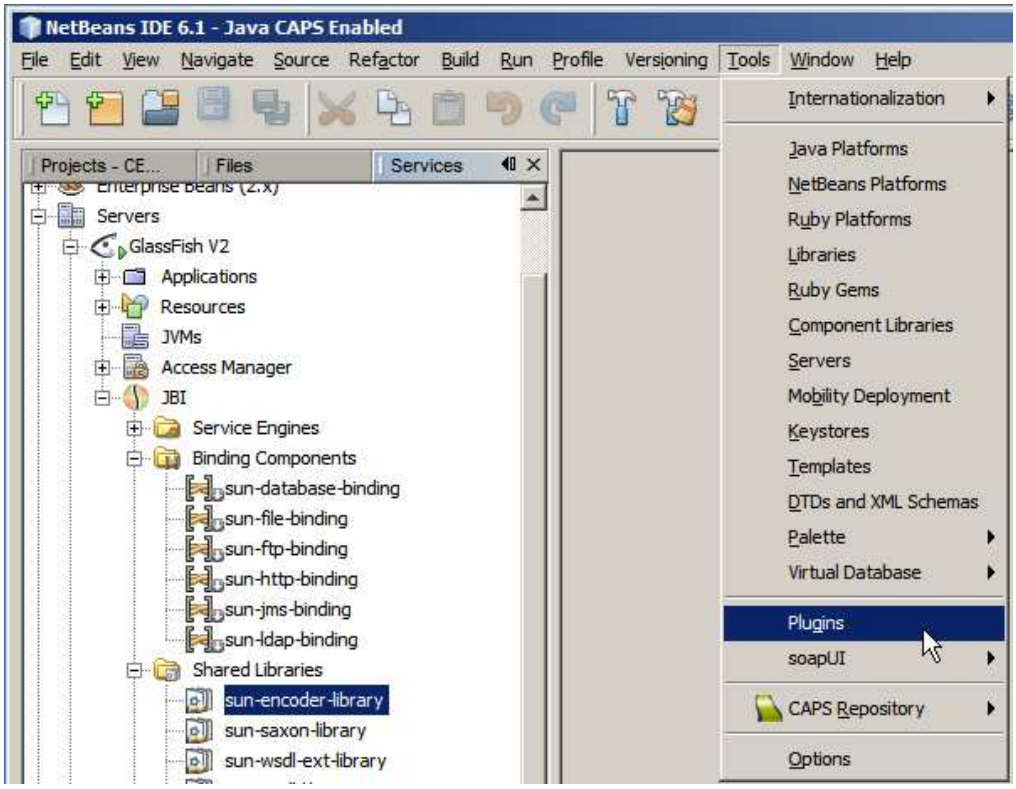


Figure 0-4, Triggering the Plugins Manager Wizard

Click on the Downloaded Tab, click on the Add Plugins … button, choose the plugins and click Open.

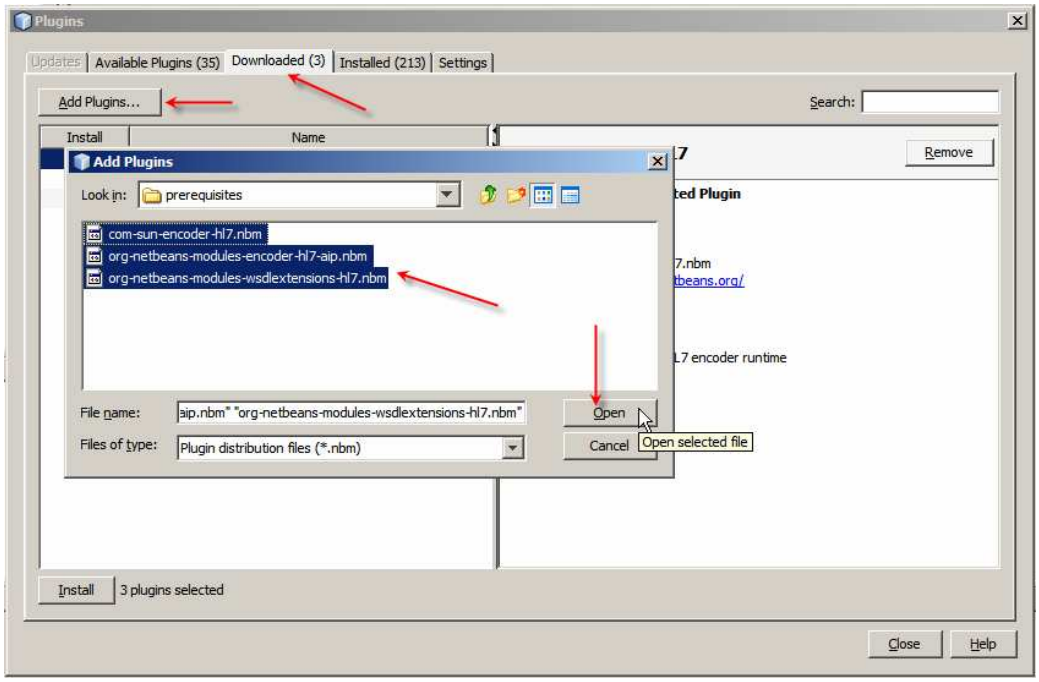


Figure 0-5 Choosing Plugins to be installed

With the plugins selected click the Install button.

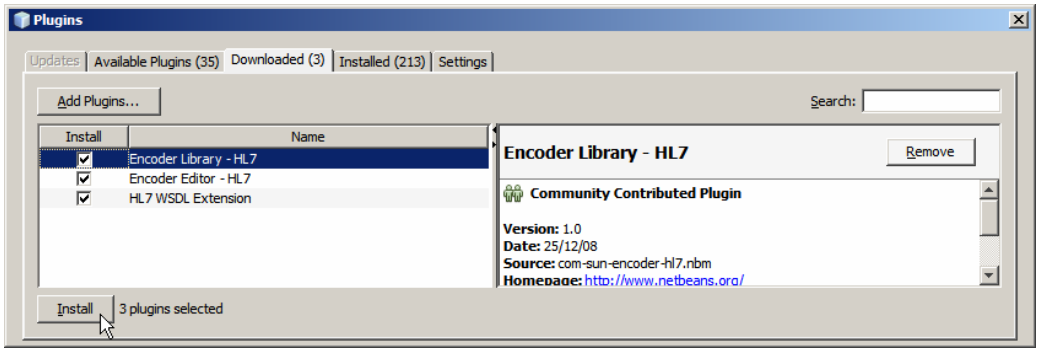


Figure 0-6 Initiating plugin installation

Click Next, check the Accept the license agreement checkbox and click Install

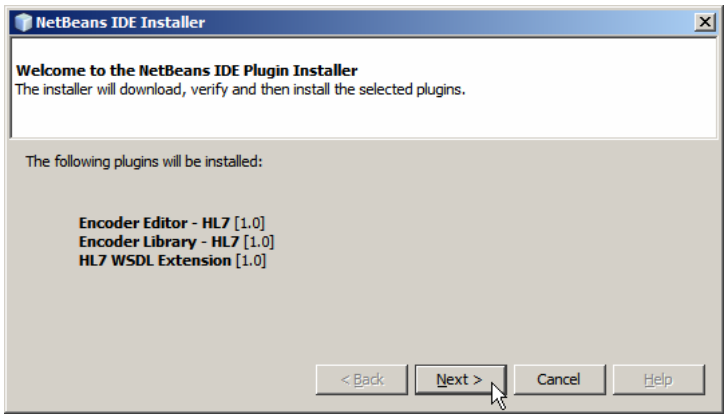


Figure 0-7 Acknowledge selection

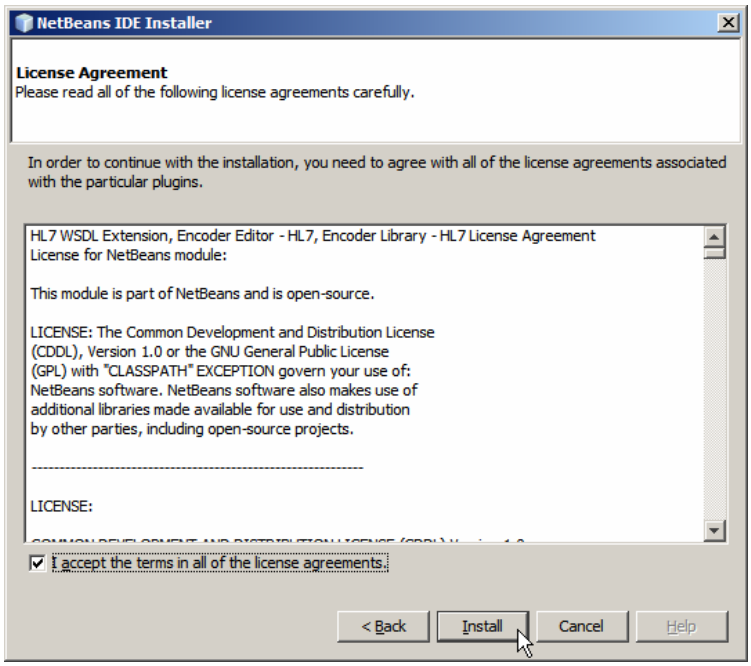


Figure 0-8 Accept license agreement and install

Ignore warning about unsigned plugins.

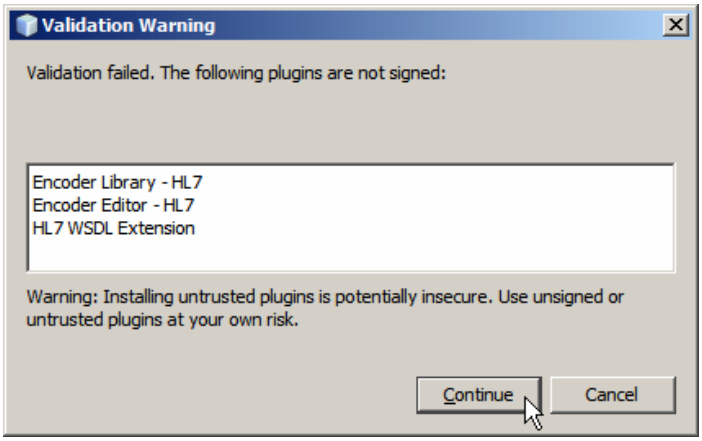


Figure 0-9 Unsigned plugins warning

Restart the IDE once plugins are installed.

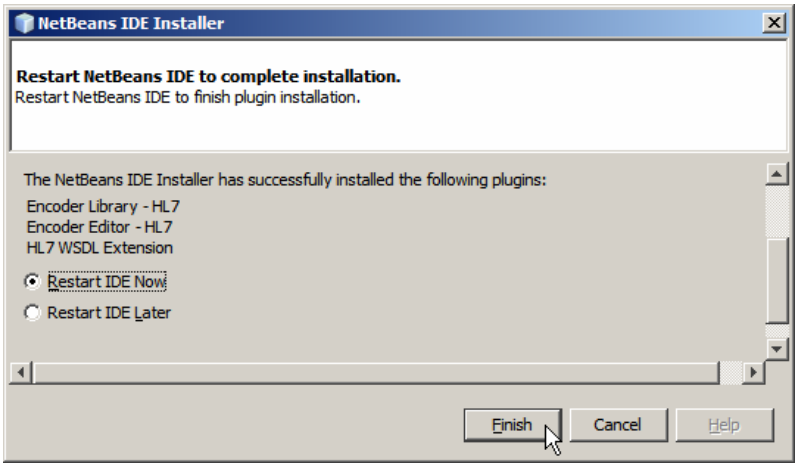


Figure 0-10 Allow IDE to be restarted

Once the IDE restarts, confirm that the plugins were installed by triggering the Plugins Manager, clicking at the Installed Tab and locating the HL7 Encoder amongst other encoders.

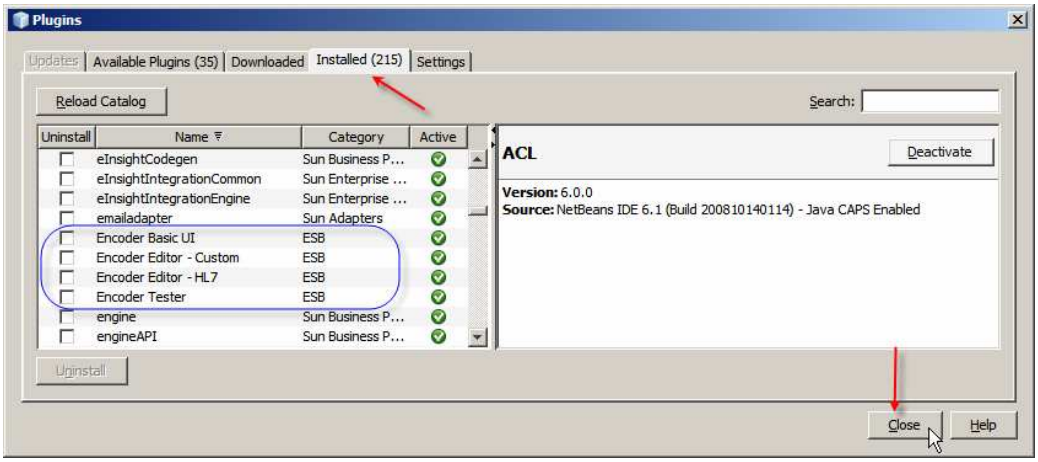


Figure 0-11 Confirming installation of the HL7 Encoder

Confirm, also, that the HL7 WSDL Extension for the HL7 Binding Component, to be installed next, is available.

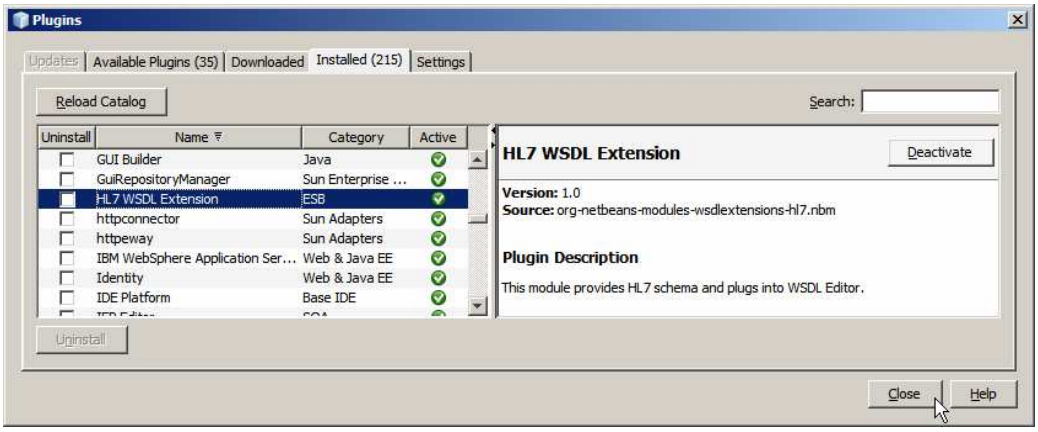


Figure 0-12 Confirming that the HL7 WSDL Extension plugin has been installed

Install the HL7 BC

Since we installed the HL7 WSDL Extensions NetBeans plugin module in the previous step half the installation is already done.

To complete installation of the HL7 Binding Component start the GlassFish Application Server that is bundled with Java CAPS 6 and switch to Services Tab in the NetBeans IDE. Expand the Servers node through to Servers->GlassFish V>JBI, right click on the Binding Components node and choose Install and Start ....

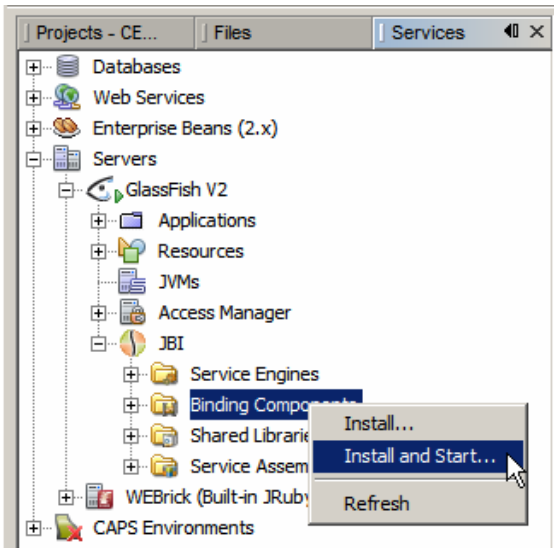


Figure 0-13 Starting Binding Component installation wizard

Locate the hl7bc.jar in the file system and click Install.

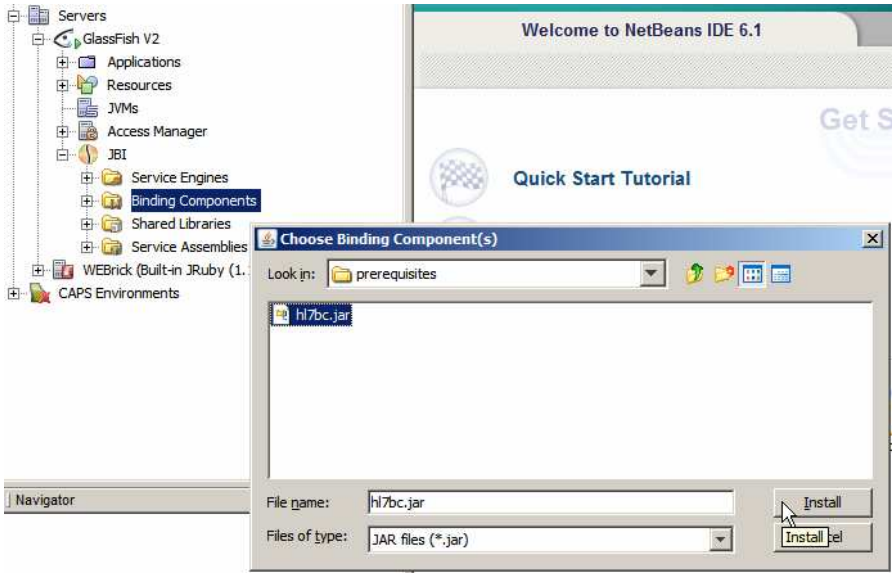


Figure 0-14 Beginning installation of the Binding Component

Check the "Allow Dynamic Endpoint" checkbox and click Install.

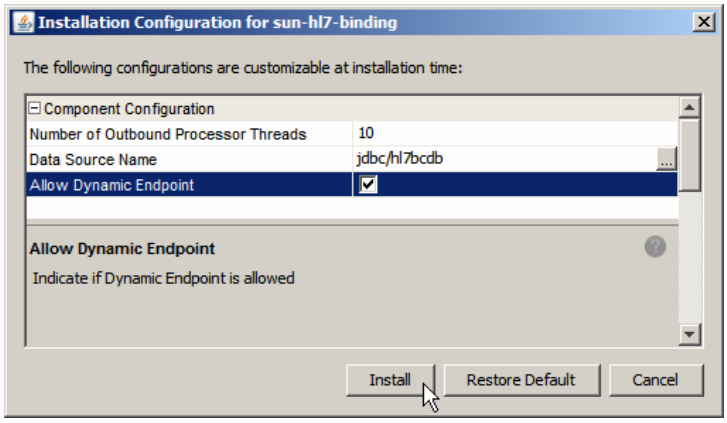


Figure 0-15 Continuing installation of the HL7 BC

Once installation finishes expand the Binding Components node to confirm that the HL7 Binding Component has been installed and started.

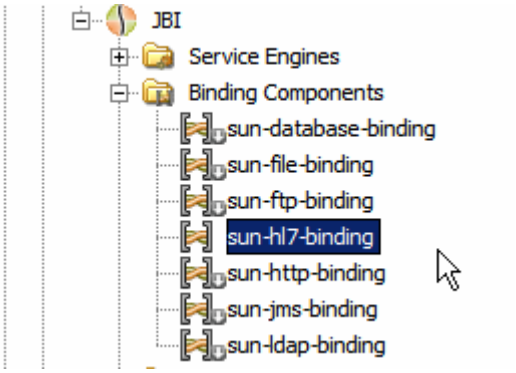


Figure 0-16 HL7 BC has been installed and started

Optional 3rd Party Software

7Scan

There are a number of 3rd party software solutions that aid in dealing with HL7. One handy solution is the 7Scan solution, http://www.7scan.com/. While one can use it to create and parse HL7 messages, which is handy, one can also use it as a sender or a receiver of HL7 messages, supporting the MLLP protocol and HL7 Acknowledgments over TCP/IP. I used it to send test messages to the HL7Consumer listener, developed later. At the time of this writing a 20 day trial download was available. I actually have a licensed copy so I did not download and install the trial. I assume the trial works the same way as a licensed copy does.

> **Note** that I am not associated with the company which develops and sells 7Scan. I don't know if it is good, bad or indifferent, amongst the many HL7 tools. I use it occasionally and it works well enough for me. You don't have to use it. If you have a tool you like use that if it helps you.

If you don't have a HL7 tool to use we will develop a JBI-based HL7 feeder project later anyway.

Assuming you have a copy of the 7Scan tool you can configure a HL7 sender. Start the 7Scan and load a HL7 message or a batch of messages – there are a number in the prerequisites/sources directory. Click the Build Tcp/ip icon and configure the sender.

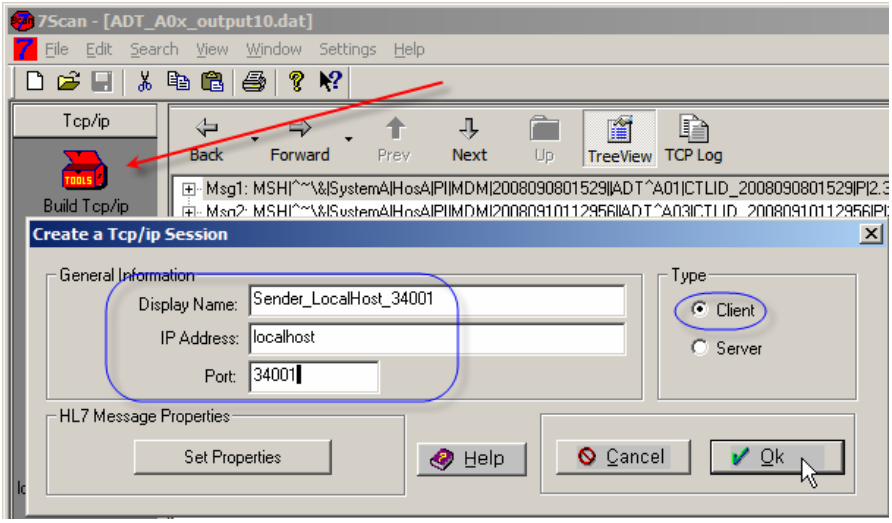


Figure 0-1 Configuring 7Scan as a HL7 sender

Click the Send Properties button to confirm or change MLLP delimiters. The default ones are:

- Start Block Character (Start Char): 11 (dec) == 0x0B (hex)
- End Block Character (1st End Chars): 28 (dec) == 0x1C (hex)
- End Data Character (2nd End Chars): 13 (dec) == 0x0D (hex)

It is important to make sure the client and the listener use the same set of characters.

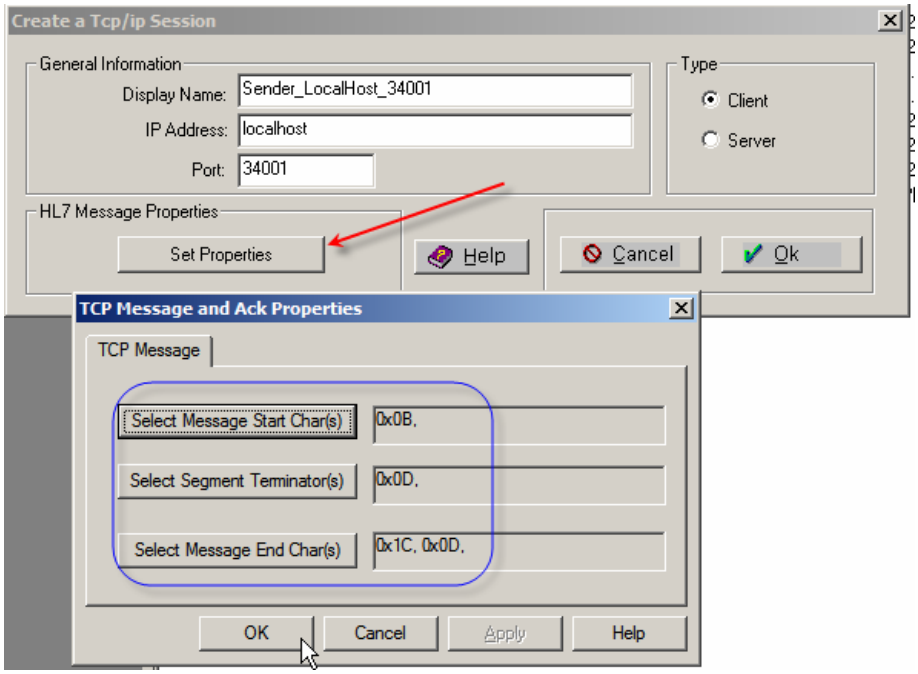


Figure 0-2 Configuring/confirming MLLP data block delimiters

Click the newly created Sender to make it connect to the remote listener – this assumes that there is a remote HL7 listener on the nominated host listening on the nominated port.

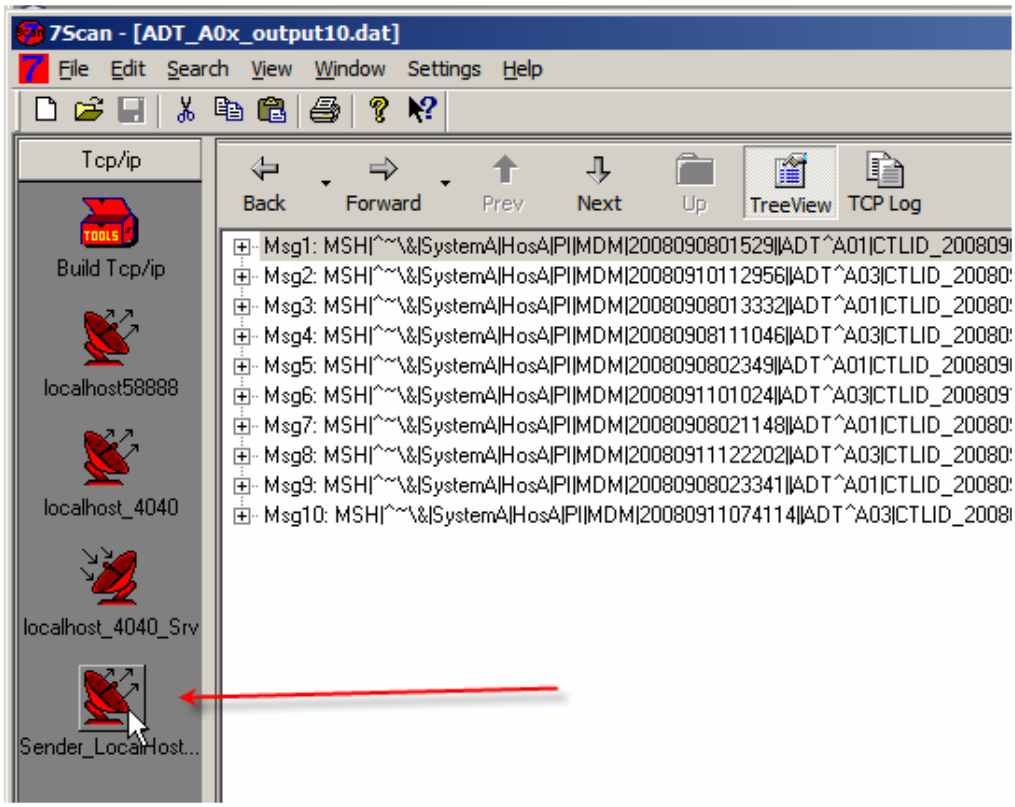


Figure 0-3 Cause the sender to connect to the listener

Once the sender is connected its icon will change colour. Right-click on one of the HL7 messages and choose Send One or Send Multiple from the context menu, to send

a single selected message or multiple messages from the set of messages in the 7Scan window.

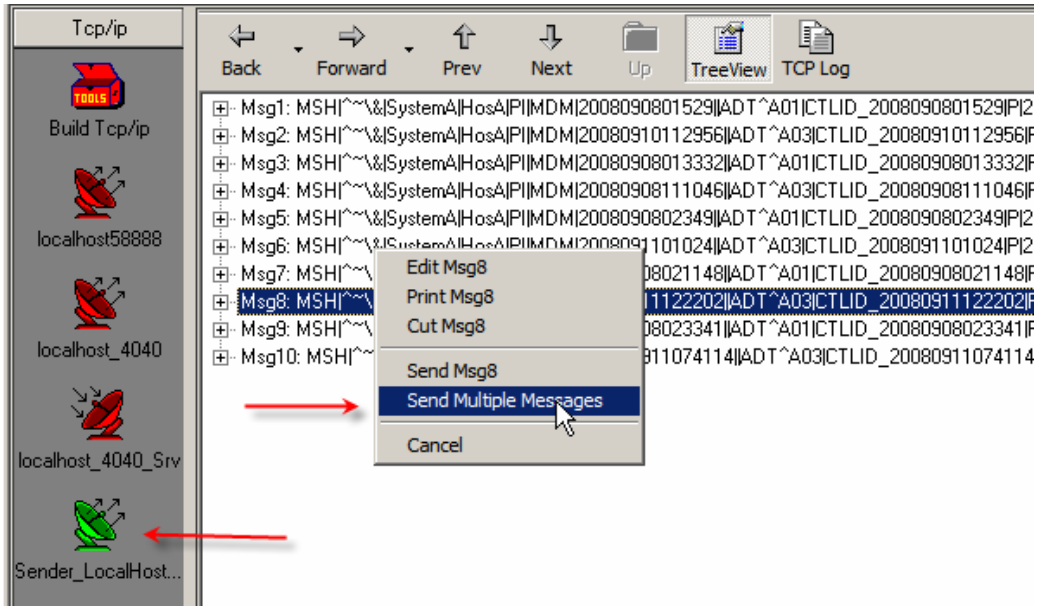


Figure 0-4 Choose to send multiple messages

If you choose to send multiple messages you will have an opportunity to send all messages in the window or selected range of messages.

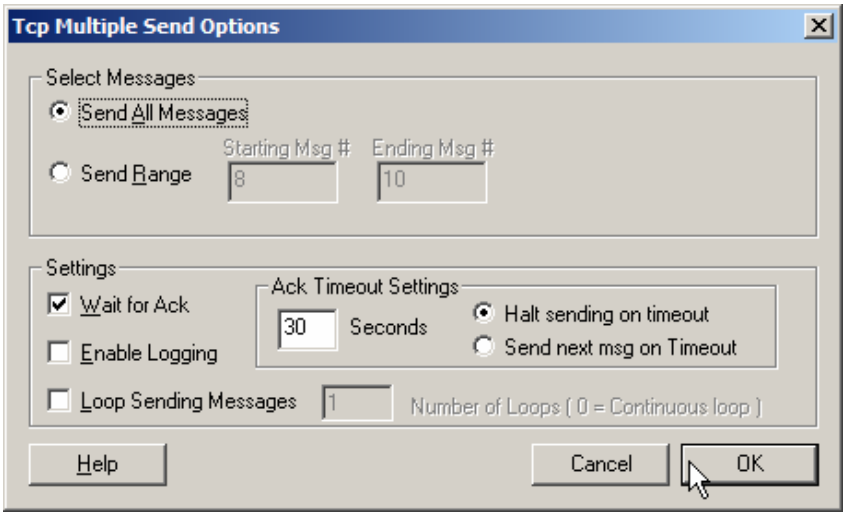


Figure 0-5 Choosing to send all messages

If all is well, and the messages were sent and acknowledged successfully, you will see a confirmation dialog box.

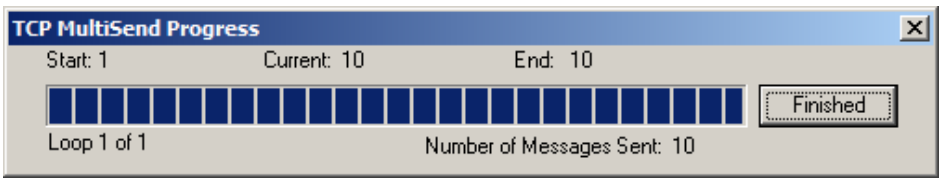
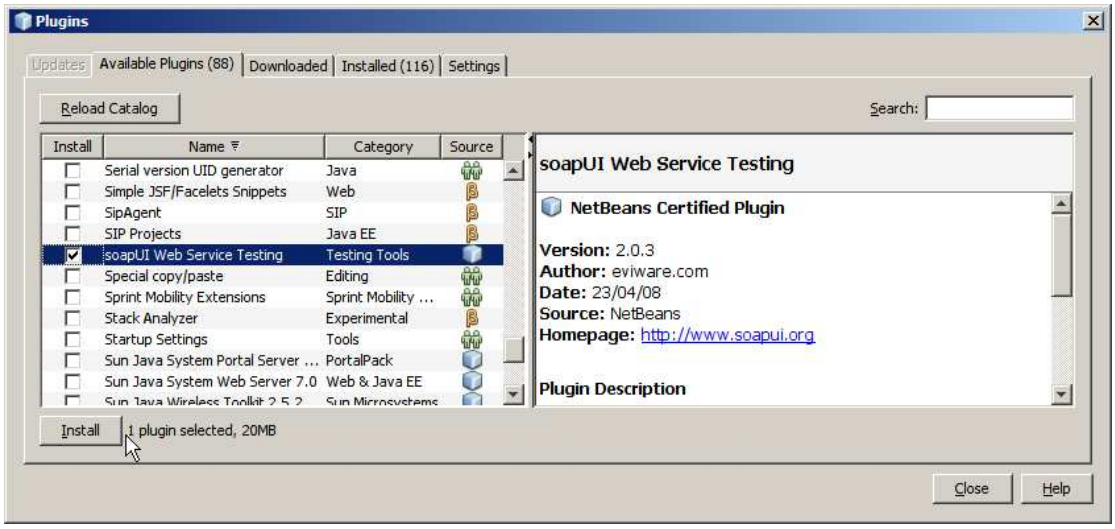


Figure 0-6 Confirmation of successful message send.

The server.log, if your HL7Consumer project is the listener, will show successful receipt and acknowledgment of each message.

SoapUI NetBeans Plugin

Install SoapUI Plugin, 2.0.3, or whatever is current, from the Tools->Plugins->Available Plugins tab in the Plugin Manager. If you don't do this you will not be able to use SopaUI as a web service testing tool.



This may require access to the Internet. Installing from the Internet is preferable as this will give you the most recent SoapUI plugin.

If you don't have access there is a copy of the plugin module, com-eviware-soapui-netbeans-module-2.0.2.nbm, in the prerequisites directory. If you need to install it from the prerequisites directory use the “Downloaded” Tab in the Plugins Manager.

Discussion

During development of this example we will build a Healthcare solution that processes HL7 version 2.3.1 delimited messages. Anyone who works or worked in healthcare in recent years will be able to relate to the subject. For these who have not, the following sections discuss the subject and the Java CAPS solution implemented in this Note.

Business Discussion

Anyone above the arbitrarily chosen age of 10, in the more affluent parts of the World where Hospitals are available for these who need specialist medical care and, alas in many cases, who can afford it, will have an intuitive grasp of the most basic administrative process in a Hospital – the admission and discharge process.

Patients are admitted, stay in a Hospital for a greater or lesser length of time and are discharged.

Whilst in a Hospital patients must be fed, tested, medicated and generally looked after as required by their condition and course of treatment. In order to do this properly different parts of the hospital organization must know that the patient has been admitted, where the patient is (ward, room, bed), what are the basic patient details in terms of gender, age and dietary requirements, where to send medical staff to collect specimen for analysis and where to send results of analysis, which doctors look after the patient, etc., etc..

Some hospitals might have an interest in knowing whether the patient has been already registered in that or another associated hospital, in one or more of the hospital information systems, possibly with a different identifier. Knowing patient identifiers in different systems allows the hospital to get a true picture of the patients it is dealing with, collect information about the same individual from different systems into the so-called Single Patient View and do a number of other things that are aided by the knowledge of all identifiers associated with a single individual. The discipline that deals with that is generally called Master Data Management and, in the case of a hospital, aims at building a Master Patient Index.

Once the patient's stay in the Hospital is over the patient is discharged. At and after discharge other things must happen to complete the process. Any outstanding results must be looked at and, possibly, forwarded to the doctor who will look after the patient from that point. In countries without socialized medicine bills must be prepared and sent for payment. In hospitals which care about quality and statistics, patient stay information must be analysed to determine if quality measures are effective, etc., etc..

Some of the hospitals in some countries are interested in things like patient's length of stay. Knowing how long a patient with a particular condition remained in the hospital can be illuminating from the clinical care and management stand point. For example in certain countries healthcare is subsidized by the tax paying public. The subsidy is sometimes calculated on the basis of the number and the severity of cases treated and does not vary with the length of time patients stayed in the hospital on the assumption

that severity of the condition is the determinant of the length of stay required to treat the condition. Whole science sprang around working out how much it costs to treat patients with certain conditions and the subsidy aims to cover these costs. In this kind of system hospital administrators would be vitally interested in knowing whether patients are kept in the hospital longer then statistically necessary to treat the condition and if so, to investigate the reasons and take corrective action. The assumption is that the shorter the length of stay the more money is left in the kitty after the patient is discharged and, conversely, if the patient stays longer then statistically necessary the hospital is out of pocket on that patient. In other cases a length of stay significantly in excess of statistical average for the patient type may indicate that complications occurred and patient had to be retained longer to deal with their effects. Hopefully rare, but not unheard of , are cases of medical instruments being left in patient's body cavities, post operative complications or infections. These things warrant investigation to determine if surgical procedures or hygiene are lax, or if there is a major outbreak of antibiotic resistant bacterial infection in the hospital.

In short, amongst other things, hospitals need to know that patients are admitted and that patients are discharged. Admission events are of interest because they are used to populate the Master Patient Index. Discharge events are of interest because they are used to calculate the length of stay and compare it to the average length of stay.

Technical Discussion

To capture admission events, discharge events and other kinds of information hospitals deploy Hospital Information Systems. To cut the long story short let's assume that the hospital uses the HL7 version 2.3.1 messages to convey information about various administrative events between its various hospital systems, which might include laboratory systems, radiology and imaging systems, dietary systems and others.

HL7 is a well established healthcare messaging standard. See www.hl7.org for details, though in the tradition of old fashioned standards bodies HL7 organization charges for its standards so you may not be able to get hold of the standards documents without paying for them. It is a pity since requiring people to pay for standards makes them hard to get therefore does not promote standardization.

HL7 version 2.3.1 is a variant of the HL7 delimited message standard. It is important to note, for these people who are under the impression that there was no World before XML, that delimited HL7 messages have been in use for over 20 years in healthcare institutions around the world and are still going strongly. Most hospital systems that use HL7 and that are older then about 5 years, will use the delimited HL7 messages. Newer systems are likely to use the XML version of the HL7 2.x messages or HL7 version 3 which is all XML, however grossly over engineered and disastrous for the environment that is.

It is not the purpose of this Note to teach anyone about healthcare or HL7. For these interested in minimal background information here are a few links. Bear in mind that the HL7 Organization is the first and the last word on the matter.

> See http://www.ringholm.de/docs/00210_en_HL7_ADT_messages.htm for a discussion of ADT (Admission, Discharge, Transfer) events in HL7.

See http://www.otechimg.com/pdf/B-112_ch1.pdf for a brief discussion of HL7 v2 and a sample message.
Here http://www.srdc.metu.edu.tr/webpage/seminars/Healthcare/HL7.ppt#282,33,Patient%20Care%20Scenarios is another overview.
And here https://knol.google.com/k/jaffa-brown/hl7-adt-message-overview/3g15mgyrpfb21/3# is another.

Here, http://www.hosinc.com/Products/Interfaces/interface_documentation.htm, is a discussion of a "HL7 interface" with several examples of messages.

For our purposes we will be dealing with a HL7 version 2.3.1 ADT A01 (Admission) and ADT A03 (Discharge) messages.

A sample ADT A01 message might look like this:

```
MSH|^~\&|SystemA|HosA|PI|MDM|20080908111907||ADT^A01|CTLID_2008090811
1907|P|2.3.1|||AL|NE\r
EVN|A01|20080908111907|||JavaCAPS6^^^^^^^USERS\r
PID|1||A000010^^^HosA^MR^HosA||Kessel^Abigail^^^^^||19460101123045|M|
||7 South 3rd Circle^^Downham Market^England -
Norfolk^20^UK^|||||||A20080908111907\r
PV1|1|I|^^^|I|||GOO^Goodlace^Andrew^^^^^^^^^^MAIN|||EMR|||||||||V2008
0908111907^^^^VISIT|||||||||||||||||||||||||20080908111907\r
```

A sample ADT A03 message might look like this:

```
MSH|^~\&|SystemA|HosA|PI|MDM|20080911033024||ADT^A03|CTLID_2008091103
3024|P|2.3.1|||AL|NE\r
EVN|A03|20080911033024|||JavaCAPS6^^^^^^^USERS\r
PID|1||A000010^^^HosA^MR^HosA||Kessel^Abigail^^^^^||19460101123045|M|
||7 South 3rd Circle^^Downham Market^England -
Norfolk^30828^UK^|||||||A20080908111907\r
PV1|1|I|^^^|I|||GOO^Goodlace^Andrew^^^^^^^^^^MAIN|||EMR|||||||||V2008
0908111907^^^^VISIT|||||||||||||||DISH DISP|disch
loc||||||20080908111907|20080913033024\r
```

Each line ending with \r is called a Segment. Each segment starts with Segment ID. The Message Header (MSH) segment is the first segment in a message. Each message has one. It identifies, amongst other things, the originating system and the intended destination system as well as the Trigger Event (ADT) and Message Type (A01, A03). The Patient Identification (PID) Segment carries patient demographic information. The Patient Visit (PV1) Segment carries patient episode of care information – for us of interest are the admission date and the discharge date.

MSH Segment Attributes for HL7 v 2.3.1 are listed in the table below.

SEQ	LEN	DT	OPT	RP/#	TBL#	ITEM #	ELEMENT NAME
1	1	ST	R			00001	Field Separator
2	4	ST	R			00002	Encoding Characters
3	180	EI	O			00003	Sending Application
4	180	EI	O			00004	Sending Facility
5	180	EI	O			00005	Receiving Application
6	180	EI	O			00006	Receiving Facility
7	26	TS	O			00007	Date/Time Of Message
8	40	ST	O			00008	Security
9	7	CM	R		0076 0003	00009	Message Type
10	20	ST	R			00010	Message Control ID
11	3	PT	R			00011	Processing ID
12	608	VID	R		0104	00012	Version ID
13	15	NM	O			00013	Sequence Number
14	180	ST	O			00014	Continuation Pointer
15	2	ID	O		0155	00015	Accept Acknowledgment Type
16	2	ID	O		0155	00016	Application Acknowledgment Type
17	2	ID	O			00017	Country Code
18	106	ID	O	Y/3	0211	00692	Character Set
19	60	CE	O			00693	Principal Language Of Message
20	20	ID	O			01317	Alternate Character Set Handling Scheme

Figure 0-1 MSH Segment Definition

EVN (Event) Segment Attributes for HL7 v2.3.1 are listed in the table below.

SEQ	LEN	DT	OPT	RP/#	TBL#	ITEM#	ELEMENT NAME
1	3	ID	B		0003	00099	Event Type Code
2	26	TS	R			00100	Recorded Date/Time
3	26	TS	O			00101	Date/Time Planned Event
4	3	IS	O		0062	00102	Event Reason Code
5	60	XCN	O	Y	0188	00103	Operator ID
6	26	TS	O			01278	Event Occurred

Figure 0-2 EVN Segment Definiton

PID Segment Attributes for HL7 v2.3.1 are listed in the table below.

SEQ	LEN	DT	OPT	RP/#	TBL#	ITEM#	ELEMENT NAME
1	4	SI	O			00104	Set ID - PID
2	20	CX	BO			00105	Patient ID
3	20	CX	R	Y		00106	Patient Identifier List
4	20	CX	BO	Y		00107	Alternate Patient ID - PID
5	48	XPN	R	Y		00108	Patient Name
6	48	XPN	O	Y		00109	Mother's Maiden Name
7	26	TS	O			00110	Date/Time of Birth
8	1	IS	O		0001	00111	Sex
9	48	XPN	O	Y		00112	Patient Alias
10	80/1	CE/IS	O	Y	0005	00113	Race
11	106	XAD	O	Y		00114	Patient Address
12	4	IS	B		0289	00115	County Code
13	40	XTN	O	Y		00116	Phone Number - Home
14	40	XTN	O	Y		00117	Phone Number - Business
15	60	CE	O		0296	00118	Primary Language
16	80/1	CE/IS	O		0002	00119	Marital Status
17	80/3	CE/IS	O		0006	00120	Religion
18	20	CX	O			00121	Patient Account Number
19	16	ST	BO			00122	SSN Number - Patient
20	25	DLN	O			00123	Driver's License Number - Patient
21	20	CX	O	Y		00124	Mother's Identifier
22	80/3	CE/IS	O	Y	0189	00125	Ethnic Group
23	60	ST	O			00126	Birth Place
24	1	ID	O		0136	00127	Multiple Birth Indicator
25	2	NM	O			00128	Birth Order
26	80/4	CE/IS	O	Y	0171	00129	Citizenship
27	60	CE	O		0172	00130	Veterans Military Status
28	80	CE	O		0212	00739	Nationality
29	26	TS	O			00740	Patient Death Date and Time
30	1	ID	O		0136	00741	Patient Death Indicator

Figure 0-3 PID Segment Definition

PV1 Segment Attributes HL7 v2.3.1 are listed in the table below.

SEQ	LEN	DT	OPT	RP/#	TBL#	ITEM#	ELEMENT NAME
1	4	SI	O			00131	Set ID - PV1
2	1	IS	R		0004	00132	Patient Class
3	80	PL	O			00133	Assigned Patient Location
4	2	IS	O		0007	00134	Admission Type
5	20	CX	O			00135	Preadmit Number
6	80	PL	O			00136	Prior Patient Location
7	60	XCN	O	Y	0010	00137	Attending Doctor
8	60	XCN	O	Y	0010	00138	Referring Doctor
9	60	XCN	O	Y	0010	00139	Consulting Doctor
10	3	IS	O		0069	00140	Hospital Service
11	80	PL	O			00141	Temporary Location
12	2	IS	O		0087	00142	Preadmit Test Indicator
13	2	IS	O		0092	00143	Re-admission Indicator
14	3	IS	O		0023	00144	Admit Source
15	2	IS	O	Y	0009	00145	Ambulatory Status
16	2	IS	O		0099	00146	VIP Indicator
17	60	XCN	O	Y	0010	00147	Admitting Doctor
18	2	IS	O		0018	00148	Patient Type
19	20	CX	O			00149	Visit Number
20	50	FC	O	Y	0064	00150	Financial Class
21	2	IS	O		0032	00151	Charge Price Indicator
22	2	IS	O		0045	00152	Courtesy Code
23	2	IS	O		0046	00153	Credit Rating
24	2	IS	O	Y	0044	00154	Contract Code
25	8	DT	O	Y		00155	Contract Effective Date
26	12	NM	O	Y		00156	Contract Amount
27	3	NM	O	Y		00157	Contract Period
28	2	IS	O		0073	00158	Interest Code
29	1	IS	O		0110	00159	Transfer to Bad Debt Code
30	8	DT	O			00160	Transfer to Bad Debt Date
31	10	IS	O		0021	00161	Bad Debt Agency Code
32	12	NM	O			00162	Bad Debt Transfer Amount
33	12	NM	O			00163	Bad Debt Recovery Amount
34	1	IS	O		0111	00164	Delete Account Indicator
35	8	DT	O			00165	Delete Account Date
36	3	IS	O		0112	00166	Discharge Disposition
37	25	CM	O		0113	00167	Discharged to Location
38	802	CE~~IS~~	O		0114	00168	Diet Type
39	2	IS	O		0115	00169	Servicing Facility
40	1	IS	B		0116	00170	Bed Status
41	2	IS	O		0117	00171	Account Status
42	80	PL	O			00172	Pending Location
43	80	PL	O			00173	Prior Temporary Location
44	26	TS	O			00174	Admit Date/Time
45	26	TS	O			00175	Discharge Date/Time
46	12	NM	O			00176	Current Patient Balance
47	12	NM	O			00177	Total Charges
48	12	NM	O			00178	Total Adjustments
49	12	NM	O			00179	Total Payments

Figure 0-4 PID Segment Definition

To find out what the different columns and entries mean you need to refer to the HL7 version 2.3.1 Standard itself. There is no significance to the colour in the tables. I happened to have a draft copy of Chapter 3 of the HL7 Version 2.3.1 standard, from which the tables came, in which Change Tracking was enabled.

After going on for a while about HL7 delimited message standard we are coming to XML. The HL7 Processor solution we will build will use BPEL 2.0 to implement processing logic. The BPEL Service Engine will need to be given a XML message to deal with and will produce a XML message. HL7 delimited message will need to be converted to their XML equivalents. The HL7 Encoder feature will be used to do the conversion as messages are received by the HL7 Binding Component. Conversion requires the presence of HL7 version 2.3.1 XML Schema Documents. See

Prerequisites section, above, for instruction on where to find these Schema documents.

Solution

As mentioned in the Introduction, the HL7 Processor will:

- receive HL7 v2.x delimited messages using the HL7 BC
- convert HL7 v2.x messages to their equivalent XML format
- split message stream into ADT A01s and ADT A03s
- convert A01s to an abbreviated Custom Patient XML format
- convert A03s to an abbreviated Custom Discharge XML format
- send Custom Patients to a JMS Queue for processing by a MDM solution
- send Custom Discharges to a JMS Queue for processing by an IEM solution

Before we can receive HL7 messages using the HL7 BC we must construct the HL7 sender. This sender will read a file of sample HL7 messages using the File BC and will send them using the HL7 BC. The overall solution architecture is shown below.

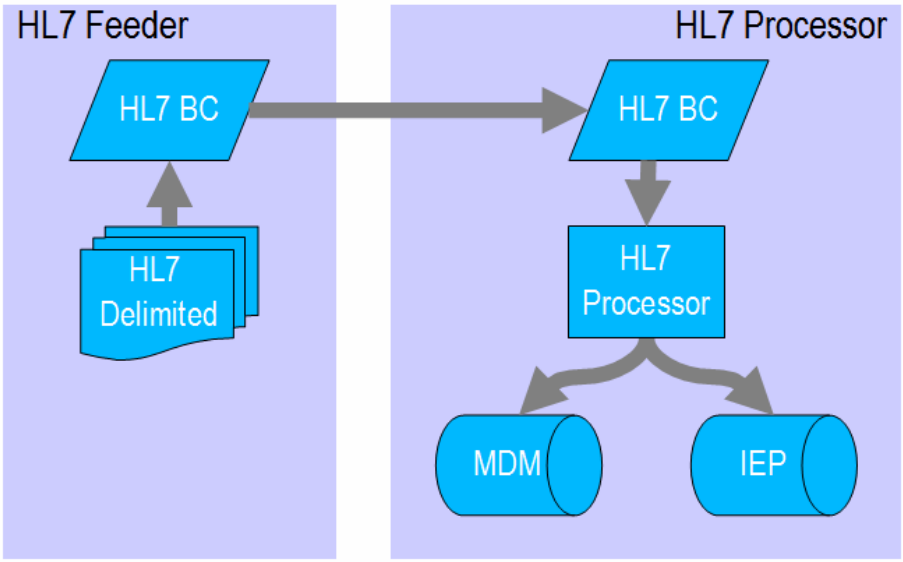


Figure 0-5 Solution Architecture Schematic

To enable us to test the HL7 Feeder we will create an abbreviated HL7 Consumer, which will also be used to explore conversion of HL7 delimited messages into HL7 XML messages, so we can see where and how this is done. The HL7 Consumer will be discarded later as it will be replaced by the HL7 Processor proper.

At the end of the process we will have implemented:

HL7Consumer	Throwaway solution receiving HL7 delimited messages using the HL7 BC and writing their XML equivalents to files in a file system using the File BC
HL7Feeder	The solution reading HL7 delimited messages from a file in the file system using the File BC and sending these messages to a HL7 Listener using the HL7 BC.
WSSDateDiff	EJB-based web service implementing “difference in days between two arbitrarily formatted dates” using the

	“Implementation First” method.
WSSCovnertDate	EJB-based web service implementing conversion of a date/time string between two arbitrary formats using the “Interface First” method.
HL7Processor	The solution receiving HL7 delimited messages using the HL7 BC, processing ADT A01 and ADT A03 separately, converting HL7 date/time formatted dates to ISO 8601 date/time formatted dates, calculating the difference in days between discharge dates and admission dates, constructing custom XML messages for forwarding to the MDM and IEP solutions developed and discussed elsewhere and sending these messages on using the JMS BC.

Implementation

Project Group

All projects developed here will be collected into a Project Group. Let's create the Project Group, HL7ProcessorGroup, using NetBeans Project Group facilities.

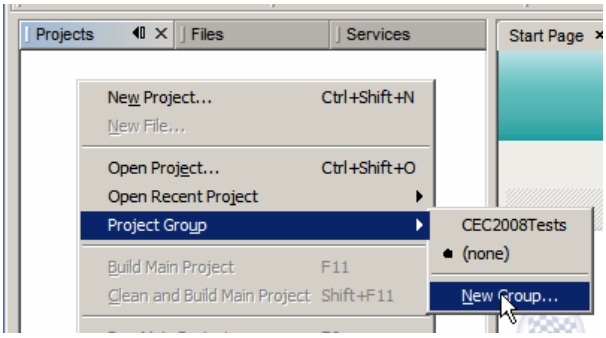


Figure 0-1 Triggering the New Project Group functionality

In the New Group dialogue box select the Folder of Projects option, navigate through the file system to the folder location where projects in this group will be collected, creating subfolders as necessary, and click Open.

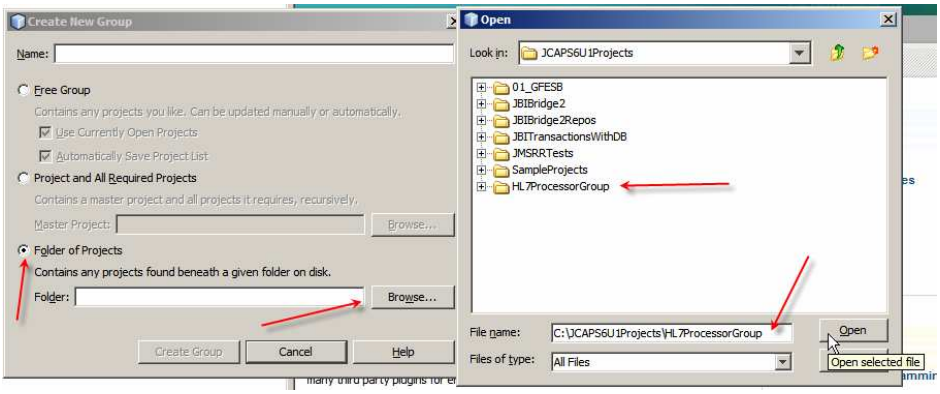


Figure 0-2 Choosing a folder to contain projects in the Project Group

Provide the name for the Project Group and create it.

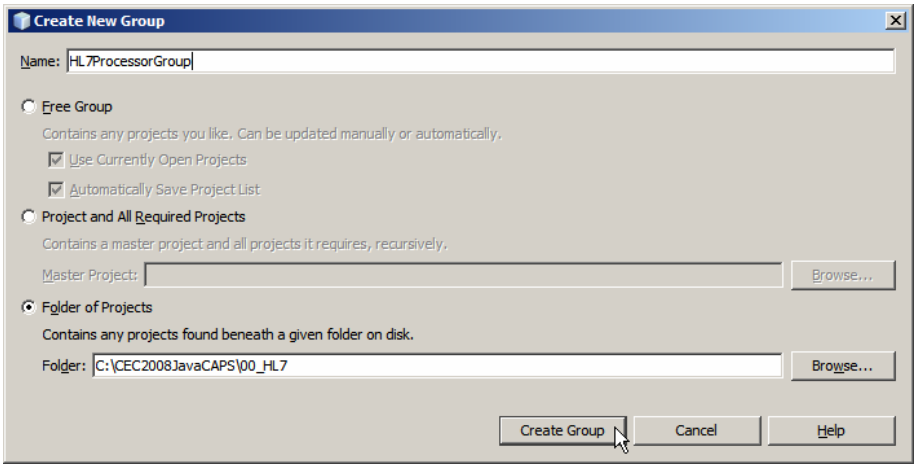


Figure 0-3 Completing Project Group creation

Make sure to choose the project group's folder for new projects, when creating projects in subsequent sections.

HL7 Consumer and XML Converter

To test the HL7 Feeder, which we will develop in the next section, we need a HL7 Consumer, a listener implementation which will listen for TCP connections on a specific port, will accept connections and receive all messages that are sent to it, sending a HL7 ACK back for each. Each message, assumed to be a HL7 message, will be converted from the HL7 Delimited to its HL7 XML format and will be written to a file in the file system.

The purpose of this consumer, which will ultimately be discarded, is to test the HL7 Feeder and to explore HL7 delimited to HL7 XML message conversion.

All conversion logic will be handled by the HL7 Encoder directly in the HL7 BC so no external logic, like Java or BPEL, will be required. Both Binding Components will be added directly to the Composite Application Service Assembly and wired together.

In the HL7ProcessorGroup project group create a new Composite Application project, named HL7Consumer_CA.

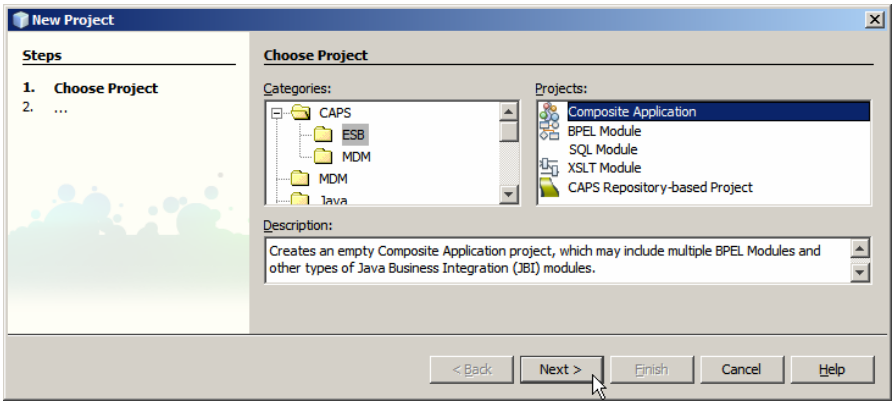


Figure 0-4 Begin creation of a new Composite Application

Make sure to provide the project location that is consistent with the project group's location and to name the project as required.

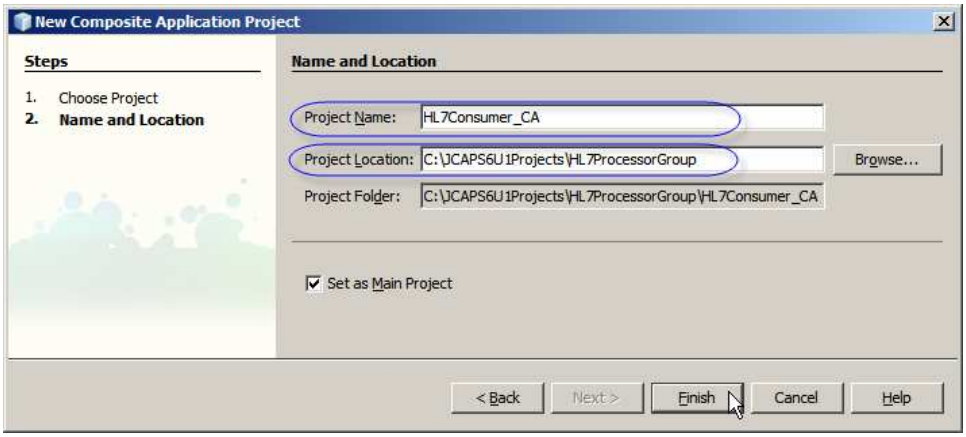


Figure 0-5 Naming the project and selecting project location

Expand the Project Files folder and create a subfolder called HL7v2 using the New->Other->Other->Folder options.

Figure 0-6 Create a folder inside the Process Files folder

Name the new folder HL7v2.

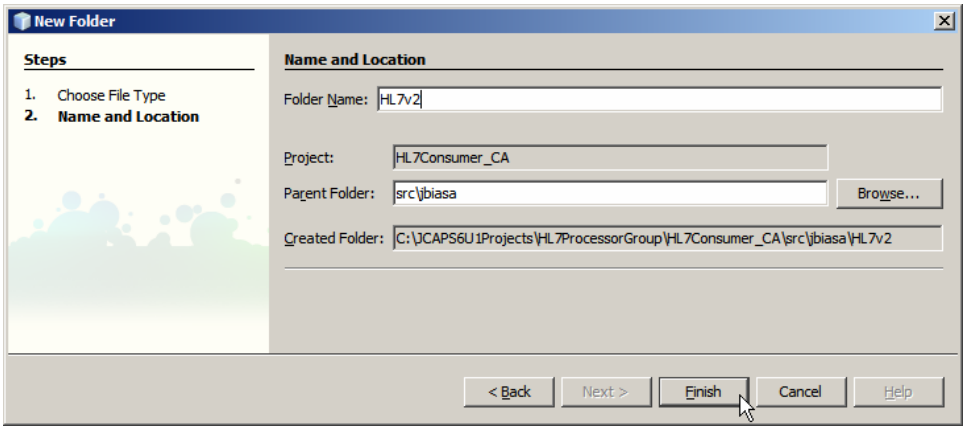


Figure 0-7 Name the new folder

Expand the Process Files folder, right-click on the HL7v2 folder and choose New->Other->XML->External XML Schema Documents.

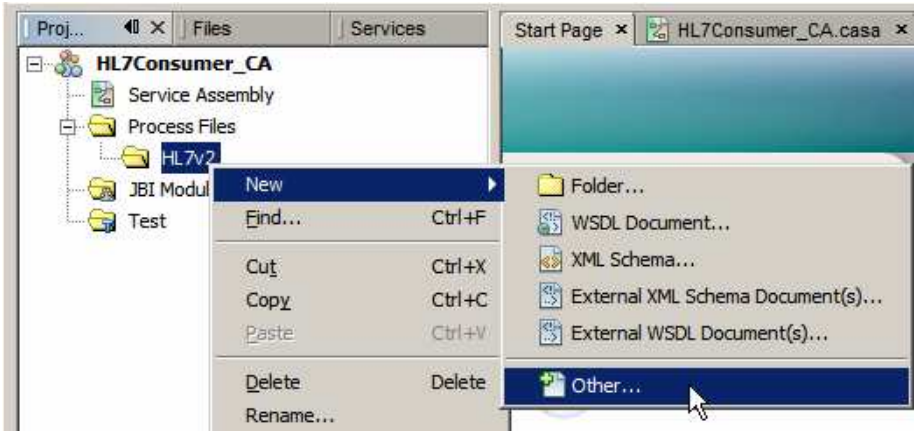


Figure 0-8 Choose New->Other…

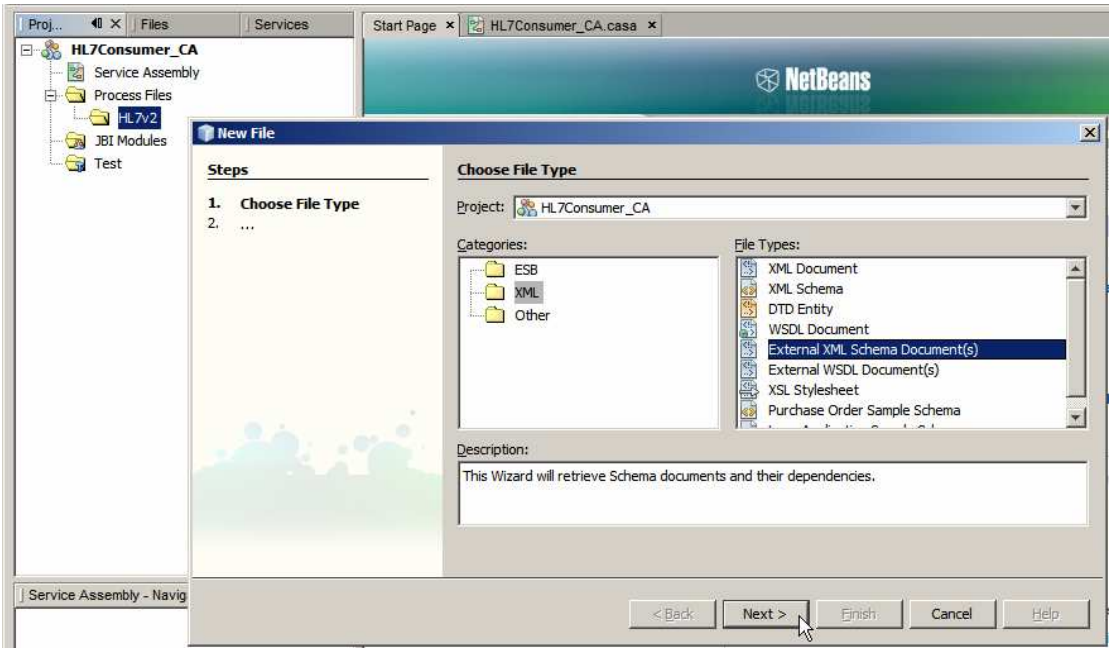


Figure 0-9 Choose XML->External XML Schema Documents

Navigate to the hl7v2/2.3.1 folder containing the HL7 v2.3.1 XML Schema Documents which were extracted from the ZIP archive downloaded earlier, and select the following XSD documents:

- ADT_A01.xsd
- ADT_A03.xsd
- batch.xsd
- datatypes.xsd
- fields.xsd
- messages.xsd
- segments.xsd

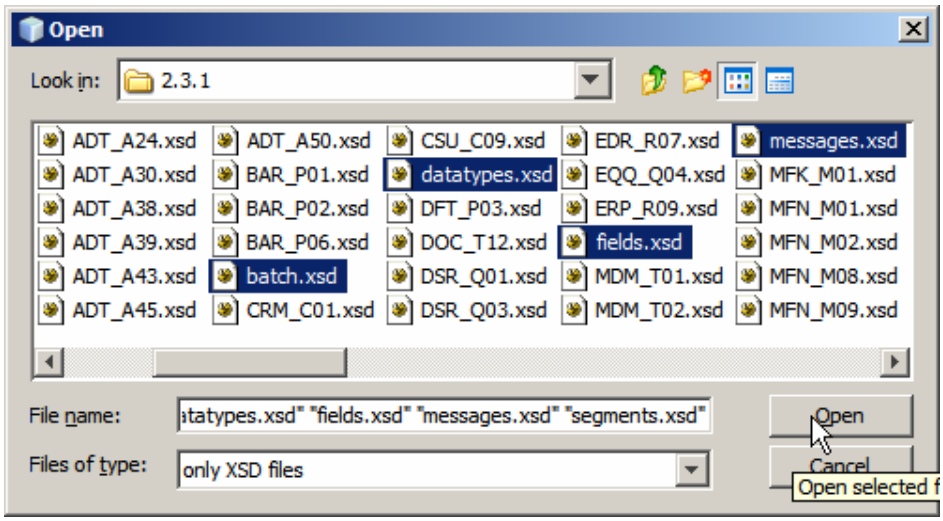


Figure 0-10 Select specific HL7 v2.3.1 XSD files

Click Finish to import all files.

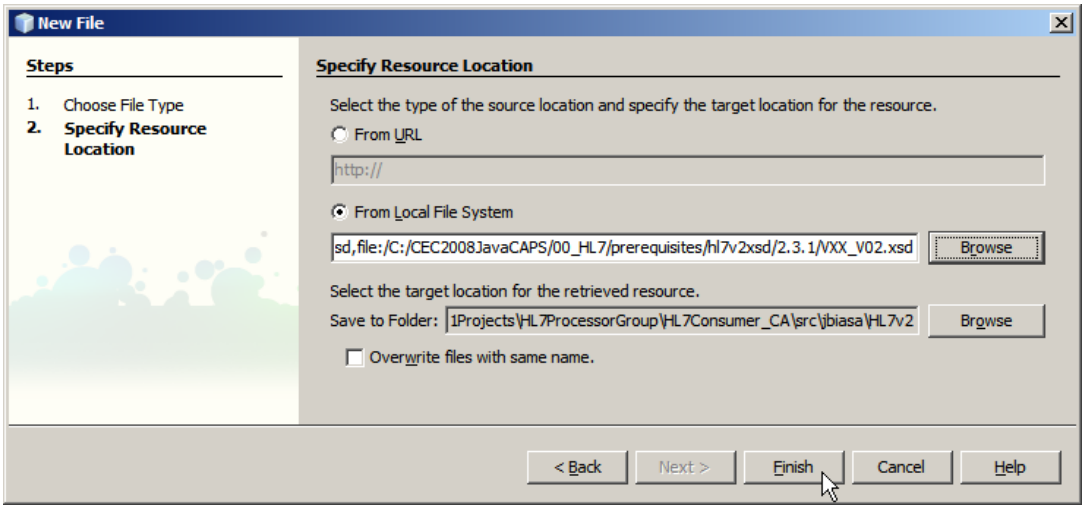


Figure 0-11 Start the import process

Once the import is finished a few files will appear in the HL7v2 folder.

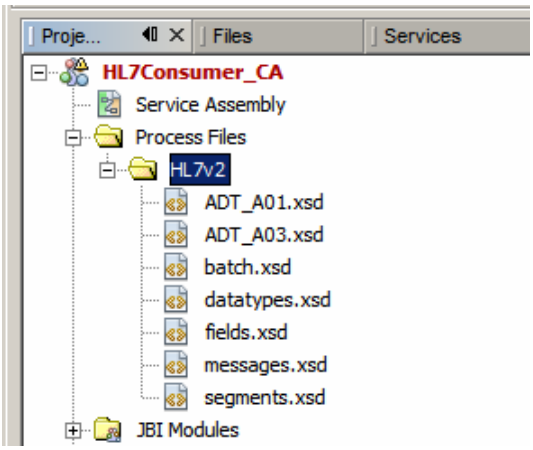


Figure 0-12 HL7 v2.3.1 XSDs

Note that we only imported the ADT A01 (Admission) and ADT A03 (Discharge) XSD files and subsidiary files referenced from them. This is because we will only deal with ADT A01 and ADT A03 messages in this project.

Expand to the Project Files folder in the folder tree under the HL7Consumer_CA project and choose New->WSDL Document … context menu option.

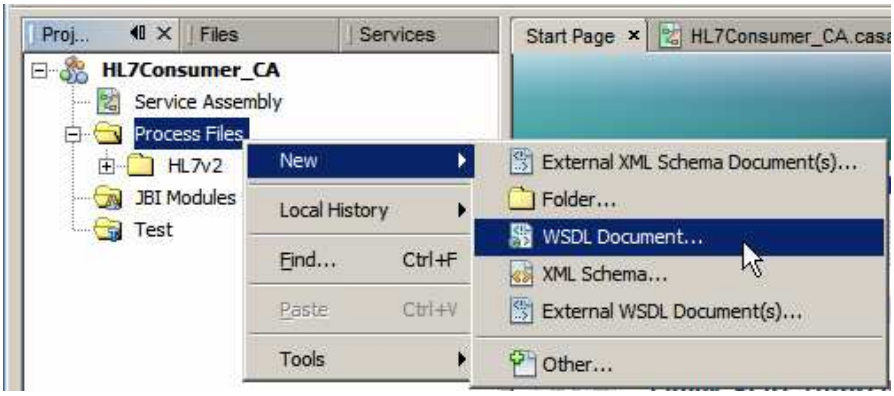


Figure 0-13 Triggering New WSDL Document wizard functionality

The WSDL we will be creating is a placeholder for the HL7 Binding Component properties, as will be seen shortly. A WSDL Wizard allows us to choose the Binding Component which to use, then to configure properties that are specific to this Binding Component.

We will listen for incoming HL7 Version 2 messages. As the HL7 BC receives the messages it will 'decode' them from delimited to XML format and will provide each message to the Normalized Message Router for routing to whatever component is interested in processing them, as defined through the Composite Application Service Assembly, more of which later.

Configure the WSDL / HL7 Binding Component as follows:

- File Name: HL7Consumer_CA_A01_A03Delim_HL7In
- WSDL Type: Concrete WSDL Document
- Binding: HL7
- Type: HL7 Version 2 – Inbound Request

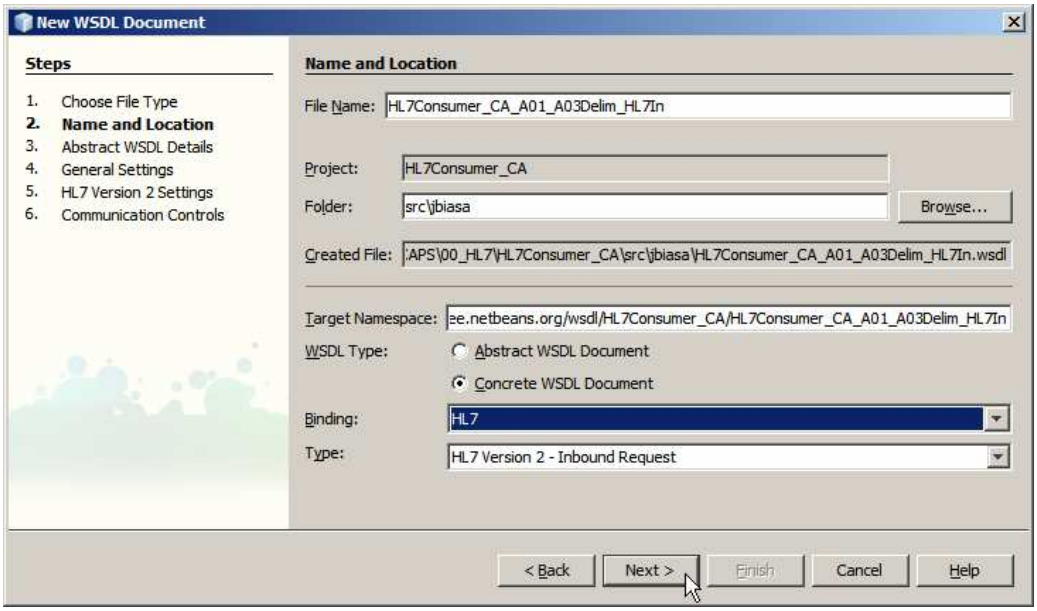


Figure 0-14 Initial HL7 Inbound configuration dialogue

Click Next to get to the next Wizard dialogue box.

This is where things get more interesting. The HL7 BC must be told what kind of messages it is dealing with. The only way at present to tell it is to choose the XML Schema Document that represents the HL7 trigger event and message type. This is why we imported all these XSDs earlier.

Click the Browse button along the Request Message data entry field, navigate to the ADT_A01.xsd, choose the ADT_A01 Element and click OK.

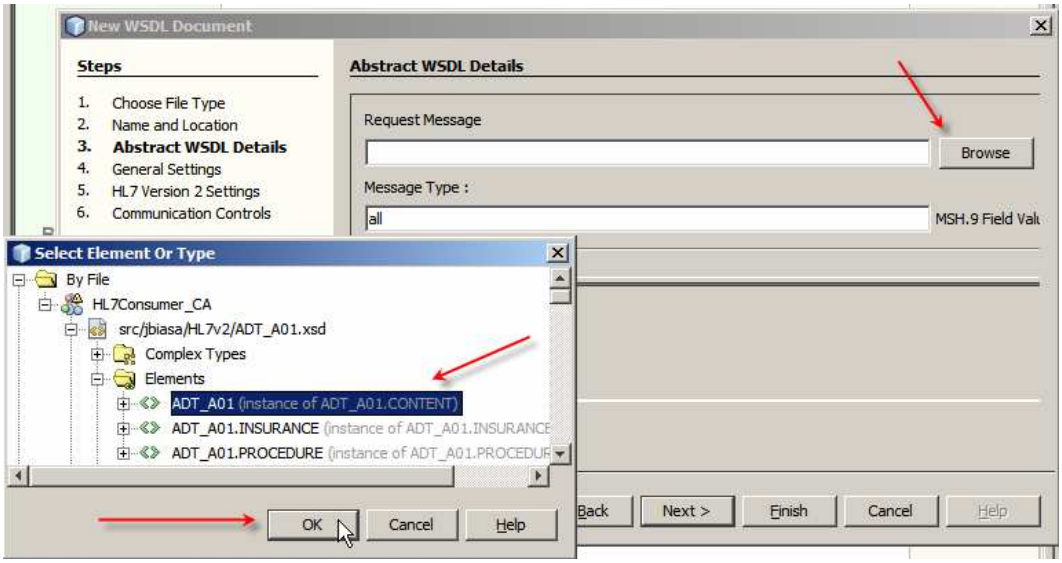


Figure 0-15 Choose ADT A01 for the request message

Manually enter "**ADT^A01**" for the message type and click Next.

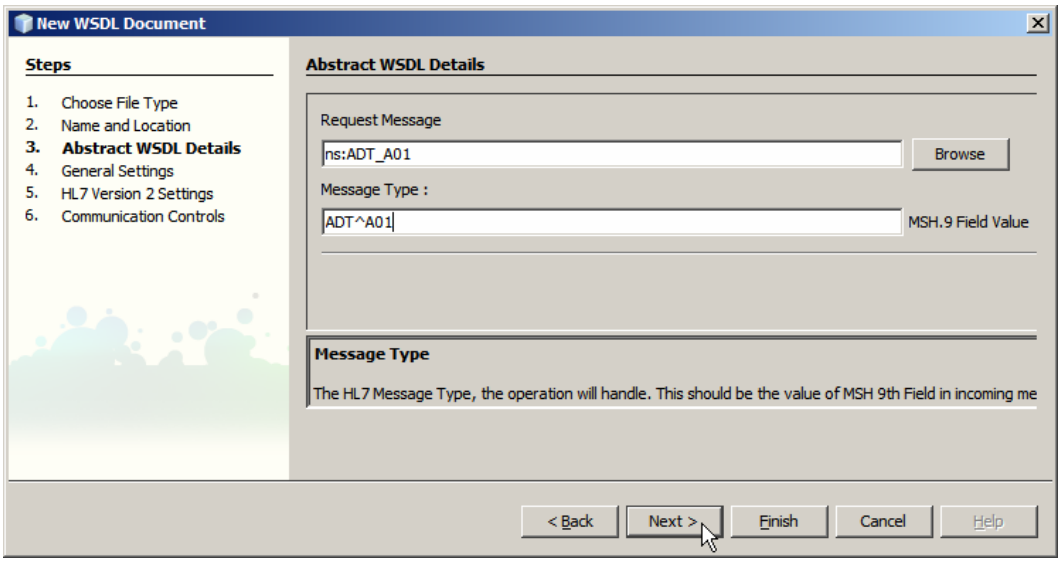


Figure 0-16 Enter message type

Modify the listening port if you need. By default it is 4040. Notice, too, the Start Block, End Block and End Data characters. Recall the brief discussion in the 3rd Party Optional Tools section. We are configuring a HL7 Listener. The HL7 client/sender will have to be configured so that its star block, end block and end data characters are identical to what is configured here.

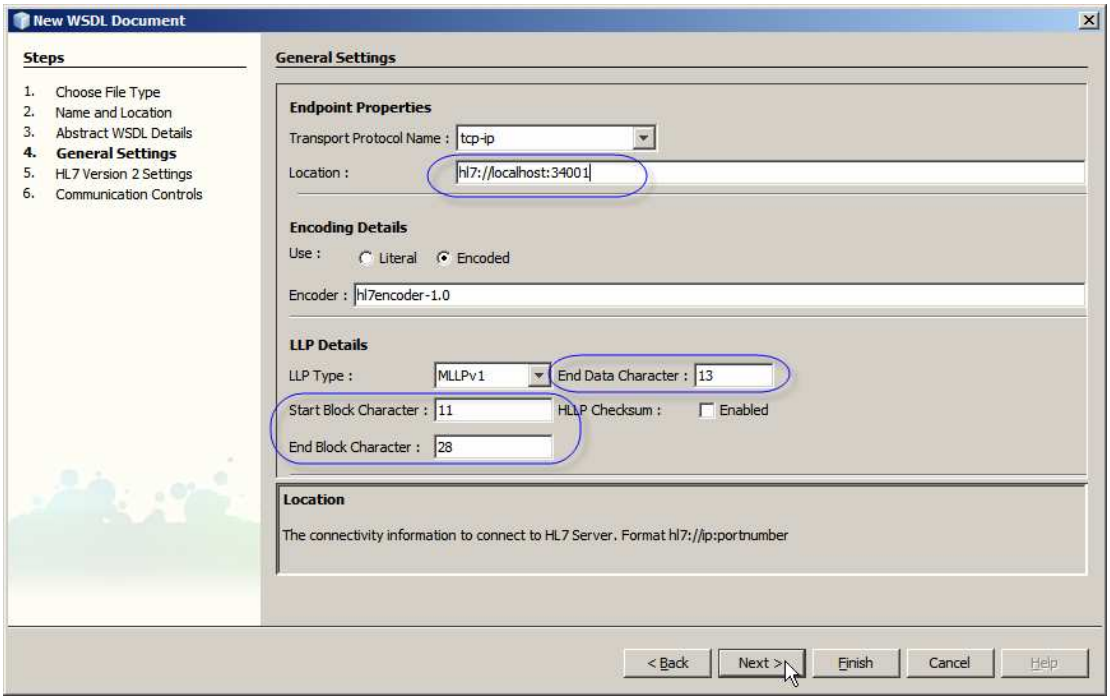


Figure 0-17 Changing the listening port and confirming MLLP delimiter characters

Click Next, accept all defaults in the HL7 Version 2 Settings and click Next again.

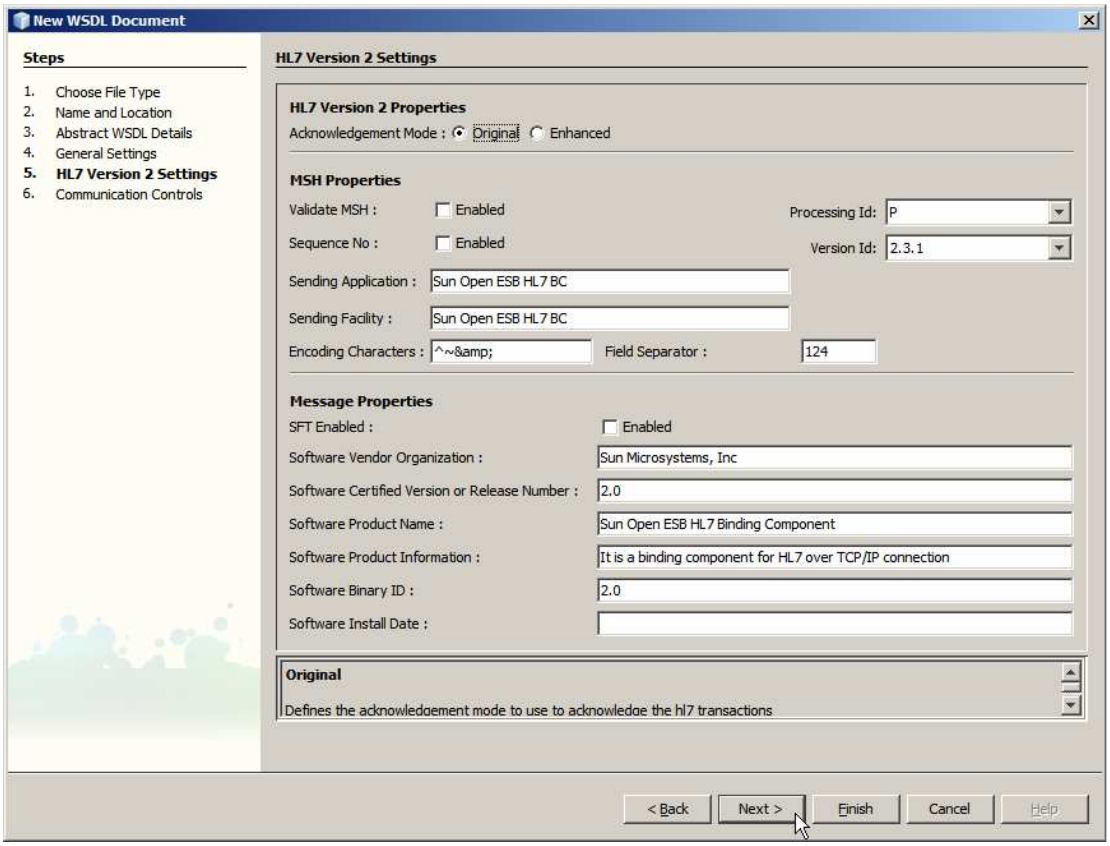


Figure 0-18 Accept default for the HL7 Version 2 Settings

Accept defaults at the Communications Controls and click Finish.

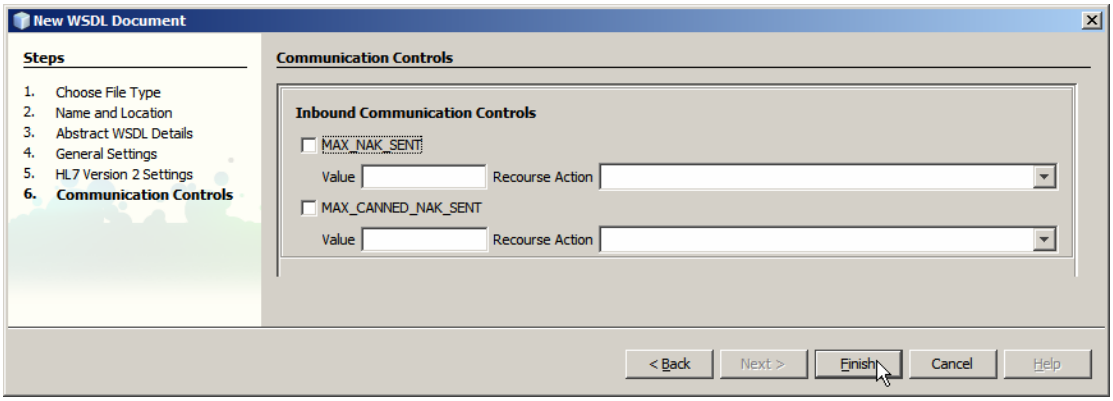


Figure 0-19 Complete the Wizard.

Outcome of this process is a WSDL which defines one operation on one type of HL7 message. It looks similar to what is shown below.

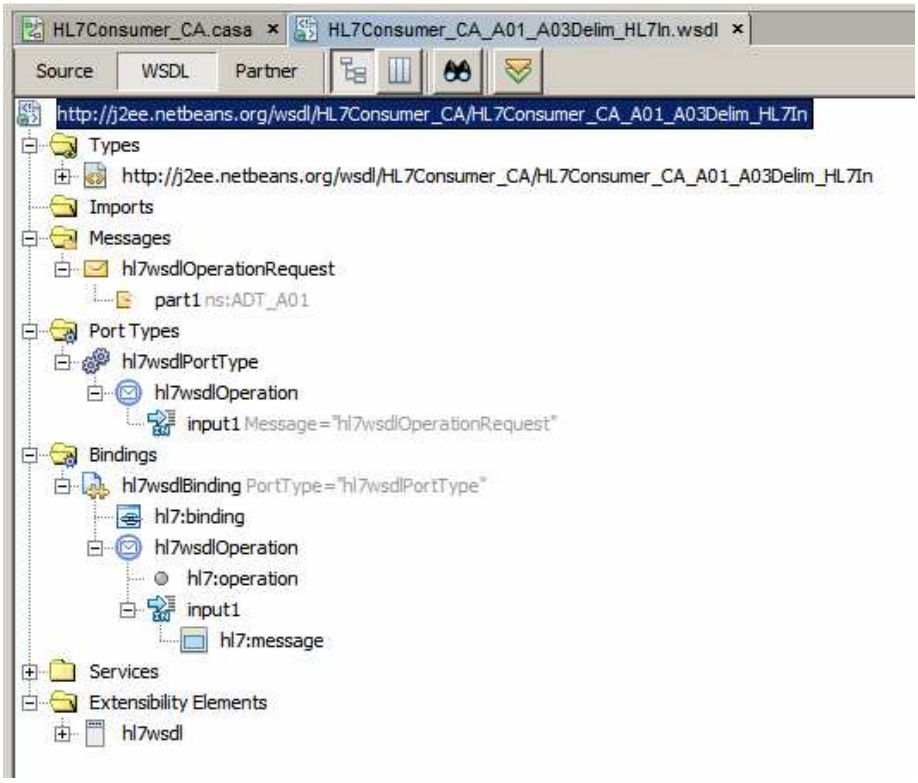


Figure 0-20 WSDL generated by the HL7 Wizard

In order for us to be able to process two different types of HL7 messages, the ADT A01, already configured, and the ADT A03, we need to add a new Message, WSDL Operation and Binding Operation. Alas, this has to be dome through the WSDL editor since at this time there is no Wizard that will assist.

First, let's add a message. Right-click on the Messages node and choose Add Message.

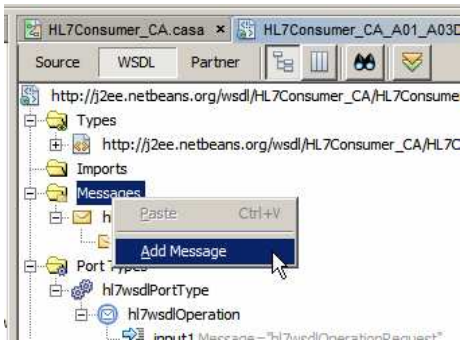


Figure 0-21 Trigger Add Message functionality

Rename the new node to msgADT_A03 by right-clicking it and choosing Refactor->Rename.

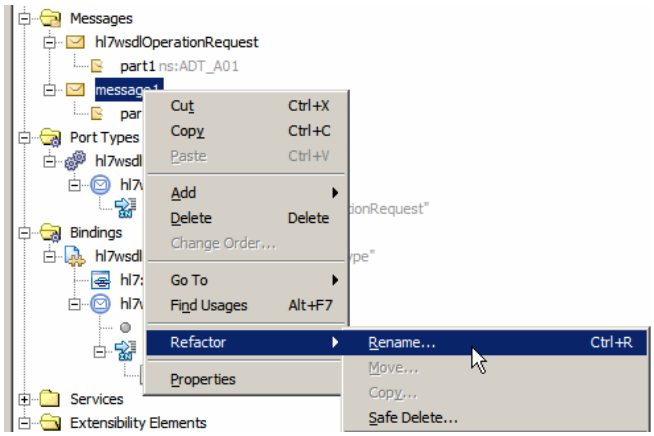


Figure 0-22 Rename the message

Right-click on the part1 node under the msgADT_A03 message and choose Properties.

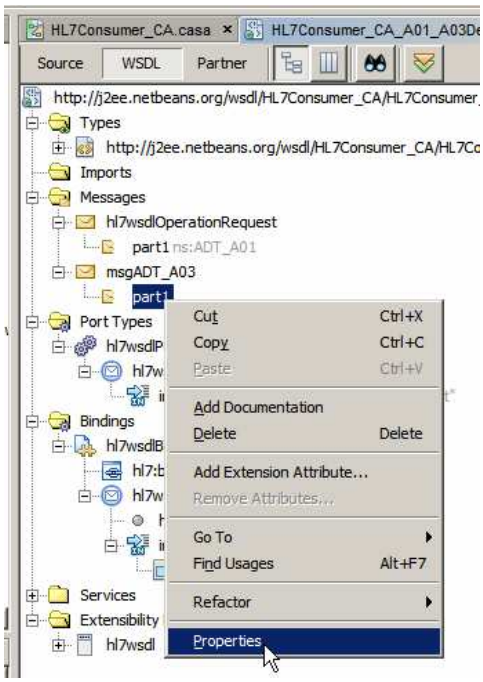


Figure 0-23 Choose properties to configure part's data type

Click the small button with ellipsis in it, alongside the Element or Type data entry box, locate the ADT A03 XSD, select the ADT_A03 Element, click OK and Close.

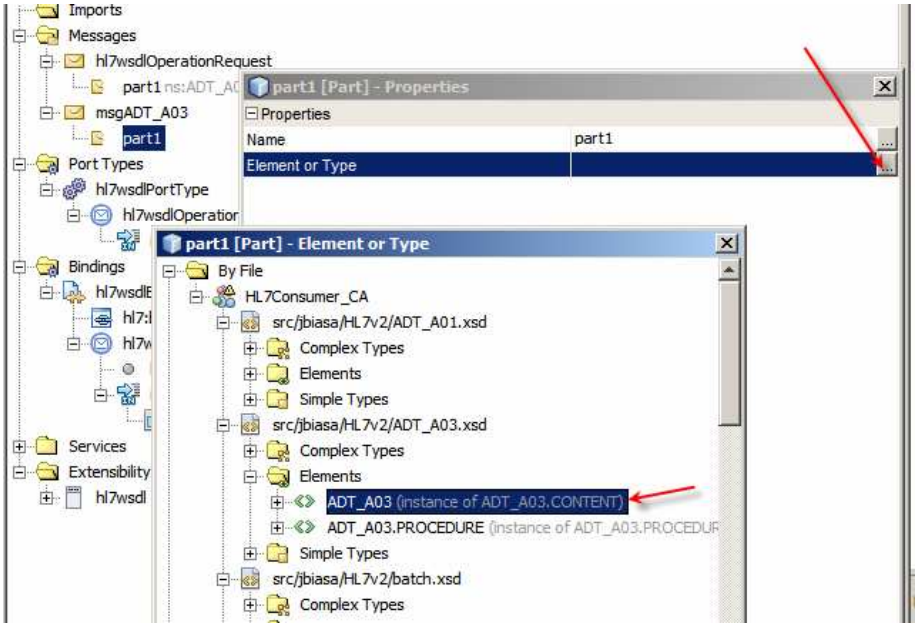


Figure 0-24 Configure data type for the message part

Right-click the hl7wsdlPortType node, choose Add, choose Operation …

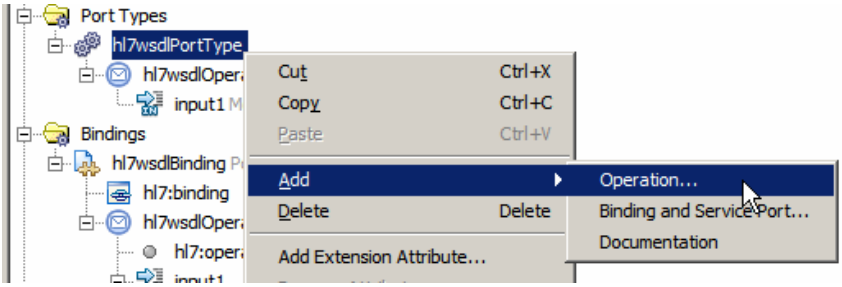


Figure 0-25 Choose to add new operation to the existing port

Name the operation opADT_A03, choose One-Way Operation for Operation Type and select msgADT_A03 message as Input. Click OK to complete the wizard.

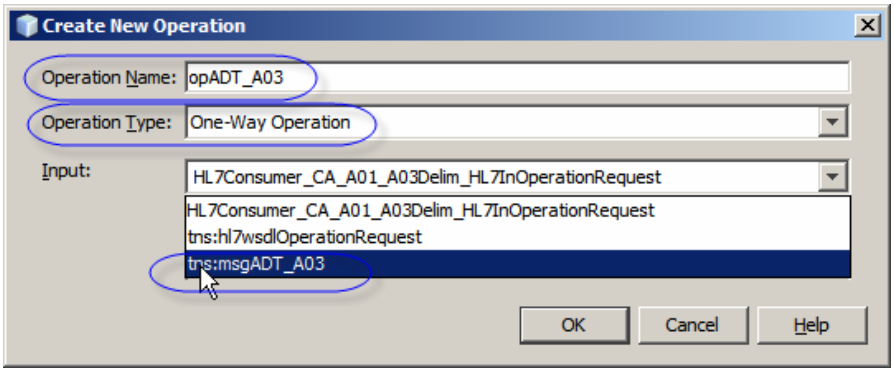


Figure 0-26 Name and configure new operation

Right-click the hl7wsdlBinding node, choose Add, choose Binding Operation.

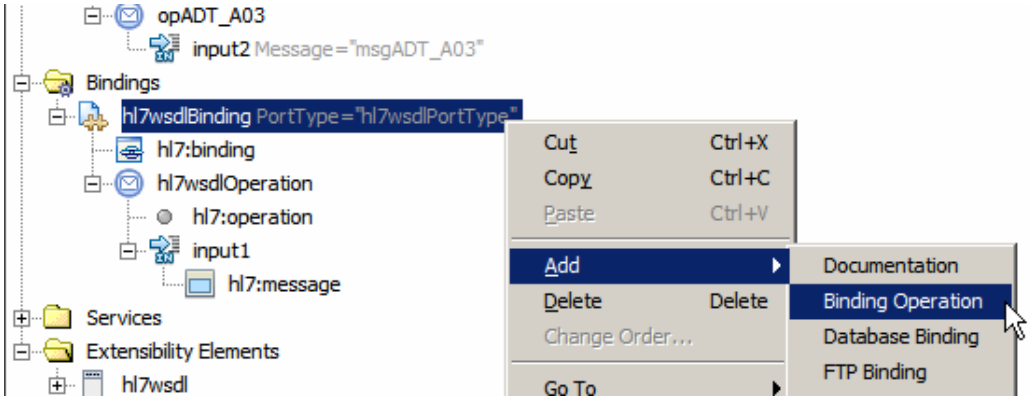


Figure 0-27 Add new Binding Operation

New binding operation with empty input2 node is added.

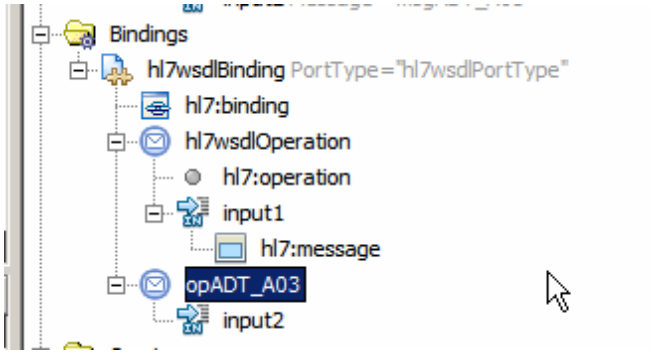


Figure 0-28 New, unconfigured Binding Operation

Right-click the input2 node, choose Add, choose HL7 Message.

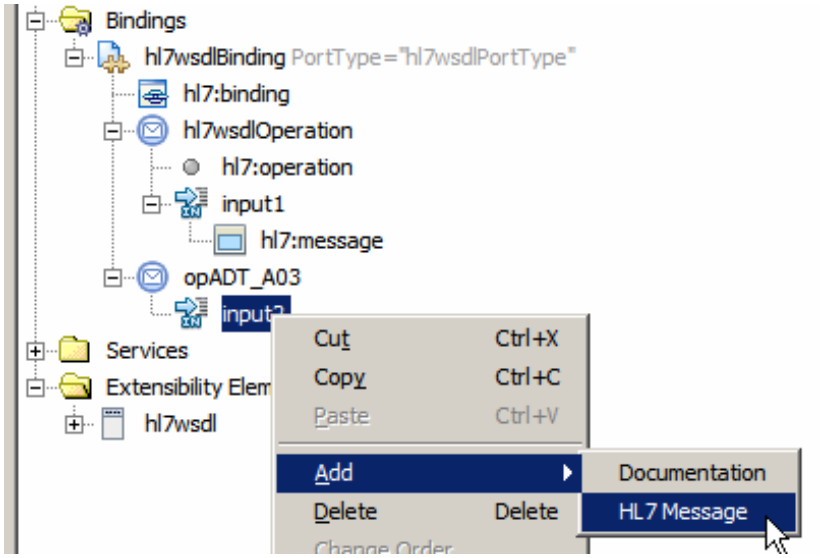


Figure 0-29 Trigger Add HL7 Message to the Binding Operation functionality

Right-click hl7:message and choose Properties.

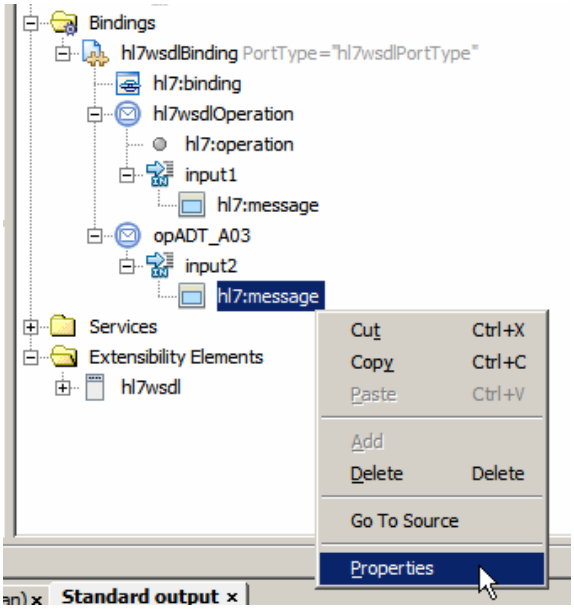


Figure 0-30 Choose Properties of the hl7:message node

Choose **part1** for part, **encoded** for use and enter **hl7encoder-1.0** for encodingStyle.

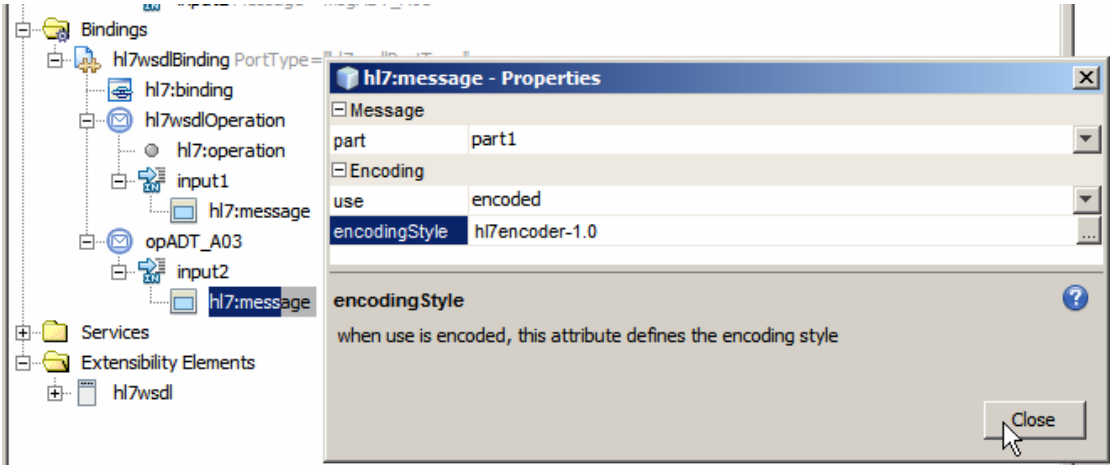


Figure 0-31 Configure hl7:message properties.

Right-click opADT_A03 node, choose Add, choose HL7 Operation.

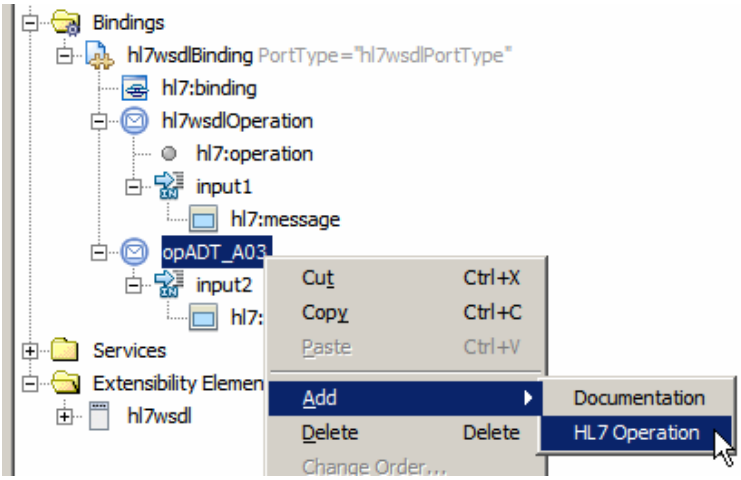


Figure 0-32 Add HL7 Operation to the Binding Operation

Right-click hl7:operation node, choose Properties, enter ADT^A03 into the messageType data entry box and close the dialogue box.

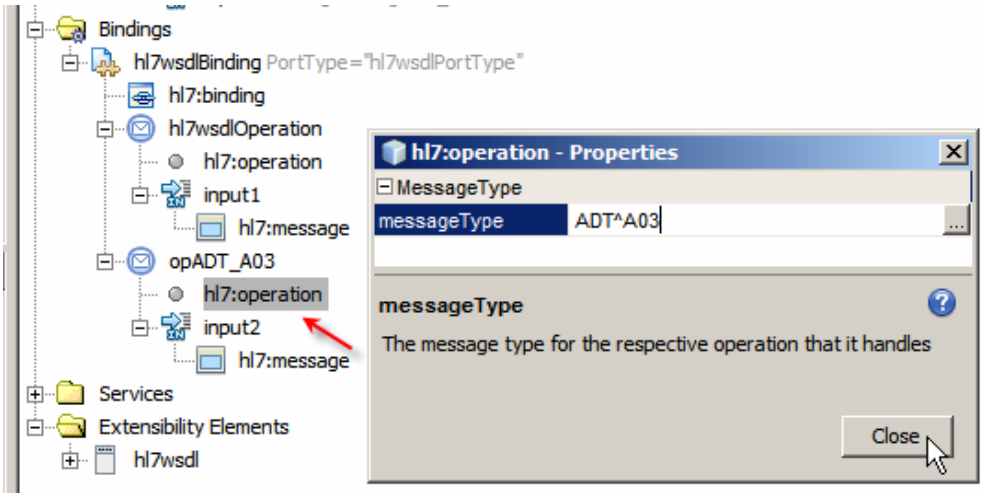


Figure 0-33 Configure messageType for the opADT_A03 operation

Save the modified WSDL. Now the HL7 BC will accept both ADT^A01 and ADT^A03 message types and will reject all others.

Now that we have a configured HL7 BC we can put together the Service Assembly that will constitute our Composite Application. The Composite Application will receive HL7 A01 and A03 delimited messages, convert them to their equivalent XML forms and write all messages to a file using the File Binding Component.

Let's open the Service Assembly, right-click in the WSDL Port swim line and choose Load WSDL Port…

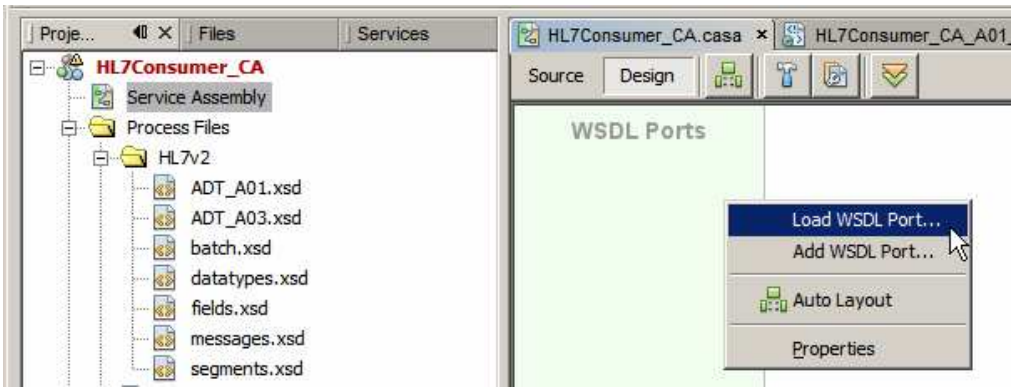


Figure 0-34 Triggering Add WSDL Port functionality

Select the one and the only WSDL Port and click OK.

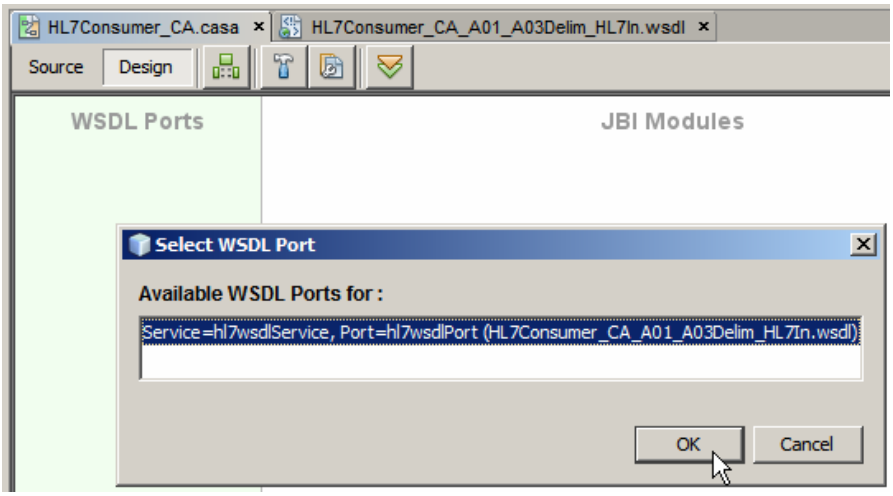


Figure 0-35 Select the WSDL Port to load

From the WSDL Bindings Palette drag the File BC onto the WSDL Ports swim line.

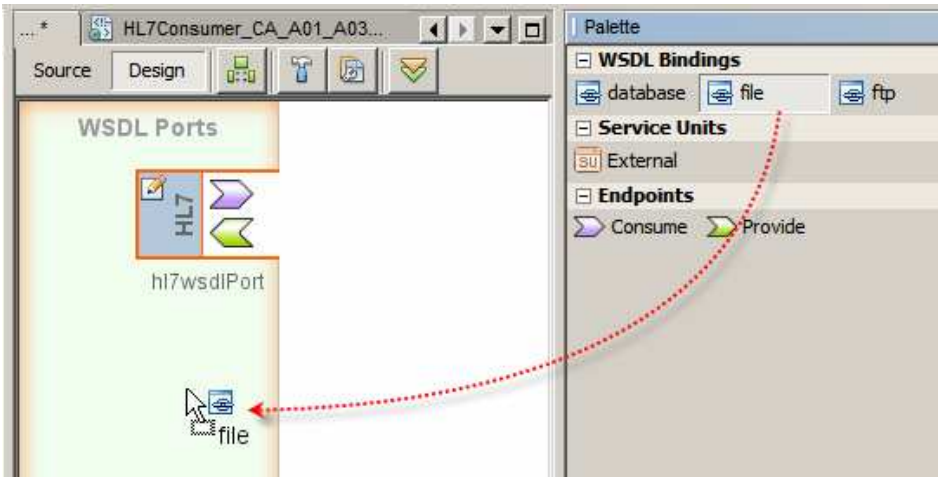


Figure 0-36 Add File BC to the Service Assembly editor canvas

Connect the Consume endpoint of the HL7 BC to the Provide endpoint of the File BC, choose hl7wsdlOperation from the drop down menu that appears and click OK.

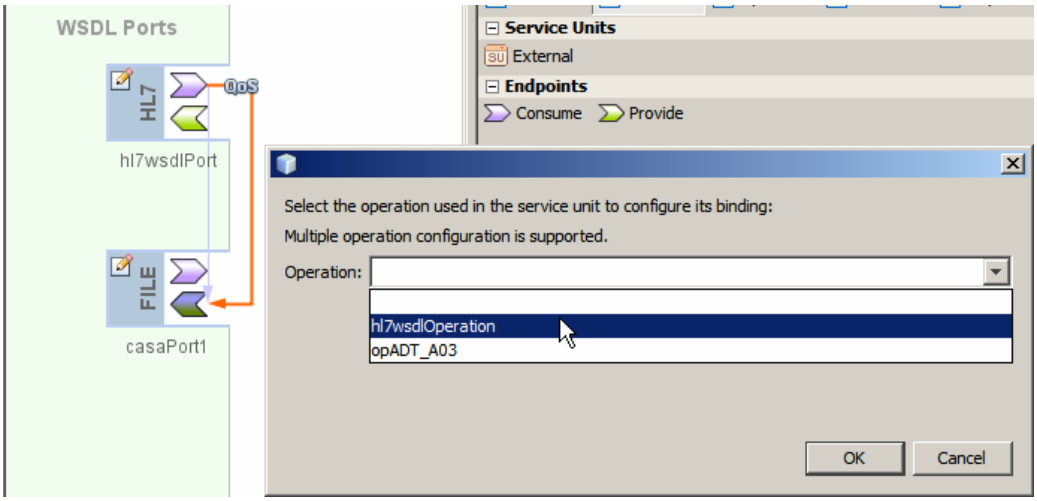


Figure 0-37 Choose to configure the A01 (hl7wsdlOperation) operation

Configure File Name * (Pattern) to read ADT_A0x_output_%d.xml and an appropriate output directory, to which the file will be written. Accept defaults for other properties and click OK.

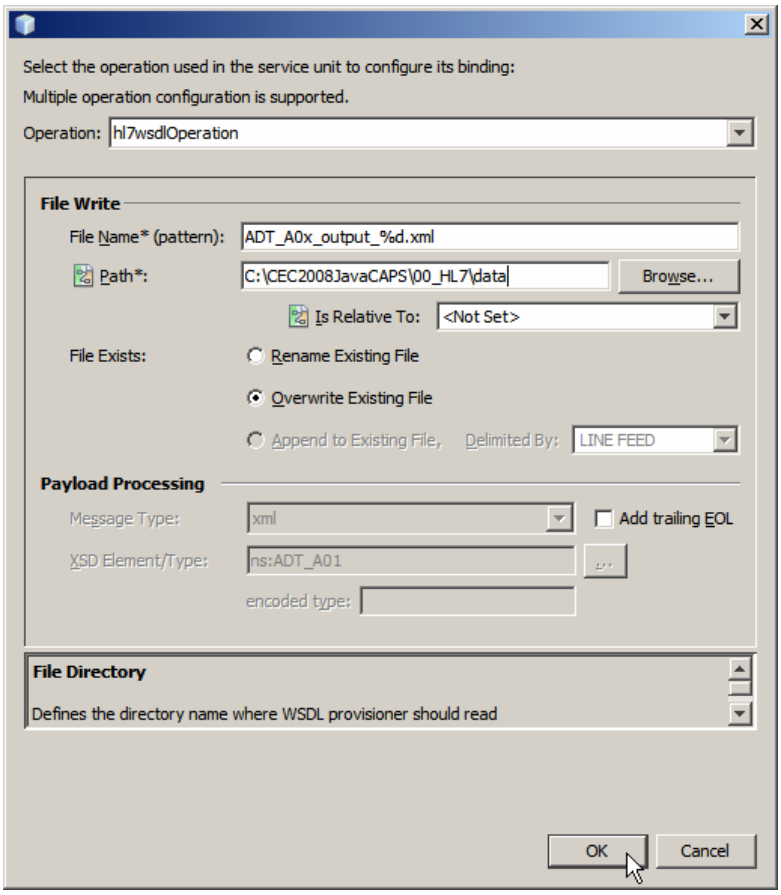


Figure 0-38 Configure file to which to write A01s

Click the paper and pencil icon on the File BC, select the opADT_A03 from the drop down and click OK.

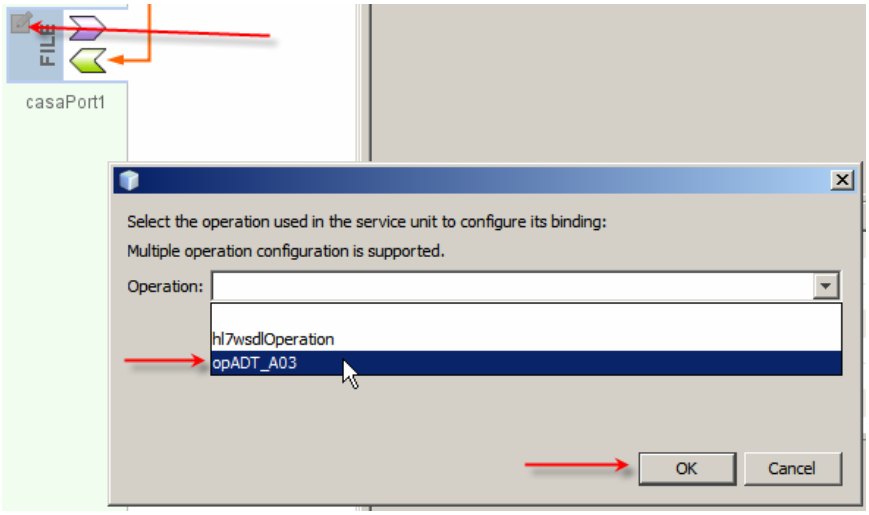


Figure 0-39 Trigger configuration of the File BC for the ADT A03 operation

Make sure to configure the file name, the output directory and all other properties identically to what was configured for the other operation before. This is necessary so that all HL7 messages get written to the same file.

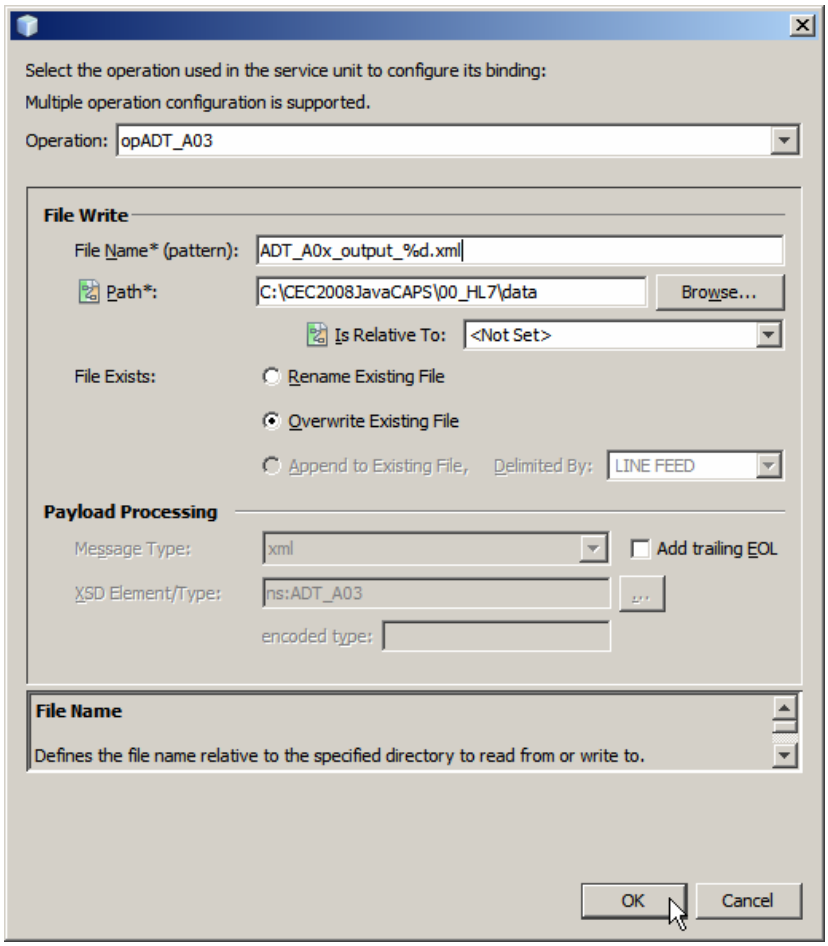


Figure 0-40 Configure File BC for the opADT_A03 operation the same as for ADT A01

Build and deploy the project.

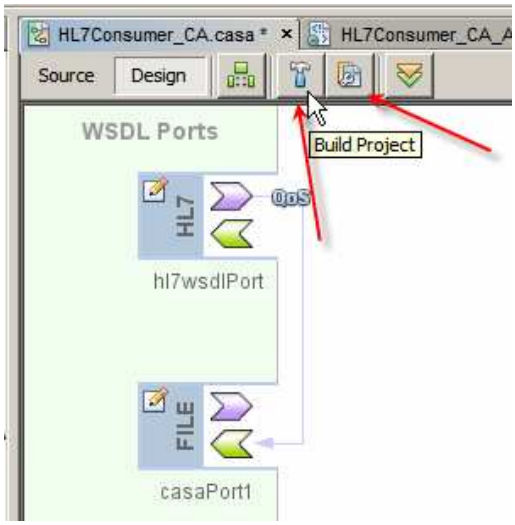


Figure 0-41 Build and Deploy buttons

This is probably one of the World’s smallest useful integration solutions. It takes a stream of HL7delimited messages coming over a TCP/IP connection, acknowledges them to the sender and writes their equivalent XML messages to files in the file system.

If you have the 7Scan tool installed and configured to talk to a HL7 listener then you can test the HL7 Consumer project by submitting to it A01 and A03 messages. Section “

Optional 3rd Party Software" discusses the 7Scan and gives a brief overview of how to configure it as a HL7 sender.

If you don't have 7Scan, or another HL7 client, then proceed to the next section where a HL7 Feeder project will be developed.

If you have 7Scan then continue with this section.

Start 7Scan. Click the Open button.

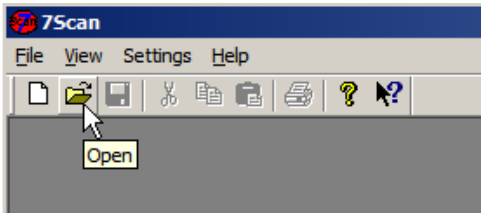


Figure 0-42 Trigger Open File functionality

Navigate to data/sources directory, select the ADT_A01_one_tx.dat file and click open.

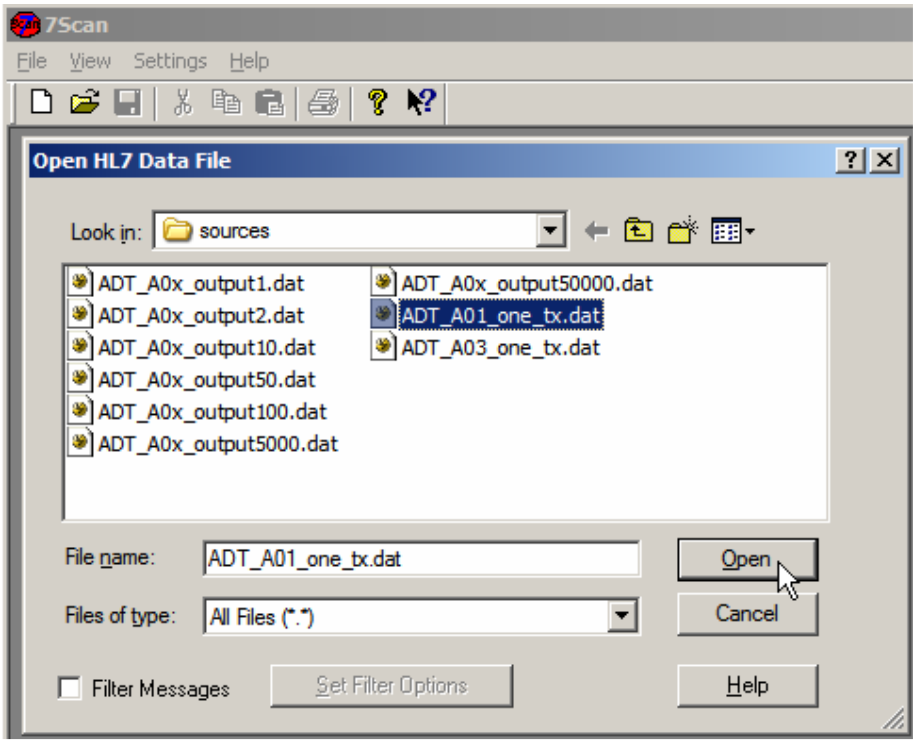


Figure 0-43 Choose ADT_A01_one_tx.dat to open

Click Sender_LocalHost_34001 to connect to the listener – this assumes that you created a sender as discussed in "

Optional 3rd Party Software". If not, create and configure one.

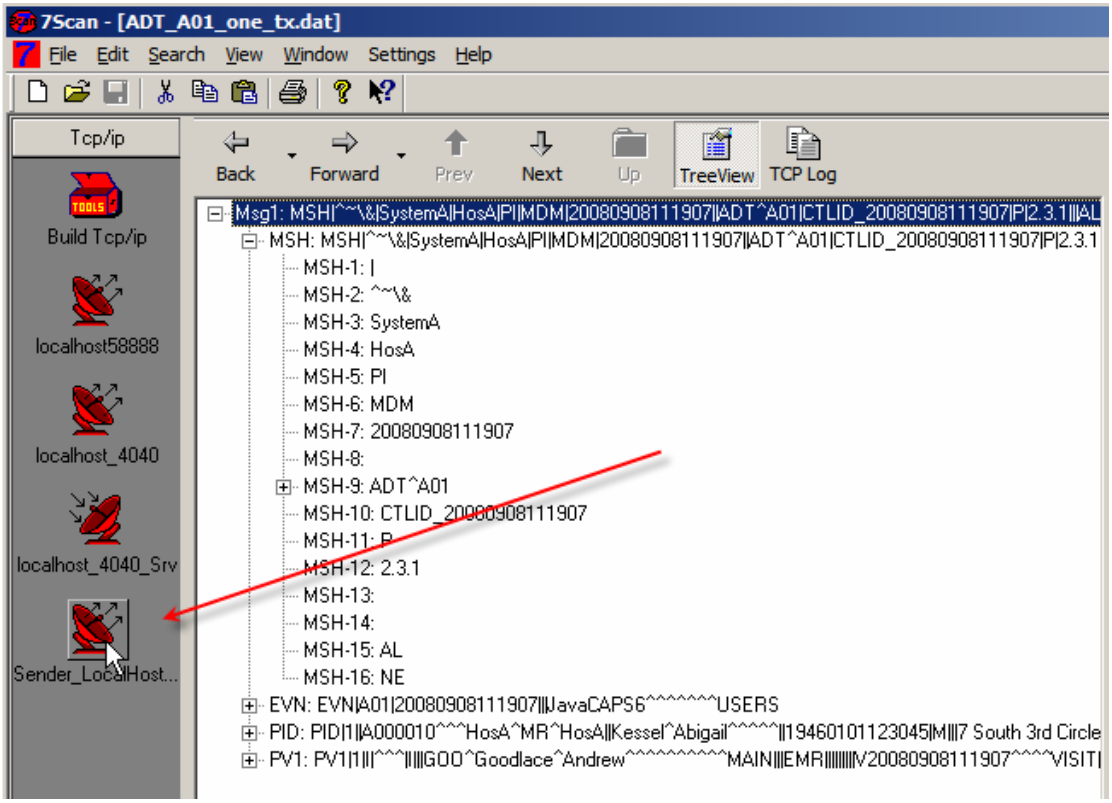


Figure 0-44 Connect to the listener

Right-click on the HL7 message in the window and choose Send Msg1.

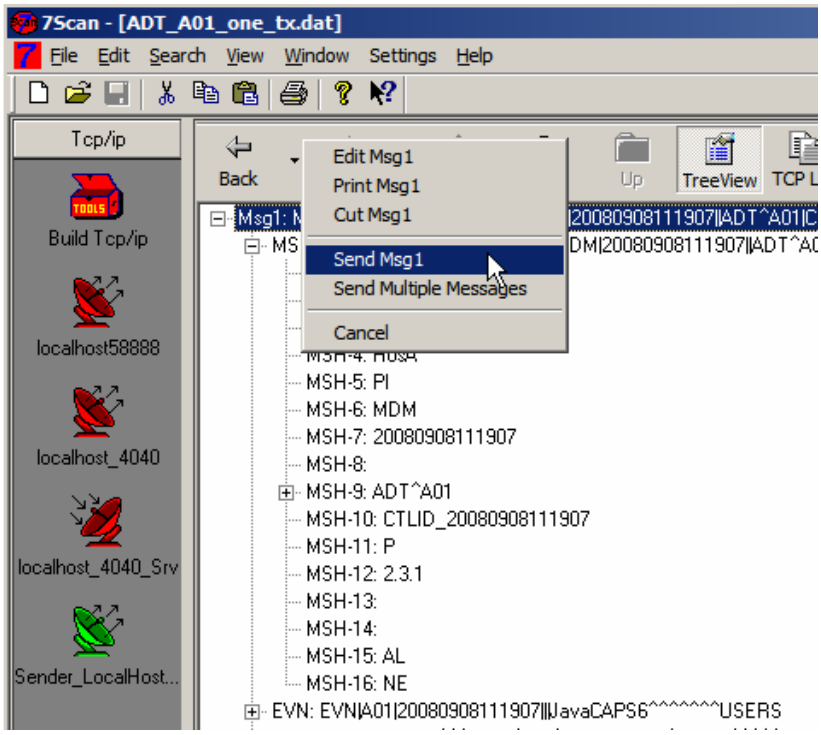


Figure 0-45 Send a message

Click the TCP Log button, observe the HL7 ACK, clear the log and dismiss the dialog box.

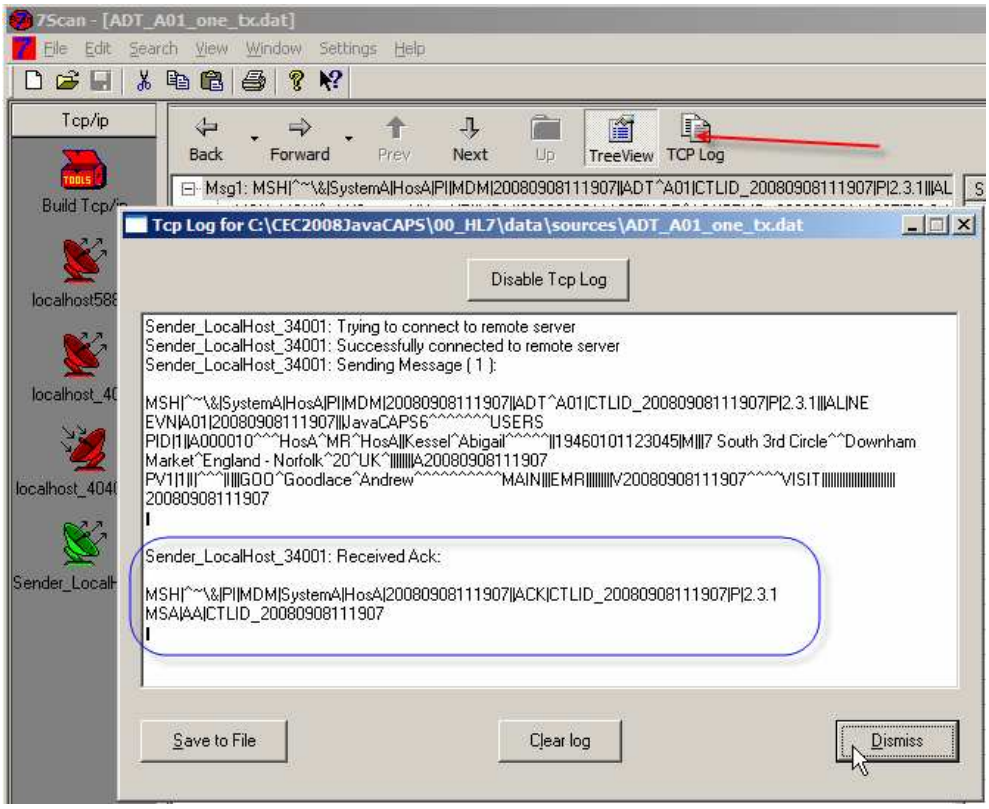


Figure 0-46 Observe HL7 ACK in the TCP Log

Open the ADT_A03_one_tx.dat message, connect the sender and send. Observe the TCP Log to confirm successful send.

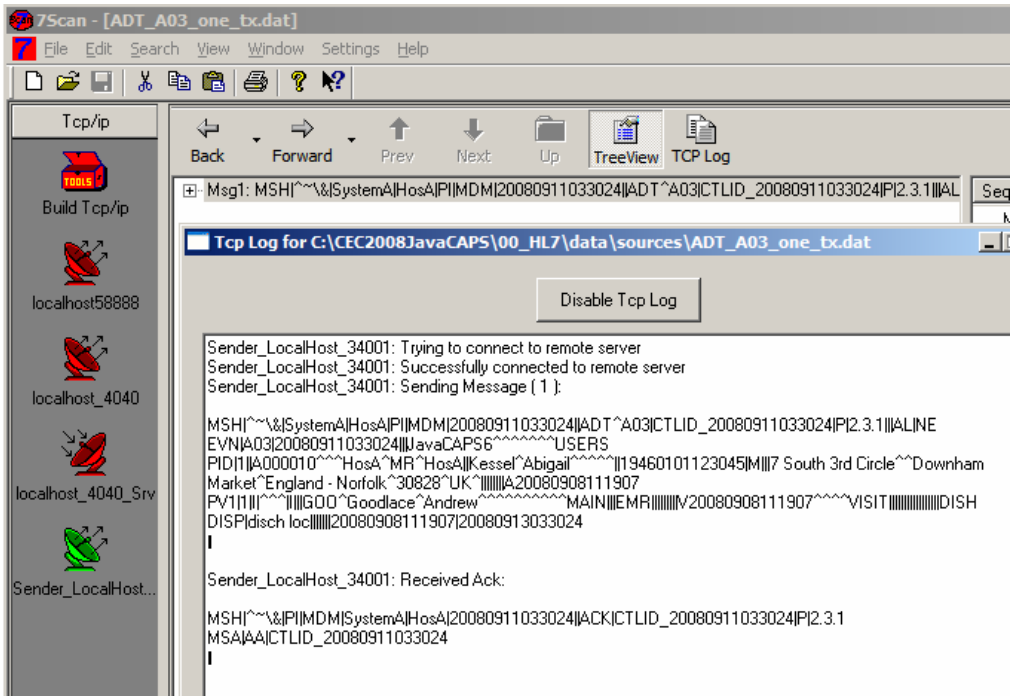


Figure 0-47 Send an ADT A03 and observe the HL7 ACK

Navigate to the directory configured as output directory for the File BC and view the two HL7 XML messages in files named ADT_A0x_output_*nn*.xml.

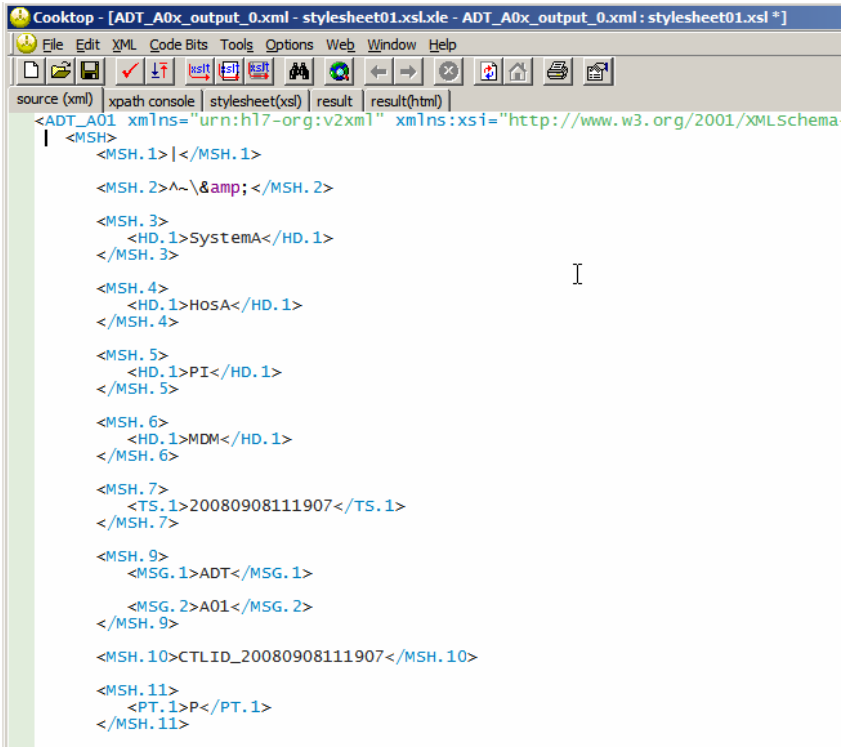


Figure 0-48 ADT A01

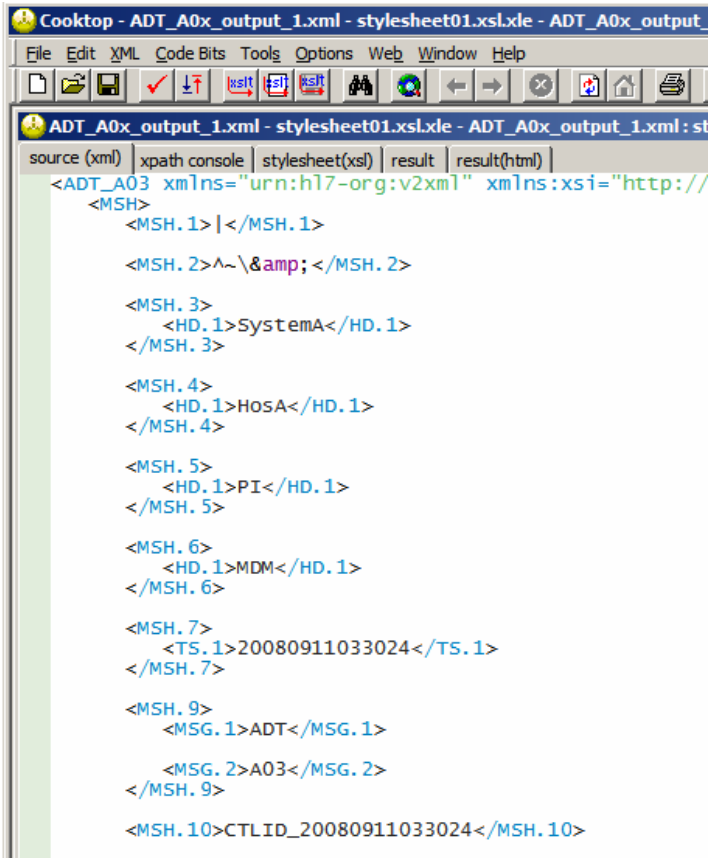


Figure 0-49 ADT A03

This completes the HL7 Consumer, which we will stop using once we test the HL7 Feeder project, developed next.

HL7 Feeder

The HL7 Feeder project will read a file containing one or more HL7 delimited records, both ADT A01 and ADT A03, and will use the HL7 Binding Component to send them to the HL7 Listener we developed in the previous section.

Alas, to send HL7 delimited messages out the HL7 BC expects to be fed HL7 XML records that it will 'encode' to delimited format before sending. To feed HL7 XML records to the HL7 BC the File BC will have to 'decode' delimited HL7 records it reads from the file. It would have been more efficient if this double conversation was not required but this is what the HL7 BC does so we must accommodate ourselves to the inevitable until and unless this requirement is relaxed.

> It is my understanding that adding functionality which avoids this double conversion is planned for the HL7 BC in early 2009.

The conversion logic is handled by the HL7 Encoder invoked directly from the Binding Components. We will add both Binding Components to the Composite Application Service Assembly, wire them together and deploy the result, much as we have done in the previous section.

Let's create a new Composite Application project, named HL7Feeder_CA. The process should be familiar by now so we will not provide detailed screenshots for some of the steps.

Let's create a folder called HL7v2 inside the Process Files folder. We will import XML Schema Documents we will need for HL7 encoding/decoding into this folder.

In the previous section we used the ADT_A01 and ADT_A03 XML Schemas to instruct the HL7 BC to accept and 'decode' both of these transactions. This time we will instruct the HL7 BC to read and send any HL7 version 2 message, whether ADT or otherwise. To do so we must create a 'generic' HL7 Version 2 message XML Schema Document.

Let's import External XML Schema Document ADT_A01 and subsidiary schemas into the HL7v2 folder, much as we have done in the previous section. What we need are:

- ADT_A01.xsd
- batch.xsd
- datatypes.xsd
- fields.xsd
- messages.xsd
- segments.xsd

Rename the ADT_A01.xsd to HL7_ANY.xsd.

Open the HL7_ANY.xsd schema and switch to Source mode.

Figure 0-50 Open HL7_ANY.xsd in Source mode

Replace lines 21 through 69, inclusive, with the following:

```
    <!--
       MESSAGE
    -->
    <xsd:element name="MSG" type="MSG.CONTENT">
        <xsd:annotation>
            <xsd:appinfo source="urn:com.sun:encoder">
                <top xmlns="urn:com.sun:encoder">true
                </top>
            </xsd:appinfo>
        </xsd:annotation>
    </xsd:element>
    <xsd:complexType name="MSG.CONTENT">
        <xsd:sequence>
            <xsd:element ref="MSH" minOccurs="1" maxOccurs="1"/>
            <xsd:any processContents="lax"
                    namespace="##any"
                    minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
```

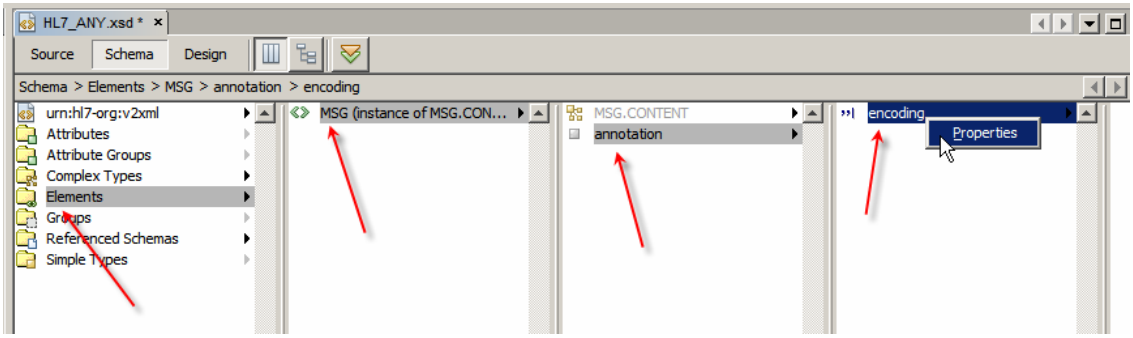
The complete XML Schema Document for any HL7 v2 message will look like the following.

```
<?xml version ="1.0" encoding="UTF-8"?>
<xsd:schema
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="urn:hl7-org:v2xml"
    xmlns:hl7="urn:com.sun:encoder-hl7-1.0"
    targetNamespace="urn:hl7-org:v2xml"
    xmlns:jaxb="http://java.sun.com/xml/ns/jaxb" jaxb:version="2.0">

    <!-- include segment definitions for version v2.3.1 -->
    <xsd:include schemaLocation="segments.xsd"/>

    <xsd:annotation>
       <xsd:appinfo source="urn:com.sun:encoder">
           <encoding
                xmlns="urn:com.sun:encoder"
                name="HL7 v2 Encoding"
                namespace="urn:com.sun:encoder-hl7-1.0"
                style="hl7encoder-1.0"/>
       </xsd:appinfo>
    </xsd:annotation>

    <!--
       MESSAGE
    -->
```

```
    <xsd:element name="MSG" type="MSG.CONTENT">
        <xsd:annotation>
            <xsd:appinfo source="urn:com.sun:encoder">
                <top xmlns="urn:com.sun:encoder">true
                </top>
            </xsd:appinfo>
        </xsd:annotation>
    </xsd:element>
    <xsd:complexType name="MSG.CONTENT">
        <xsd:sequence>
            <xsd:element ref="MSH" minOccurs="1" maxOccurs="1"/>
            <xsd:any processContents="lax" namespace="##any"
            minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

Save the XSD and switch to Schema mode.

Expand Elements->MSG->annotation->encoding. Right-click encoding and choose Properties.



Check the Top checkbox and close the properties.

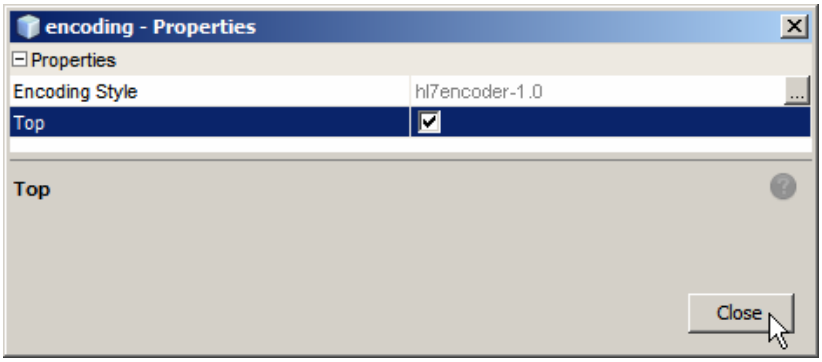


Figure 0-51 Check the Top element.

Save the XSD again.

Now that the XML Schema is ready we can use it to configure the HL7 BC to process any HL7 version 2 message.

Let's create a new concrete WSDL Document, HL7Feeder_CA_HL7_ANY_HL7Out, using the HL7 binding of type HL7 Version 2 – Outbound Request.

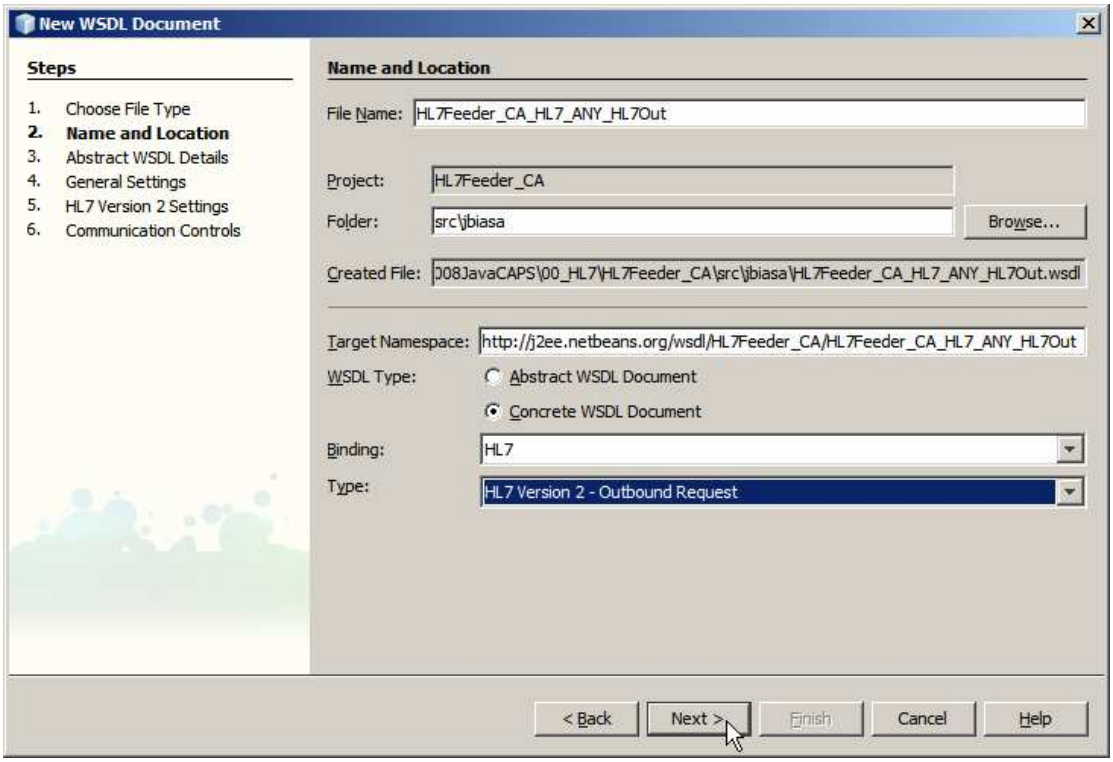


Figure 0-52 HL7 Outbound Request

Choose HL7_ANY->MSG Element as the Request Message and click Next.

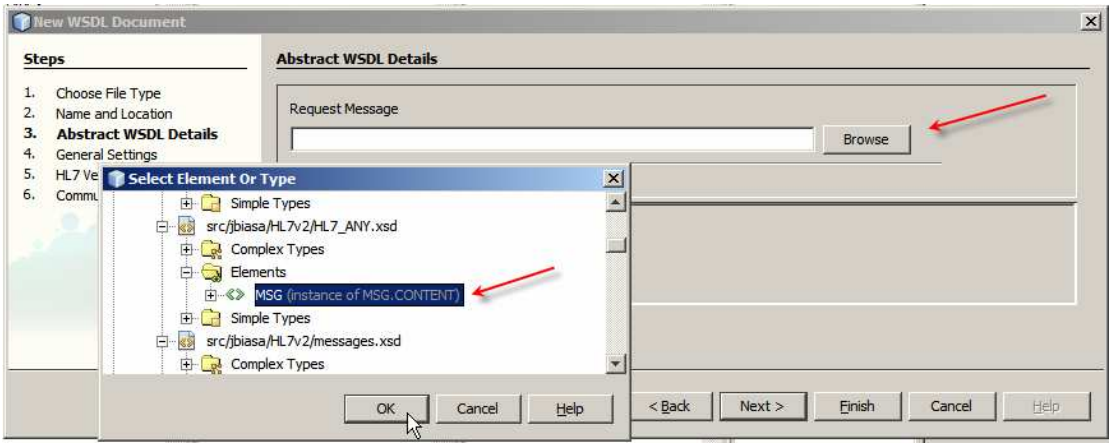


Figure 0-53 Choose HL7_ANY MSG Element

Make sure to verify that the Listener location is configured as required and the LLP encoding characters are configured to agree with these of the listener to which this BC will connect. Click Finish, leaving remaining settings at their defaults.

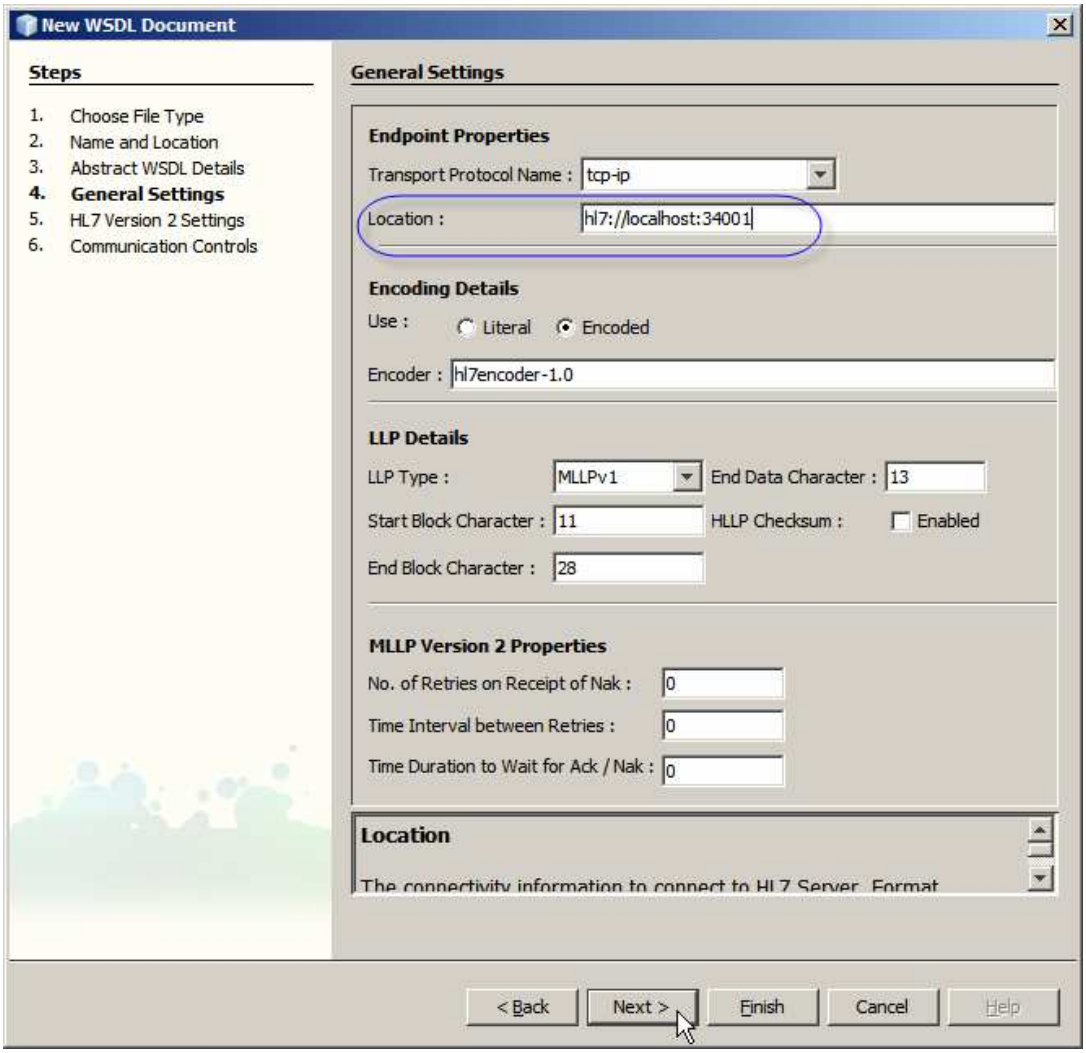


Figure 0-54 Confirm General Settings

Open the Service Assmebly, right-click inside the WSDL Ports swim line, choose Load WSDL Port, choose the one and only WSDL Port and click OK.

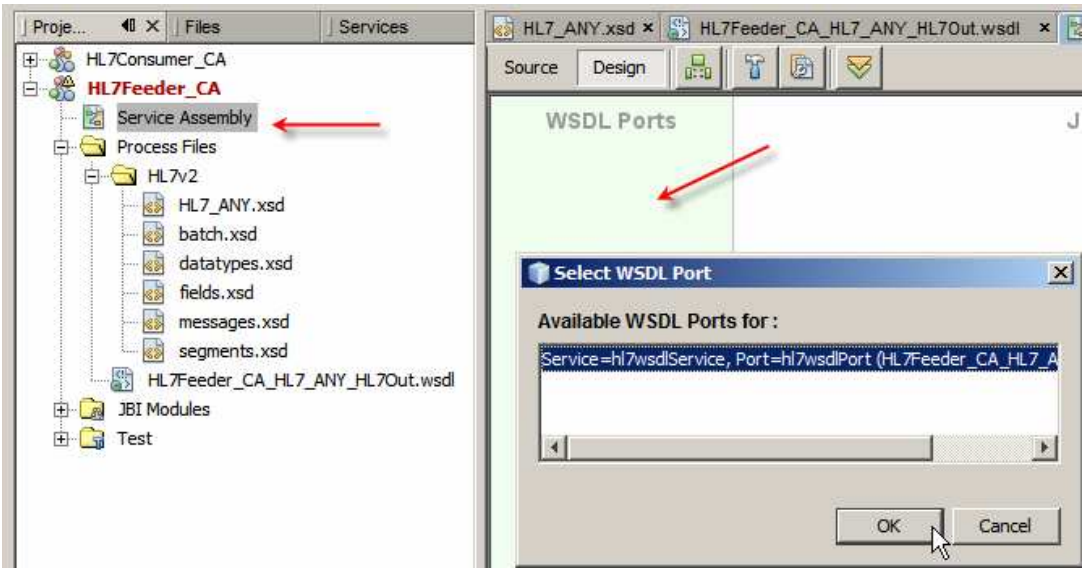


Figure 0-55 Load WSDL Port into the Service Assembly

Drag the File BC from the WSDL Bindings Palette onto the WSDL Ports swim line. Connect the Consume endpoint of the File BC to the Provide endpoint of the HL7 BC, configure File BC properties to read a file named ADT_A0x_output%d.dat from the data directory. Make sure to check the Multiple Records checkbox, specify \r\r\n for Delimited By and check the Remove Trailing EOL checkbox. Then click OK.

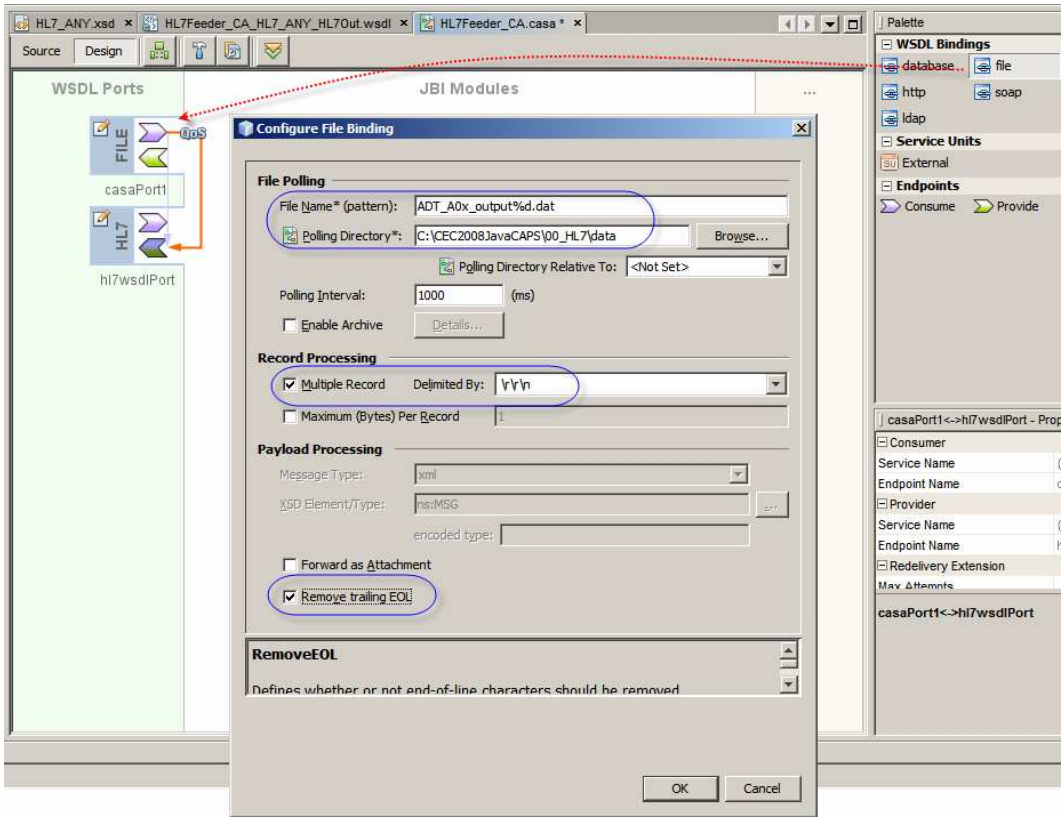


Figure 0-56 Add File BC to the Service Assmebly, connect and configure

The CASA editor adds another WSDL to the configuration under Process Files. This WSDL, HL7Feeder_CA.wsdl, must be modified manually to make it support on-the-fly ‘decoding’ of HL7 delimited messages.

Open the WSDL HL7Feeder_CA.wsdl, expand Bindings->casaBinding1->hl7wsdlOperation->input1. Right-click file:message and configure the use property to encoded and the encodingType property to hl7encoder-1.0.

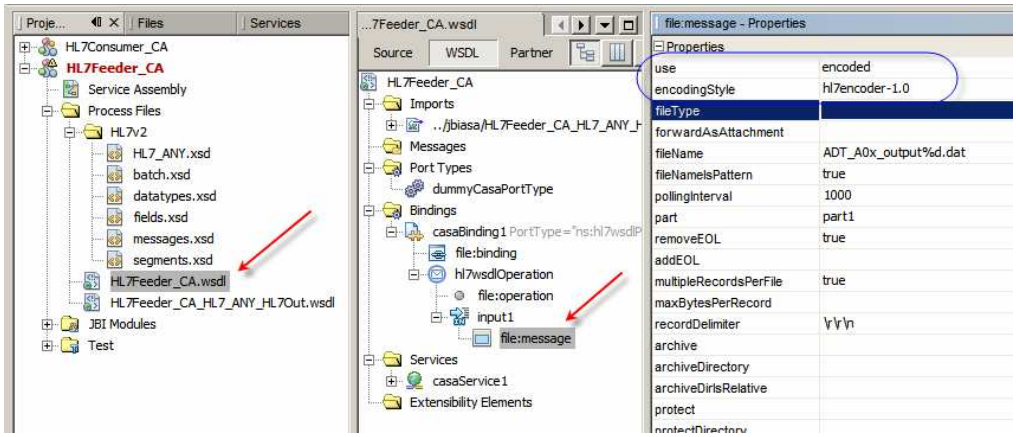


Figure 0-57 Fix the File BC WSDL to support ‘decoding’ of delimited HL7 messages

Switch back to CASA editor, build and deploy the project.

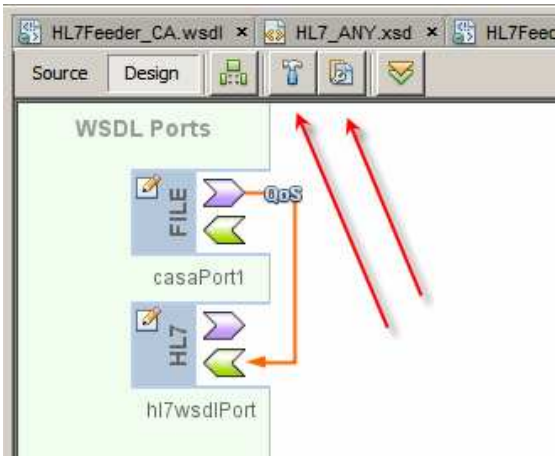


Figure 0-58 Build and deploy the project

Copy the sample file, ADT_A0x_output1.dat, from **data/sources** to **data** directory to provide input for the File BC.

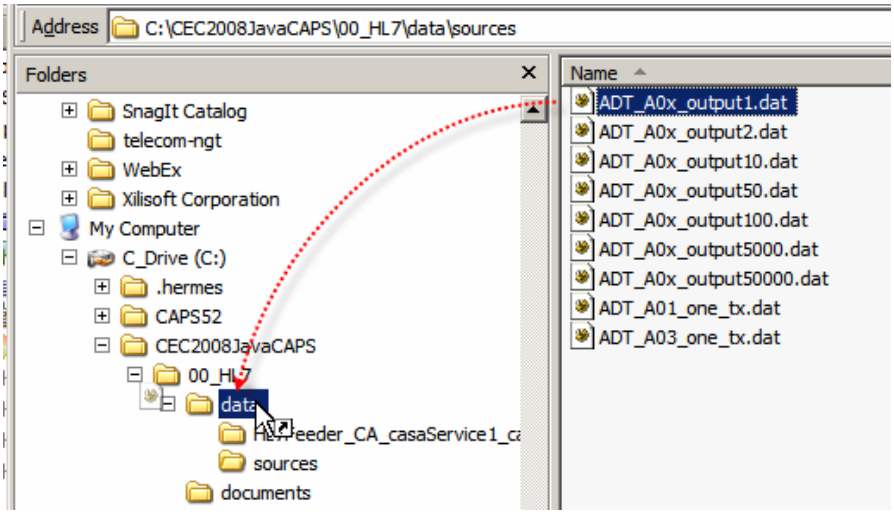


Figure 0-59 Provide input for the File BC.

Observe the output file, ADT_A0x_output_*nn*.xml, produced by the HL7 Consumer we developed in the previous section.

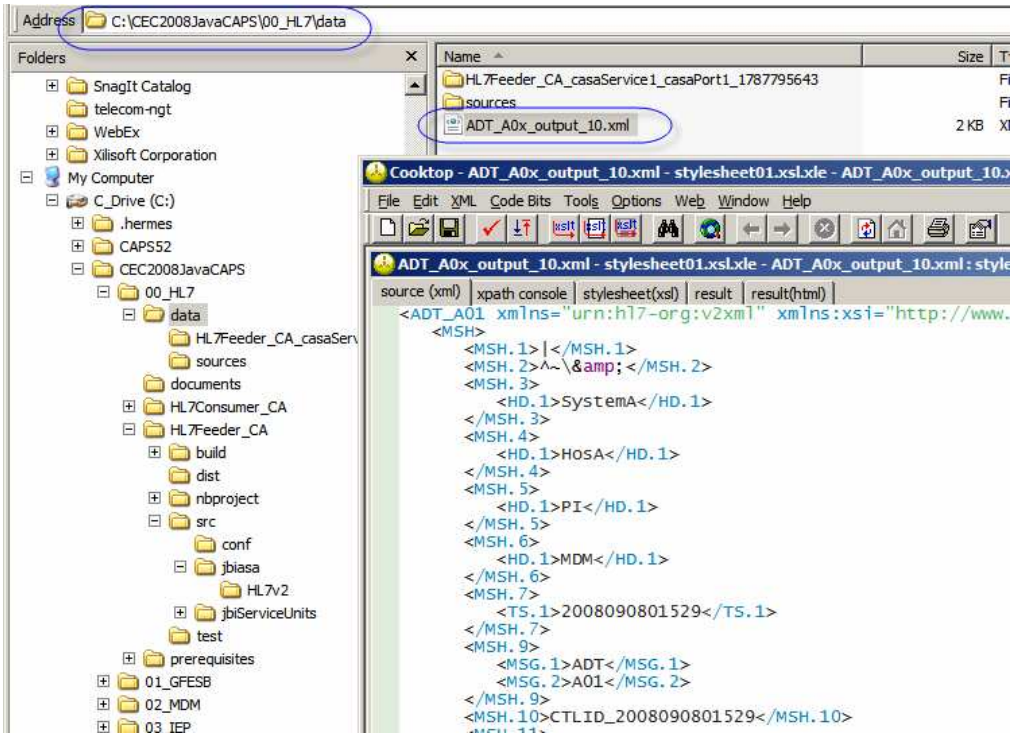


Figure 0-60 HL7 XML output produced by the HL7 Consumer

Clearly, the HL7 Feeder and the HL7 Consumer work together to read HL7 delimited messages from a file, pass them through a pair of HL7 BCs to eventually write the XML versions to an output file.

Now that we have tested with one message manually submitted we need to ensure that the order of message submission is preserved. This is very important in Healthcare. Let's right-click on the link between the BCs in the CASA map and configure the Throttling Extension->Max Concurrency Limit property to 1, build and deploy the project again. This will ensure that messages are sent one at a time.

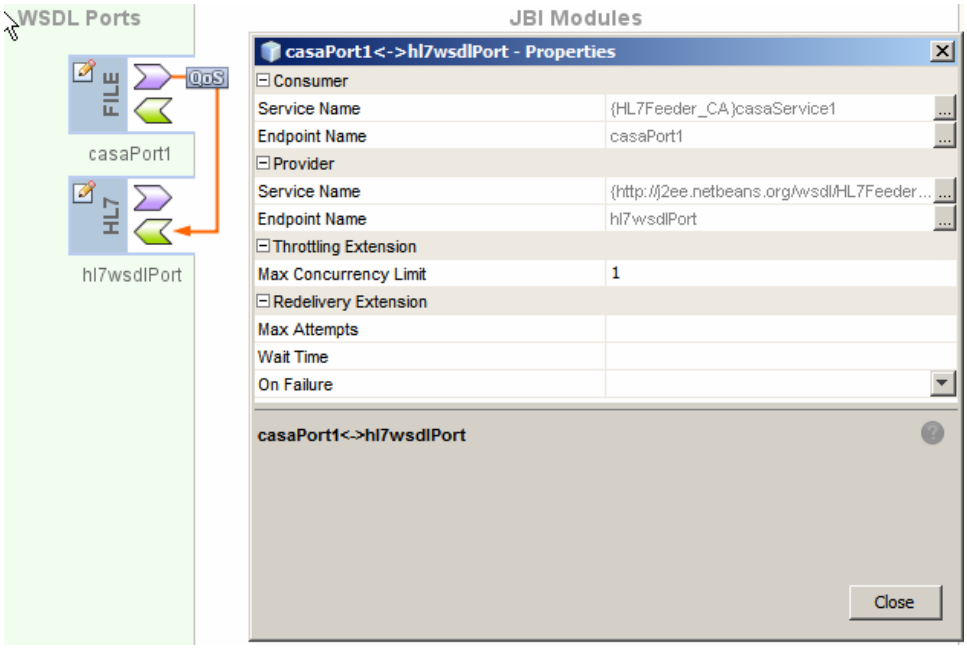


Figure 0-61 Configuring Max Concurrency Limit

Our HL7 Feeder is ready and tested. We can now undeploy the HL7 Consumer so it does not get in a way of the HL7 Processor we will develop shortly.

Date Difference Utility

The HL7 Processor processes ADT A01 (Admission) and ADT A03 (Discharge) messages. The A03 Discharge messages are converted to the Custom Discharge XML format which is the feed to the Intelligent Event Processor-based Excessive Length of Stay solution, discussed in http://blogs.sun.com/javacapsfieldtech/entry/glassfish_esb_illustrated_solution_development Java CAPS Essentials IEP Lab. In order to provide appropriate data to the IEP solution we need to work out the length of stay in days, based on the difference between the Discharge Date and the Admission Date contained in the ADT A03 Discharge message.

As a generally useful thing we will develop an EJB-based Web Service that will accept two dates and return a difference in days between them. This section walks through the process of developing and testing this utility.

> **Note** that there are at least three ways in which a piece of Java logic can be invoked from a BPEL 2.0 process – using a Web Service implementation, much as what we will be doing below, using a POJO Service Engine and using an embedded reference to a Java Class.

Note that we are departing, for a time, the Java Business Integration (JBI) word and are entering into the Java Enterprise world. The two will meet and intersect later when we include the web service we are developing in the JBI Service Assembly of the HL7 Processor Composite Application.

Let's create an Enterprise->EJB Module project called WSSDateDiff.

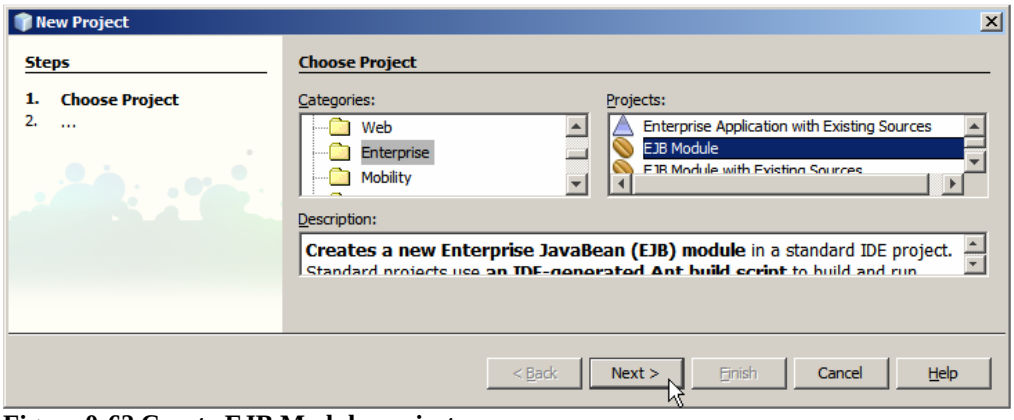


Figure 0-62 Create EJB Module project

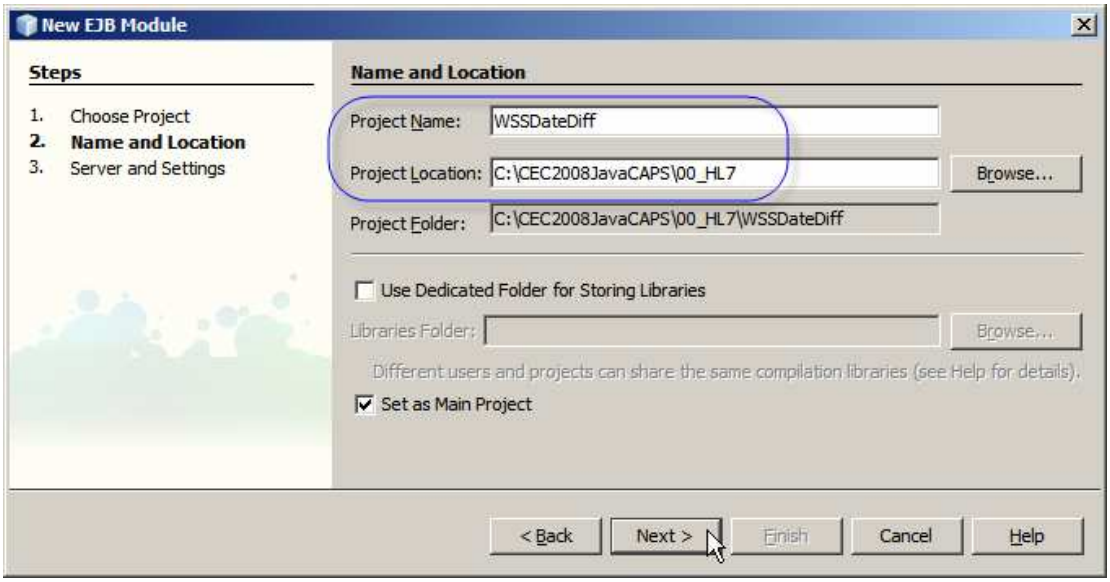


Figure 0-63 Name the project WSSDateDiff

We are developing a Web Service implementation using the “Implementation First” method, that is we will implement the service using Web Service Annotations and the service WSDL will be generated at build time. This is opposite to the “Interface First” method where the WSDL is defined first and the service is built to conform to the WSDL-defined interface specification.

Right-click on the name of the project, WSSDateDiff, choose New, choose Other …, choose Web Service category, choose Web Service and click Next.

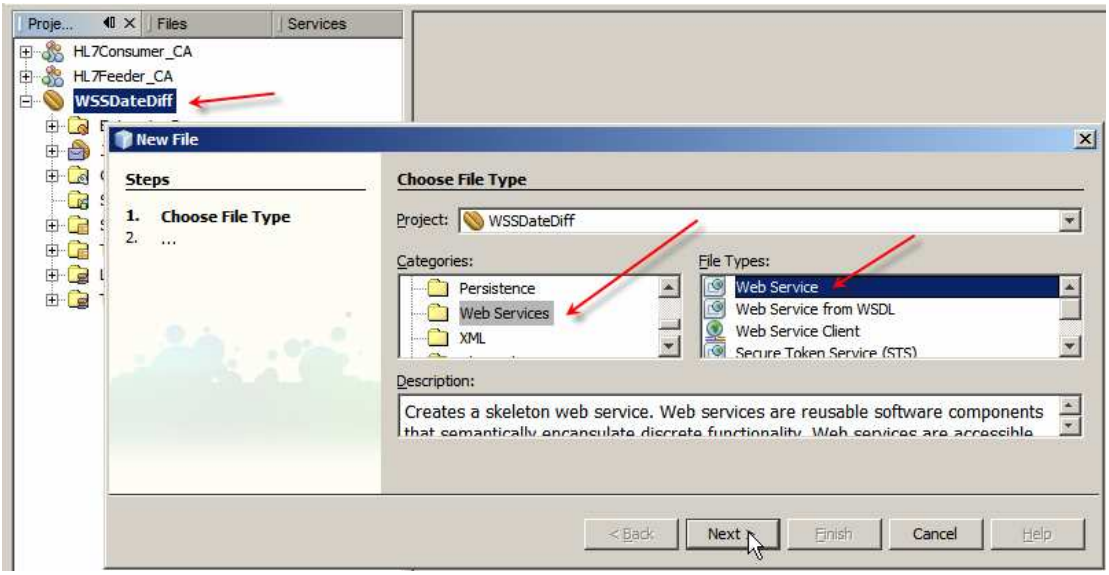


Figure 0-64 Create a new web service

Name this service WSSDateDiff, give the package the name pkg.WSSDateDiff and click Finish.

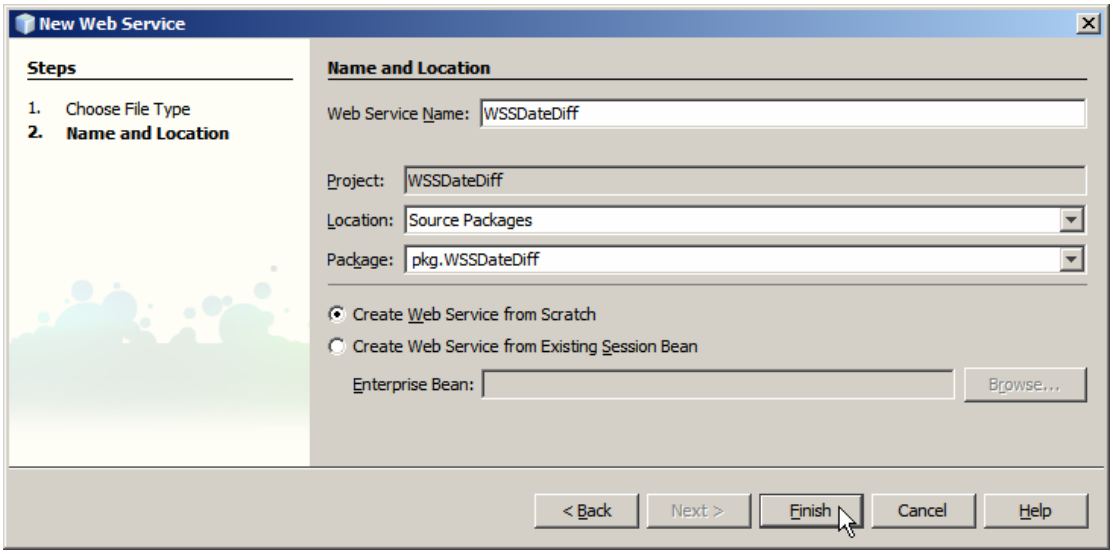


Figure 0-65 Complete wizard sequence

Define the web service interface be defining the method signature. Click Add Operation …, name the operation opDateDiff, change the Return Type to int, add four parameters of type java.lang.String named sEarlierdate, sEarlierDateFormat, sLaterdate, sLaterDateFormat, then click OK.

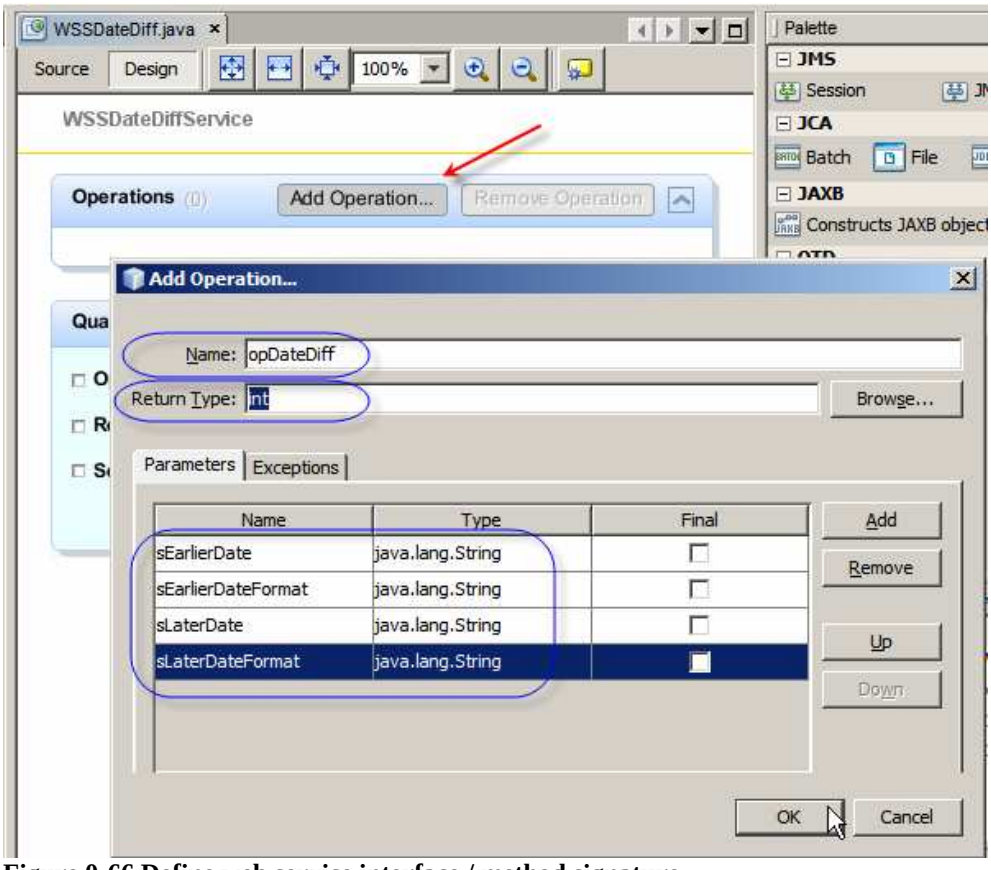


Figure 0-66 Define web service interface / method signature

The Design view should look like that shown below.

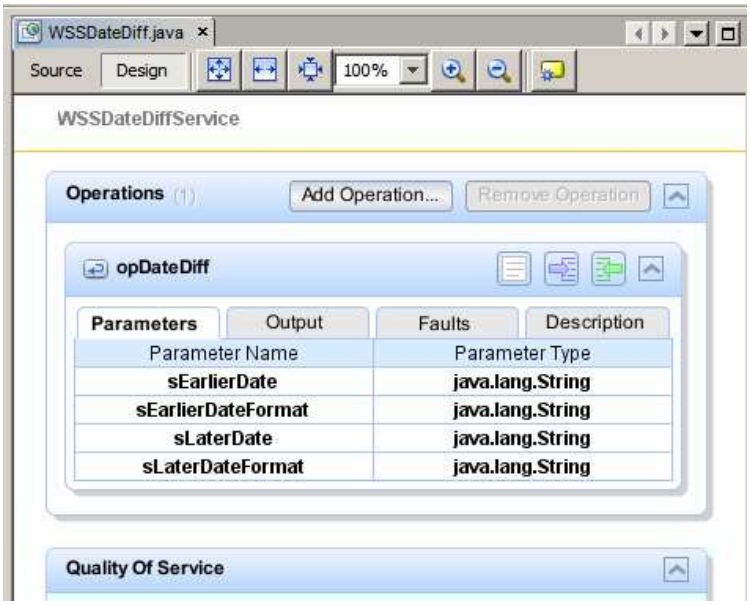


Figure 0-67 Web Service interface in Design View

Switch to the Source view and add the following immediately before the @WebMethod annotation:

```
    static Object oBj = new Object();
```

Modify the following

```
       String sLaterDateFormat) {
```

to read

```
       String sLaterDateFormat) throws Exception {
```

Replace

```
        //TODO write your implementation code here:
        return 0;
```

With

```
        java.text.DateFormat fmtEarlierDT =
                new java.text.SimpleDateFormat(sEarlierDateFormat);
        java.text.DateFormat fmtLaterDT =
                new java.text.SimpleDateFormat(sLaterDateFormat);
        java.util.Date dtMinuend = null;
        java.util.Date dtSubtrahend = null;
        long lDiffInDays = 0;
        synchronized (oBj) {
            try {
                dtMinuend = fmtLaterDT.parse(sLaterDate);
                dtSubtrahend = fmtEarlierDT.parse(sEarlierDate);
                long lDiffInMillis = dtMinuend.getTime()
                          - dtSubtrahend.getTime();
                long lDiffInSecs = lDiffInMillis / 1000;
                long lDiffInMins = lDiffInSecs / 60;
                long lDiffInHours = lDiffInMins / 60;
                lDiffInDays = (lDiffInHours / 24) + 1;
            } catch (java.text.ParseException ex) {
                throw new Exception(ex);
            }
        }
        return (int)lDiffInDays;
```

Build and deploy the project.

To test the project we will use the built-in web service testing functionality.

Expand the Web Services node under the project WSSDateDiff node, right-click the name of the service WSSDateDiff and choose the Test Web Service option.

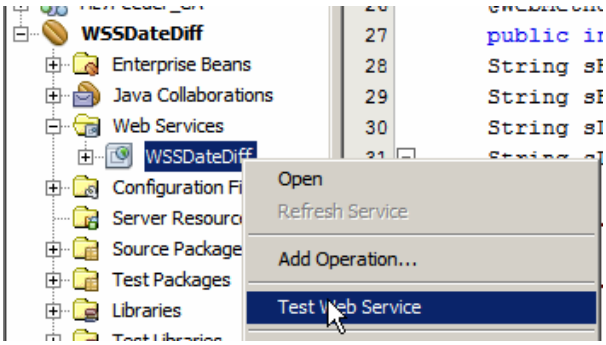


Figure 0-68 Invoke NetBeans Web Service Testing functionality

Once the NetBeans constructs the Web Services client and the Java Server Page to collect service parameter values, it will open a Web Browser window with the HTML form similar to that shown below. Provide dates and formats remembering that they are Earlier Date, Earlier Date Format, Later Date, Later date Format, then click the button labelled opDateDiff.

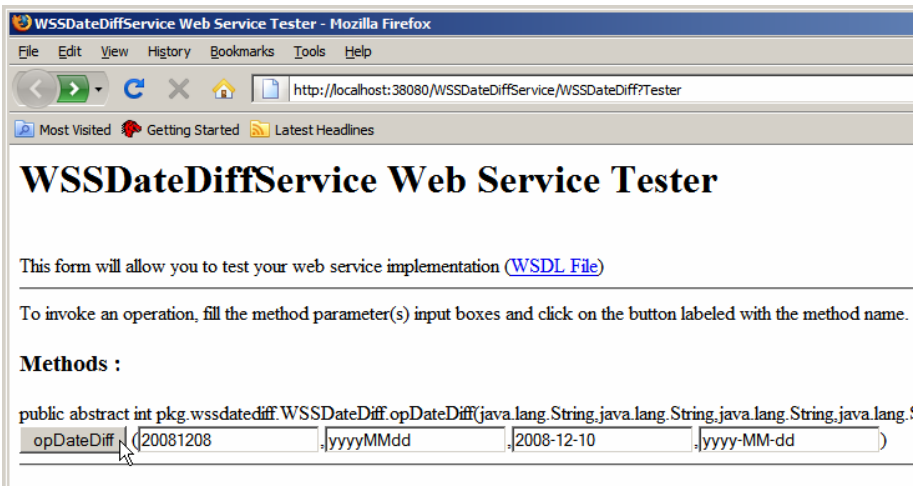


Figure 0-69 Web Service testing HTML Form

Note the (WSDL File) link. Once the service executes the results will be shown on the next web page and will look similar to the following.

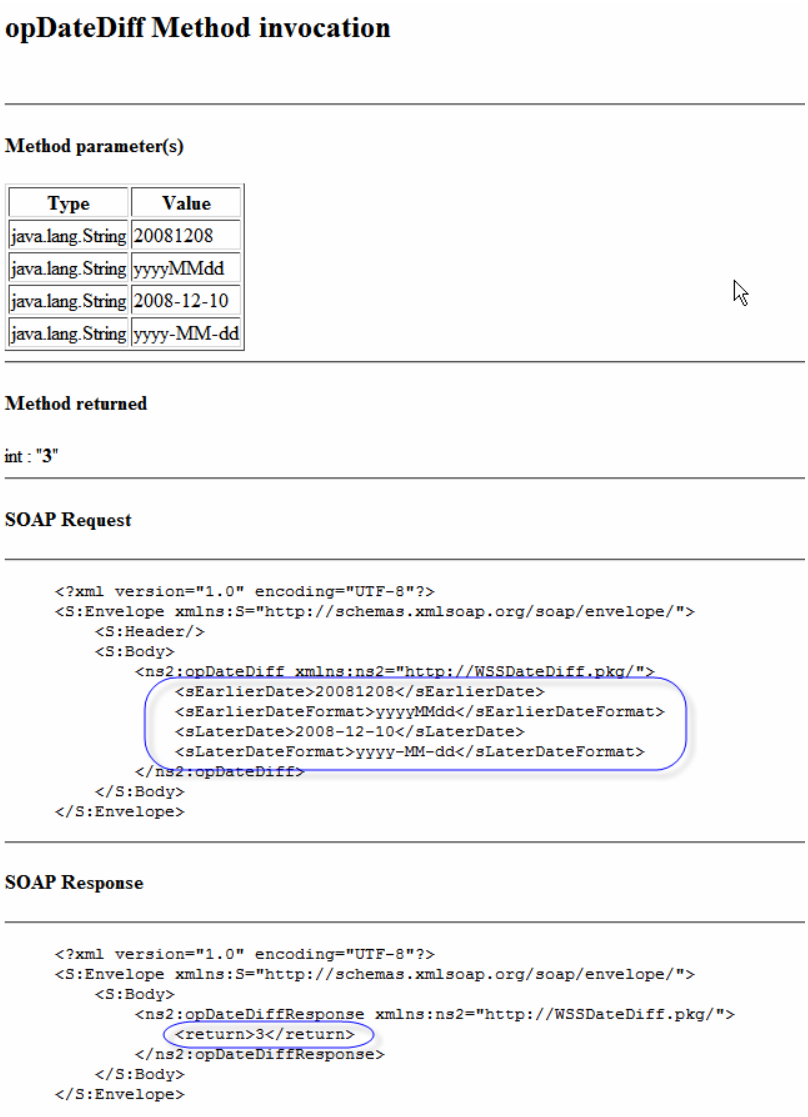


Figure 0-70 Service Test results

The service functions as expected, producing a difference between two dates in whole days. If the two dates were the same the result would be 0.

Click the Back button in the browser and click the (WSDL File) link to inspect the WSDL interface generated by the build process. This WSDL reference will be used later to provide the interface specification to the BPEL 2.0 process we will be building in a little while.

For the service I just built and deployed the WSDL URL is:

`http://localhost:38080/WSSDateDiffService/WSSDateDiff?WSDL`

Leave the service deployed.

If you had the SoapUI NetBeans plugin installed the NetBeans IDE would offer additional web service testing functionality. Let’s assume you have SoapUI plugin installed.

Right-click the Web Service WSSDateDiff node and choose Create Web Services Tests.

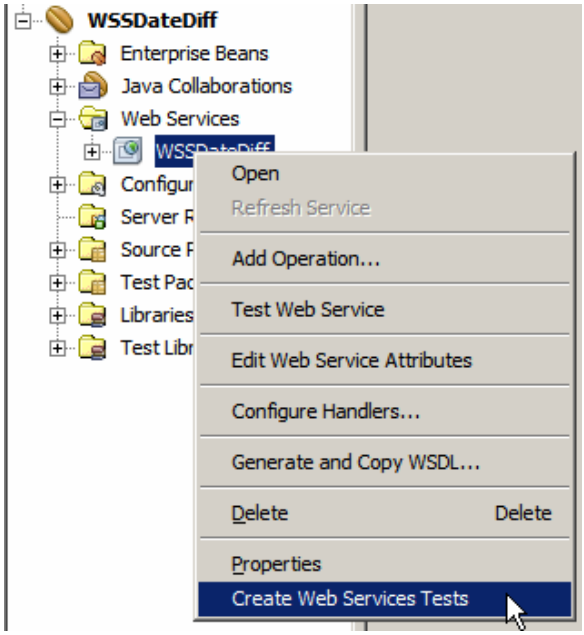


Figure 0-71 Triggering SoapUI Web Services Tests functionality

The SoapUI web service testing project will be created, the WSDL will be loaded and parsed and a dialog box with a variety of options will be shown. Accept all defaults by clicking OK.

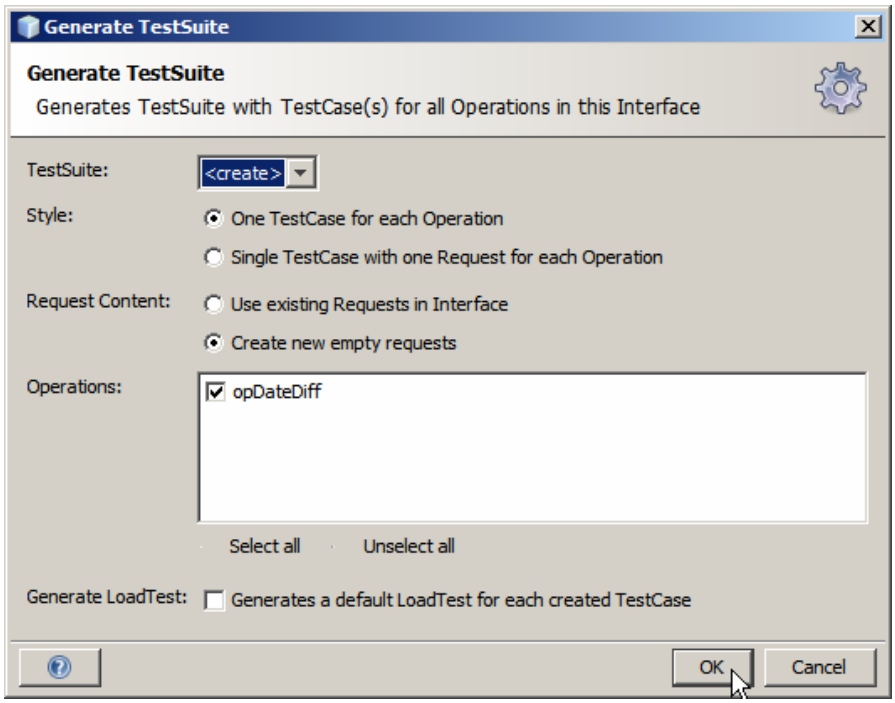


Figure 0-72 Test suite options

Accept the default name for the testing suite or provide one of your own.

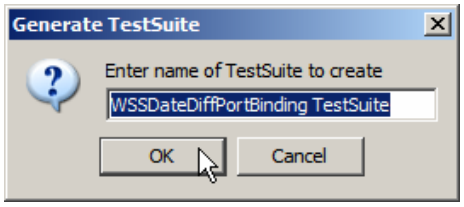


Figure 0-73 Name the testing suite

When the test suite folder tree appears expand it all the way to Request 1. Double click Request 1 and modify the request XML document to provide test data.

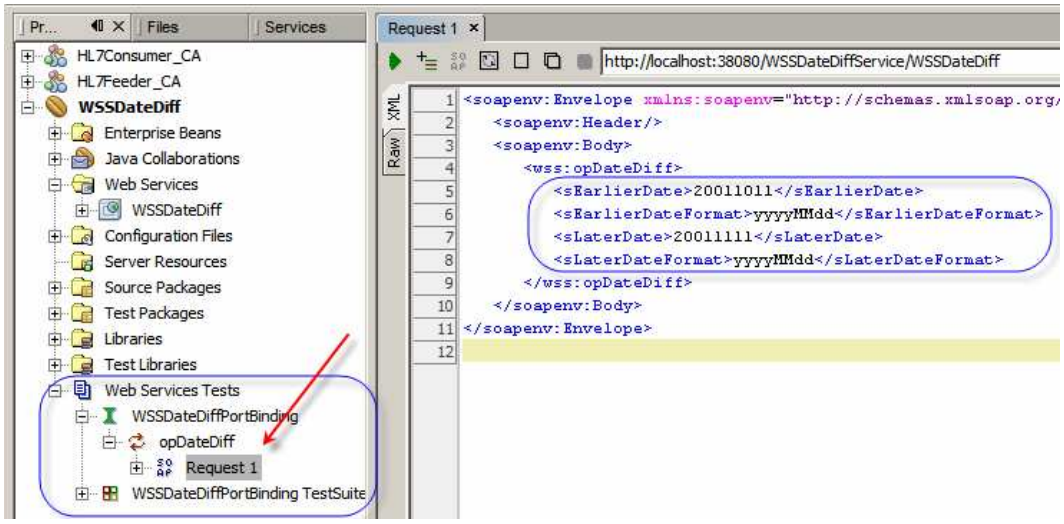


Figure 0-74 Locate and modify default Request 1 to provide data for testing

Submit the request.

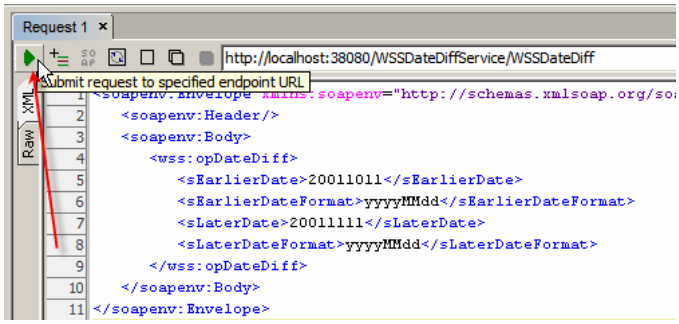


Figure 0-75 Submit the request

Observe the response.

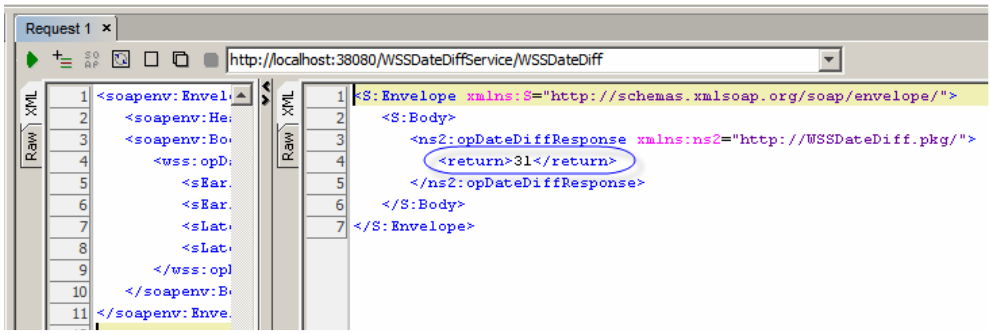


Figure 0-76 Service Response

There are a variety of ways in NetBeans to implement web services. What was shown here is just one way.

There are a variety of ways in NetBeans to create and execute web service tests. What was shown here are just two ways.

The service created in this section is a generic Date Difference web service. It can be used wherever a web service can be invoked and whenever date difference is required.

Looking at the WSDL observe that this is a document/literal web service; therefore it ought to be WS-Interoperability compliant. WS-I compliance is important when creating interoperable web services is important. With properly configured SoapUI Plugin one can check WS-I compliance right inside the NetBeans tool.

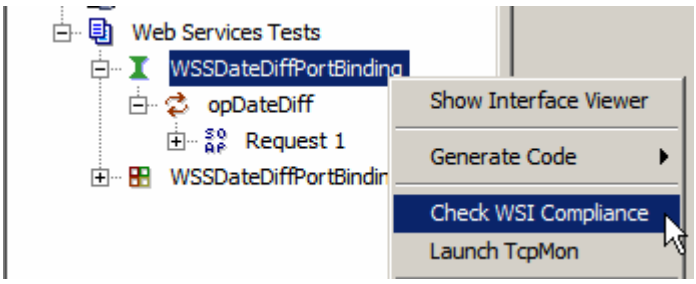


Figure 0-77 Triggering WS-I compliance checking

Date Conversion Utility

By default, HL7 uses the format yyyyMMddhhmmss for the TimeStamp datatype.

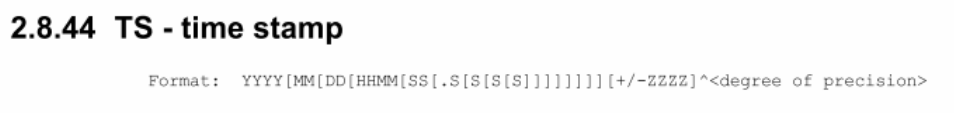


Figure 0-78 HL7 Version 2.3.1 TS (Timestamp) data type definition

Dates in our sample data are of that format.

Custom XML Schemas, which we will define for output from the HL7 Processor for the MDM and the IEP projects, use ISO 8601 date/time formats – xsd:dateTime data type. The format is yyyy-MM-ddThh:mm:ss.SSSS. See, for example, http://en.wikipedia.org/wiki/ISO_8601. Look for “Combined date/time representation”.

The Web Service we will develop in this section provides the generic Date/Time conversion service. To show another way of implementing an EJB-based web service we will use the “Interface First” method, that is we will first create the WSDL document then implement the service which conforms to it.

Note that we are again departing, for a time, from the Java Business Integration (JBI) word and are entering again into the Java Enterprise world. The two will meet and intersect later when we include the web service we are developing in the JBI Service Assembly of the HL7 Processor Composite Application.

Let’s create an Enterprise->EJB Module project called WSSConvertDate.

Our web service will accept an input Date/Time String, the format String denoting the format of the input Date/Time and the format String denoting the format of the Date/Time string we wish to be retuned by the service. As can be easily guessed, the web service will return a Data/Time String in the format requested.

It behoves us to create a web service which is interoperable as much as possible. To ensure maximum interoperability we should create a web service which is WS-Interoperability compliant. A WS-I compliant web service, amongst other requirements it must meet, must use a document/literal style. To use document/literal style the WSDL must use XSD Elements for both input, output and fault messages. XSD Elements come from XML Schema Documents.

Let’s create an XML Schema Document, ConvertDateMessages.xsd, to represent the request, the response and the fault messages. Let’s right-click on the name of the EJB Module project, choose New, choose Other, choose XML, choose XMS Schema.

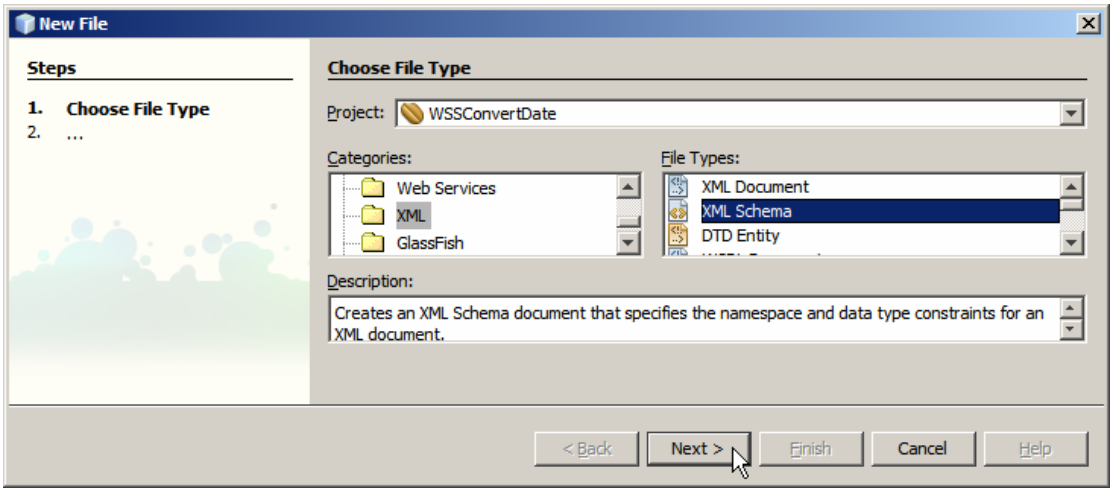


Figure 0-79 Trigger create new XML Schema functionality

Name the Schema ConvertDateMessages and click Finish.

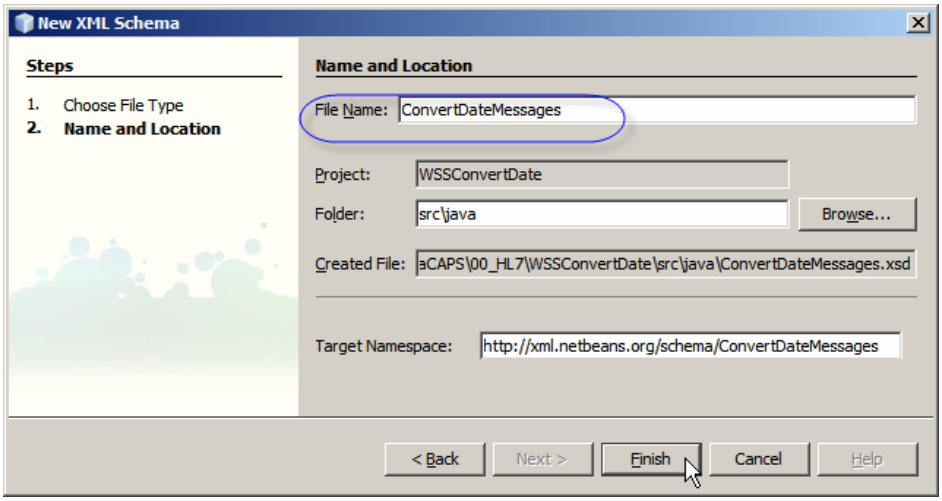


Figure 0-80 Name the schema

Right-click Elements and choose Add Element …

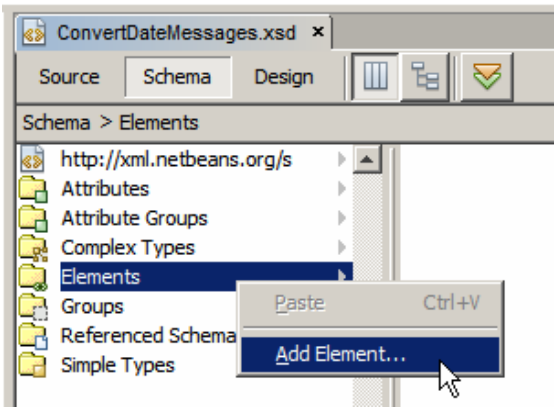


Figure 0-81 Trigger Add Element

Name the Element ConvertDateRequest and click OK to accept other values at defaults.

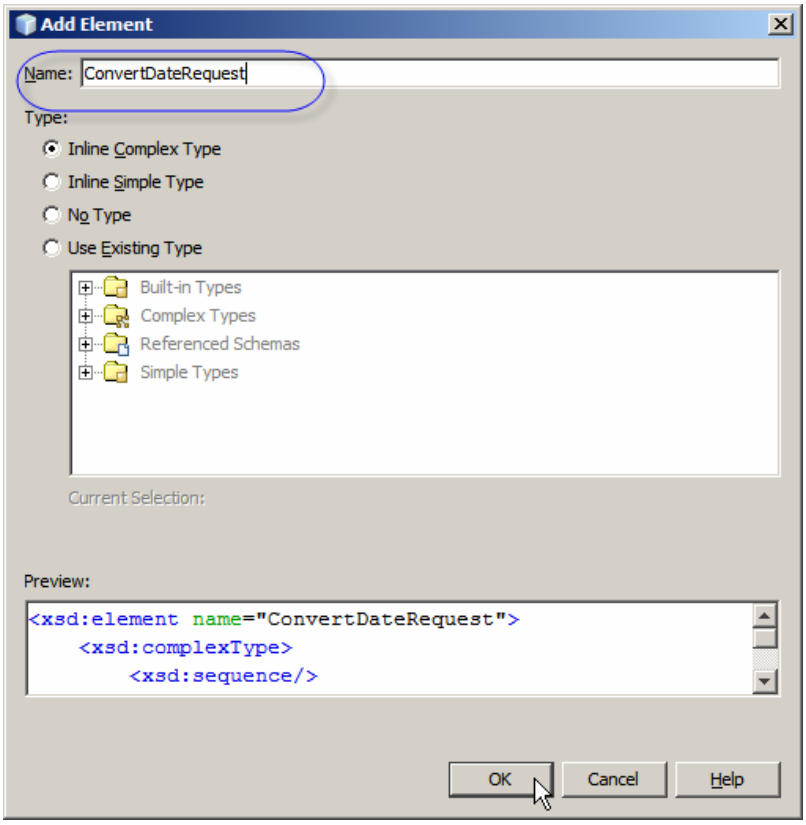


Figure 0-82 Name the element

Click on the ConvertdateRequest, then complexType, then right-click on sequence and choose Add Element. This will add a new element to the ConvertDateRequest complex type.

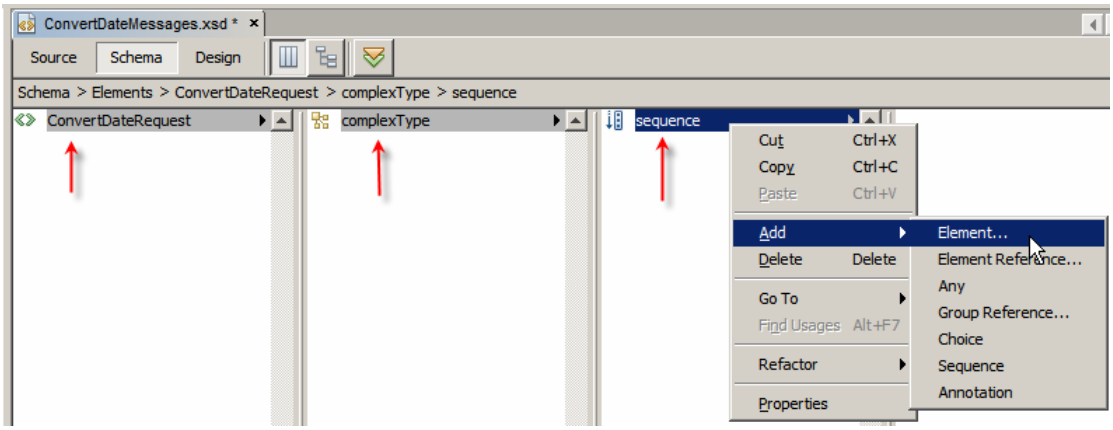


Figure 0-83 Add element to the complex type

Name the Element sDateTimeIn, choose Use Existing Type, choose Built-in Type, choose string and click OK.

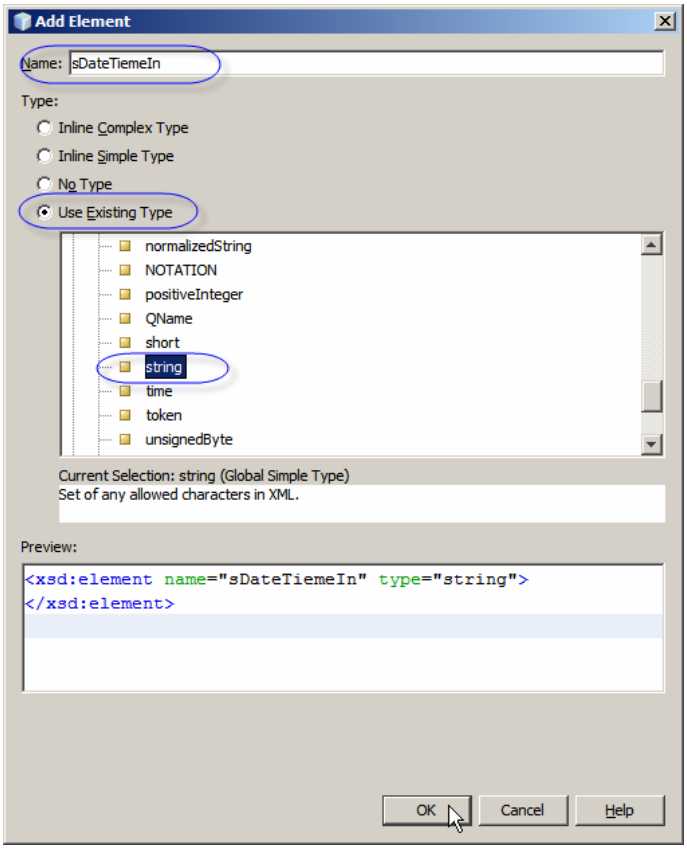


Figure 0-84 Add a leaf element

Using the same method add two more string elements, sDateTimeInFormat and sDateTimeOutFormat.

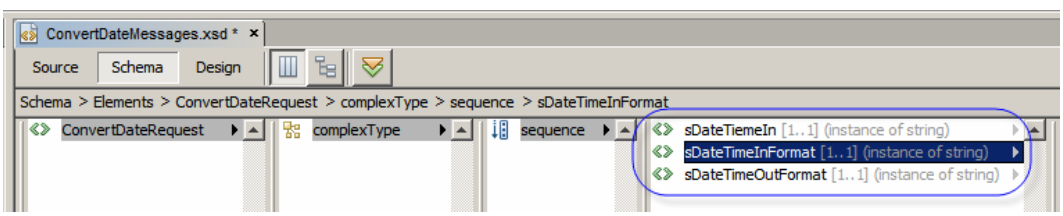


Figure 0-85 ConvertDatyeRequest defined

Scroll back to the left, right click on the Elements node and add an element named ConvertDateResponse at the same level as the ConvertDateRequest. Leave it as Inline Complex Type (default).

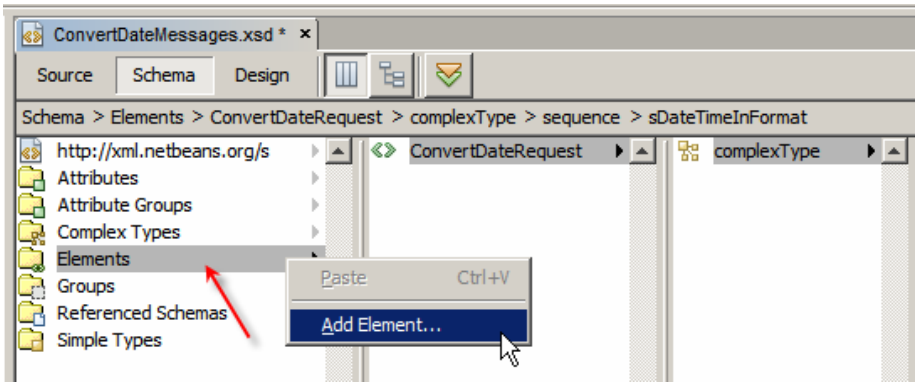


Figure 0-86 Add an element at the root level

Expand through to sequence, right click on sequence and choose Add Element.

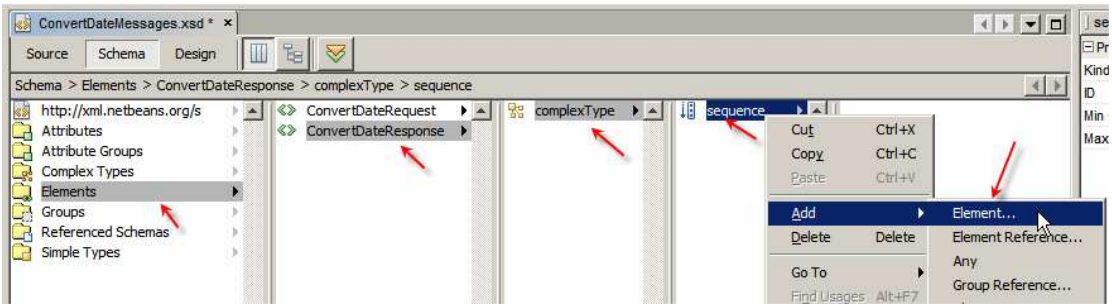


Figure 0-87 Add element to the complex type

Name the new element sDateTimeOut and make it a built-in type of string.

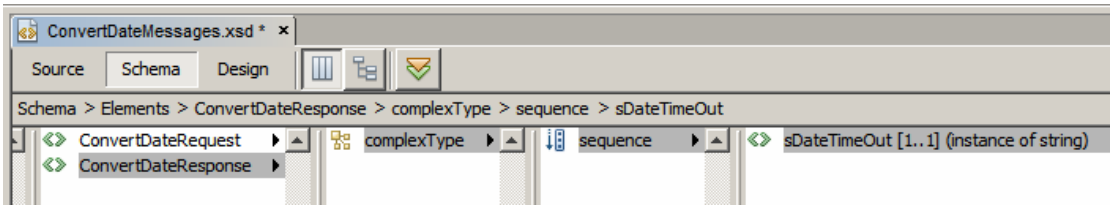


Figure 0-88 Add new element to the complex type

Let's scroll back to the left and add another root Element, ConvertDateFault, containing a built-in string type Element sFault, much as we have done for the ConvertDateResponse above.

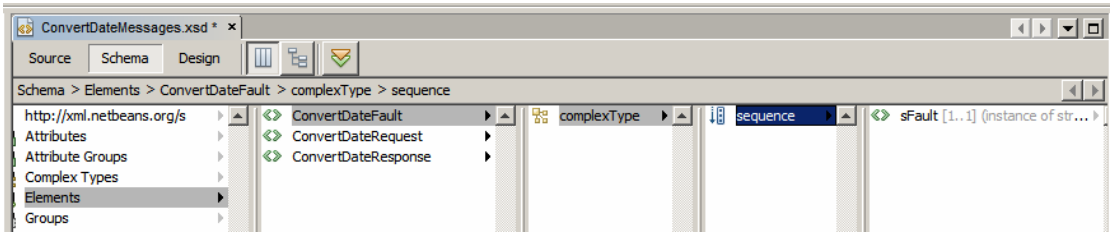


Figure 0-89 All elements configured

Save the new XML Schema Document, switch to Source mode and view the complete XSD. It should look like that shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://xml.netbeans.org/schema/ConvertDateMessages"
            xmlns:tns="http://xml.netbeans.org/schema/ConvertDateMessages"
            elementFormDefault="qualified">
    <xsd:element name="ConvertDateRequest">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="sDateTimeIn"
                    type="xsd:string"></xsd:element>
                <xsd:element name="sDateTimeInFormat"
                    type="xsd:string"></xsd:element>
                <xsd:element name="sDateTimeOutFormat"
                    type="xsd:string"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="ConvertDateResponse">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="sDateTimeOut" type="xsd:string"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="ConvertDateFault">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="sFault" type="xsd:string"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

Here is what it looks like in the Design view.

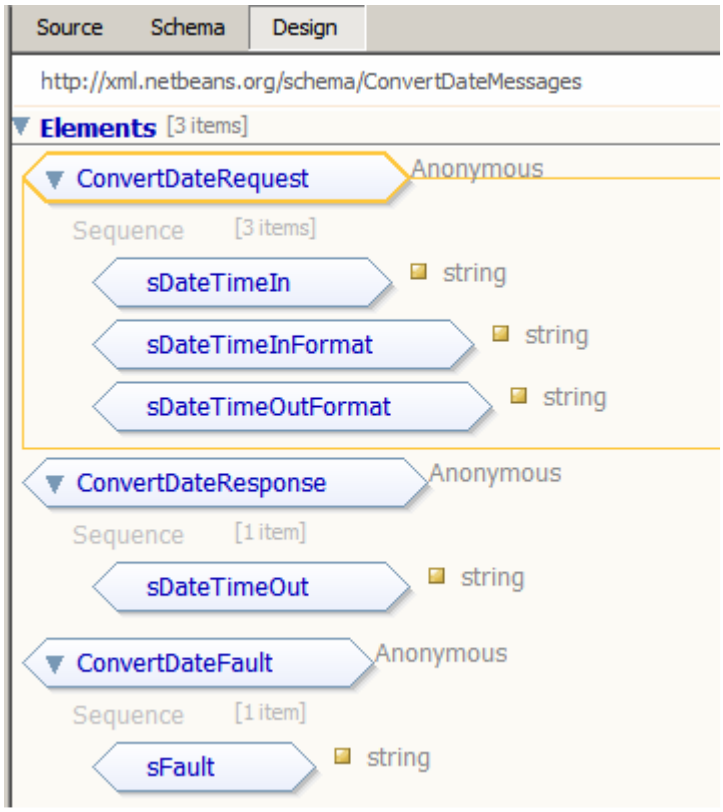


Figure 0-90 XML Schema in design view

Now that we have the XML Schema document to represent our web service request, response and Fault messages we are in a position to define the WSDL that uses these messages. Right-click on the name of the EJB Module, choose New, choose WSDL Document, name the WSDL document WSSConvertDate, make it complex, SOAP, document/literal service WSDL.

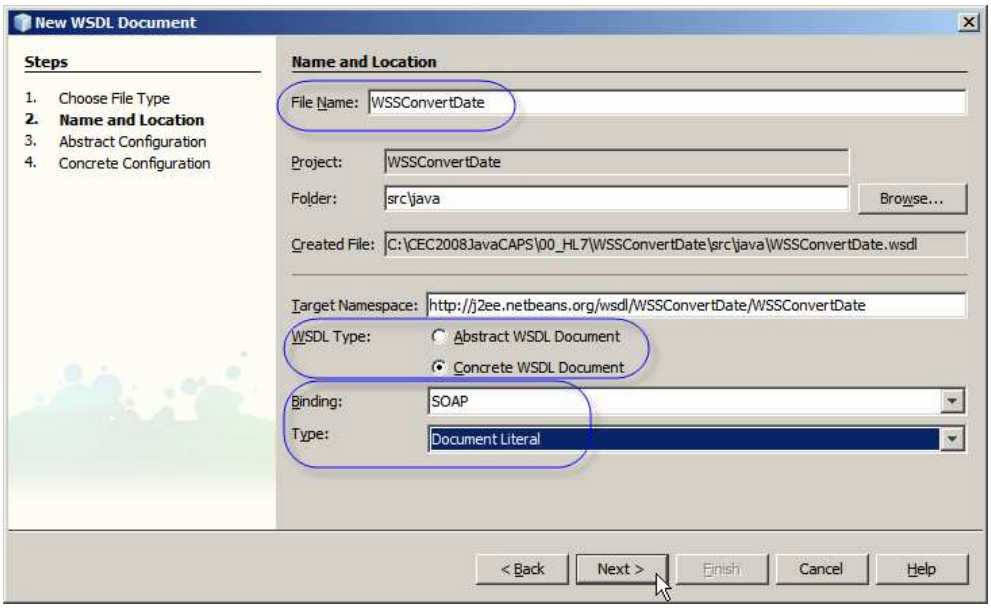


Figure 0-91 Trigger new WSDL functionality

Change the Operation Name to opConvertDate, change Input Message Part Name to msgRequest, click on the ellipsis button, choose the ConvertDateRequest element and click OK.

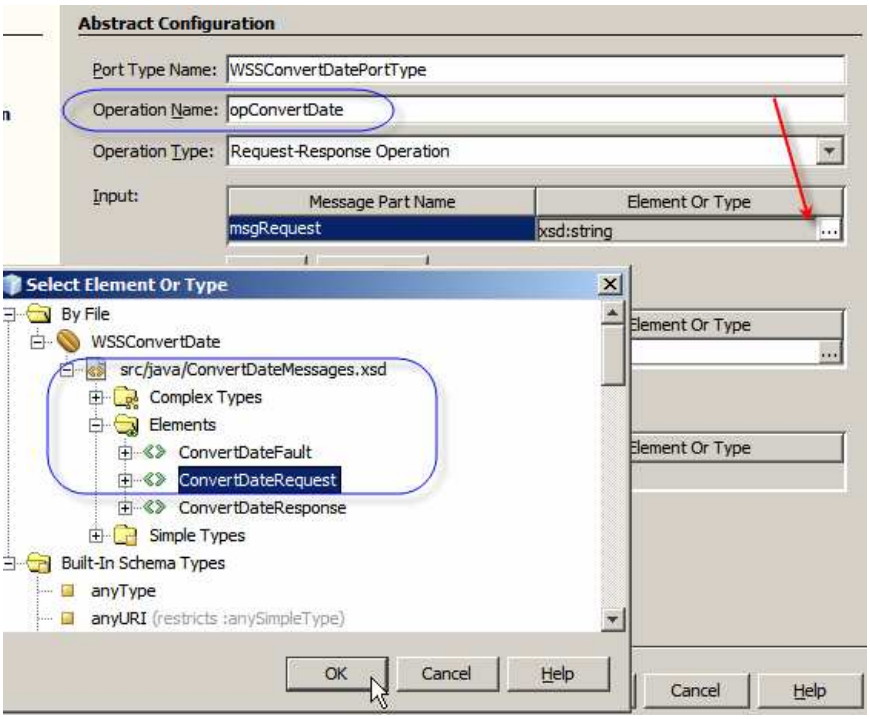


Figure 0-92 Configure operation name and the Input message part

Change the name of the Output Message Part Name to msgResponse, click the ellipsis button, choose the ConvertdateResponse element and click OK.

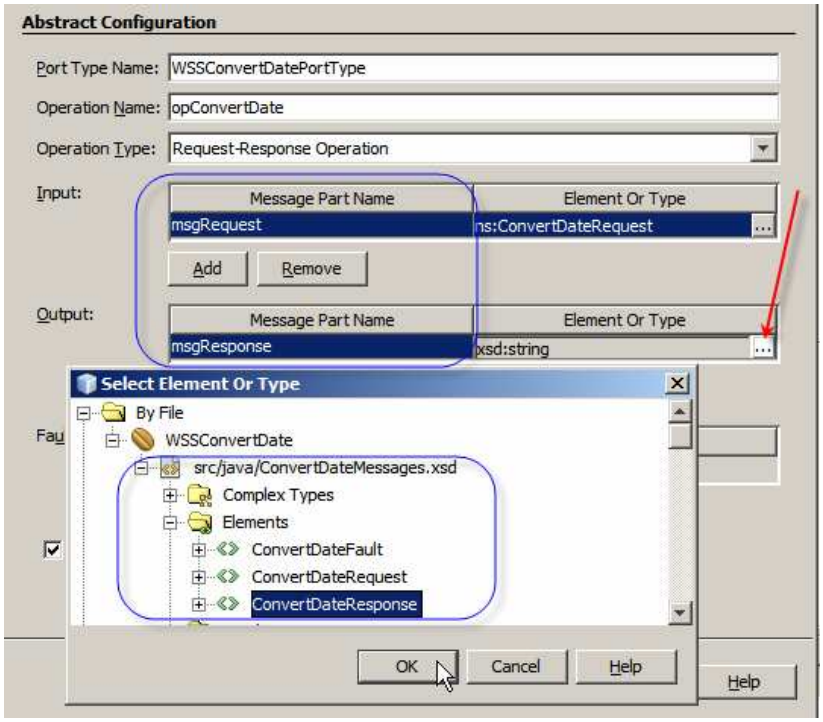


Figure 0-93Change name and type of the Output Message Part

Add a Fault element, sFault, of type ConvertdateFault.

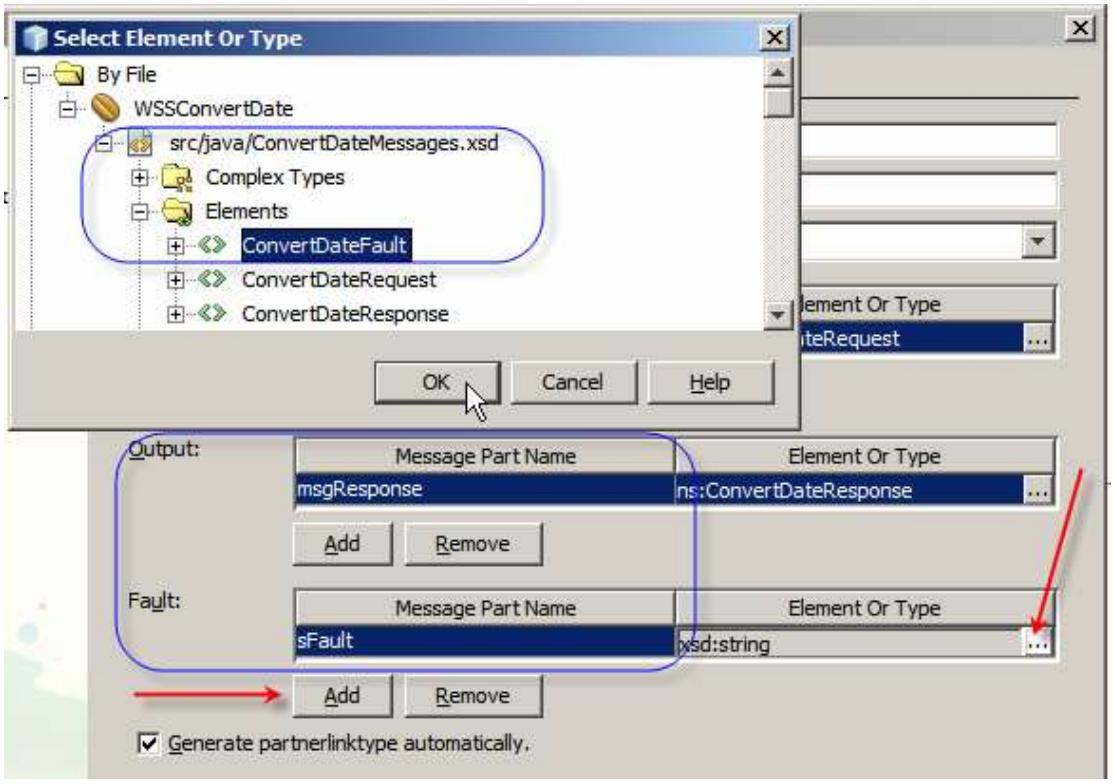


Figure 0-94 Add xsd:string Fault element

With all parts named and typed as required click Next. Leave the default values in the next dialogue box, unless you desire to change them, and click Finish.

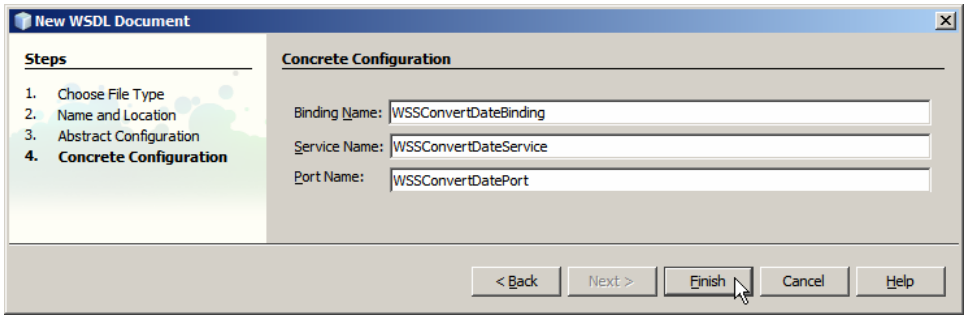


Figure 0-95 Complete the dialogue

Open the WSDL, expand the Services node tree all the way to the soap:address node, right-click on the soap:address node and choose Properties.

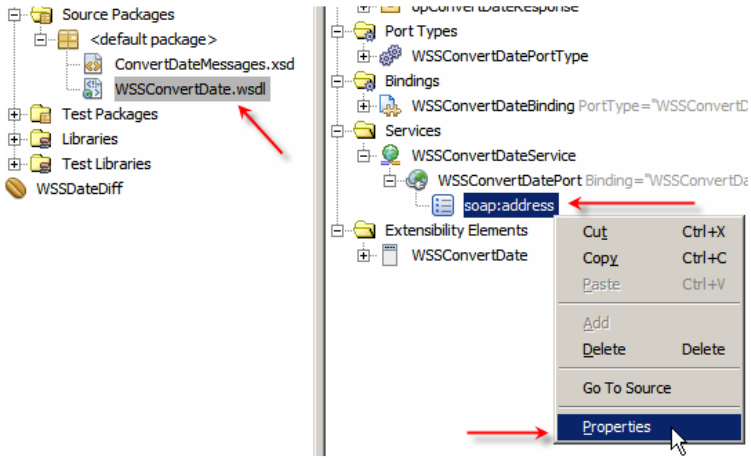


Figure 0-96 Choose soap:address properties

Replace the string ${HttpDefaultPort} with the port number of your web container – by default it will be 8080. For me it is 38080.

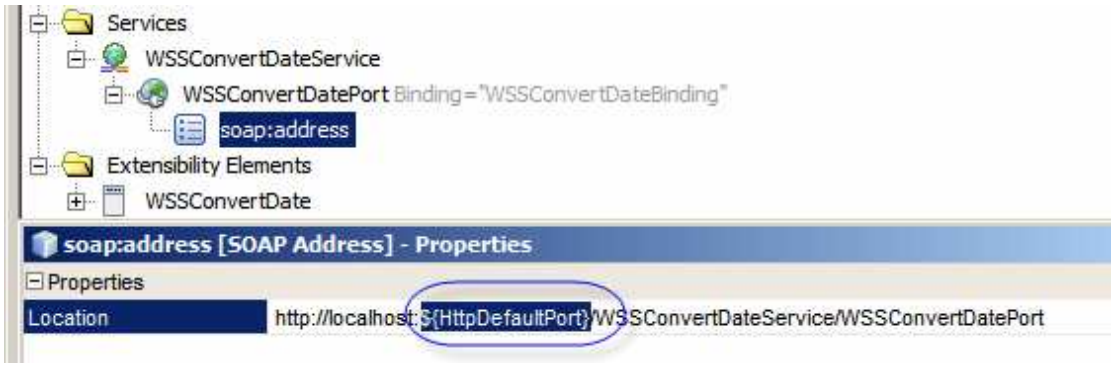


Figure 0-97 Fix the port number in the endpoint address

Now that we have the WSDL that describes our interface let's right-click on the name of the EJB Module, choose New, choose Other, choose Web Services, choose Web Service from WSDL …

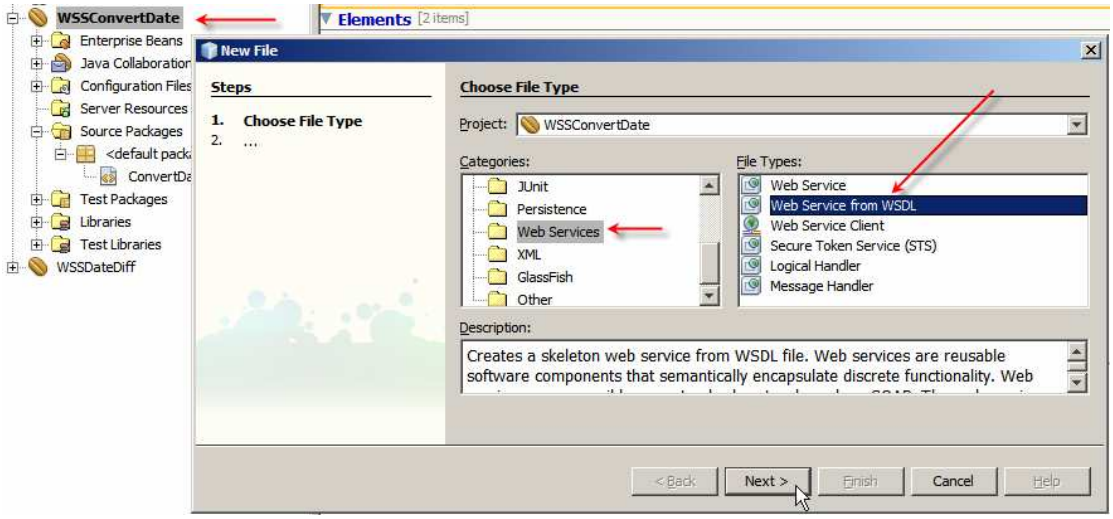


Figure 0-98 Trigger Create Web Service from WSDL functionality

Name the web service WSSConvertDate, give it the package name of pkg.WSSConvertDate, navigate the file system to the location of the WSDL we just created (which will be in the src/java directory of the project), choose the WSSConvertDate.wsdl WSDL and click Open.

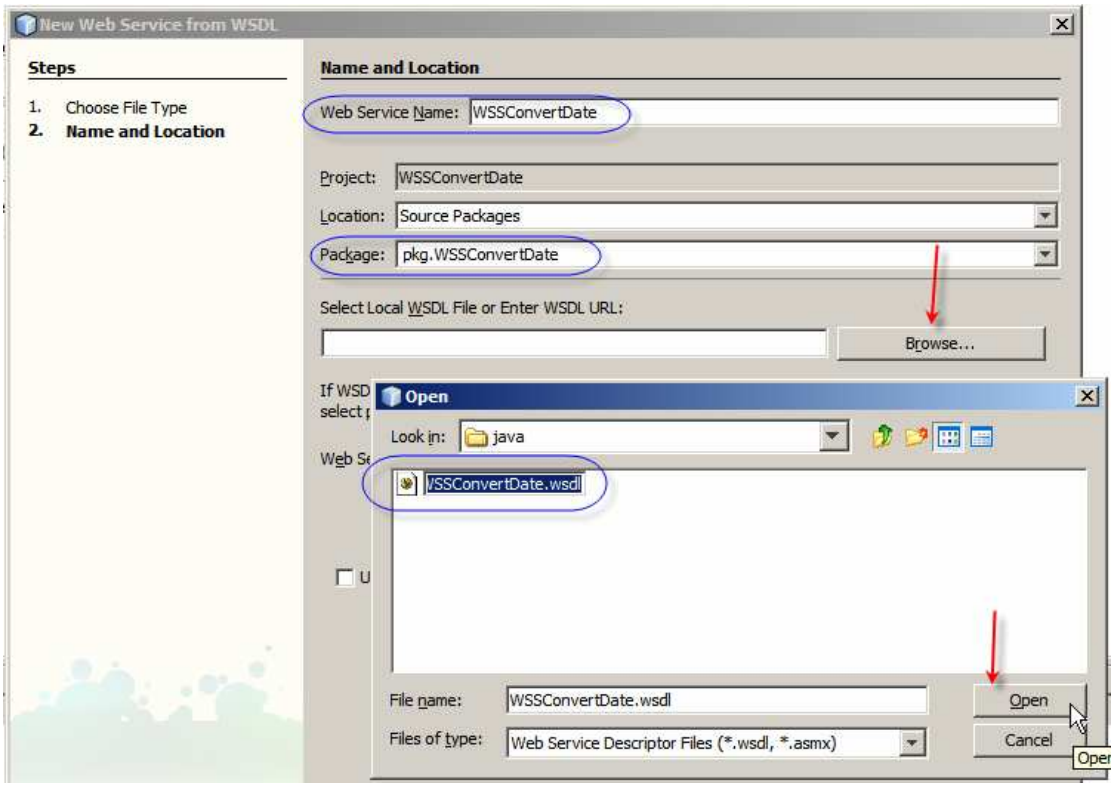


Figure 0-99 Name the service and choose the WSDL

While you were looking for the WSDL file the dialogue box had an error message to the effect that “there is no port in the WSDL file”. Depending on the speed of your machine and other factors there may be a noticeable delay as the WSDL file is being parsed before the error message disappears. Be patient and you shall be rewarded. When the error disappears click the Finish button.

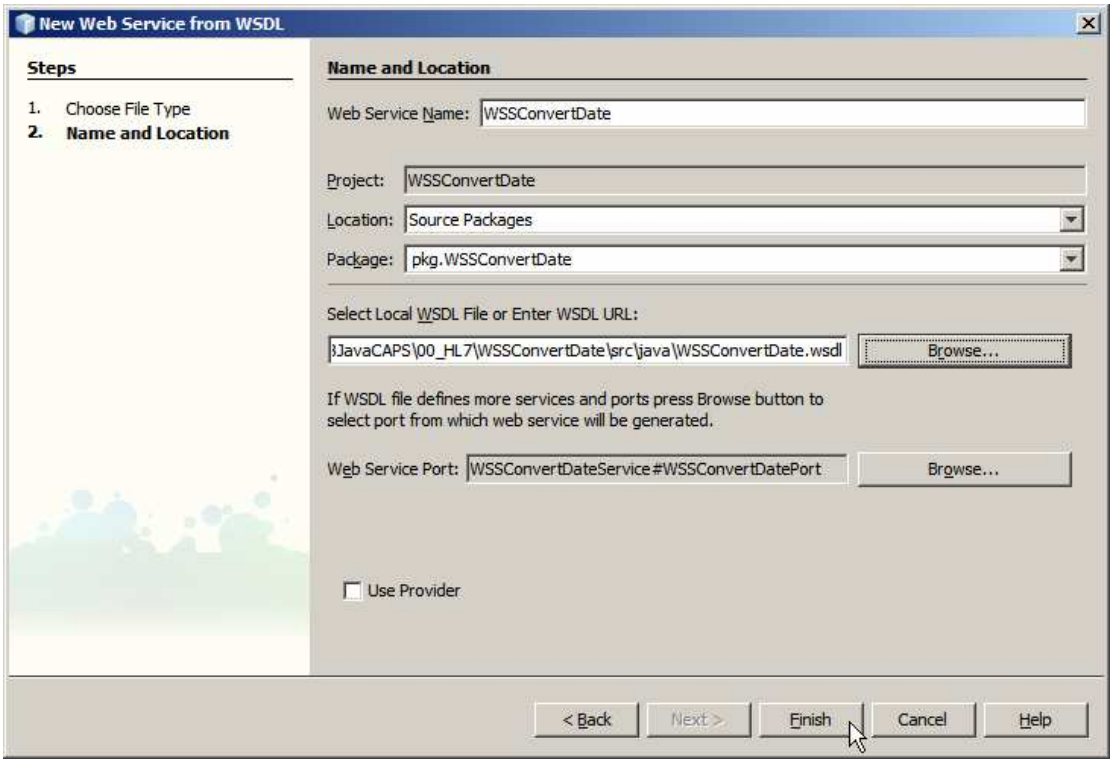


Figure 0-100 Complete the dialogue

The WSDL is parsed, new nodes appear in the node tree and the web service skeleton implementation appears in the web service editor window in the Design mode.

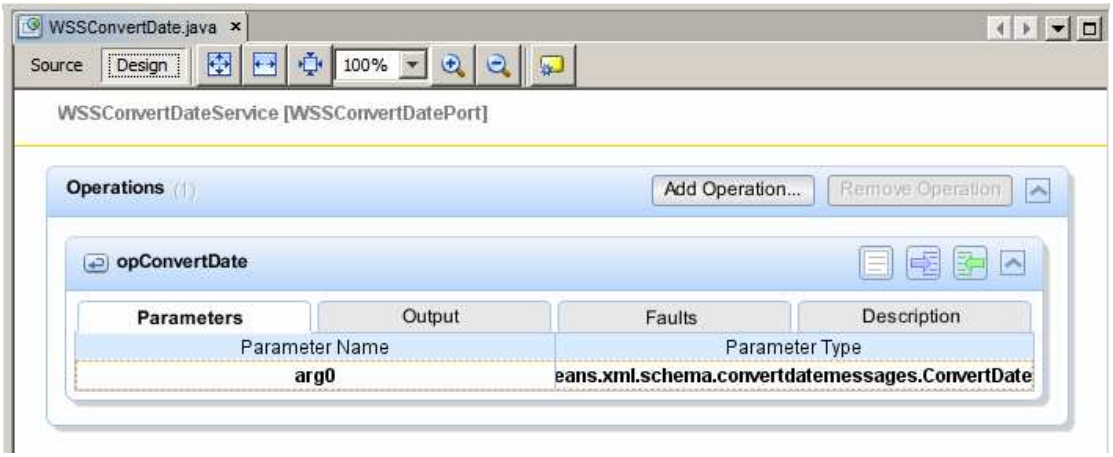


Figure 0-101 Web Service skeleton in Design view

Note the operation name and the data type of the input message as what we specified when creating the WSDL.

Here is the source code of the skeleton implementation reformatted to fit into the margins.

```
6   package pkg.WSSConvertDate;
7
8   import javax.ejb.Stateless;
9   import javax.jws.WebService;
10  import org.netbeans.j2ee.wsdl.wssconvertdate.wssconvertdate.OpConvertDateFault;
11  import org.netbeans.j2ee.wsdl.wssconvertdate.wssconvertdate.WSSConvertDatePortType;
12
13  /**
14   *
15   * @author mczapski
16   */
17  @WebService
18      (serviceName = "WSSConvertDateService"
19      , portName = "WSSConvertDatePort"
20      , endpointInterface =
21          "org.netbeans.j2ee.wsdl.wssconvertdate.wssconvertdate.WSSConvertDatePortType"
22      , targetNamespace =
23          "http://j2ee.netbeans.org/wsdl/WSSConvertDate/WSSConvertDate"
24      , wsdlLocation = "META-INF/wsdl/WSSConvertDate/WSSConvertDate.wsdl")
25  @Stateless
26  public class WSSConvertDate implements WSSConvertDatePortType {
27
28      public
29          org.netbeans.xml.schema.convertdatemessages.ConvertDateResponse
30              opConvertDate
31                  (org.netbeans.xml.schema.convertdatemessages.ConvertDateRequest msgRequest)
32                      throws OpConvertDateFault {
33          //TODO implement this method
34          throw new UnsupportedOperationException("Not implemented yet.");
35      }
36
37  }
```

Figure 0-102 Reformatted skeleton code

Let's add the following just after the

```
public class WSSConvertDate
```

statement to get

```
public class WSSConvertDate
    static Object oBj = new Object();
```

Let's delete the lines with `//TODO` and `throw new Unsupported`... (31 and 32 in the original source) and add the following in their place:

```
java.text.DateFormat fmtIn
    = new java.text.SimpleDateFormat(msgRequest.getSDateTimeInFormat());
java.text.DateFormat fmtOut
    = new java.text.SimpleDateFormat(msgRequest.getSDateTimeOutFormat());

org.netbeans.xml.schema.convertdatemessages.ConvertDateResponse sDT
    = new org.netbeans.xml.schema.convertdatemessages.ConvertDateResponse();
java.util.Date dt = null;

synchronized(oBj) {
    try {
        dt = fmtIn.parse(msgRequest.getSDateTimeIn());
        sDT.setSDateTimeOut(fmtOut.format(dt));
    } catch (java.text.ParseException ex) {
        ConvertDateFault flt = new ConvertDateFault();
        flt.setSFault(ex.getMessage());
        throw new OpConvertDateFault
                ("Fault converting from "
                    + msgRequest.getSDateTimeIn() + " of format "
                    + msgRequest.getSDateTimeInFormat() + " to format "
                    + msgRequest.getSDateTimeOutFormat()
                 ,flt, ex);
    }
}
return sDT;
```

The important part of the code block looks like this:

```
@WebService
    (serviceName = "WSSConvertDateService"
    , portName = "WSSConvertDatePort"
    , endpointInterface =
        "org.netbeans.j2ee.wsdl.wssconvertdate.wssconvertdate.WSSConvertDatePortType"
    , targetNamespace =
        "http://j2ee.netbeans.org/wsdl/WSSConvertDate/WSSConvertDate"
    , wsdlLocation = "META-INF/wsdl/WSSConvertDate/WSSConvertDate.wsdl")
@Stateless
public class WSSConvertDate implements WSSConvertDatePortType {
    static Object oBj = new Object();
    public
        org.netbeans.xml.schema.convertdatemessages.ConvertDateResponse
            opConvertDate
                (org.netbeans.xml.schema.convertdatemessages.ConvertDateRequest msgRequest)
                    throws OpConvertDateFault {

        java.text.DateFormat fmtIn
            = new java.text.SimpleDateFormat(msgRequest.getSDateTimeInFormat());
        java.text.DateFormat fmtOut
            = new java.text.SimpleDateFormat(msgRequest.getSDateTimeOutFormat());

        org.netbeans.xml.schema.convertdatemessages.ConvertDateResponse sDT
            = new org.netbeans.xml.schema.convertdatemessages.ConvertDateResponse();
        java.util.Date dt = null;

        synchronized(oBj) {
            try {
                dt = fmtIn.parse(msgRequest.getSDateTimeIn());
                sDT.setSDateTimeOut(fmtOut.format(dt));
            } catch (java.text.ParseException ex) {
                ConvertDateFault flt = new ConvertDateFault();
                flt.setSFault(ex.getMessage());
                throw new OpConvertDateFault
                        ("Fault converting from "
                            + msgRequest.getSDateTimeIn() + " of format "
                            + msgRequest.getSDateTimeInFormat() + " to format "
                            + msgRequest.getSDateTimeOutFormat()
                         ,flt, ex);
            }
        }
        return sDT;
    }
}
```

Figure 0-103 Completed code block

Let's build and deploy this project.

The Web Service uses a complex message as input. To test the service we need to use the SoapUI Plugin rather then the simple "Test Web Service" option. The simple Test Web Service option will create a Web Service client and will display the JSP in a web browser but there will be a single data entry box for the complex message rather then 3 data entry boxes for each of the fields. We would need to provide an entire properly formatted XML Instance Document, which would get butchered by the browser anyway. It is too much trouble so we will use the SoapUI plugin functionality instead.

Right-click on the name of the web service in the Web Services node under the web service project name and choose Create Web Service Tests.

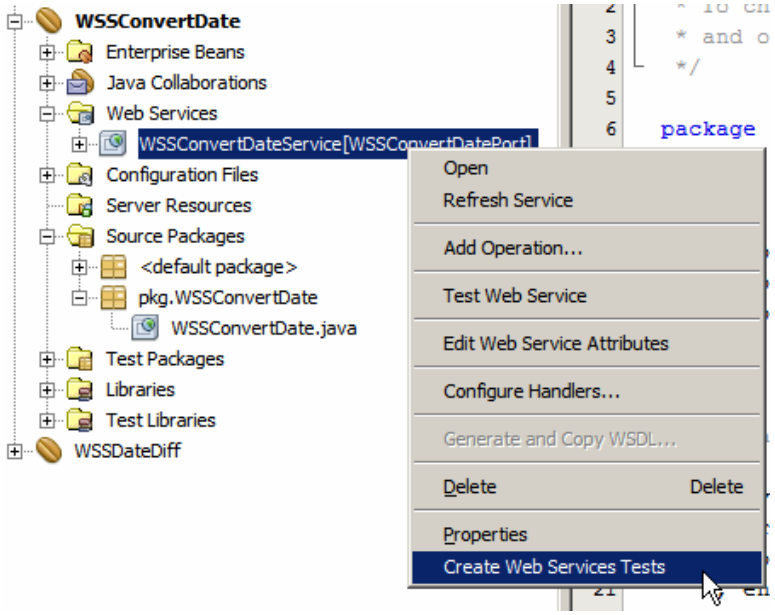


Figure 0-104 Create web service testing project for testing the service

Accept defaults and click OK.

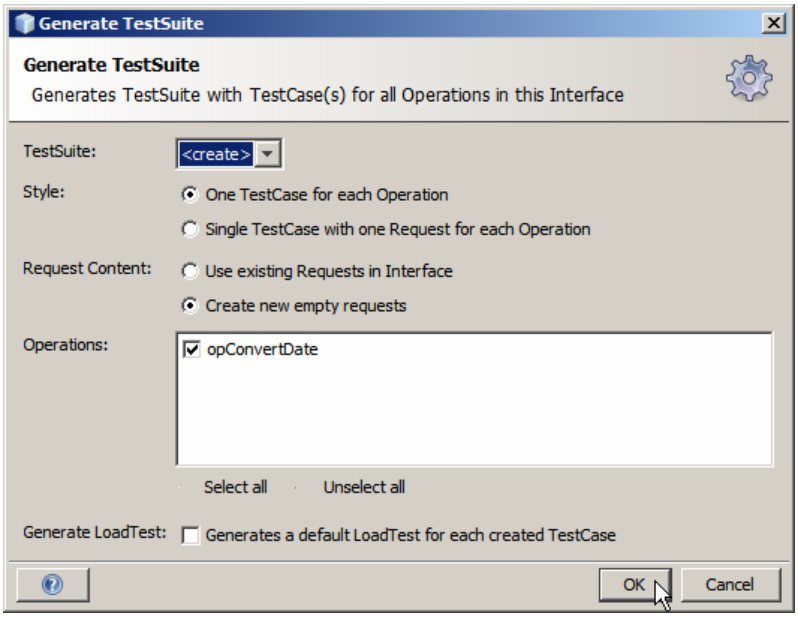


Figure 0-105 Accept defaults

Accept default name and click OK.

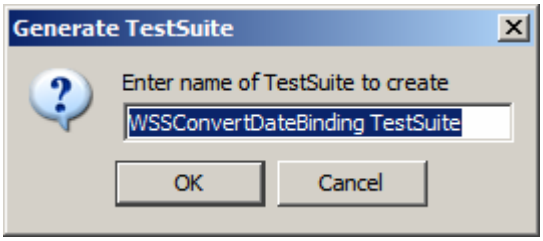


Figure 0-106 Accept name

Expand nodes under the Web Services Tests node all the way to Request 1. Double-click Request 1 to open it in the request editor and modify to provide appropriate data.

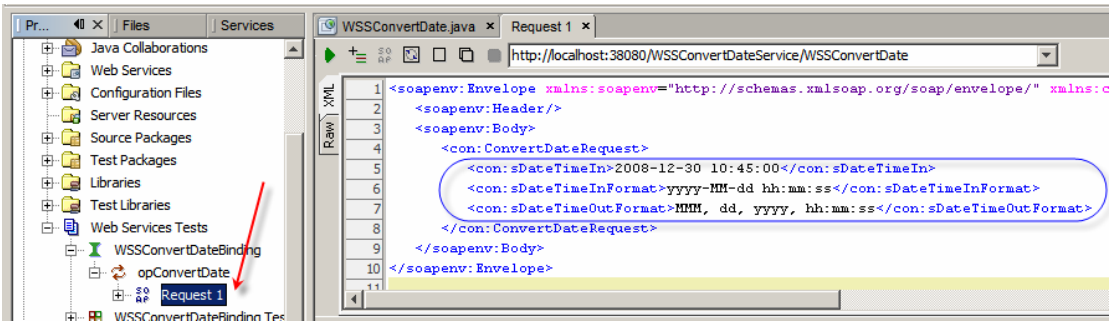


Figure 0-107 Prepare request message

Submit request.

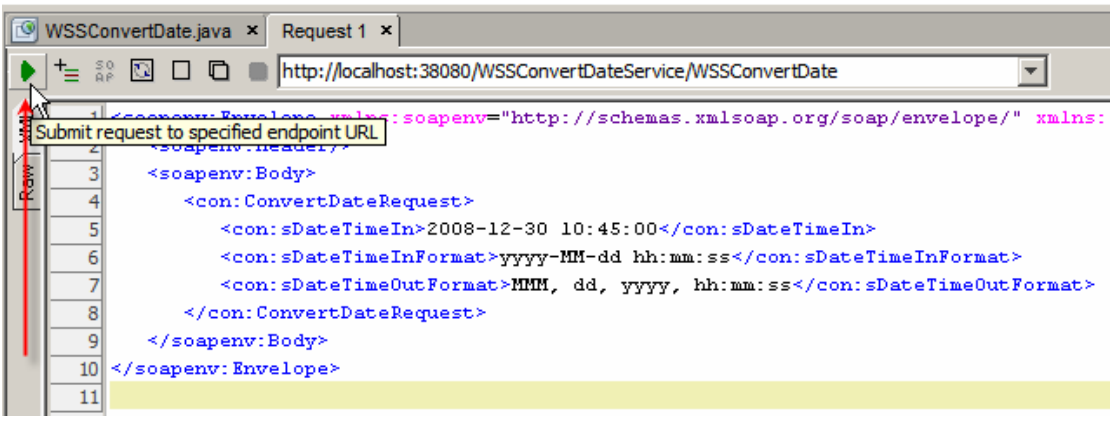


Figure 0-108 Submit request

Observe result.

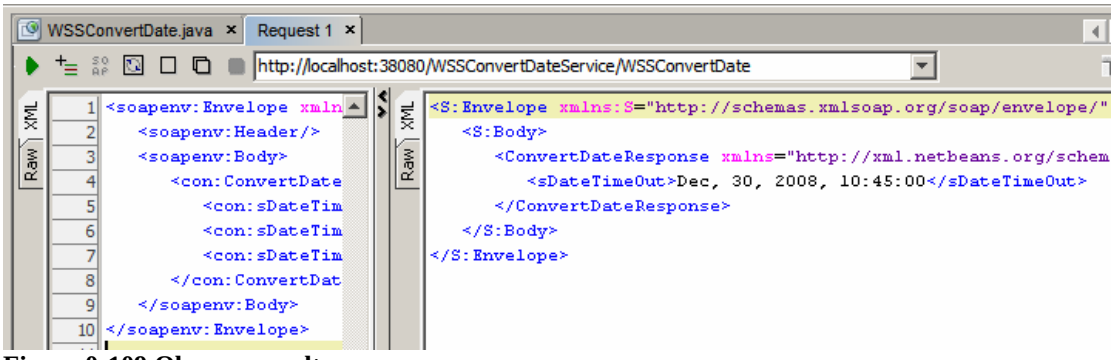


Figure 0-109 Observe result

Break the request by adding some non-numeric characters to the sDateTimeIn value so the date/time in and the format in don't agree. Submit the request and observe the SOAP Fault response.

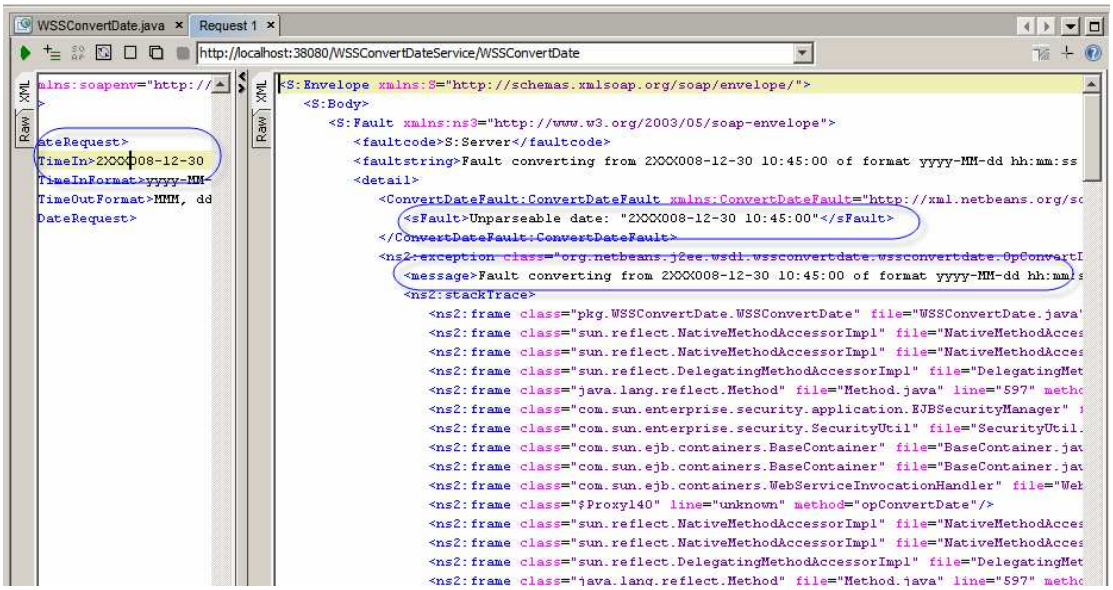


Figure 0-110 SOAP Fault with feedback on invalid input date

The service created in this section is a generic Date/Time Format Conversion web service. It can be used wherever a web service can be invoked and whenever date/time conversion is required.

HL7 Processor

Finally, we are getting around to developing the HL7 Processor project, which is what we set off to do at the beginning. Let's reproduce the overall solution schematic and expand it a bit to show what is supposed to be built.

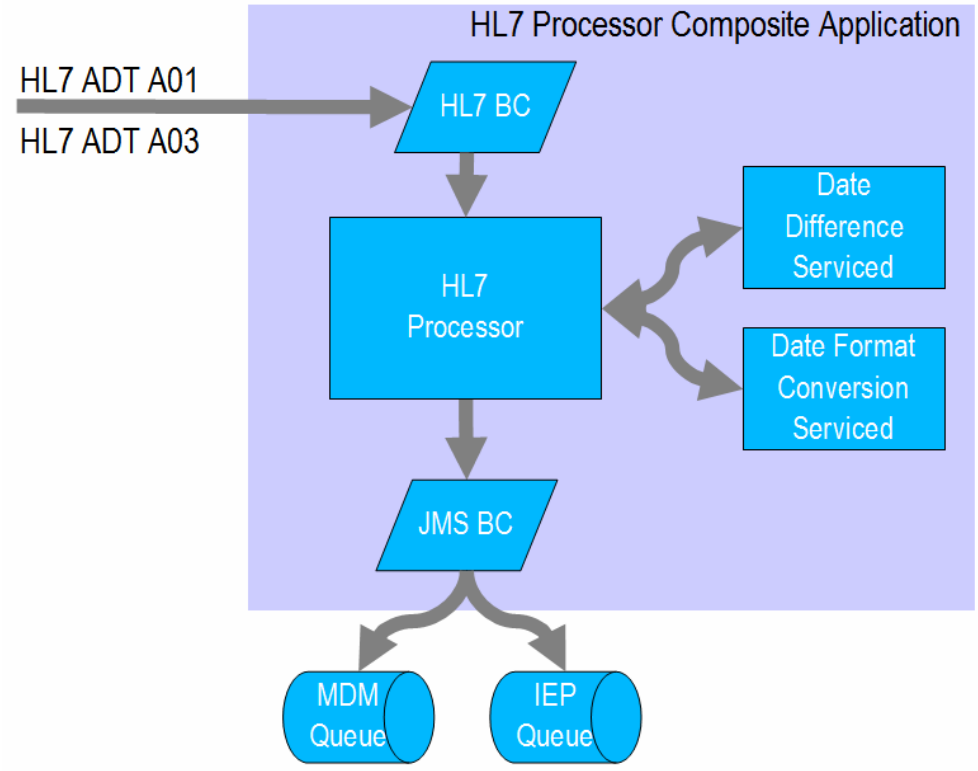


Figure 0-111 HL7 Processor solution schematic

In words, a BPEL 2.0 Business Process will receive streams of messages – A01s and A03s, from the HL7 BC. It will map a subset of the content of A01 messages to a Custom Patient XML structure, converting Date of Birth to the ISO 8601 format, and send the Custom Patient messages to a JMS Queue using the JMS BC. It will map a subset of the content of A03 messages to a Custom Discharge XML structure, converting Admission Date and Discharge Date to ISO 8601 format and calculating the difference in days between the two, then, it will send the Custom Discharge messages to a JSM Queue using the JMS BC.

Let's create a new CAPS->ESB->BPEL Module project HL7Processor.

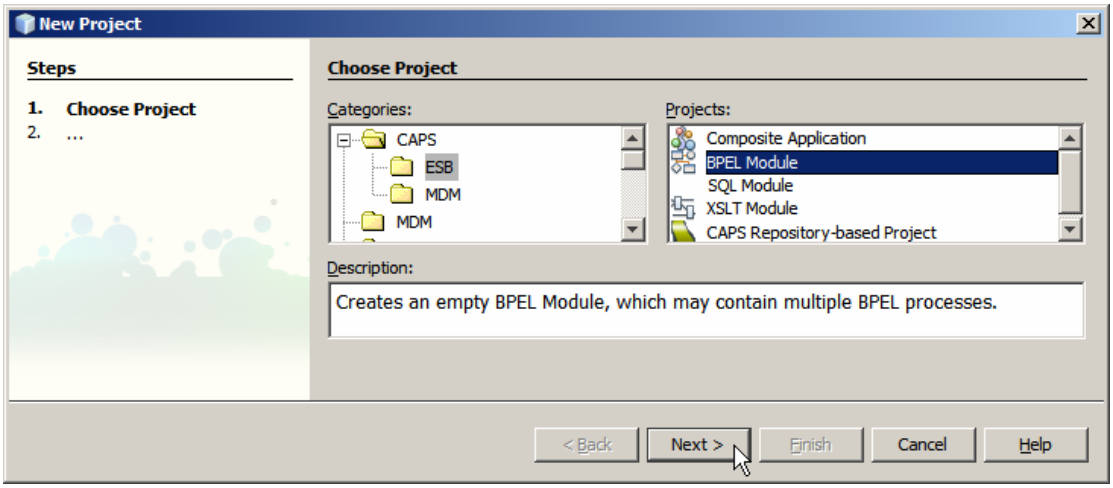


Figure 0-112 Create a new BPEL Module project

We are receiving a stream of HL7 delimited messages from the HL7 Feeder. These delimited messages must be converted to their XML equivalents in order to allow BPEL 2.0 process access to their individual field values. All conversion logic will be handled by the HL7 Encoder directly in the HL7 BC so no external logic, like Java or BPEL, will be required.

As implemented, the HL7 BC expects a stream of messages defined by the XML Schemas, one for the ADT A01 and one for the ADT A03 message type. This has been discussed and illustrated in section "HL7 Consumer and XML Converter". The following discussion is essentially a reproduction of what was already said there so both the subject and the method should be familiar.

Expand the Project Files folder and create a subfolder called HL7v2 using the New->Other->Other->Folder options.

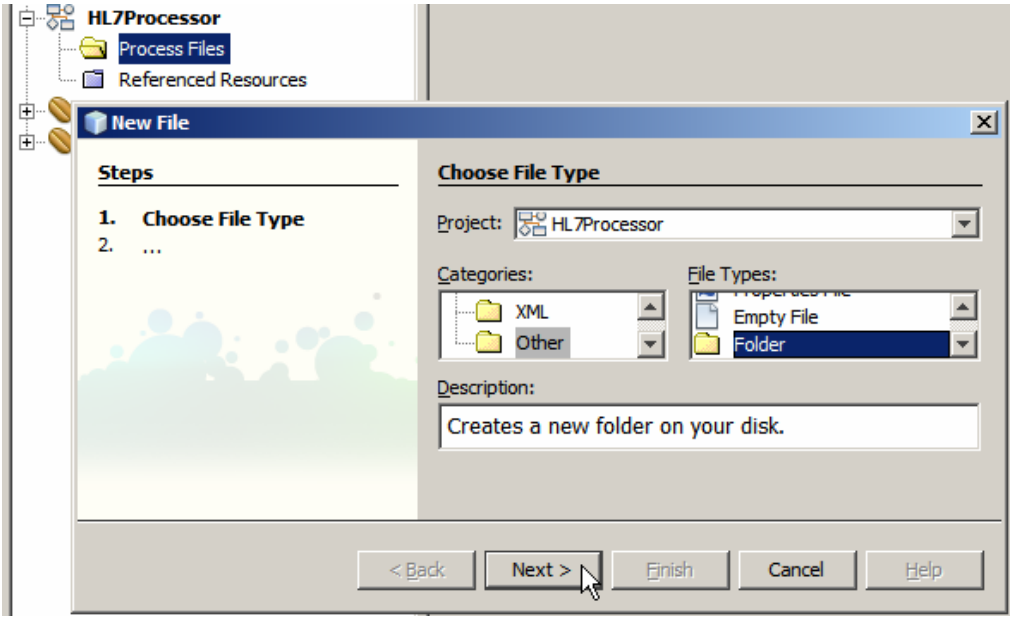


Figure 0-113 Create a folder inside the Process Files folder

Name the new folder HL7v2.

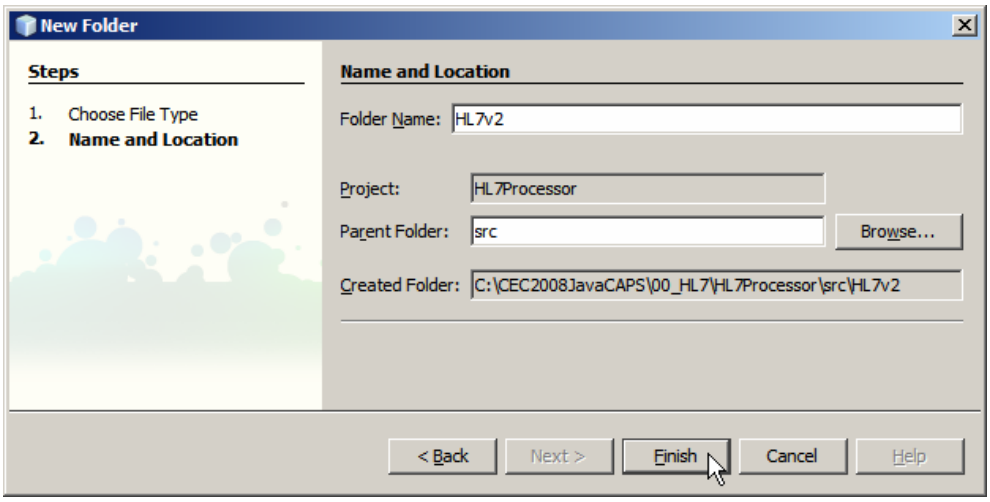


Figure 0-114 Name the new folder

Expand the Process Files folder, right-click on the HL7v2 folder and choose New->Other->XML->External XML Schema Documents.

Navigate to the hl7v2/2.3.1 folder containing the HL7 v2.3.1 XML Schema Documents which were extracted from the ZIP archive downloaded earlier, and select the following XSD documents:

- ADT_A01.xsd
- ADT_A03.xsd
- batch.xsd
- datatypes.xsd
- fields.xsd
- messages.xsd
- segments.xsd

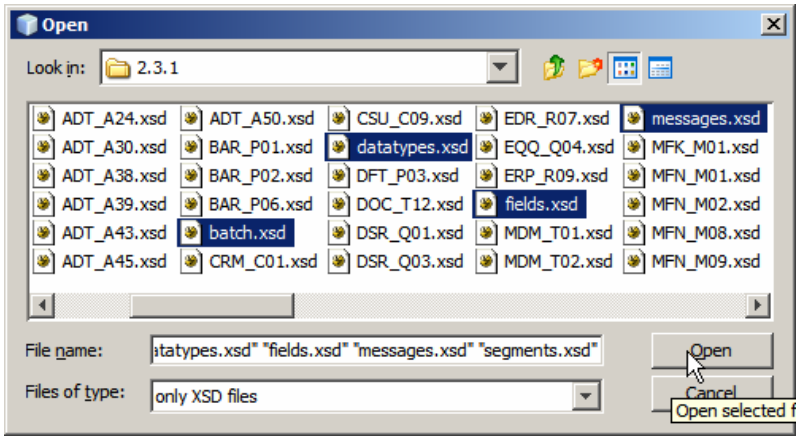


Figure 0-115 Select specific HL7 v2.3.1 XSD files

Click Finish to import all files.

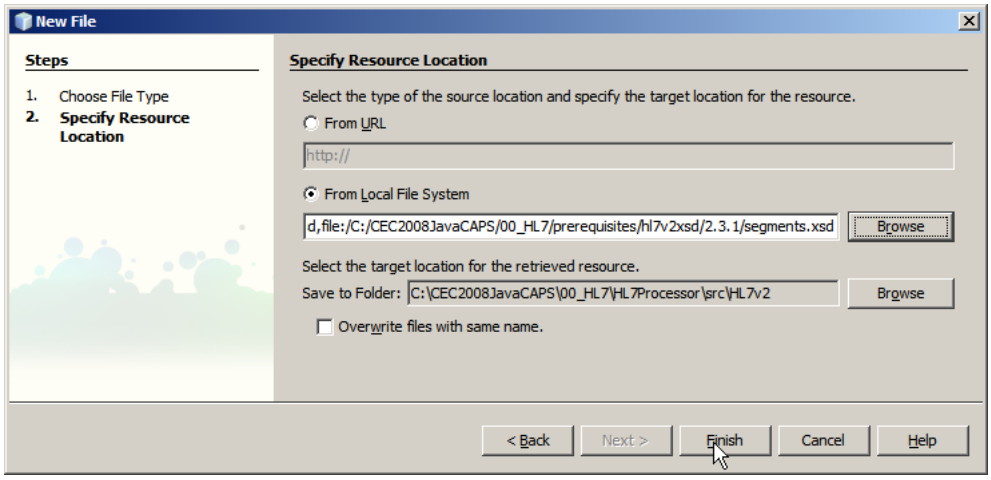


Figure 0-116 Start the import process

Once the import is finished a 7 new files will appear in the HL7v2 folder.

Note that we only imported the ADT A01 (Admission) and ADT A03 (Discharge) XSD files and subsidiary files referenced from them. This is because we will only deal with AT A01 and ADT A03 messages in this project.

Expand to the Project Files folder in the folder tree under the HL7Processor project and choose New->WSDL Document … context menu option. The WSDL we will be creating is a placeholder for the HL7 Binding Component properties, as will be seen shortly. A WSDL Wizard allows us to choose the Binding Component which to use, then to configure properties that are specific to this Binding Component.

We will listen for incoming HL7 Version 2 messages. As the HL7 BC receives the messages it will ‘decode’ them from delimited to XML format and will provide each message to the Normalized Message Router for routing to whatever component is interested in processing them, as defined through the Composite Application Service Assembly.

Configure the WSDL / HL7 Binding Component as follows:

- File Name: HL7Processor_A01_A03Delim_HL7In

- WSDL Type: Concrete WSDL Document
- Binding: HL7
- Type: HL7 Version 2 – Inbound Request

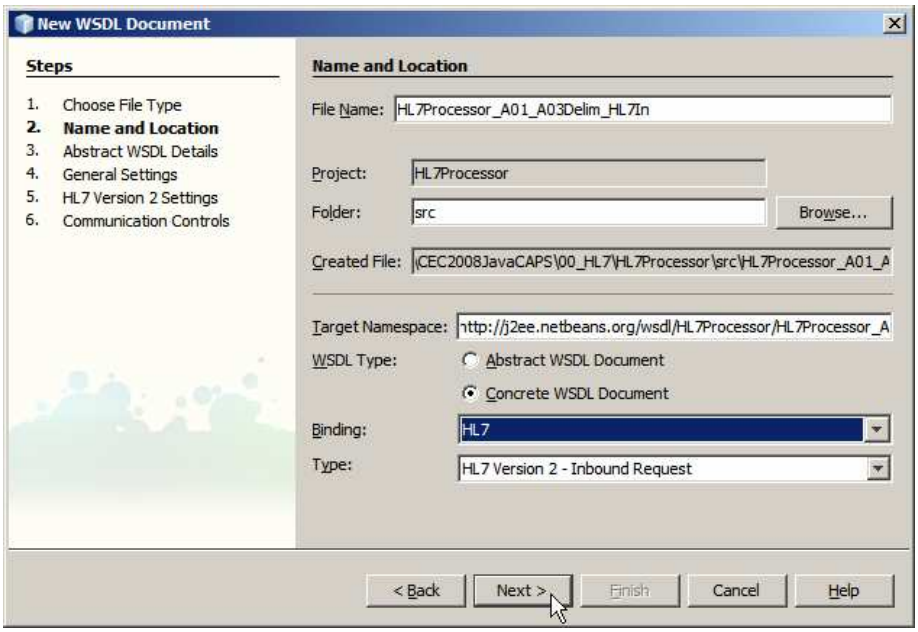


Figure 0-117 Initial HL7 Inbound configuration dialogue

Click Next to get to the next Wizard dialogue box.

This is where things get more interesting. The HL7 BC must be told what kind of messages it is dealing with. The only way at present to tell it is to choose the XML Schema Document that represents the HL7 trigger event and message type. This is why we imported all these XSDs earlier.

Click the Browse button along the Request Message data entry field, navigate to the ADT_A01.xsd, choose the ADT_A01 Element and click OK.

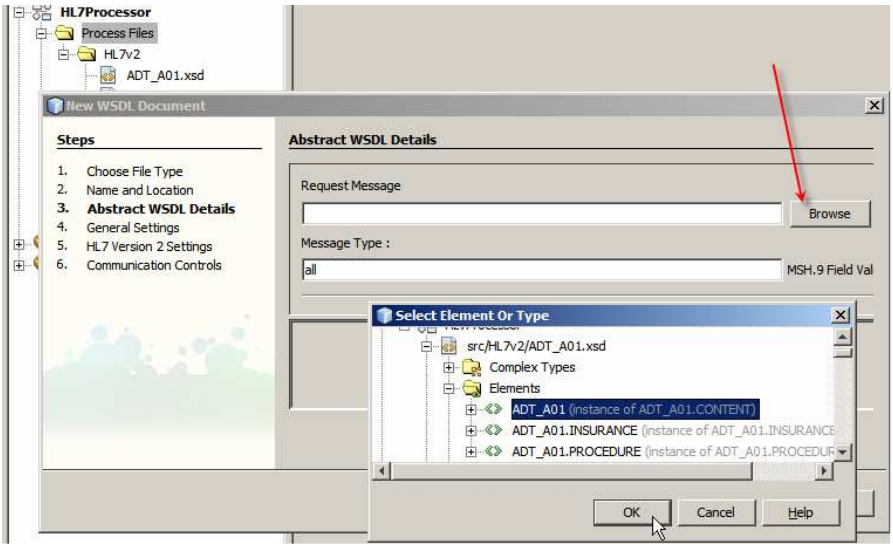


Figure 0-118 Choose ADT A01 for the request message

Manually enter "**ADT^A01**" for the message type and click Next.

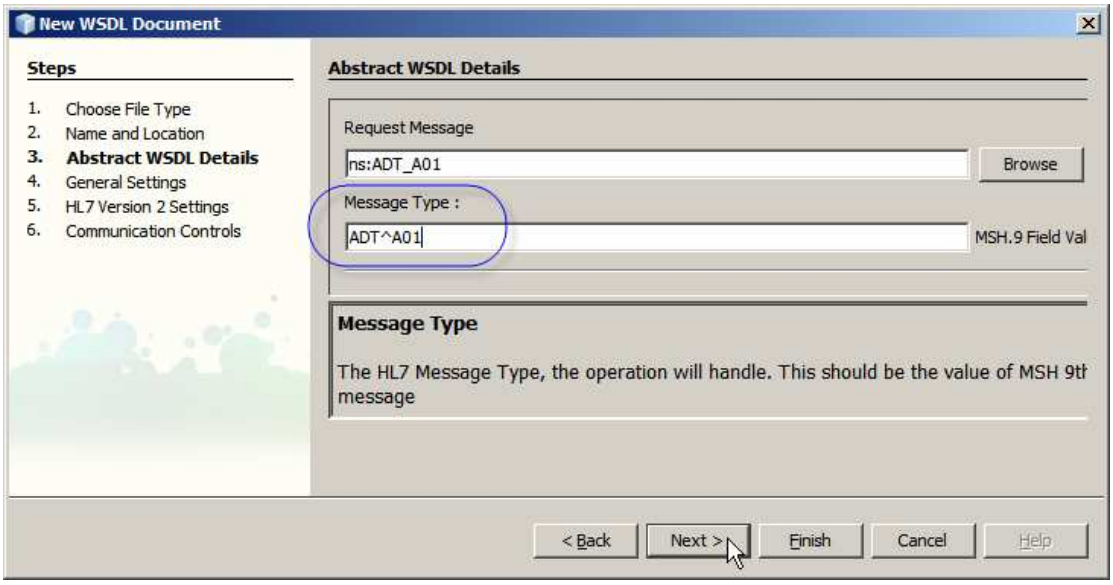


Figure 0-119 Enter message type

Modify the listening port if you need to. By default it is 4040. Notice, too, the Start Block, End Block and End Data characters. Recall the brief discussion in the "7Scan" section. We are configuring a HL7 Listener. The HL7 client/sender will have to be configured so that its star block, end block and end data characters are identical to what is configured here.

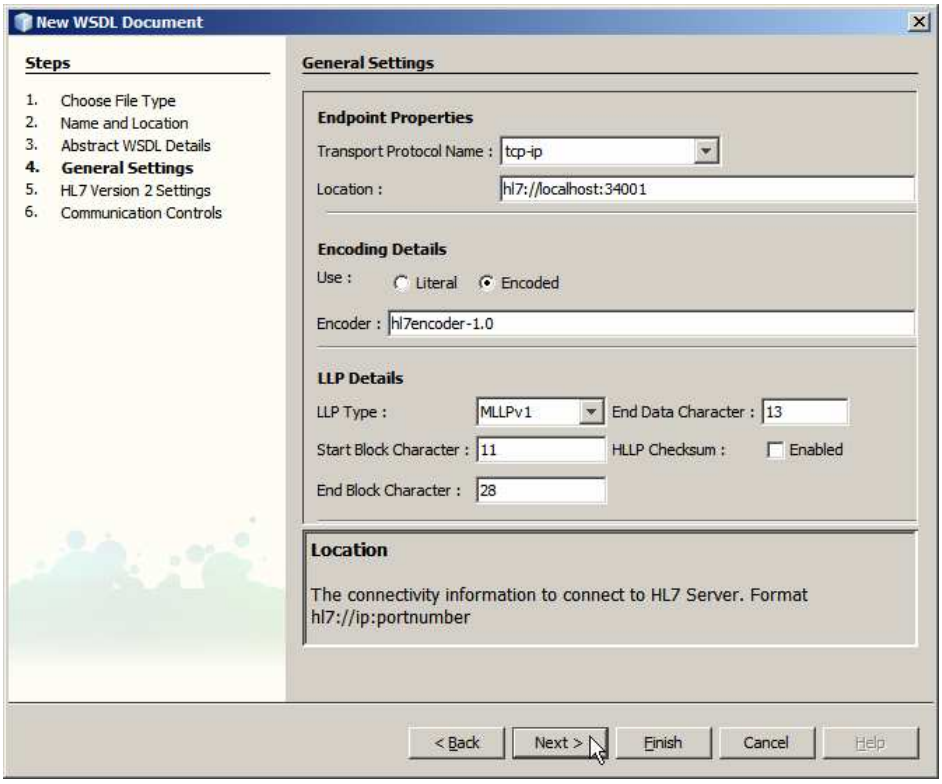


Figure 0-120 Changing the listening port and confirming MLLP delimiter characters

Click Next, accept all defaults in the HL7 Version 2 Settings and click Next again.

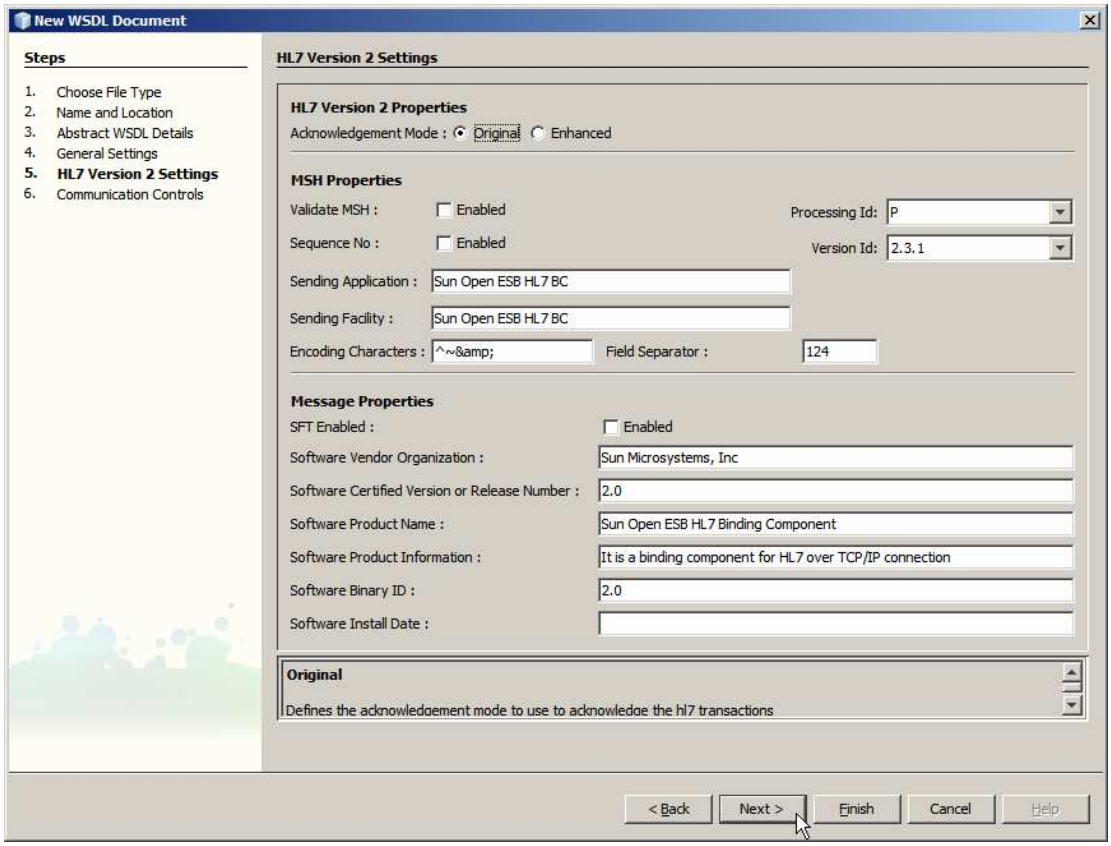


Figure 0-121 Accept default for the HL7 Version 2 Settings

Accept defaults at the Communications Controls and click Finish.

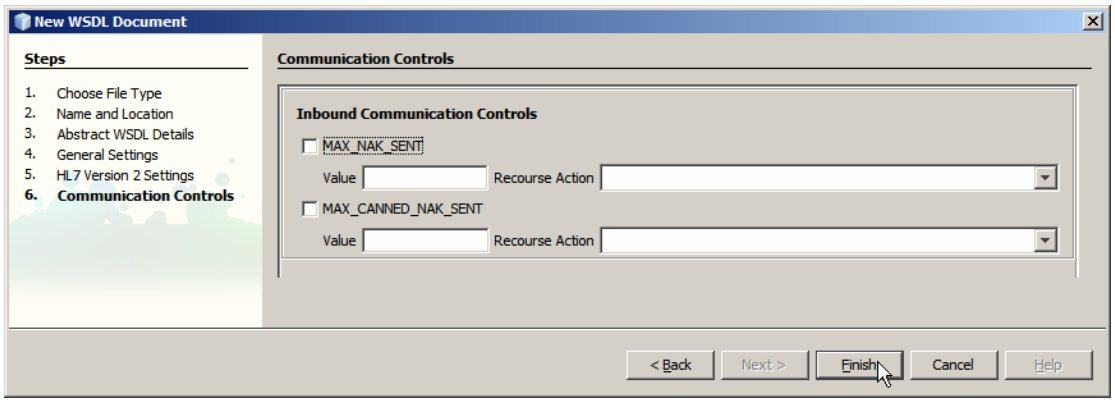


Figure 0-122 Complete the Wizard.

Outcome of this process is a WSDL which defines one operation on one type of HL7 message. It looks similar to what is shown below.

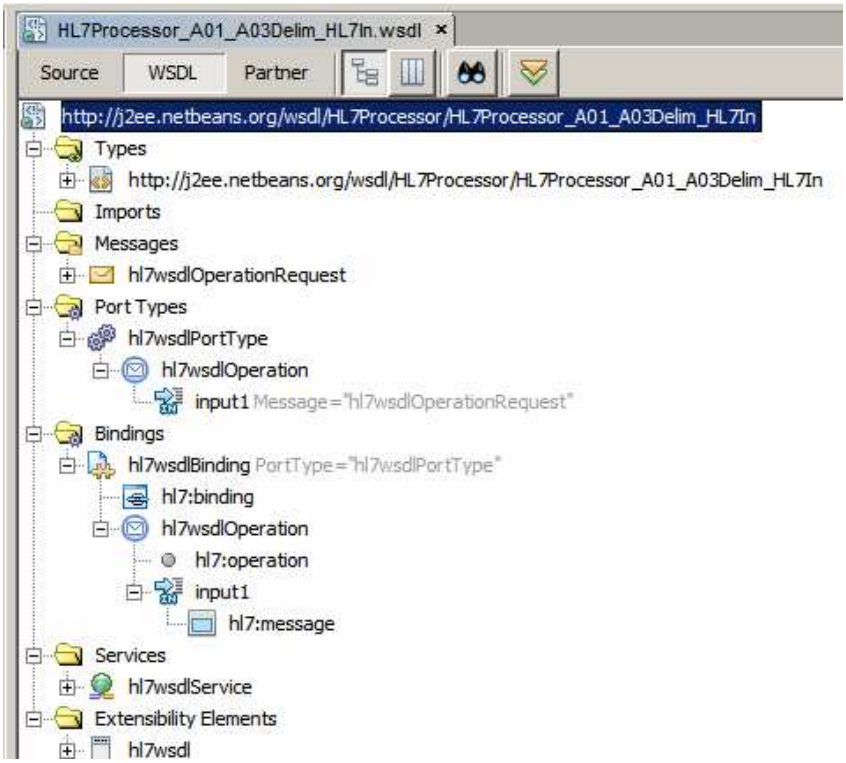


Figure 0-123 WSDL generated by the HL7 Wizard

In order for us to be able to process two different types of HL7 messages, the ADT A01, already configured, and the ADT A03, we need to add a new Message, WSDL Operation and Binding Operation. This has to be done through the WSDL editor since at this time there is no Wizard that will assist.

First, let's add a message. Right-click on the Messages node and choose Add Message.

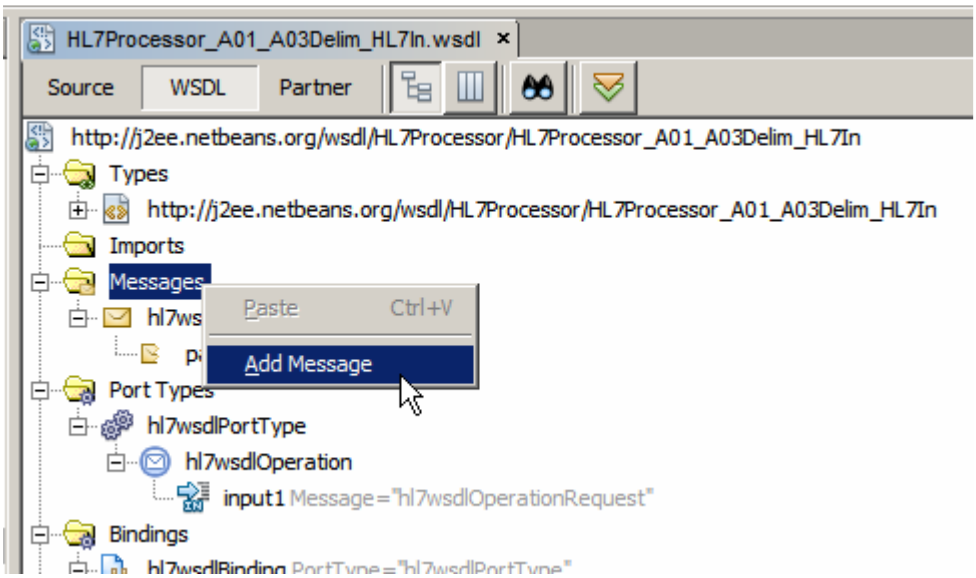


Figure 0-124 Trigger Add Message functionality

Rename the new node to msgADT_A03 by right-clicking it and choosing Refactor->Rename.

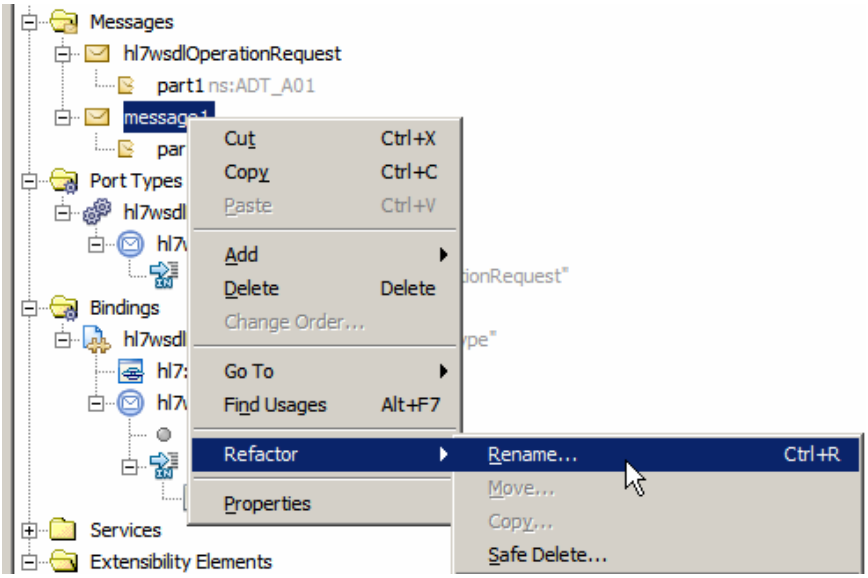


Figure 0-125 Rename the message

Right-click on the part1 node under the msgADT_A03 message and choose Properties.

Click the small button with ellipsis in it alongside the Element or Type data entry box, locate the ADT A03 XSD, select the ADT_A03 Element, click OK and Close.

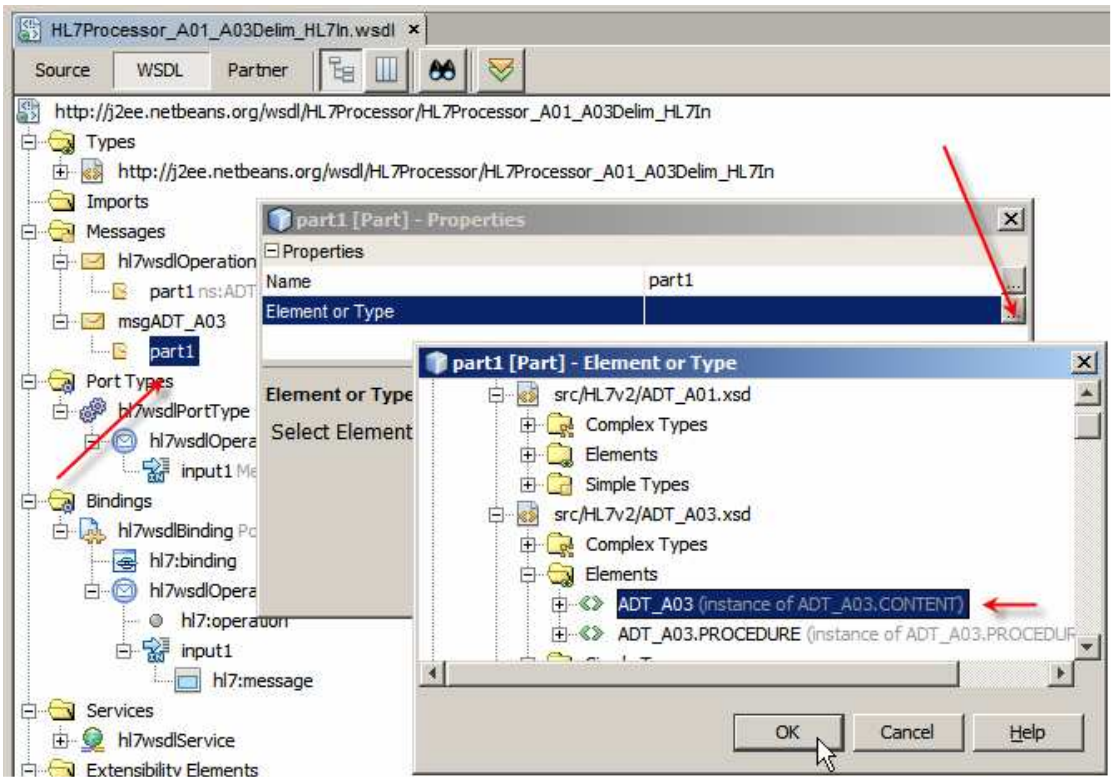


Figure 0-126 Configure data type for the message part

Right-click the hl7wsdlPortType node, choose Add, choose Operation …

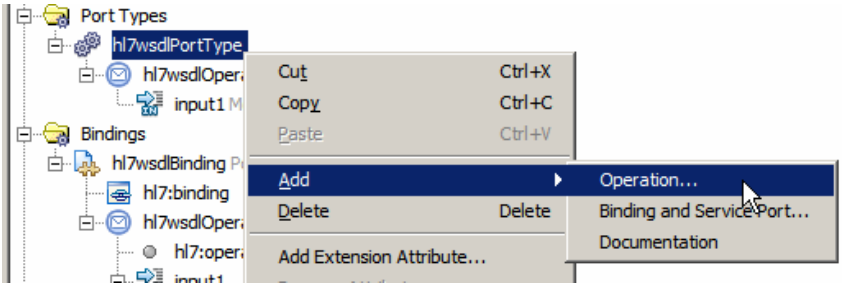


Figure 0-127 Choose to add new operation to the existing port

Name the operation opADT_A03, choose One-Way Operation for Operation Type and select msgADT_A03 message as Input. Click OK to complete the wizard.

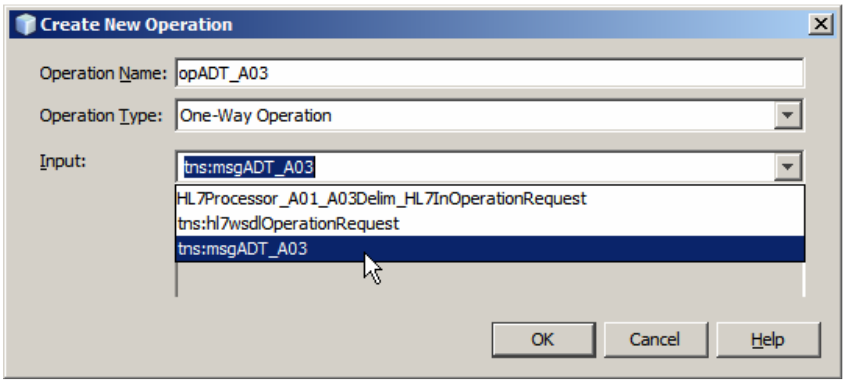


Figure 0-128 Name and configure new operation

Right-click the hl7wsdlBinding node, choose Add, choose Binding Operation.

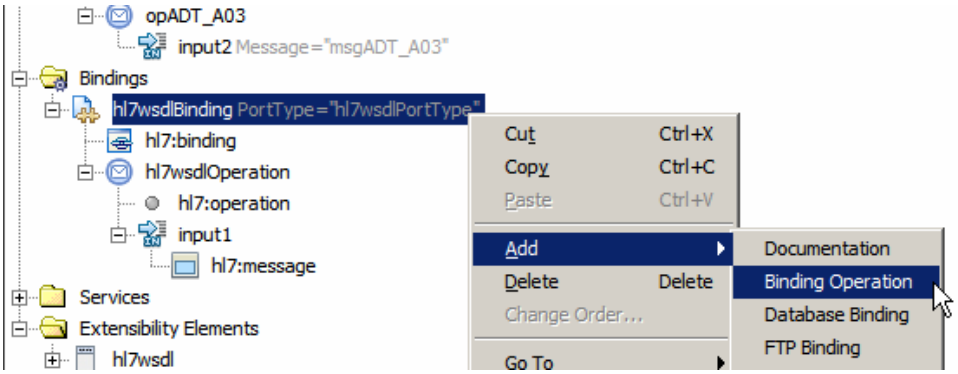


Figure 0-129 Add new Binding Operation

New binding operation with empty input2 node is added.

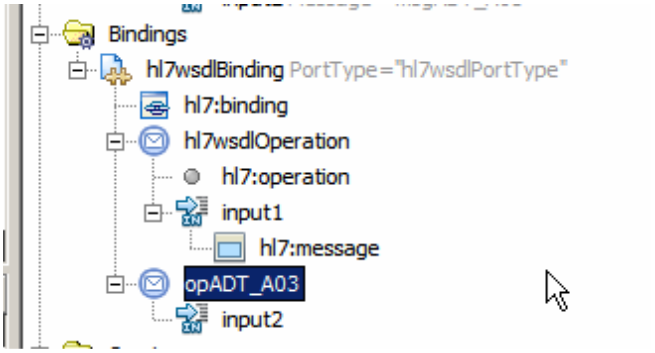


Figure 0-130 New, unconfigured Binding Operation

Right-click the input2 node, choose Add, choose HL7 Message.

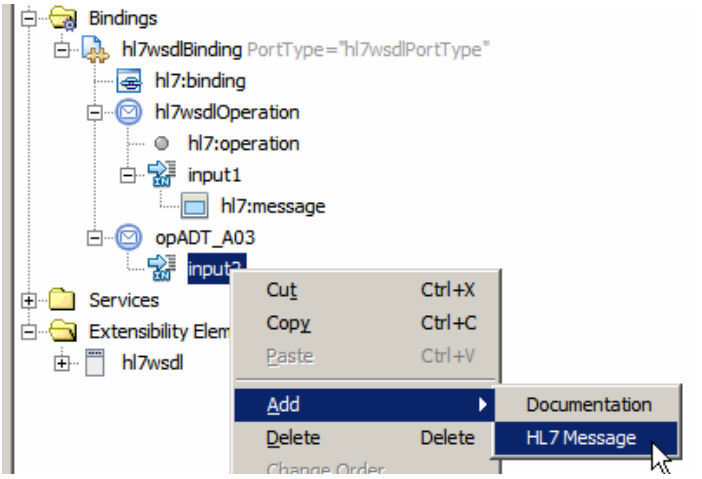


Figure 0-131 Trigger Add HL7 Message to the Binding Operation functionality

Right-click hl7:message and choose Properties. Configure **part1** for part, **encoded** for use and enter **hl7encoder-1.0** for encodingStyle.

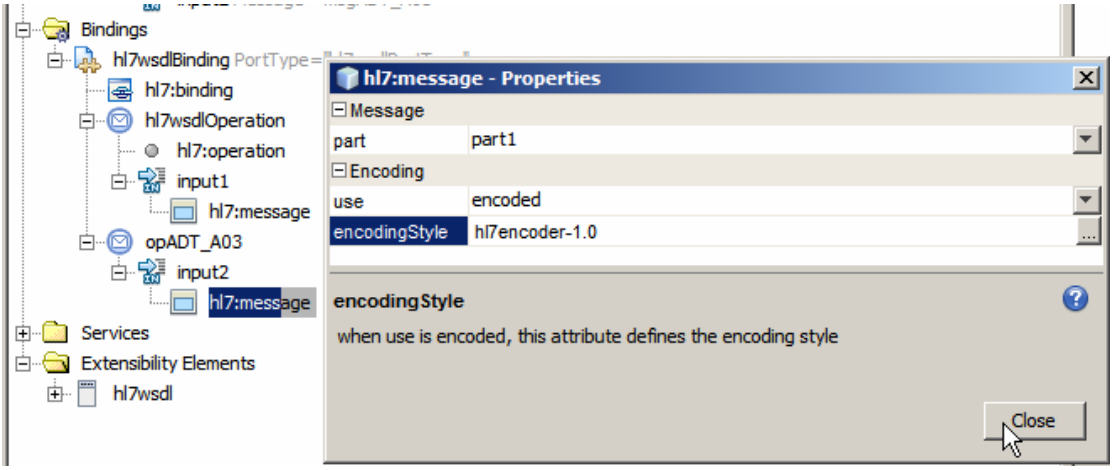


Figure 0-132 Configure hl7:message properties.

Right-click opADT_A03 node, choose Add, choose HL7 Operation.

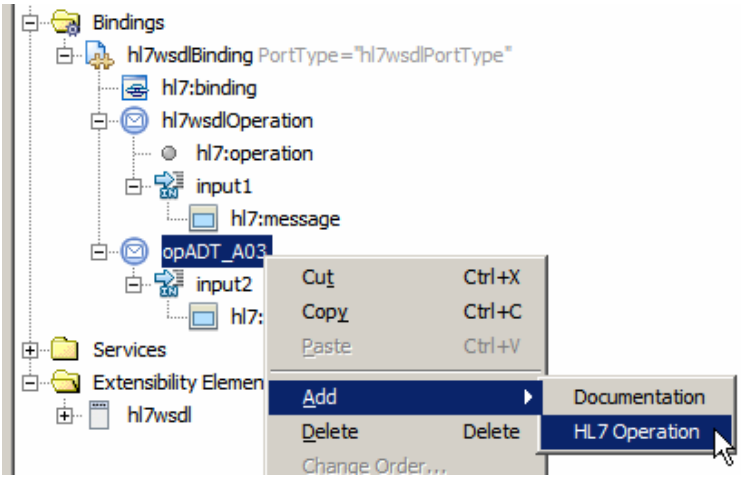


Figure 0-133 Add HL7 Operation to the Binding Operation

Right-click hl7:operation node, choose Properties, enter **ADT^A03** into the messageType data entry box and close the dialogue box.

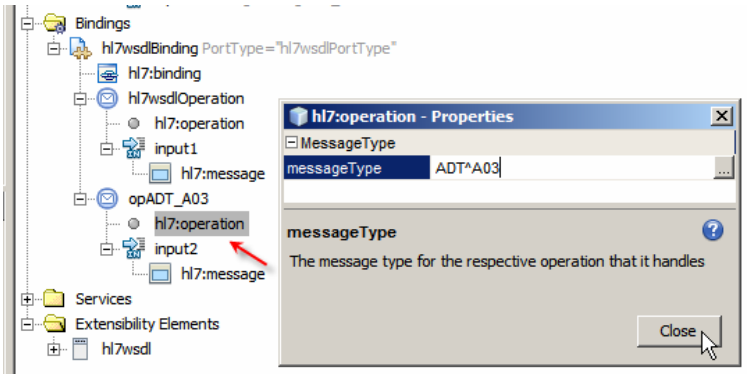


Figure 0-134 Configure messageType for the opADT_A03 operation

Save the modified WSDL. Now the HL7 BC will accept both ADT^A01 and ADT^A03 message types and will reject all others.

The HL7Processor will receive a stream of HL7 messages using the HL7 BC. It will produce two streams of MDMCustomPatient and IEPCustomDischarge messages to be sent to two different JMS Queues using the JMS BC.

We need to define XML Schemas for the MDMCustomPatient and IEPCustomDischarge XML messages. Rather then going through the process of describing how to do that in the user interface let's create an empty XML Schema definitions and paste the XML code right in.

Let's create the MDMCustomPatient XML Schema Document.

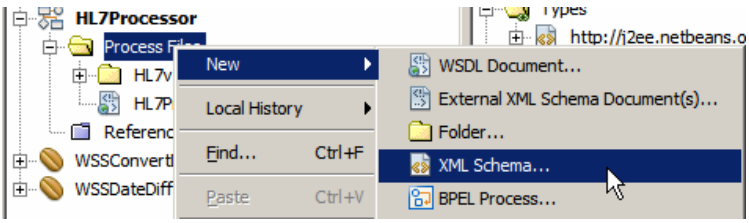


Figure 0-135 Trigger create xml schema document functionality

Name the XML schema MDMCustomPatient.

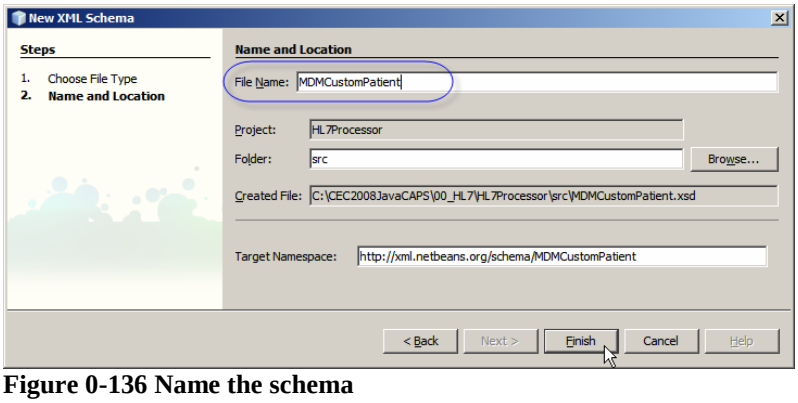


Figure 0-136 Name the schema

Switch to Source mode, select all of the content.

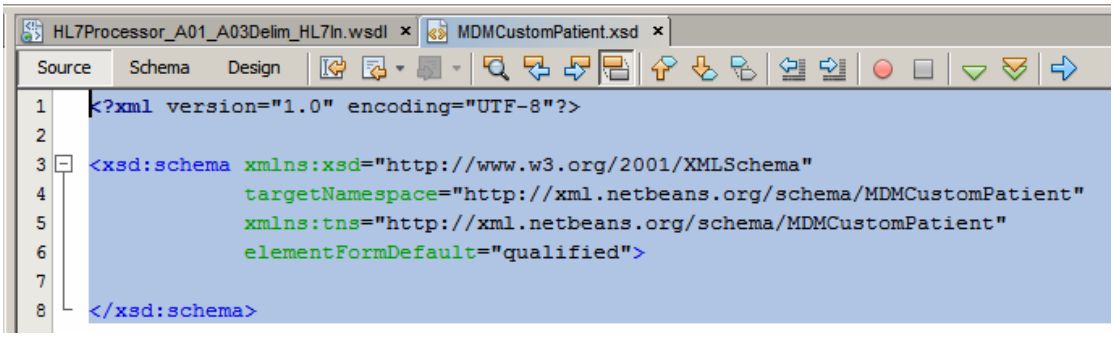


Figure 0-137 Switch to Source mode and select all content

Paste the following XML code in its place.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       targetNamespace="http://xml.netbeans.org/schema/MDMCustomPatient"
       xmlns:tns="http://xml.netbeans.org/schema/MDMCustomPatient"
       elementFormDefault="qualified">
    <xsd:element name="elMDMCustomPatient">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="MSH">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element
                                  name="MSH_3_SENDING_APPLICATION"
                                  type="xsd:string"
                                  minOccurs="0" maxOccurs="1" />
                            <xsd:element
                                  name="MSH_4_SENDING_FACILITY"
                                  type="xsd:string"
                                  minOccurs="0" maxOccurs="1" />
                            <xsd:element
                                  name="MSH_7_DATE_TIM_OF_MESSAGE_ISO8601"
                                  type="xsd:dateTime"
                                  minOccurs="0" maxOccurs="1" />
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
                <xsd:element name="EVN">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element
                                  name="EVN_1_TRIGGER_EVENT"
                                  type="xsd:string"
                                  minOccurs="0" maxOccurs="1"/>
                            <xsd:element
                                  name="EVN_5_1_OPERATOR_ID"
                                  type="xsd:string"
                                  minOccurs="0" maxOccurs="1"/>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
                <xsd:element name="PID">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element
                                  name="PID_3X_1_ID"
                                  type="xsd:string"
```

```
            minOccurs="0" maxOccurs="1"/>
          <xsd:element
            name="PID_3X_4_ASSIGNING_AUTHORITY"
            type="xsd:string"
            minOccurs="0" maxOccurs="1"/>
          <xsd:element
            name="PID_3X_5_ID_TYPE_CODE"
            type="xsd:string"
            minOccurs="0" maxOccurs="1"/>
          <xsd:element
            name="PID_3X_6_ASSIGNING_FACILITY"
            type="xsd:string"
            minOccurs="0" maxOccurs="1"/>
          <xsd:element
            name="PID_5_1_PATIENT_NAME_FAMILY"
            type="xsd:string"
            minOccurs="0" maxOccurs="1"/>
          <xsd:element
            name="PID_5_2_PATIENT_NAME_GIVEN"
            type="xsd:string"
            minOccurs="0" maxOccurs="1"/>
          <xsd:element
            name="PID_5_3_PATIENT_NAME_MIDDLE"
            type="xsd:string"
            minOccurs="0" maxOccurs="1"/>
          <xsd:element
            name="PID_5_4_PATIENT_NAME_SUFFIX"
            type="xsd:string"
            minOccurs="0" maxOccurs="1"/>
          <xsd:element
            name="PID_5_5_PATIENT_NAME_PREFIX"
            type="xsd:string"
            minOccurs="0" maxOccurs="1"/>
          <xsd:element
            name="PID_5_6_PATIENT_NAME_DEGREE"
            type="xsd:string"
            minOccurs="0" maxOccurs="1"/>
          <xsd:element
            name="PID_7_DATE_TIME_OF_BIRTH_ISO8601"
            type="xsd:dateTime"
            minOccurs="0" maxOccurs="1"/>
          <xsd:element
            name="PID_8_ADMINISTRATIVE_SEX"
            type="xsd:string"
            minOccurs="0" maxOccurs="1"/>
          <xsd:element
            name="PID_11X_1_PATIENT_ADDRESS_STREET"
            type="xsd:string"
            minOccurs="0" maxOccurs="1"/>
          <xsd:element
        name="PID_11X_2_PATIENT_ADDRESS_OTHER_DESIGNATION"
            type="xsd:string"
            minOccurs="0" maxOccurs="1"/>
          <xsd:element
            name="PID_11X_3_PATIENT_ADDRESS_CITY"
            type="xsd:string"
            minOccurs="0" maxOccurs="1"/>
          <xsd:element
            name="PID_11X_4_PATIENT_ADDRESS_STATE"
            type="xsd:string"
            minOccurs="0" maxOccurs="1"/>
          <xsd:element
            name="PID_11X_5_PATIENT_ADDRESS_POST_CODE"
                  type="xsd:string"
                  minOccurs="0" maxOccurs="1"/>
          <xsd:element
            name="PID_19_MEDICARE_NUMBER"
            type="xsd:string"
```

```
                                minOccurs="0" maxOccurs="1"/>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

This is a fairly large body of XML. Creating it in a graphical environment, even assisted by the NetBeans IDE, would take a little while. Describing and illustrating the process in a step-by-step manner would take too many pictures and too many pages.

To make sure the paste operation was correct let's check XML and validate XML.

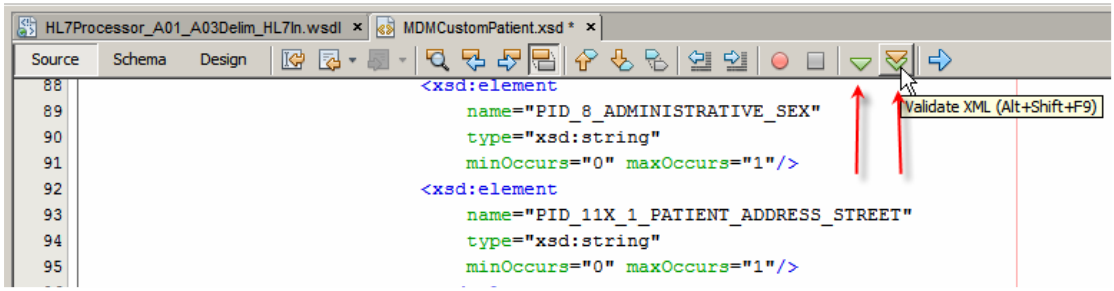


Figure 0-138 Check and validate XML

Let's create the IEPCustomDischarge XML Schema much the same way as the MDMCustomPatient schema before. This time paste the following to replace the XML schema template in the Source window.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://xml.netbeans.org/schema/IEPCustomDischarge"
    xmlns:tns="http://xml.netbeans.org/schema/IEPCustomDischarge"
    elementFormDefault="qualified"
    >
    <xsd:element name="elIEPCustomDischarge">
    <xsd:complexType>
        <xsd:sequence>
        <xsd:element name="MSH">
            <xsd:complexType>
            <xsd:sequence>
                <xsd:element  name="MSH_3_SENDING_APPLICATION"
                    type="xsd:string" minOccurs="0" maxOccurs="1" />
                <xsd:element name="MSH_4_SENDING_FACILITY"
                    type="xsd:string" minOccurs="0" maxOccurs="1" />
                <xsd:element name="MSH_7_DATE_TIM_OF_MESSAGE"
                    type="xsd:string" minOccurs="0" maxOccurs="1" />
            </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
        <xsd:element name="PID">
            <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="PID_3X_1_ID"
                    type="xsd:string" minOccurs="0" maxOccurs="1"/>
                <xsd:element name="PID_3X_6_ASSIGNING_FACILITY"
                    type="xsd:string" minOccurs="0" maxOccurs="1"/>
                <xsd:element name="PID_5_1_PATIENT_NAME_FAMILY"
                    type="xsd:string" minOccurs="0" maxOccurs="1"/>
```

```
                <xsd:element name="PID_5_2_PATIENT_NAME_GIVEN"
                    type="xsd:string" minOccurs="0" maxOccurs="1"/>
                <xsd:element name="PID_7_DATE_TIME_OF_BIRTH"
                    type="xsd:string" minOccurs="0" maxOccurs="1"/>
                <xsd:element name="PID_8_ADMINISTRATIVE_SEX"
                    type="xsd:string" minOccurs="0" maxOccurs="1"/>
            </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
        <xsd:element name="PV1">
            <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="PV1_19_1_VISIT_NUMBER"
                    type="xsd:string" minOccurs="0" maxOccurs="1"/>
                <xsd:element name="PV1_44_ADMIT_DATE_TIME"
                    type="xsd:string" minOccurs="0" maxOccurs="1"/>
                <xsd:element name="PV1_45_DISCHARGE_DATE_TIME"
                    type="xsd:string" minOccurs="0" maxOccurs="1"/>
                <xsd:element name="LOS"
                    type="xsd:int"/>
            </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

Check and validate XML.

Our two custom XML schemas are ready to be used in mapping and endpoint configuration.

We could create two separate WSDLs to configure the two JMS clients, one for each JMS Queue, or we could create a single two-operation WSDL to configure the single JMS client to send messages to two separate queues on the same JMS Server.

To make the process a bit more challenging we will do the latter. First we will create a WSDL and configure it so the JMS BC sends messages to the MDM queue. This is what we would do anyway if we were configuring the JMS BC to send to just one queue. Once we have that we will add another operation to configure the sender for the IEP queue. This is really not too different from what we have done twice already to configure the HL7 BC to receive two different HL7 message types.

Let’s create a new WSDL, HL7Processor_XML_JMSOut, a concrete WSDL, using JMS Binding, of type send.

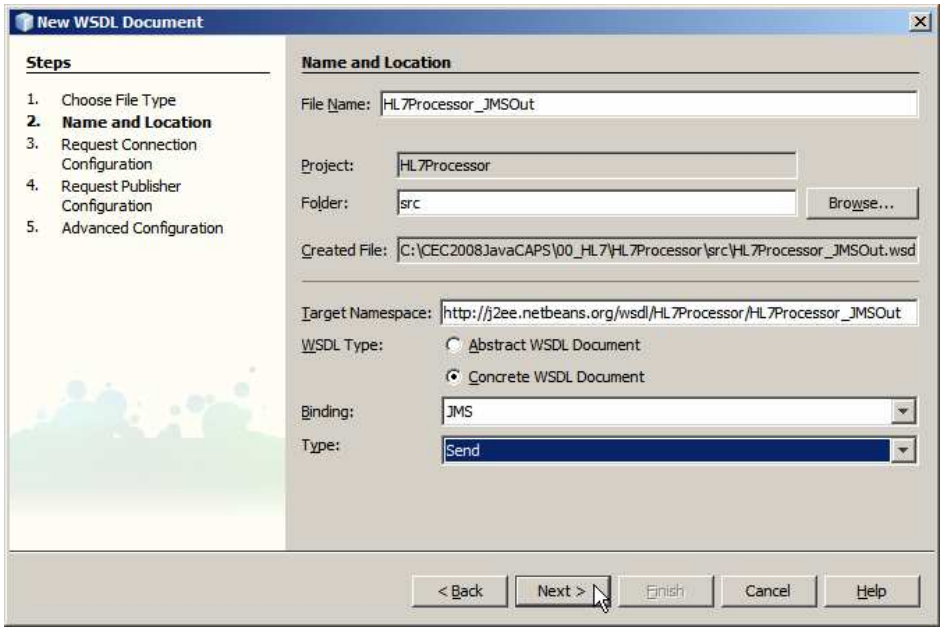


Figure 0-139 Starting creation of the JMS BC configuration

Let's configure the Connection URL to mq://localhost:37676 (you would provide the host and port appropriate to your environment). The URL scheme **mq** indicates that we are using the Java MQ rather then the STCMS JMS implementation. Provide admin and admin as username and password unless your installer changed them form the defaults. Change Message Type to XML, locate the XML Schema elMDMCusotmPatient to use as XSD Element/Type, click OK and Next.

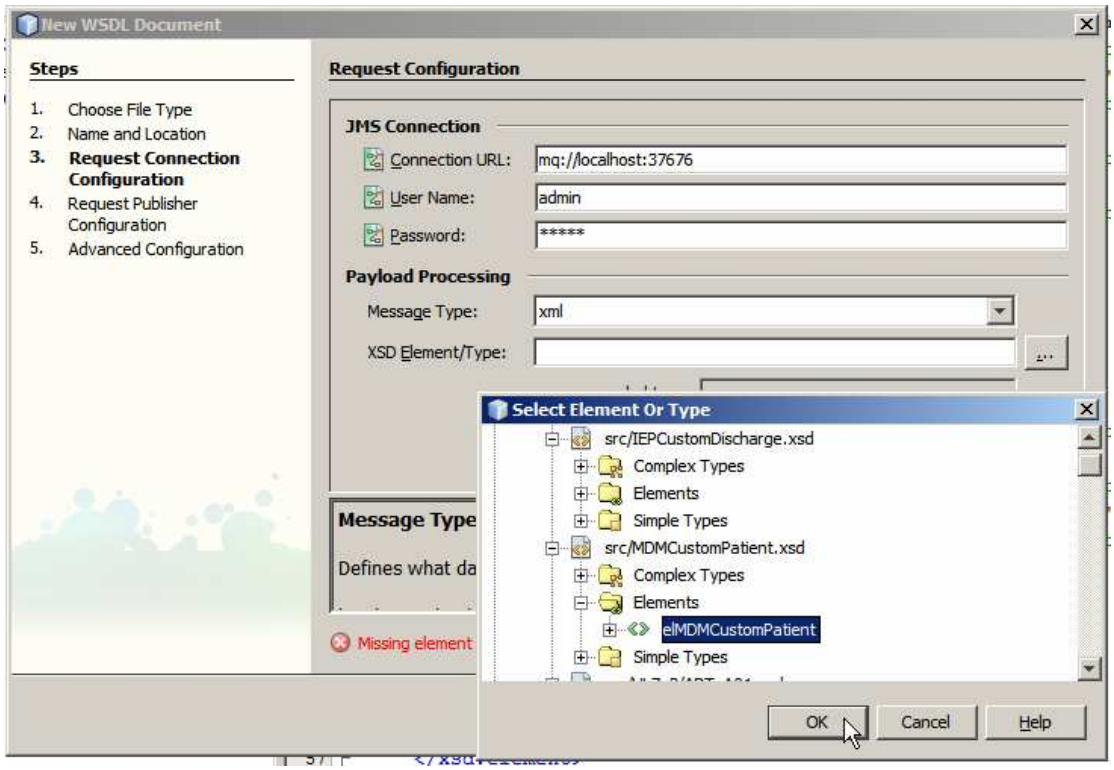


Figure 0-140 Configure Request

Name the Queue qMDM, change Delivery Mode to Non-persistent and click Finish. There is really no need to change delivery mode. Persistent will be fine as well.

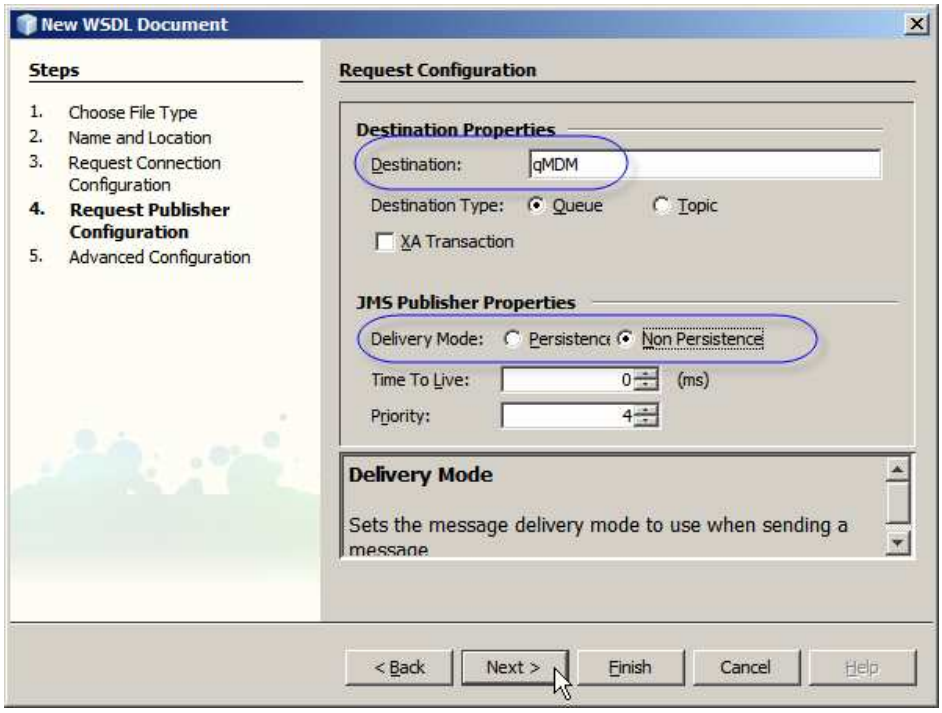


Figure 0-141 Name the queue and change delivery mode

This completes configuration of the JMS BC that will send a single XML message type to a specific queue.

Let's now modify this WSDL to allow the same JMS BC to send a different message type to a different queue on the same JMS Server. We will add a message, an operation to the Port and a Binding operation and message.

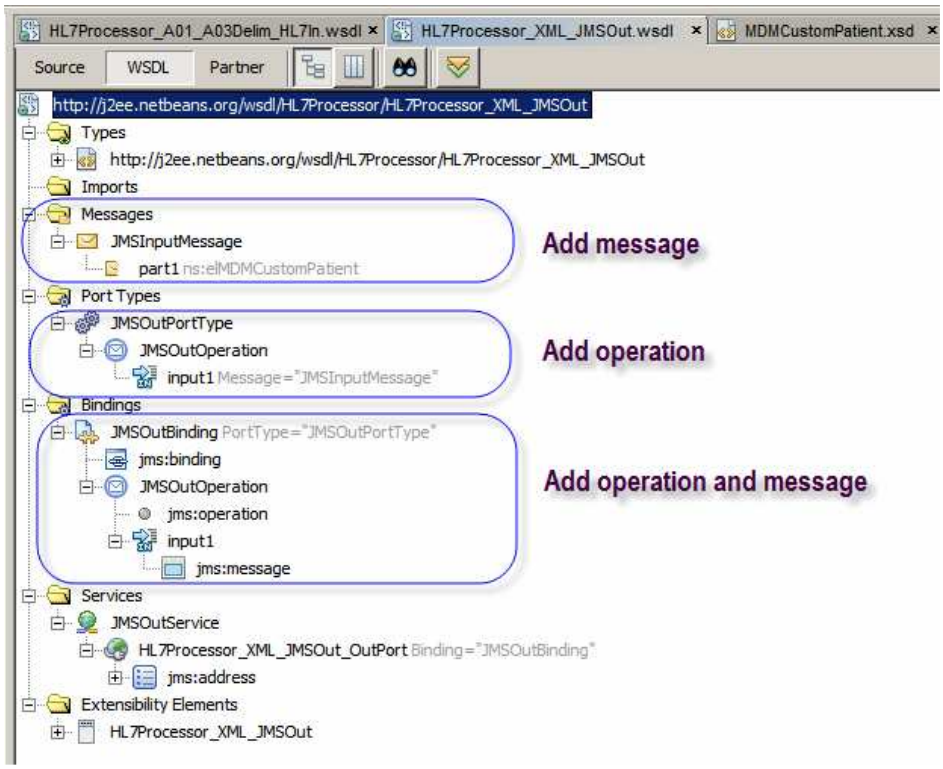


Figure 0-142 What will be modified

Right-click on Messages, choose Add Message, rename message1 to msgToIEP, edit properties of part1 and set the Element or Type property to elIDPCustomDischarge Element.

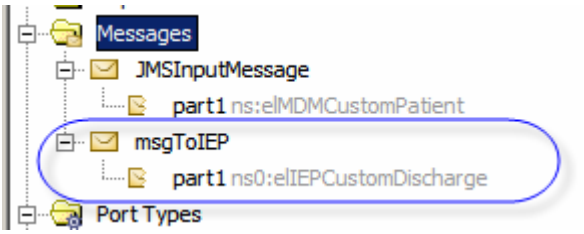


Figure 0-143 Add message of type elIEPCustomDischarge

Right-click on PortTypes->JMSOutPortType, choose Add Operation, change Operation name to opToIEP, change Operation Type to One-Way Operation, choose tns:msgToIEP as Input and click OK.

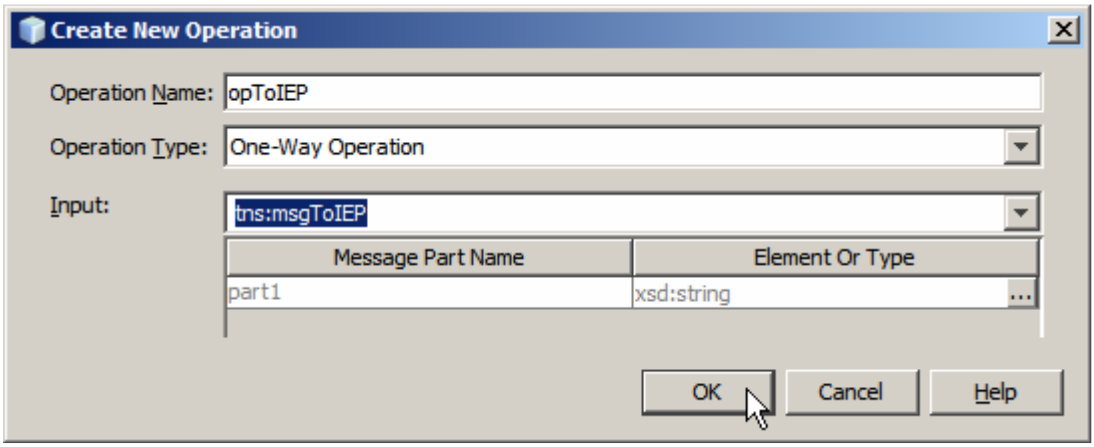


Figure 0-144 Add operation

Right-click on Bindings->JMSOutBinding, choose Add Binding Operation, right-click on input2 node under the new opToIEP node, choose Add JMS Message, modify jms:message properties so that messageType is TextMessage and textPart is part1.

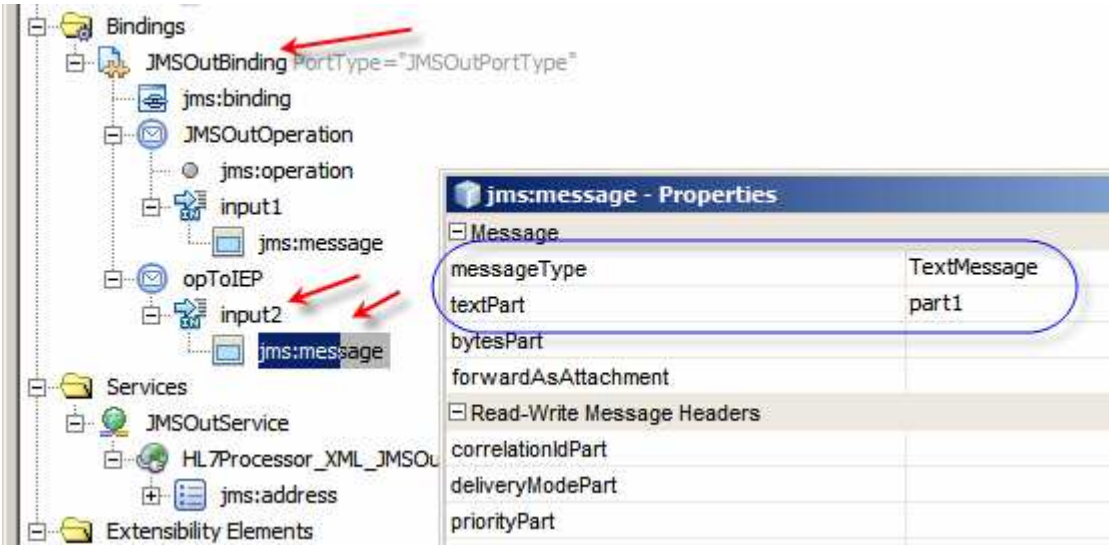


Figure 0-145 Adding Binding Operation and configuring jms:message properties

Right-click opToIEP under the Bindings tree, choose Add, choose JMS Operation, modify jms:operation properties so that destination is qIEP, destinationType is Queue, transaction is NoTransaction, redeliveryHandling is 1:move(same:$_DLQ) and deliveryMode is NON_PERSISTENT.

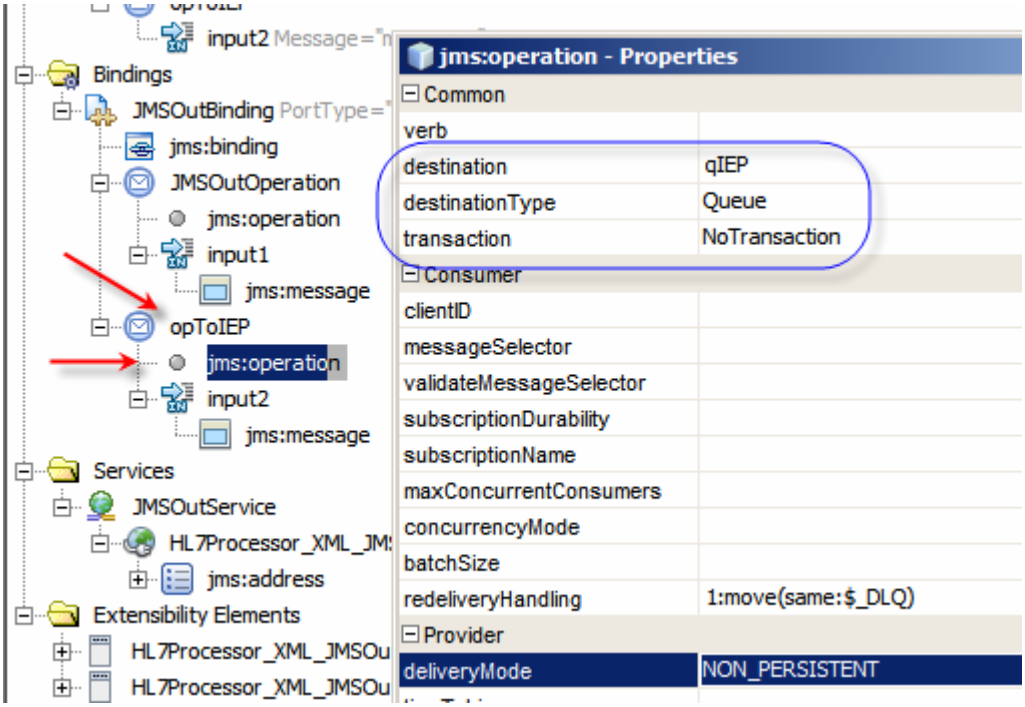


Figure 0-146 Add jms:operatin to opToIEP and configure its properties

Note the redeliveryHandling incantation "1:move(same:$_DLQ)". Read this as "Try to deliver the message **1** time. If not successful, **move** the message to a JMS destination of the **same** type, whose name is the same as the original destination name (**$**) with the string literal **_DLQ** appended". If you like you can configure the same or different redelivery handling for the other queue - qMDM. Also set Priority to 4 and Timeout to 30000 or so.

Save the WSDL.

We now have the WSDLs for the inbound HL7 BC, outbound JMS BC and the two utility web services – WSSDateDiff and WSSConvertDate. We can finally start designing out BPEL 2.0 business process.

Let's create a new BPEL Process, bpHL7Processor.

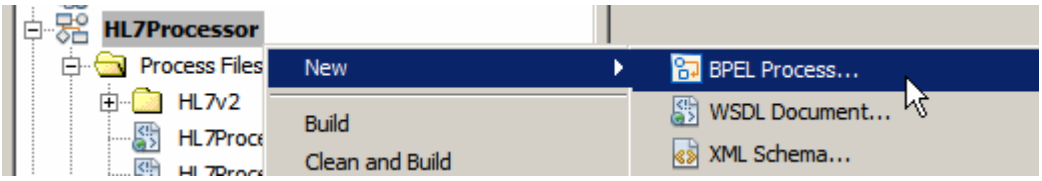


Figure 0-147 Create a New BPEL Process

There may be missing data values in our inbound messages. If we map from inbound fields to outbound fields and data is missing we will get an exception. If missing data

in optional fields is expected and we are not concerned about this we can configure the BPEL process to ignore missing data and continue mapping regardless.
Let's click on the bpHL7Processor process scope and set the Ignore Missing From Data property to Yes.

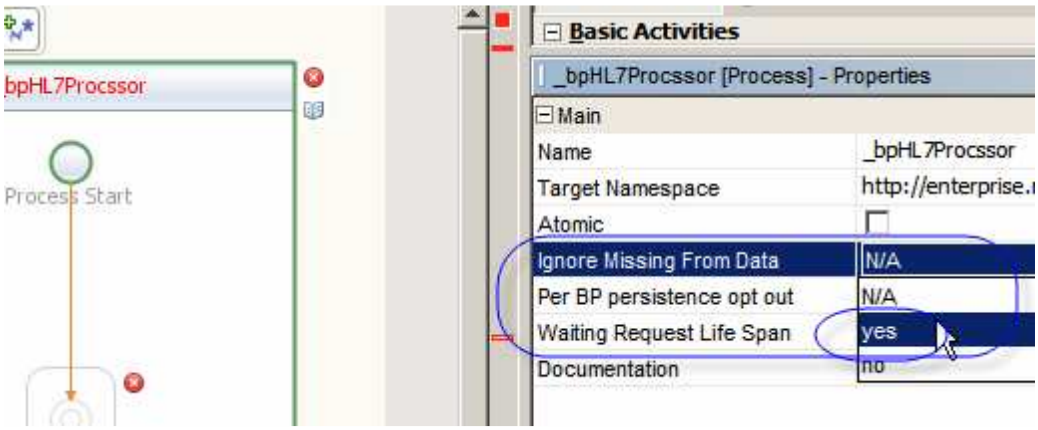


Figure 0-148 Configure Ignore Missing From Data to Yes

Take note of the 'swim lines' where the Inbound/Receive and Outbound/Invoke Binding Component WSDLs will be dropped, as illustrated in the Figure below.

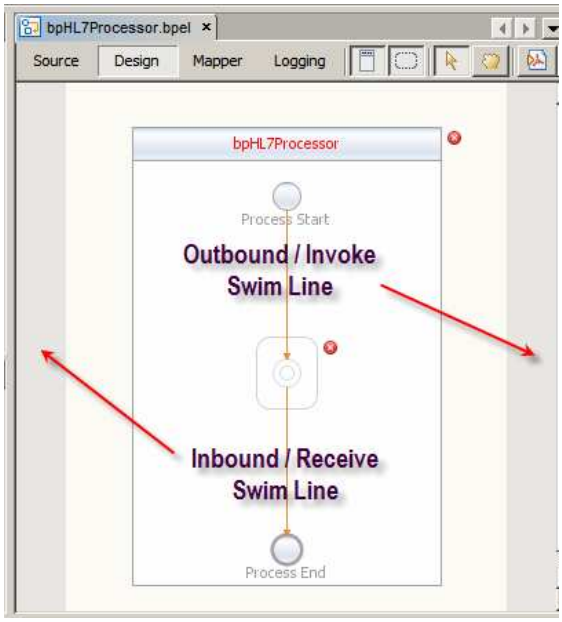


Figure 0-149 Inbound and Outbound swim lines

Drop the HL7Processor_A01_A03Delim_HL7In WSDL onto the target marker in the Inbound/Receive swim line and the HL7Processor_XML_JMSOut onto the target marker in the Outbound/Invoke swim line.

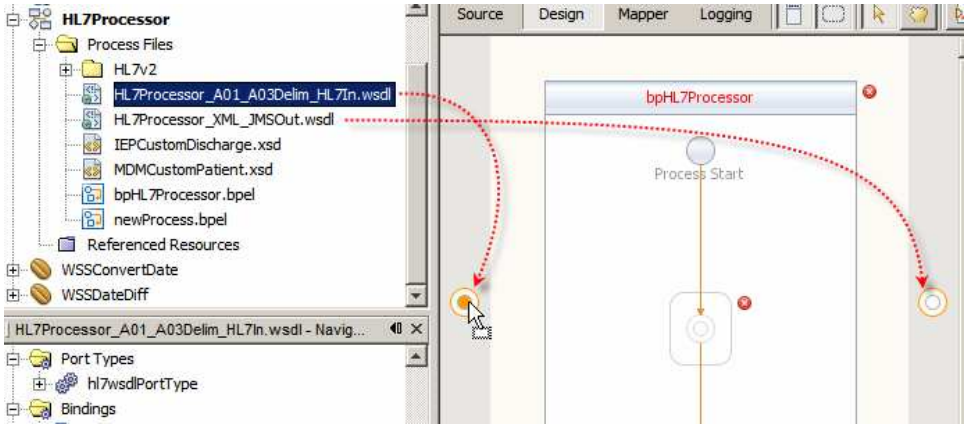


Figure 0-150 Drop WSDLs onto appropriate target markers

Rename Partner Links HL7In and JMSOut respectively.

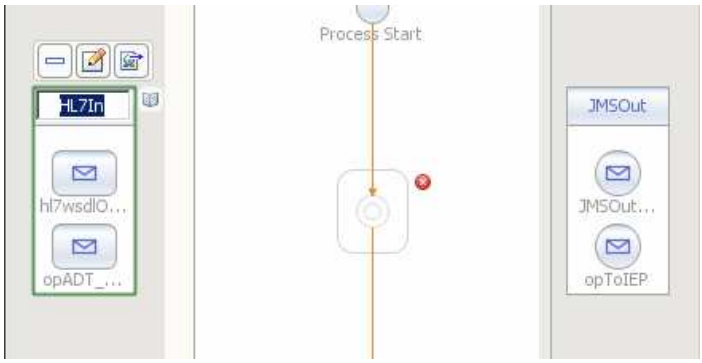


Figure 0-151 Renamed partner links

Note that both partner links have two operations. The HL7 BC has one operation to receive ADT A01 message and the other to receive ADT A03 messages. The JMS BC has one operation to send elMDMCustomPatient messages to qMDM and one to send elIEPCustomDischarge message to qIEP.

To receive two or more message types where the inbound BC exposed more then one operation a BPEL Pick activity is required. The Pick activity can trigger the process when one of its configured receive activities receives a message or when a time expires, if one is configured. In this case we need one of two messages and we don't use the timer. Let's drag the Pick activity from the Structured Activities palette onto the target marker inside the bpHL7Processor scope box.

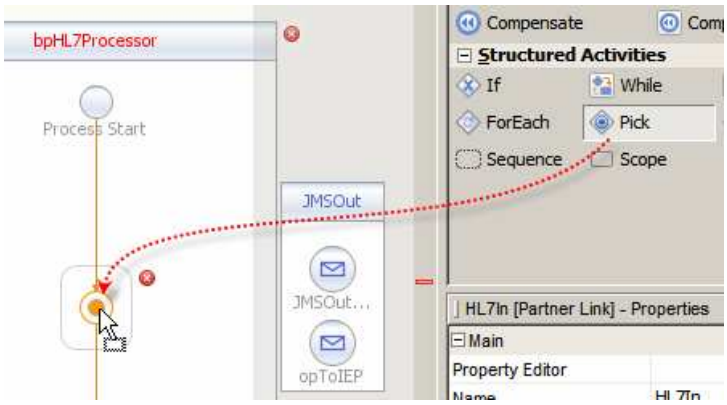


Figure 0-152 Add Pick activity to the process

Make sure to check the Create Instance checkbox so that the process instance gets created regardless of which message arrives.

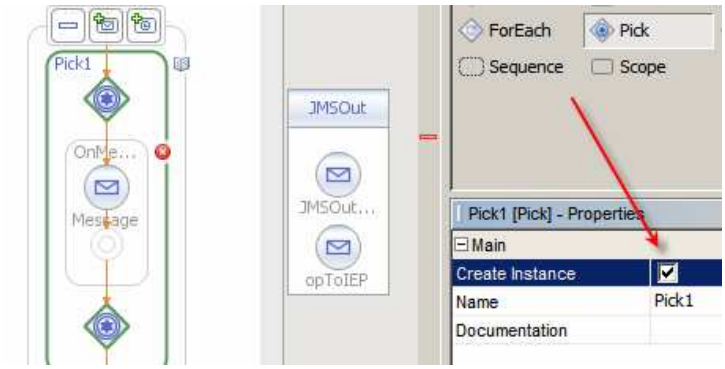


Figure 0-153 Check the Create Instance checkbox

Notice that the Pick1 scope has on OnMessage activity already on the canvas. This is a Receive activity in disguise. We will use that OnMessage activity to receive the A01 message, if any. We need another OnMessage activity. Let's click on the icon with a plus sign and a small envelope, second from the left above the Pick1 scope. These icons don't appear unless the pick activity is 'active'. If it is not, click the Pick1 diamond to make them appear.

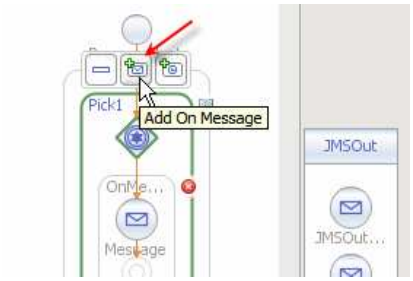


Figure 0-154 Trigger Add OnMessage functionality

Once this is done the process will look like that in the Figure below.

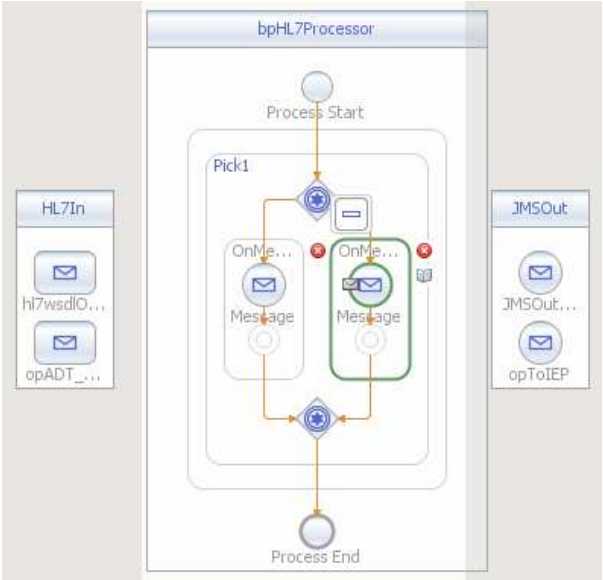


Figure 0-155 The process with two OnMessage activities in the Pick scope.

Select the left hand OnMessage activity by clicking on it and drag from the tiny envelope onto the hl7wsdlOperation operation of the HL7In partner.

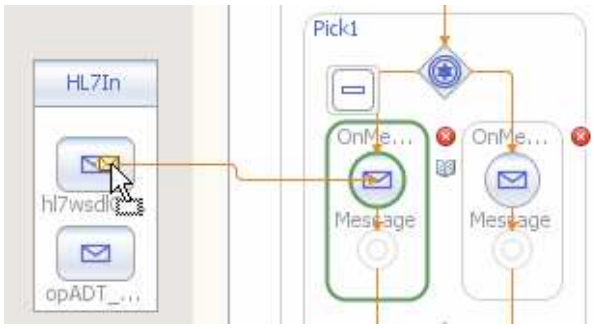


Figure 0-156 Drag to connect OnMessage to partner operation

Double-click on the OnMessage activity connected to the hl7wsdlOperation partner, or right-click on it and choose Edit, to open the property Editor.

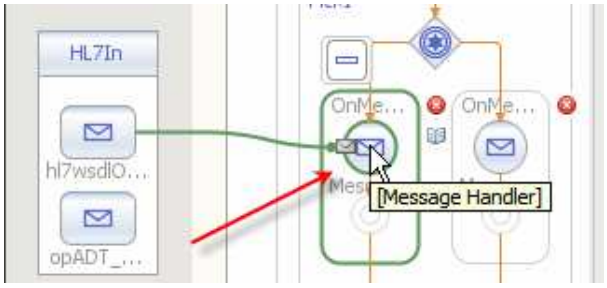


Figure 0-157 Activate Property Editor for the OnMessage activity

When the Property Editor window appears click the Create … button, change the Name property of the New Input Variable, click OK and OK again. This creates a variable (called container in BPEL 1.x) which will have the incoming message content.

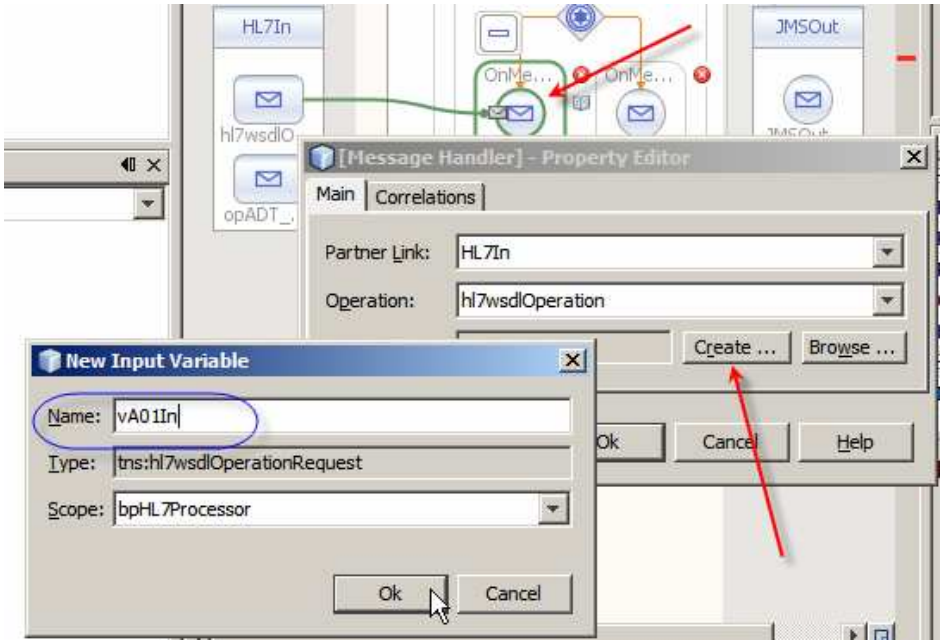


Figure 0-158 Create Input Variable to contain A01 messages

Connect the other OnMessage activity to the opADT_A03 operation and create an Input Variable vA03In following the steps just described.

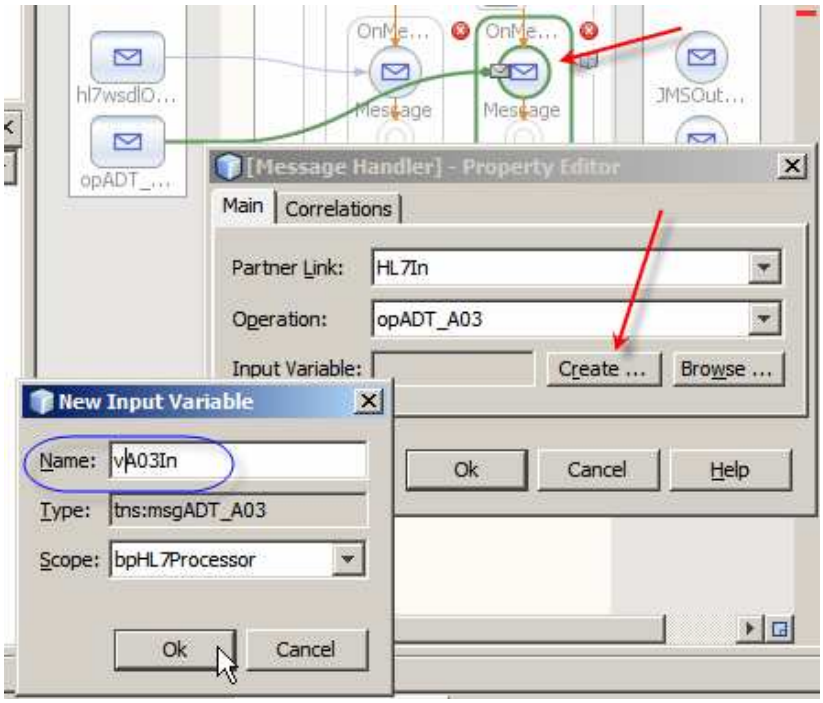


Figure 0-159 Create input variable for A03 messages

Add Assign and Invoke activities below both OnMessage activities by dragging them onto the appropriate target markers from the Basic Activities and Web Services palettes.

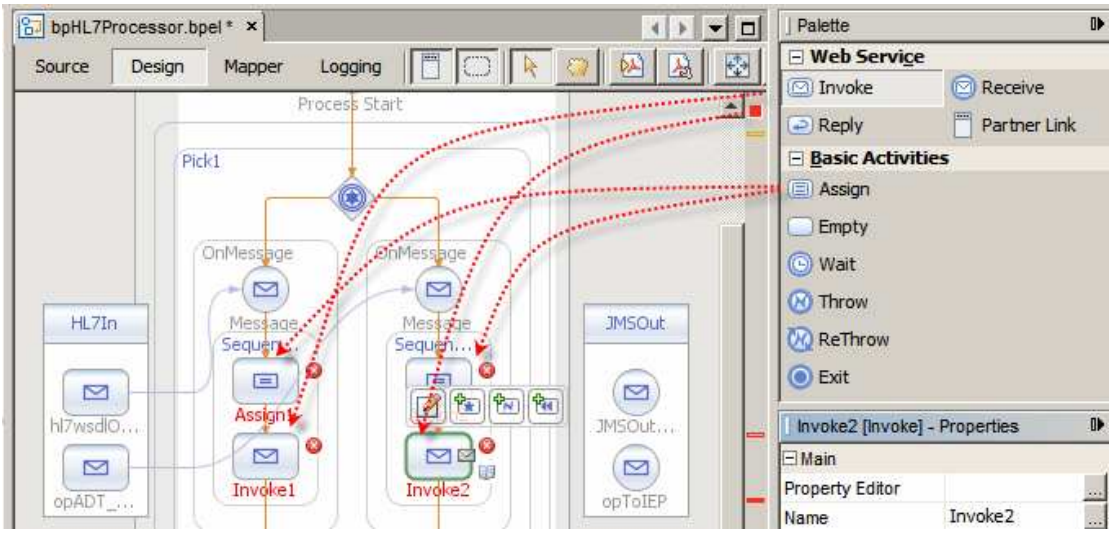


Figure 0-160 Add Assign and Invoke activities

Select Invoke1 activity, click on the Edit icon and add an Input Variable vMDMOut to hold the MDMCustomPatient message on its way out to the JMS Queue.

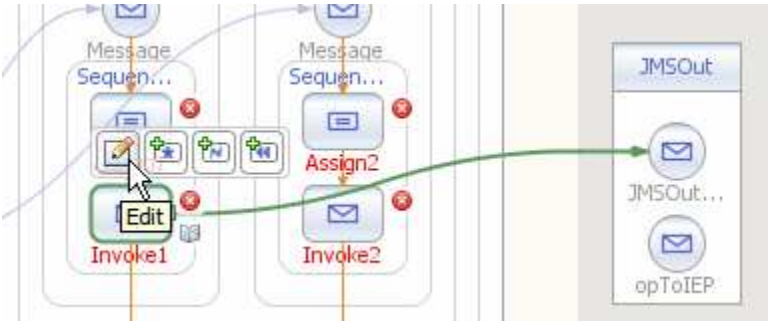


Figure 0-161 Trigger Property Editor Functionality

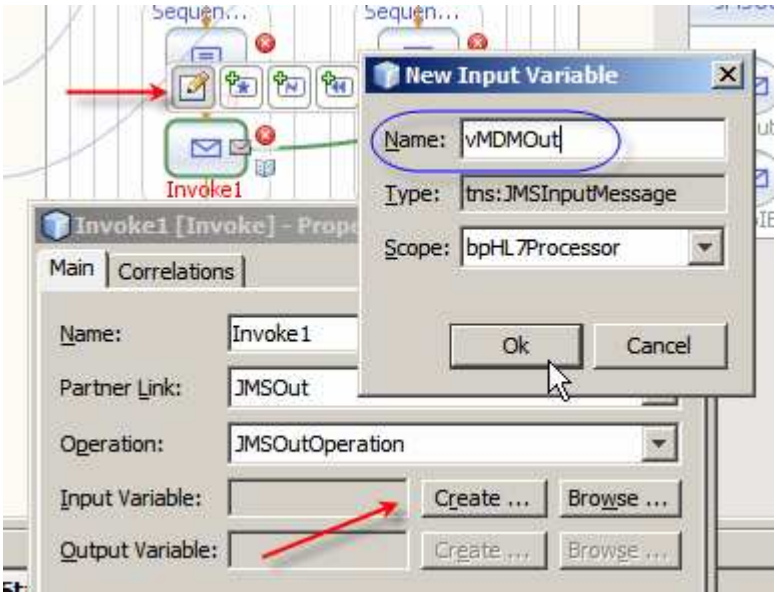


Figure 0-162 Create Input variable for the Custom Patient message

Connect the Invole2 activity to the opToIEP operation and repeat the process of creating an Input Variable vIEPOut to hold the IEPCustomDischarge message.

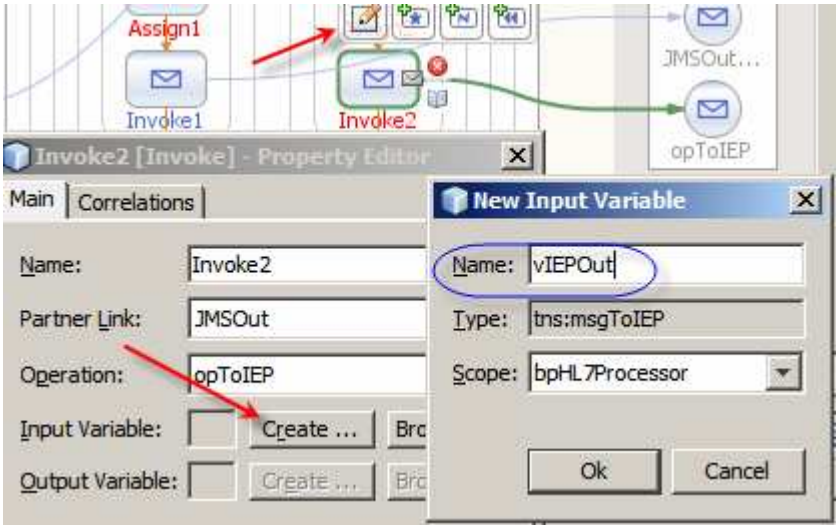


Figure 0-163 Create an Input Variable for the Custom Discharge message

Right-click the Assign1 activity, choose Go To and choose BPEL Mapper. We are going to map these fields of the A01 message to the CustomPatient message which are _not_ date/time fields. In HL7-speak these are fields that are not of TS type.

The fields are enumerates in a table below. The date/time fields will have to be converted using the WSSConvertDate service, which we have not yet added to the canvas, but will when the time comes.

vA01In>part1>MSH>MSH.3>HD.1	vMDMOut>part1>MSH>MSH_3_SENDING_APPLICATION
vA01In>part1>MSH>MSH.4>HD.1	vMDMOut>part1>MSH>MSH_4_SENDING_FACILITY
vA01In>part1>ENV>EVN.1	vMDMOut>part1>EVN>EVN_1_TRIGGER_EVENT
vA01In>part1>ENV>EVN.5>XCN.1	vMDMOut>part1>EVN>EVN_1_TRIGGER_EVENT
vA01In>part1>PID>PID.3>CX.1	vMDMOut>part1>PID>PID_3X_1_ID
vA01In>part1>MSH>MSH.3>HD.1	vMDMOut>part1>PID>PID_3X_4_ASSIGNING_AUTHORITY
vA01In>part1> MSH>MSH.3>HD.1	vMDMOut>part1>PID>PID_3X_6_ASSIGNING_FACILITY
vA01In>part1>PID>PID.5>XPN.1>FN.1	vMDMOut>part1>PID>PID_5_1_PATIENT_NAME_FAMILY
vA01In>part1>PID>PID.5>XPN.2	vMDMOut>part1>PID>PID_5_2_PATIENT_NAME_GIVEN
vA01In>part1>PID>PID.5>XPN.3	vMDMOut>part1>PID>PID_5_3_PATIENT_NAME_MIDDLE
vA01In>part1>PID>PID.5>XPN.4	vMDMOut>part1>PID>PID_5_4_PATIENT_NAME_SUFFIX
vA01In>part1>PID>PID.5>XPN.5	vMDMOut>part1>PID>PID_5_5_PATIENT_NAME_PREFIX
vA01In>part1>PID>PID.5>XPN.6	vMDMOut>part1>PID>PID_5_6_PATIENT_NAME_DEGREE
vA01In>part1>PID>PID.8	vMDMOut>part1>PID>PID_8_ADMINISTRATIVE_SEX
vA01In>part1>PID>PID.11>XAD.1	vMDMOut>part1>PID>PID_11X_1_PATIENT_ADDRESS_STREET
vA01In>part1>PID>PID.11>XAD.3	vMDMOut>part1>PID>PID_11X_3_PATIENT_ADDRESS_CITY
vA01In>part1>PID>PID.11>XAD.5	vMDMOut>part1>PID>PID_11X_5_PATIENT_ADDRESS_PORT_CODE
vA01In>part1>PID>PID.19	vMDMOut>part1>PID>PID_10_MEDICARE_NUMBER

Notice that we did not map the following date/time fields:

- MSH_7_DATE_TIME_OF_MESSAGE_ISO8601
- PID_7_DATE_TIM_OF_BIRTH

We will convert source values for these dates from the HL7 date/time to ISO 8601 date/time format and map later.

We are also not mapping the following fields:

- PID_3X_5_ID_TYPE_CODE
- PID_11X_2_PATIENT_ADDRESS_OTHER_DESIGNATION
- PID_11X_4_PATIENT_ADDRESS_STATE

This is because we are not interested in ID Type, we know our data and we know Address Other Designation never has value in it when messages come from certain system and we know our data and we know that the supposed state codes in the source system are really free text state names and do not map to our state code field. In a more robust solution we would endeavour to address this issue, perhaps by providing state name to code translation table.

Notice, also, names that contain strings like _3X_ and _11X_. The source fields for these are repeating. To keep the mapping simple in this project we are ignoring the fact and, knowing that our data only ever has a single instance of the value anyway, we map just the first repetition. In a more robust solution our target structure would probably have corresponding fields repeating as well and we would map all repetitions.

Here is a fragment of the mapping in the Mapper view.

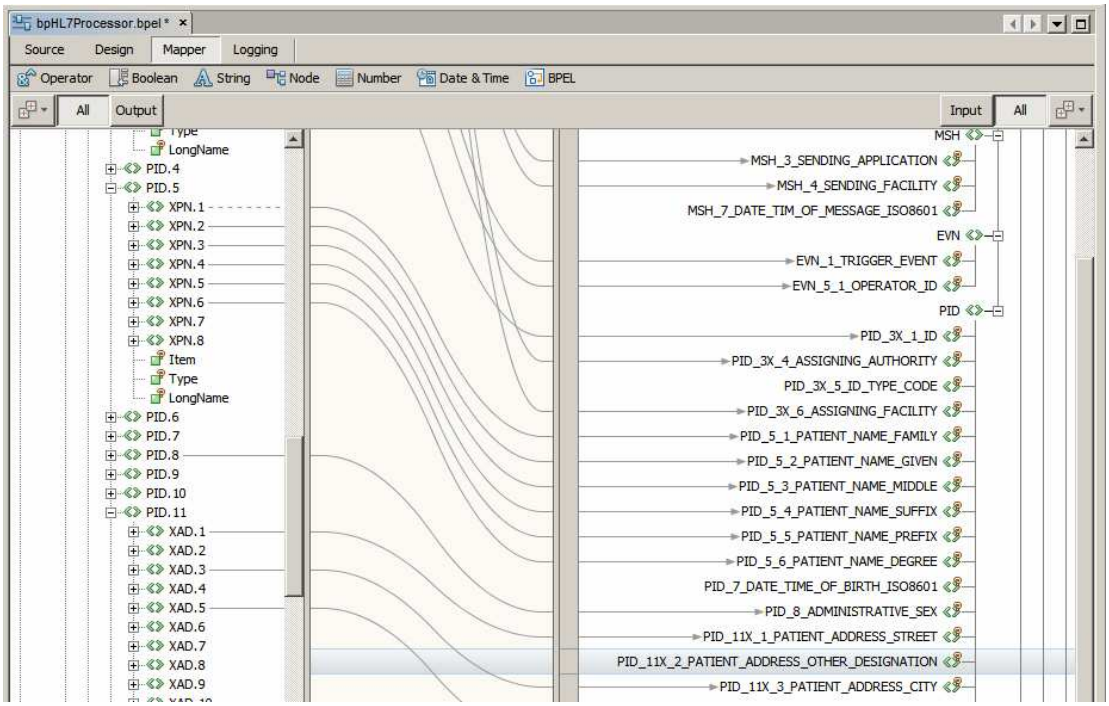


Figure 0-164 Fragment of A01 to MDMCustomPatient mapping

Switch back to the Design View.

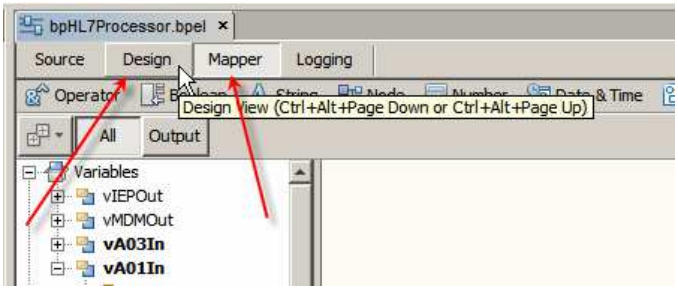


Figure 0-165 View switching Tabs

Click at the tiny Warning icon to show the warning messages – these particular messages are benign and are caused by the HL7 XML schema design – appropriate conversion will be performed.

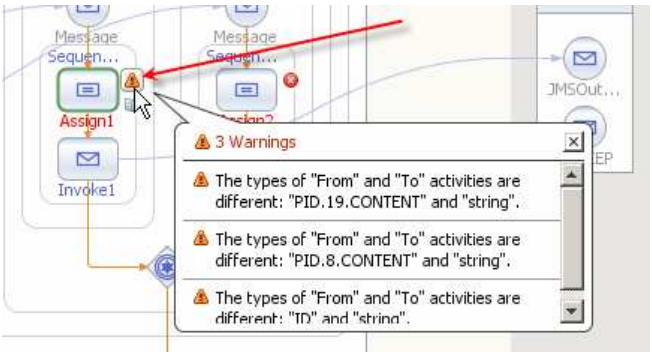


Figure 0-166 Benign (in this case) mapping warnings

Select the Assign2 activity on the second OnMessage's branch and switch to the Mapper view. We will map fields from the A03 message to the fields of the IEPCustomDischarge message which are _not_ date/time fields. In HL7-speak these

are fields that are not of TS type. We will also not map, at this point, the LOS (Length of Stay) field as we will have to derive the value from the difference in days between the discharge and the admission dates. The value will be derived using the WSSDateDiff service, which we developed earlier.

The fields are enumerates in a table below. The date/time fields will have to be converted using the WSSConvertDate service, which we have not yet added to the canvas, but will when the time comes.

vA03In>part1>MSH>MSH.3>HD.1	vIEPOut>part1>MSH>MSH_3_SENDING_APPLICATION
vA03In>part1>MSH>MSH.4>HD.1	vIEPOut>part1>MSH>MSH_4_SENDING_FACILITY
vA03In>part1>PID>PID.3>CX.1	vIEPOut>part1>PID>PID_3X_1_ID
vA03In>part1>MSH>MSH.3>HD.1	vIEPOut>part1>PID>PID_3X_6_ASSIGNING_FACILITY
vA03In>part1>PID>PID.5>XPN.1>FN.1	vIEPOut>part1>PID>PID_5_1_PATIENT_NAME_FAMILY
vA03In>part1>PID>PID.5>XPN.2	vIEPOut>part1>PID>PID_5_2_PATIENT_NAME_GIVEN
vA03In>part1>PID>PID.8	vIEPOut>part1>PID>PID_8_ADMINISTRATIVE_SEX
vA03In>part1>PV1>PV1.19>CX.1	vIEPOut>part1>PID>PV1_19_1_VISIT_NUMBER

Notice that we did not map the following fields:

- MSH_7_DATE_TIME_OF_MESSAGE_ISO8601
- PID_7_DATE_TIM_OF_BIRTH
- PV1_44_ADMIT_DATE_TIME
- PV1_45_DISCHARGE_DATE_TIM
- LOS

We will convert source values for these dates from the HL7 date/time to ISO 8601 date/time format and map later and, in the case of LOS, will derive the value.

Here is a fragment of the mapping rules in the Mapper view.

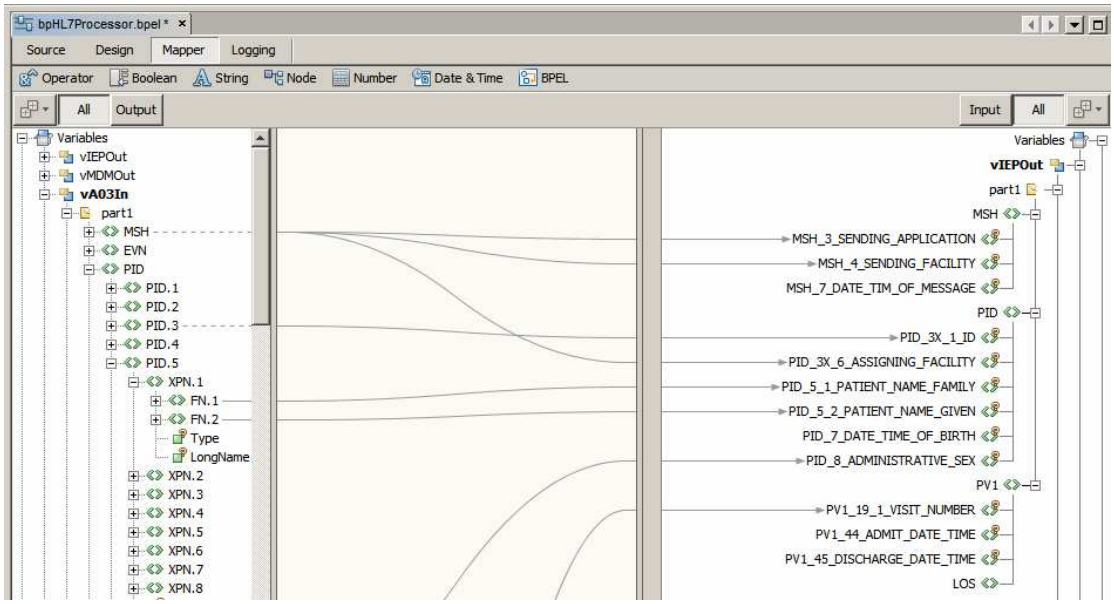


Figure 0-167 IEPCustomDischarge mapping rules fragment

Switch to Design view. Notice the warning icon for the Assign2 activity as well. This, too, is benign.

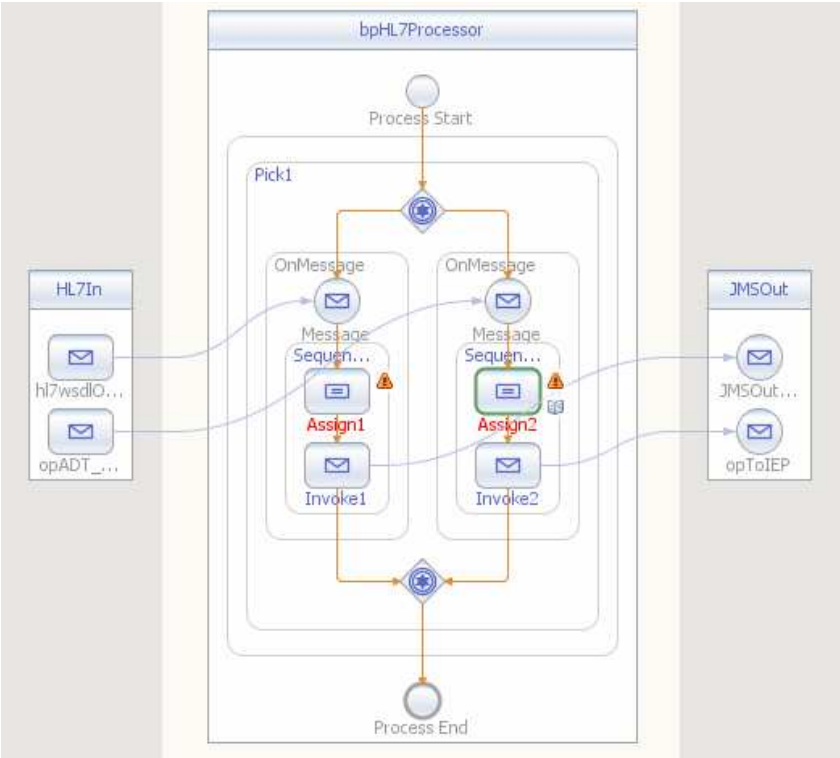


Figure 0-168 Business process so far

Since the output fields which we did not populate are optional we can actually build, deploy and exercise this solution as it is. We can, and will, come back to add the data manipulation and population rules later. Let's do that. Build the project.

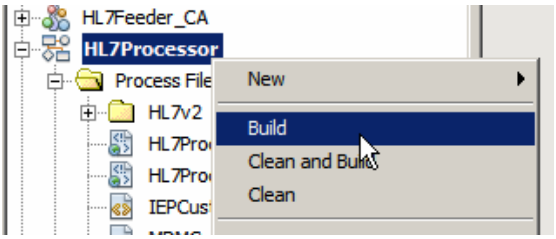


Figure 0-169 Build the project

If you get BUILD SUCCESSFUL message at the end of the process then all is well – ignore warning messages. If the build fails then investigate and resolve ERROR and SEVERE issues and build again.

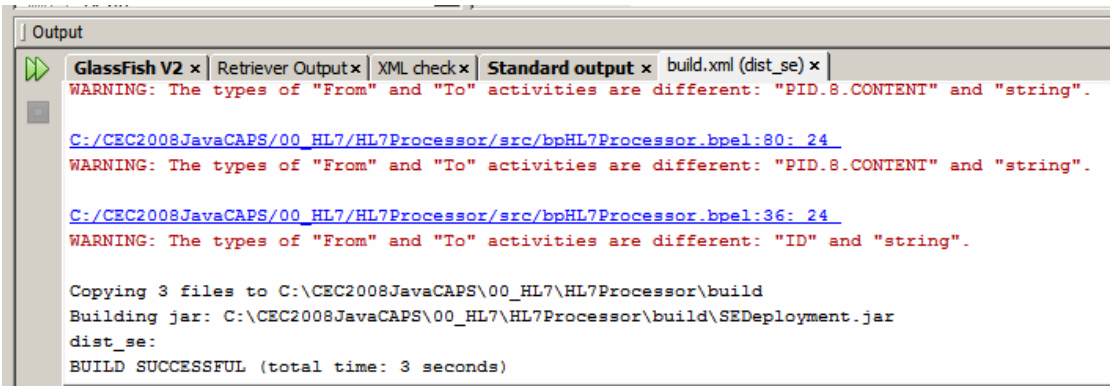


Figure 0-170 All is well

As before, when developing JBI solutions, we must have a Composite Application to which to add the BPEL module we just developed. Let's create a New Composite Application called HL7Processor_CA.

When the Composite Application Service Assembly editor windows opens, drag the HL7Processor module onto the JBI Modules part of the canvas.

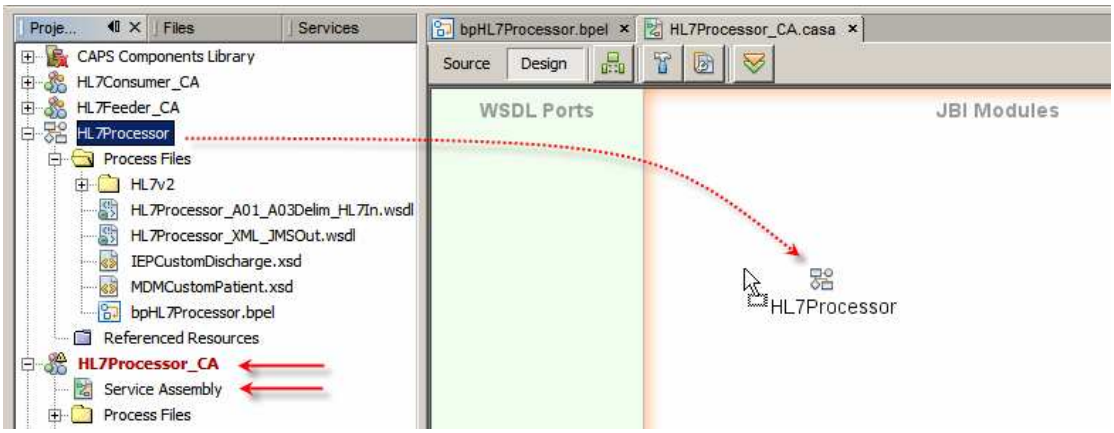


Figure 0-171 Add JBI Module to the Composite Application

Build the project.

Before deploying the Composite Application make sure that the HL7Consumer composite application we built earlier is not deployed. If it is, the HL7Processor will fail to start because the HL7 BC in the HL7Processor wants to use the same TCP port as that already used by the HL7Consumer.

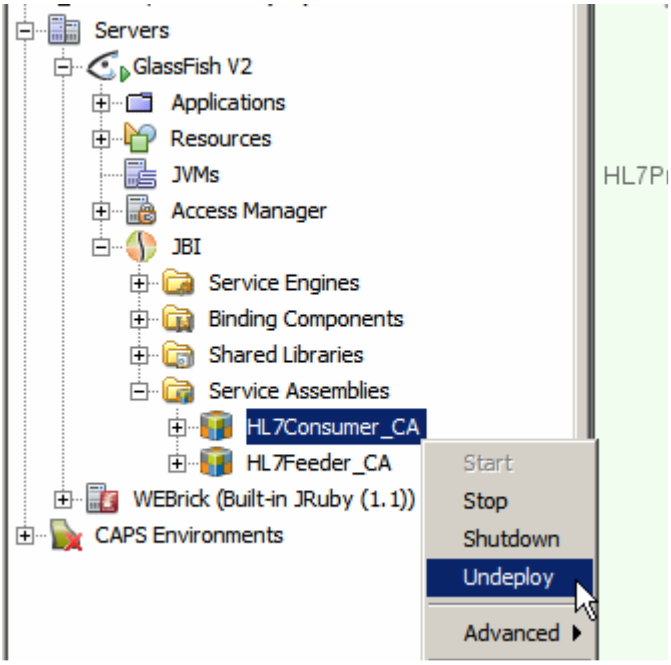


Figure 0-172 Make sure to undeploy the HL7Consumer

Deploy the Composite Application.

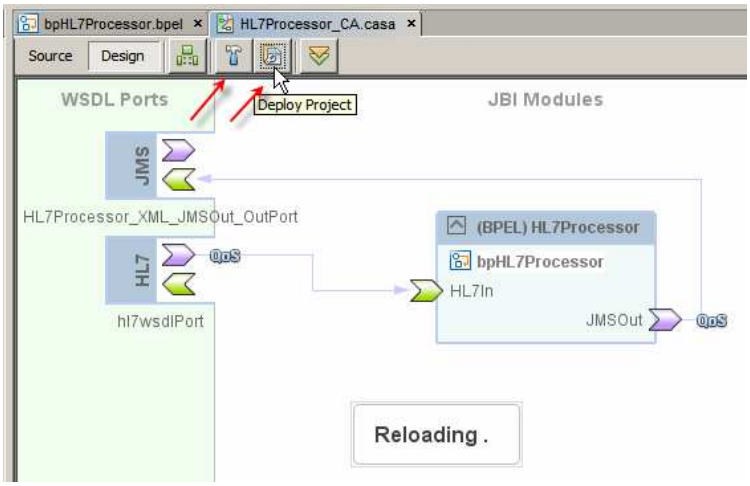


Figure 0-173 Deploying the Composite Application

We are now in a position to test the HL7Processor to see if our binding components are correctly configured and the mapping so far is correct.

In the normal course of events, in the absence of Atomic Transaction on the BPEL process or QoS limits, the HL7 BC will receive a message, deliver it to the BPEL process and wait for the next message regardless of whether the BPEL process completed processing the message it just received. With many messages being delivered there may be a number of concurrently executing BPEL process instances, each processing a different message. It is possible that messages will be processed and delivered to the downstream component in a different order to that in which they were received y the HL7 BC. In Healthcare this is typically undesirable. Let's ensure serialisation of messages by setting the "Quality of Service" (QoS) property Max Concurrency Limit to 1. Right-click on the link between the HL7 BC and the BPEL Process in the Composite Application Service Assembly editor and set the Max Concurrency Limit property to 1.

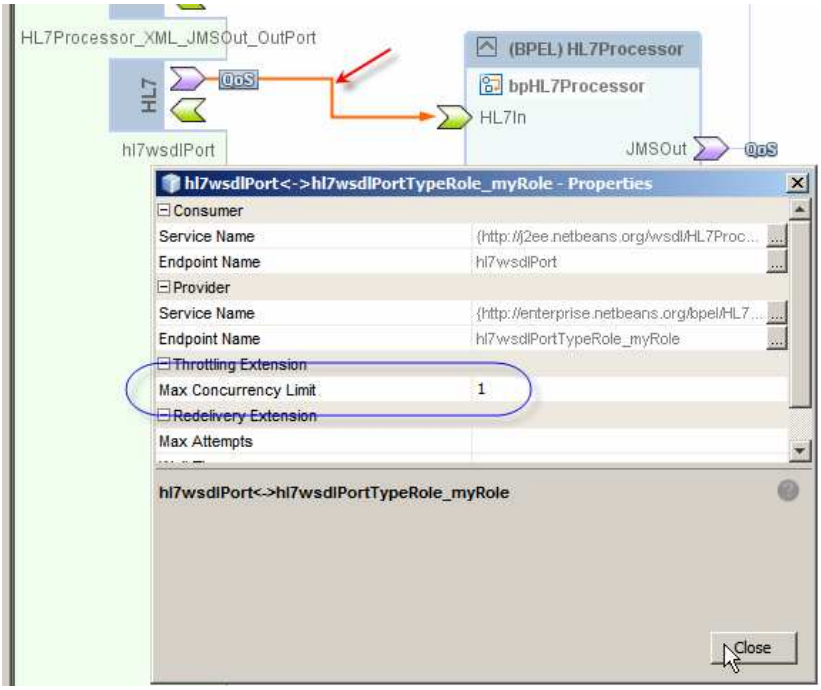


Figure 0-174 Set Max Concurrency Limit

Build and Deploy the composite application again.

Let's test both the A01 and A03 paths by copying the file ADT_A0x_output2.dat from **data/sources** directory to **data** directory. The HL7Feeder project will pick it up and deliver it to the HL7Processor's HL7 BC, which will process both messages in separate instances and will deposit each in the appropriate JMS Queue.

Let's confirm this by inspecting the Queues and messages in the queues using the Java CAPS Enterprise Manager.

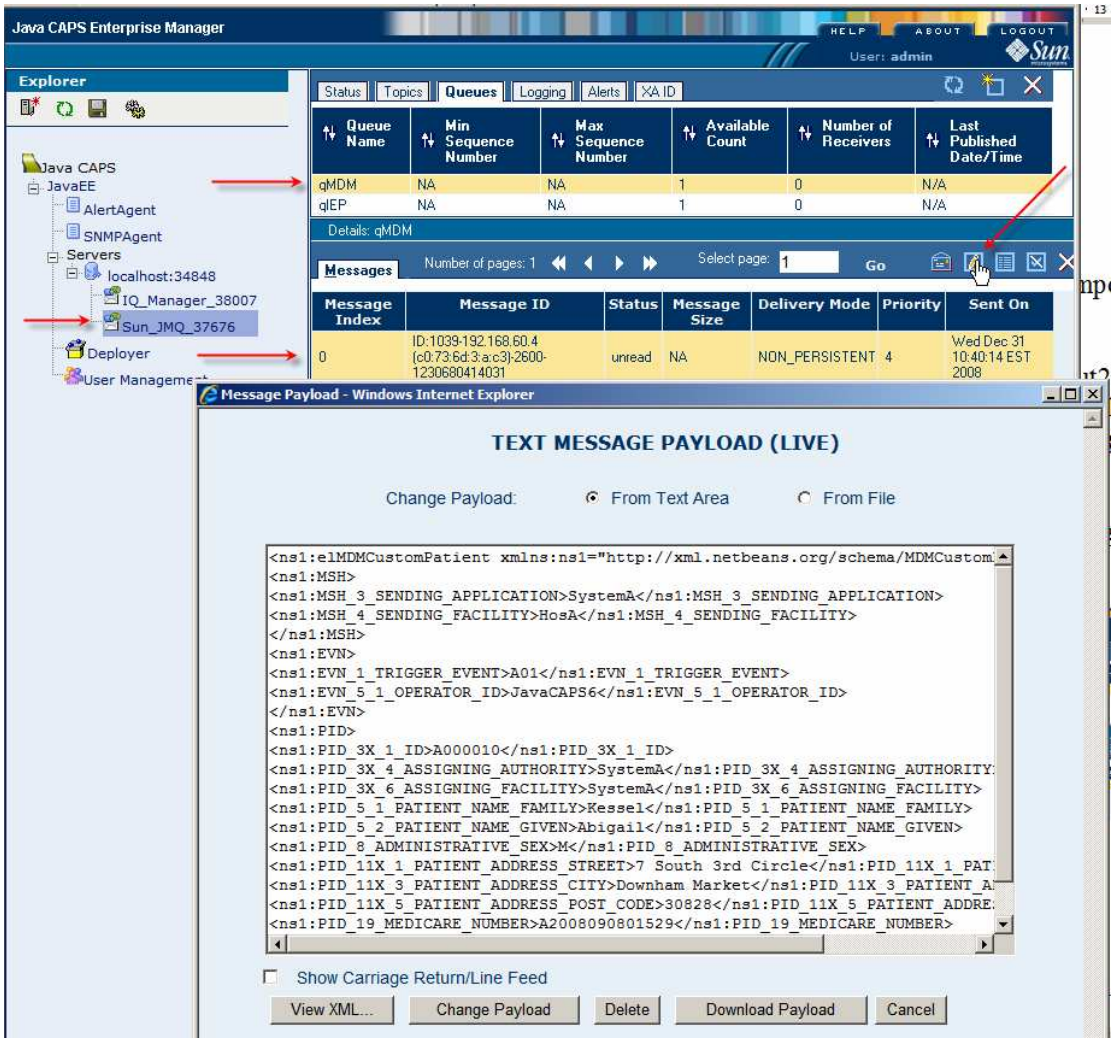


Figure 0-175 Looking at message in queues in Enterprise Manager

Indeed, there is 1 message in each queue, as indicated by Available Count of 1 in both queues. We can inspect the message itself by choosing the queue in the upper pane, choosing the message in the lower pane and clicking the View/Edit button.

> If you are implementing this solution using OpenESB you will not have the Enterprise Manager. You can use the Hermes JMS open source tool, discussed in http://blogs.sun.com/javacapsfieldtech/entry/using_hermes_jms_with_java, to view JMS destinations in the Java Message Queue instead.

Let’s now modify the BPEL Process by adding the Length of Stay calculation so that we can populate the LOS field in the IEPCustomDischarge message.

Let’s open the WSSDateDiff EJB Module and expand the hierarchy through the Web Services node. Right-click on the WSSDateDiff node and choose Test Web Service.

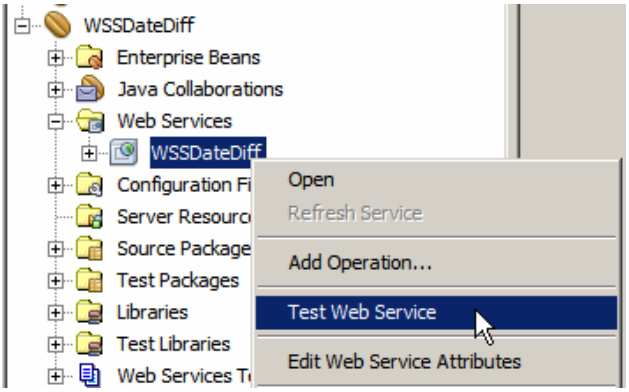


Figure 0-176 Choose test Web Service option

If the service is not deployed, deploy it.

Once the web browser window opens copy the URL pointed at by the (WSDL File) link.

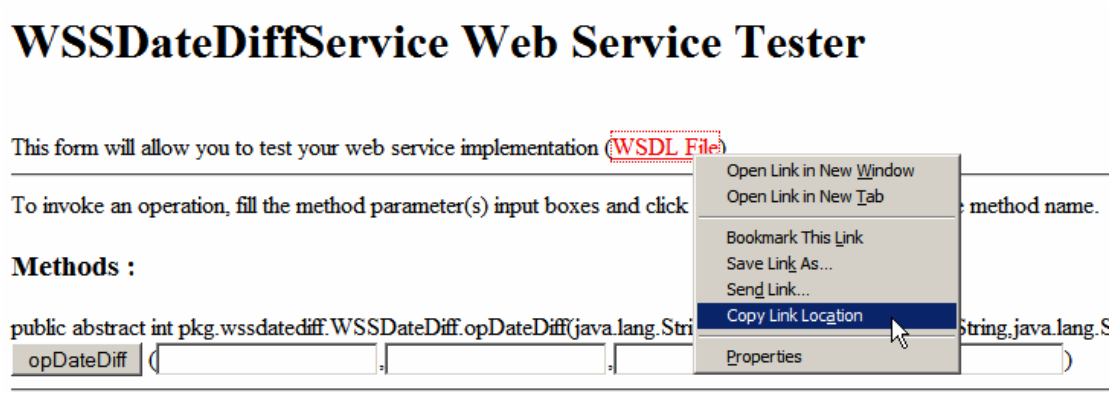


Figure 0-177 Copy the URL of the WSDL

Close the browser window.

Back in the HL7Processor project, right-click on the HL7Processor project name, choose New, choose Other, choose Web Services, choose External WSDL Document(s), paste the URL just copied into the ‘From URL:’ text box and click Finish.

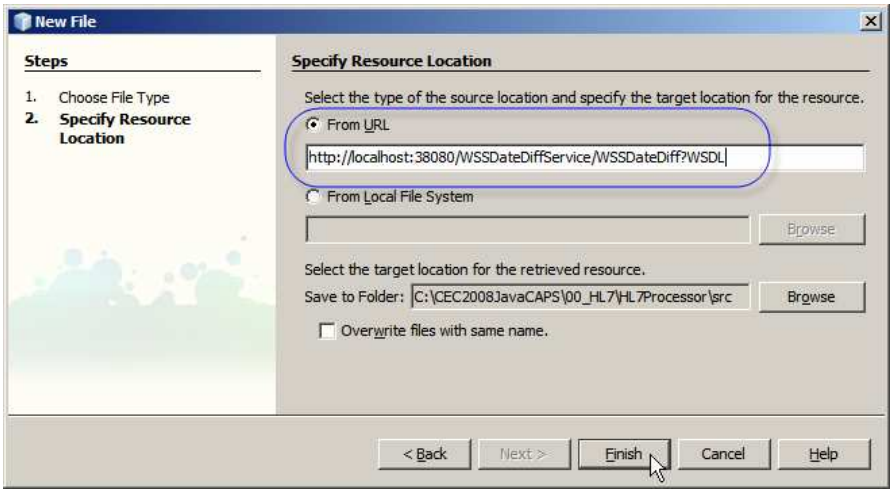


Figure 0-178 Create WSDL Document using WSDL URL

Repeat the process for the WSSConverDate web service – we will use it later.

A number of additional artefacts appeared in our HL7Processor projects. Note the two WSDL documents – we will use them shortly.

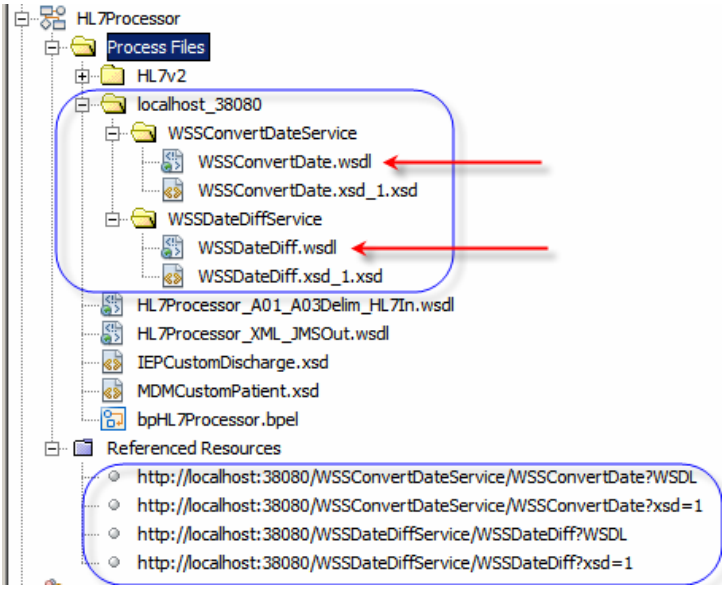


Figure 0-179 Additional artefacts in the project

Let's open the BPEL process and drag the WSSDateDiff onto the outbound, right hand swim line, much as we have done with the JMS BC WSDL before. Let's drop it at the target marker just above the JMSOut partner and name it DiffDate. Let's repeat the process with the WSSConvertDate service, dropping it onto the target marker just above the DiffDate partner and naming it CvtDate.

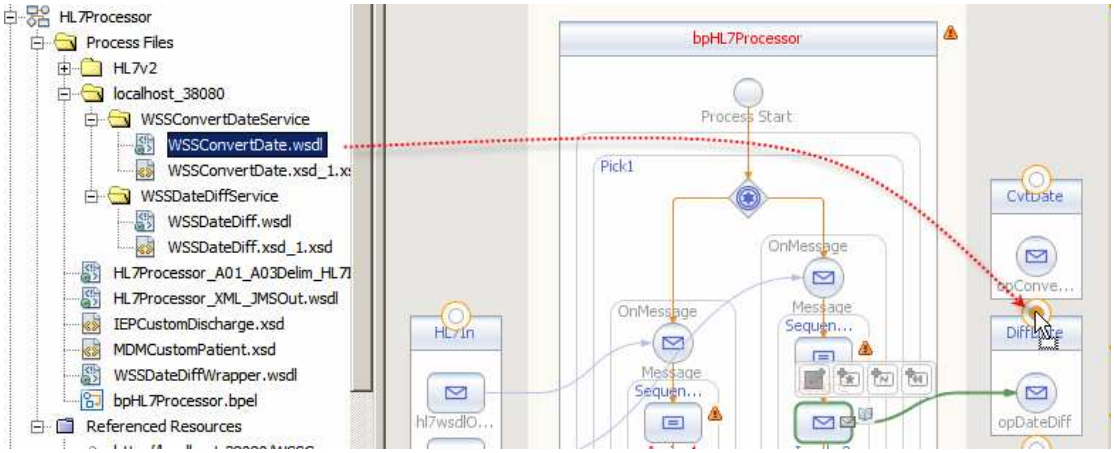


Figure 0-180 Add WSSConvertDate service to the process canvas

Let's add Invoke and Assign activities between the existing Assign2 and Invoke2 activities in the A03 OnMessage stream, connect the Invoke3 to the DateDiff service and create two variables, Input Variable vDIffReq and Output Variable vDiffRes.

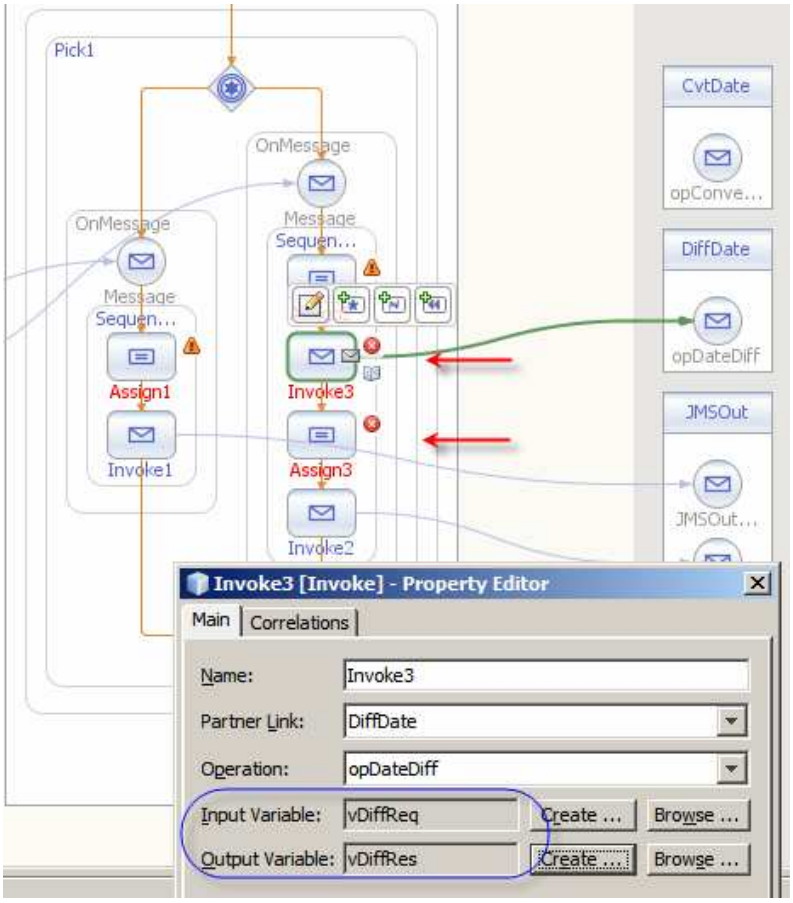


Figure 0-181 Configure Invoke3

Let's use the Assign2 mapping to map values to the input of the WSSDateDiff service.

Map vA03In>part1>PV1>PV1.44>TS.1 to vDiffReq>parameters>sEarlierDate and a string literal “yyyyMMddhhmmss” to vDiffReq>parameters>sEarlierDateFormat. This is the Admission Date.

Map vA03In>part1>PV1>PV1.45>TS.1 to vDiffReq>parameters>sLaterDate and a string literal “yyyyMMddhhmmss” to vDiffReq>parameters>sLaterDateFormat. This is the Discharge Date.

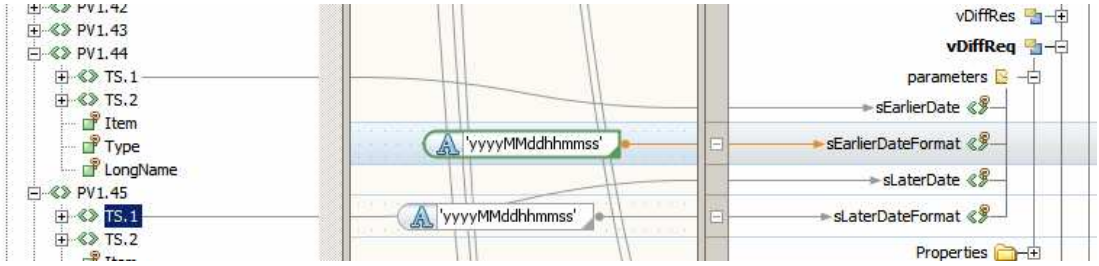


Figure 0-182 Map inputs of the WSSDateDiff service

Switch to Design view, double-click the Assign3 activity and map vDiffRes>parameters>return field to vIEPOut>part1>PV1>LOS field. The value of the return from the WSSDateDiff with is the number of days between the two given dates – the length of stay in our case.

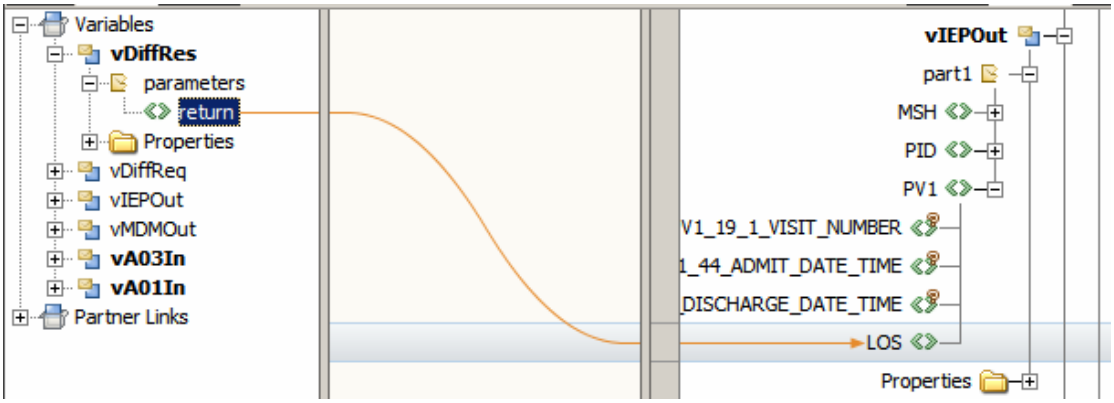


Figure 0-183 Map return from WSSDateDiff to the LOS field.

To complete the IEP side of things we must convert four dates from HL7 to ISO8601 format:

- MSH_7_DATE_TIME_OF_MESSAGE_ISO8601
- PID_7_DATE_TIM_OF_BIRTH
- PV1_44_ADMIT_DATE_TIME
- PV1_45_DISCHARGE_DATE_TIME

This requires us to add one Invoke and one Assign activity for each conversion.

Connect Invoke4, Invoke5, Invoke6 and Invoke7 to the CvtDate partner.

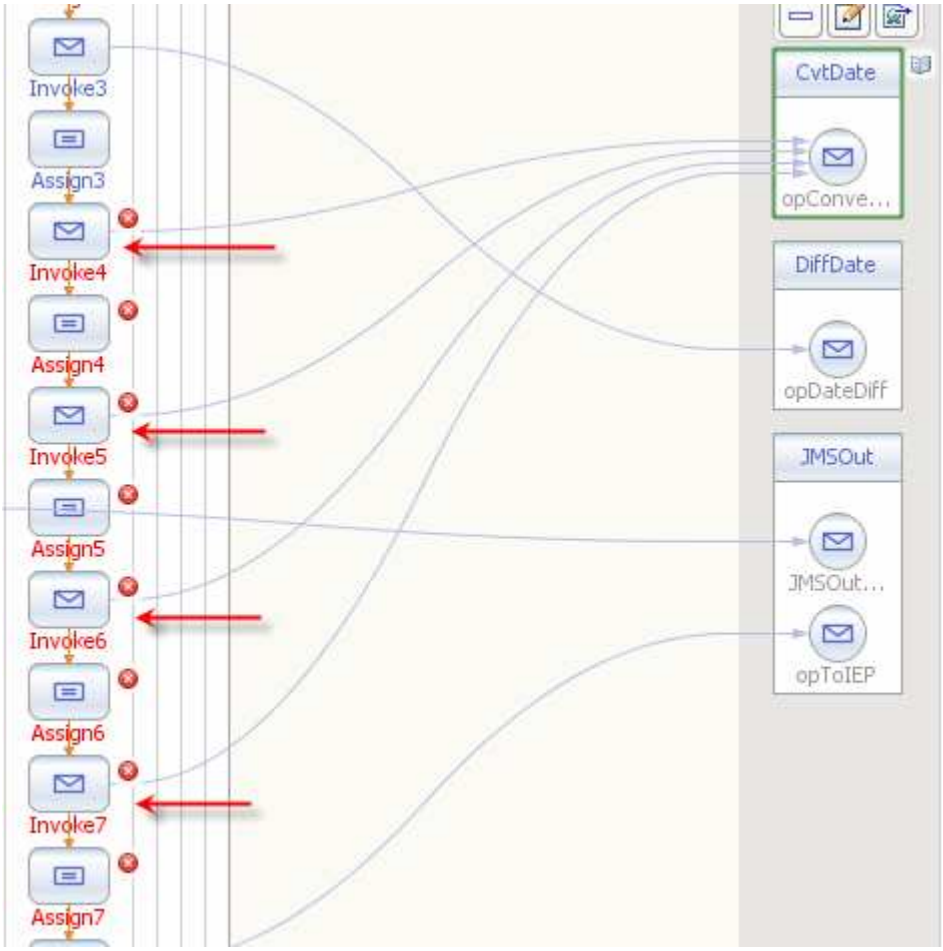


Figure 0-184 Connect Invoke4 through Invoke7 to CvtDate

Edit properties of each Invoke4 through Invoke7 and add an Input Variable and an Output Variable. Name them vDTOfMsgReq and vDTOfMsgRes (Invoke4), vDOBReq and vDOBRes (Invoke5), vDTOfAdmitReq and vDTOfAdmitRes (Invoke6), vDTOfDischReq and vDTOfDischRes (Invoke7), respectively

As you can gather Invoke4 will be used to convert MSH_7_DATE_TIME_OF_MESSAGE_ISO8601, Invoke5 will be used to convert PID_7_DATE_TIM_OF_BIRTH, Invoke6 will be used to convert PV1_44_ADMIT_DATE_TIME and Invoke7 will be used to convert PV1_45_DISCHARGE_DATE_TIME.

Select Assign3, switch to Mapper view and map vA03In>part1>MSH>MSH.7>TS.1 to vDTOfMsgReq>msgRequest>sDateTimeIn. Assign literal string “yyyyMMddhhmmss” to vDTOfMsgReq>msgRequest>sDateTimeInFormat and “yyyy-MM-dd'T'hh:mm:ss” to vDTOfMsgReq>msgRequest>sDateTimeOutFormat. Note the single quotes surrounding the capital letter T. This provides inputs to the WSSConvertDate.

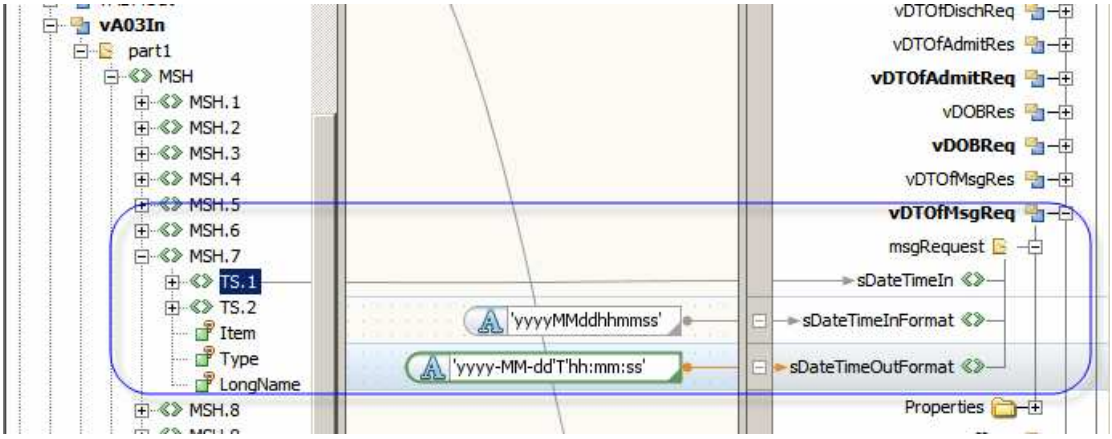


Figure 0-185 Input to WSSConvertDate converting MSH-7 Date and Time of Message

Switch to Design view, select Assign4 and switch to Mapper view.

Let's first assign the output of the previous invocation of the WSSConvertDate service, vDTOfMsgRes>msgResponse>sDateTimeOut to the vIEPOut>part1>MSH>MSH_7_DATE_TIM_OF_MESSAGE field.

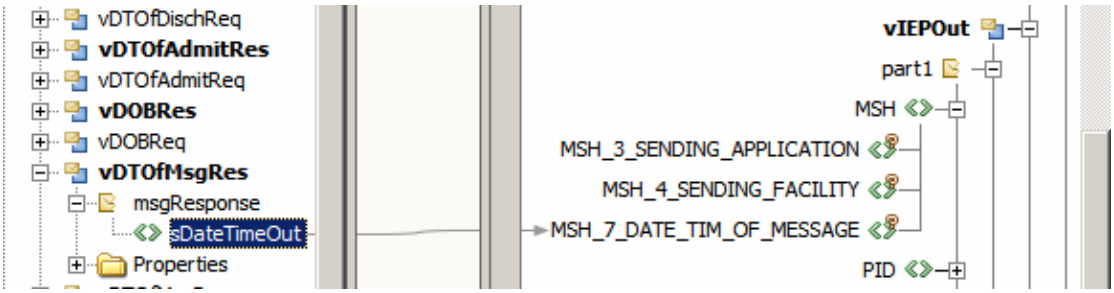


Figure 0-186 populate MSH-7 Date Time of Message field

Still with Assign4 selected, in the Mapper, let's map the inputs for the vDOBReq . Assign vA03In>part1>PID>PID.7>TS.1 to vDOBReq>msgRequest>sDateTimeIn. Assign literal string "yyyyMMddhhmmss" to vDOBReq >msgRequest>sDateTimeInFormat and "yyyy-MM-dd'T'hh:mm:ss" to vDOBReq >msgRequest>sDateTimeOutFormat. Note the single quotes surrounding the capital letter T. This provides inputs to the WSSConvertDate.

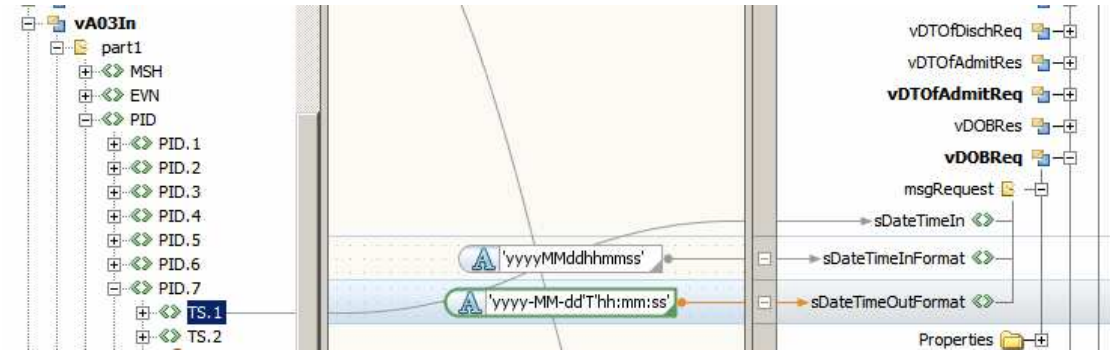


Figure 0-187 Populate inputs for CvtDate partner to convert date of Birth to ISO8601

Switch to Design view, select the Assign5 activity, switch to Mapper view and map the output of the previous invocation of WSSConvertDate,

vDOBRes>msgResponse>sDateTimeOut to vIEPOut>part1>PID>PID_7_DATE_TIME_OF_BIRTH.

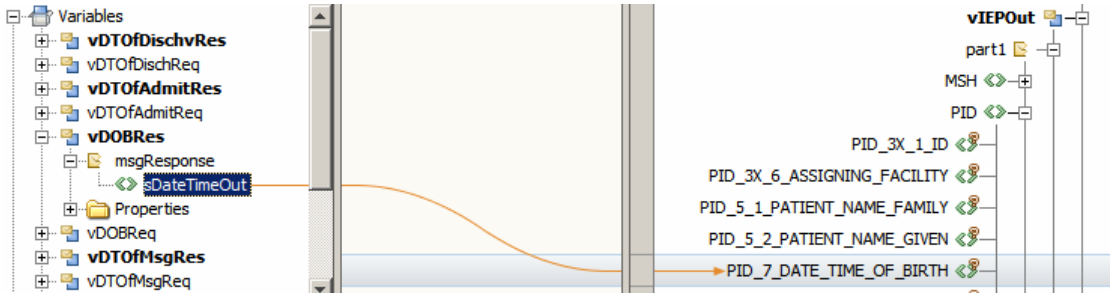


Figure 0-188 Map conversion result to the Date of Birth field

Still with Assign5 selected, in the Mapper, let's map the inputs for the vDTOfAdmitReq. Assign vA03In>part1>PV1>PV1.44>TS.1 to vDTOfAdmitReq>msgRequest>sDateTimeIn. Assign literal string "yyyyMMddhhmmss" to vDTOfAdmitReq>msgRequest>sDateTimeInFormat and "yyyy-MM-dd'T'hh:mm:ss" to vDTOfAdmitReq>msgRequest>sDateTimeOutFormat. Note the single quotes surrounding the capital letter T. This provides inputs to the WSSConvertDate.

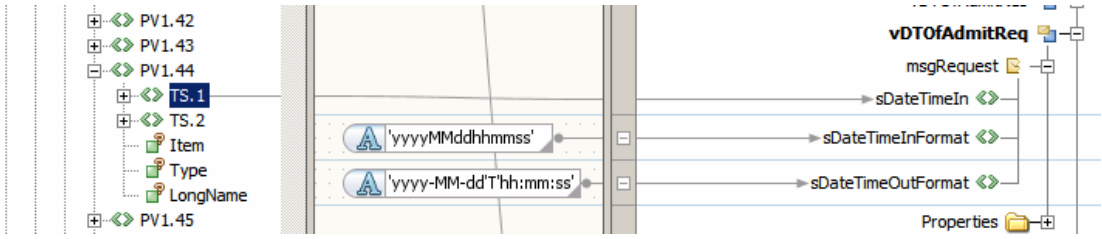


Figure 0-189 Map inputs to convert Admission Date

Switch to Design view, select the Assign6 activity, switch to Mapper view and map the output of the previous invocation of WSSConvertDate, vDTOfAdmitRes>msgResponse>sDateTimeOut to vIEPOut>part1>PV1>PV1_44_ADMIT_DATE_TIME.

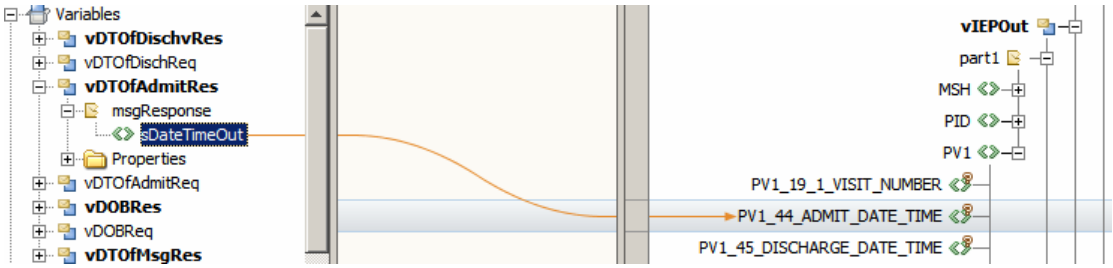


Figure 0-190 Maps output of WSSConvertDate to Admission Date field

Still with Assign6 selected, in the Mapper, let's map the inputs for the vDTOfDischReq. Assign vA03In>part1>PV1>PV1.45>TS.1 to vDTOfDischReq>msgRequest>sDateTimeIn. Assign literal string "yyyyMMddhhmmss" to vDTOfDischReq>msgRequest>sDateTimeInFormat and "yyyy-MM-dd'T'hh:mm:ss" to vDTOfDischReq>msgRequest>sDateTimeOutFormat. Note the single quotes surrounding the capital letter T. This provides inputs to the WSSConvertDate.

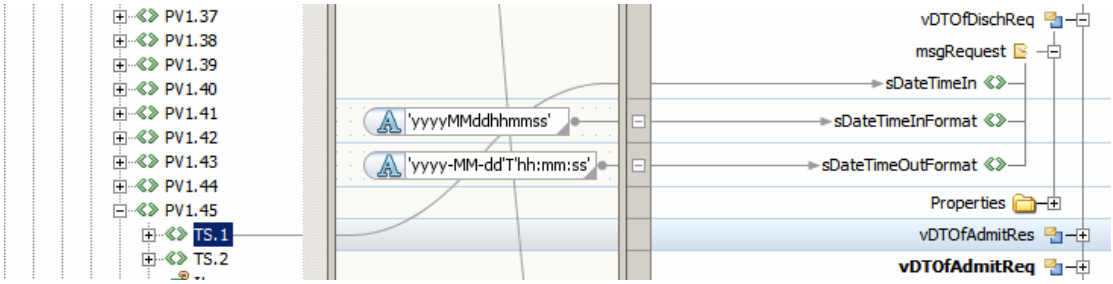


Figure 0-191 Map inputs for WSSConvertDate to convert Discharge Date to ISO8601

Finally, switch to Design view, select the Assign7 activity, switch to Mapper view and map the output of the previous invocation of WSSConvertDate, vDTOfDischRes>msgResponse>sDateTimeOut to vIEPOut>part1>PV1>PV1_45_DISCHARGE_DATE_TIME.

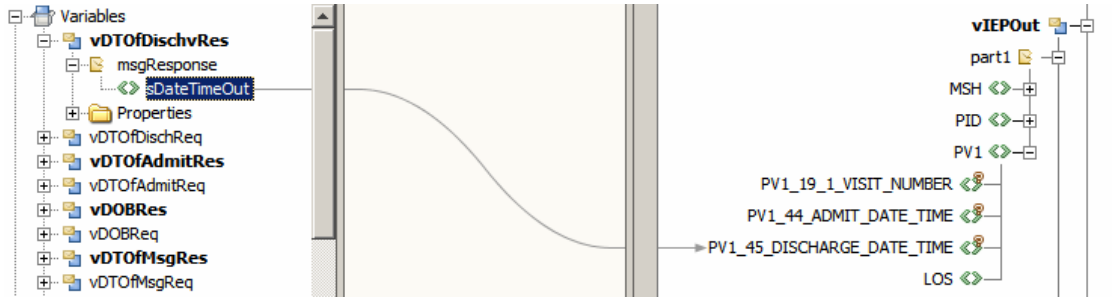


Figure 0-192 Map converted Discharge Date

Notice Error icons disappearing from the process in Design view. The BPEL editor progressively validates the process as we modify it.

Before converting 3 dates in the MDM part of the process let's build the process and build the composite application. Notice, when the composite application is built, that we have additional Consume endpoints in the BPEL Process in the CASA map and additional SOAP Binding Components to which they are connected.

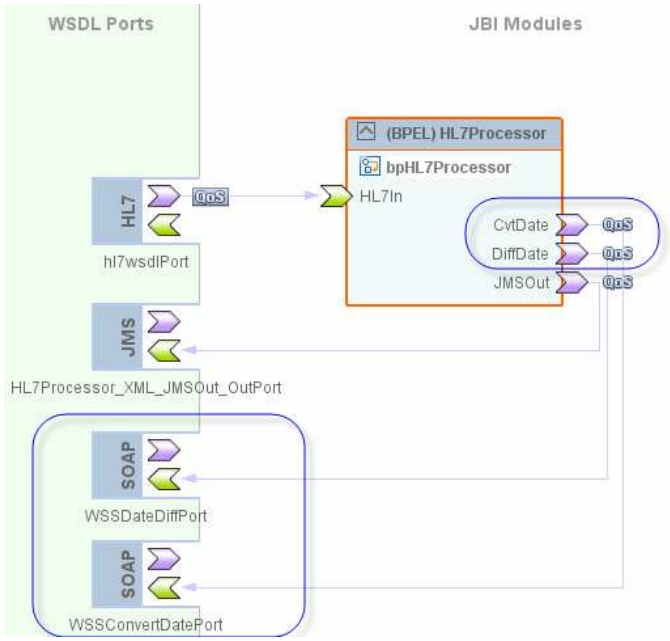


Figure 0-193 Changed CASA map

We will leave it as is for the time being. Later we will add the EJB Web Services modules to the CASA map to allow them to be invoked through the good offices of the JavaEE Service Engine using the in-memory mechanism and avoid SOAP over HTTP inefficiency.

For now, let's deploy the composite application and submit the same test file with two HL7 delimited transactions to see what the IEPCustomDischarge message looks like. It should have the LOS field and the date/time fields populated. By exercising the process at this point we will be validating the work we have done just now.

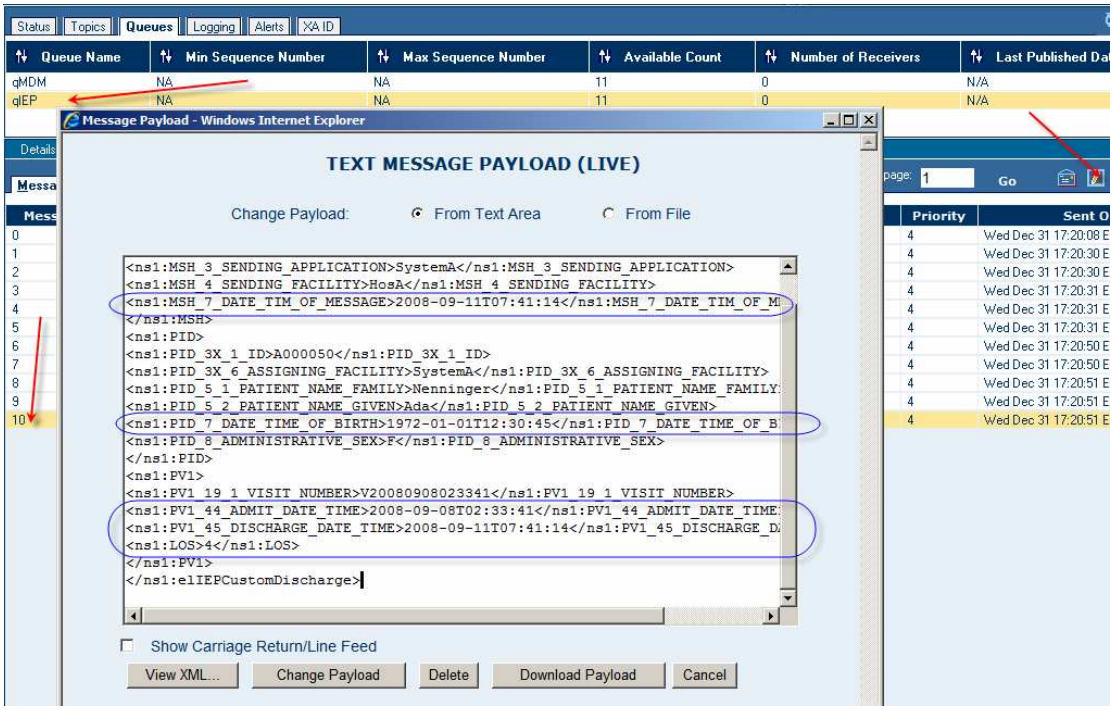


Figure 0-194 Dates were converted and Length of Stay was calculated

Let's turn our attention back to the bpHL7Processor BPEL process in order to complete the MDM-related logic stream.

To complete the MDM logic stream me must convert and map the following date/time fields:

- MSH_7_DATE_TIME_OF_MESSAGE_ISO8601
- PID_7_DATE_TIM_OF_BIRTH

Add two pairs of Invoke and Assign activities between Assign1 and Invoke1 on the MEM branch of the Pick activity.

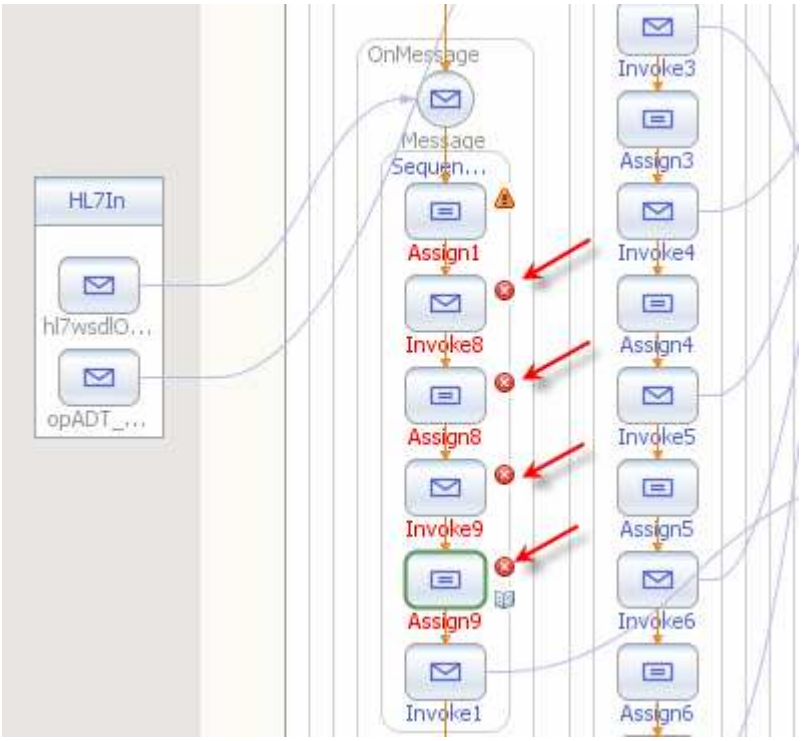


Figure 0-195 Add two pairs of Invoke and Assign activities

Connect Invoke8 and Invoke9 to the CvtDate partner.

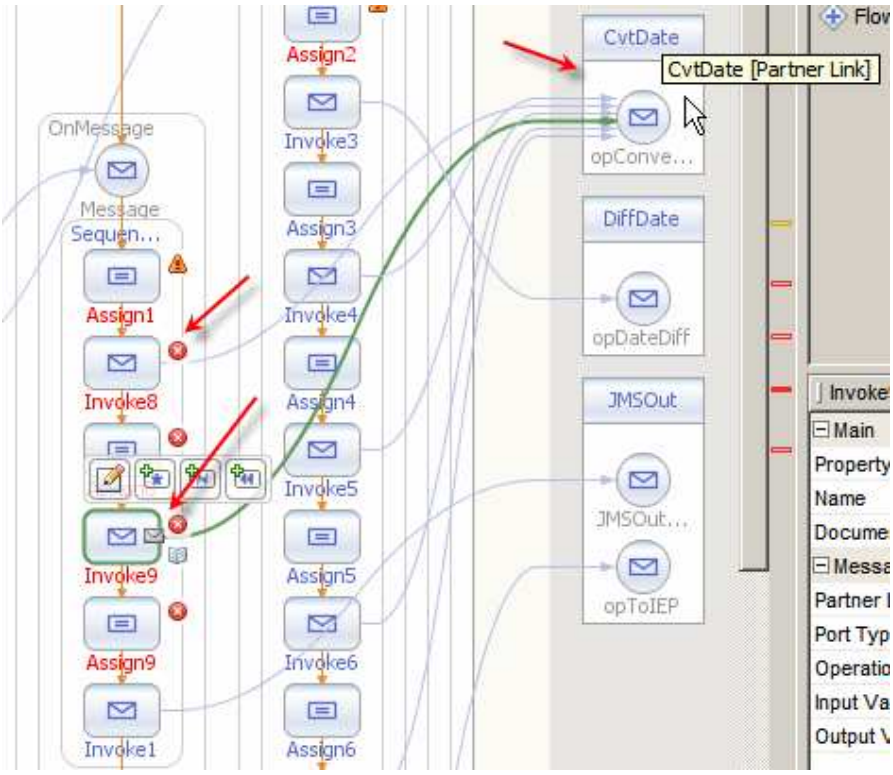


Figure 0-196 Connect Invoke8 and Invoke9 to CvtDate

Edit properties of Invoke8 and Invoke9, add Input Variable and Output Variable to each by browsing to the existing variables and choosing the appropriate variable for each – vDTOdMsgReq, vDTOfMsgRes (Invoke8), vDOBReq and vDOBRes (Invoke9).

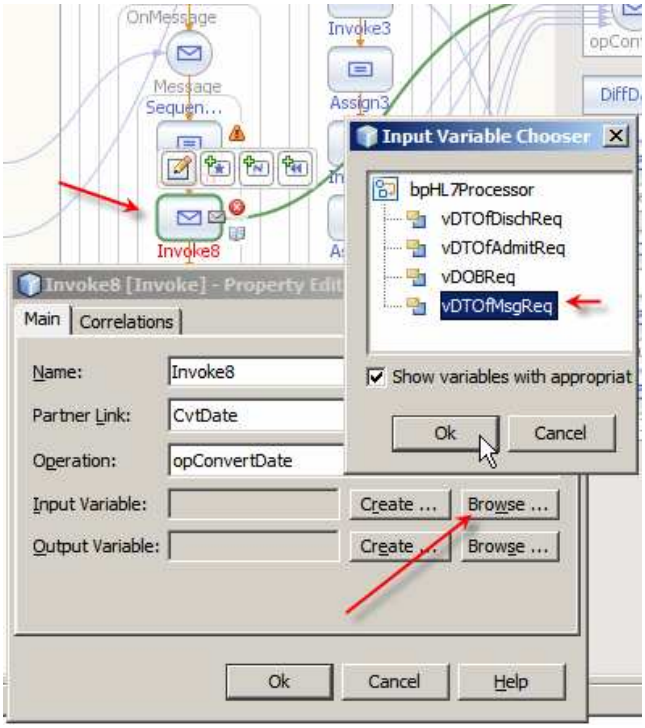


Figure 0-197 Choose existing variable vDTOfMsgReq as Input Variable

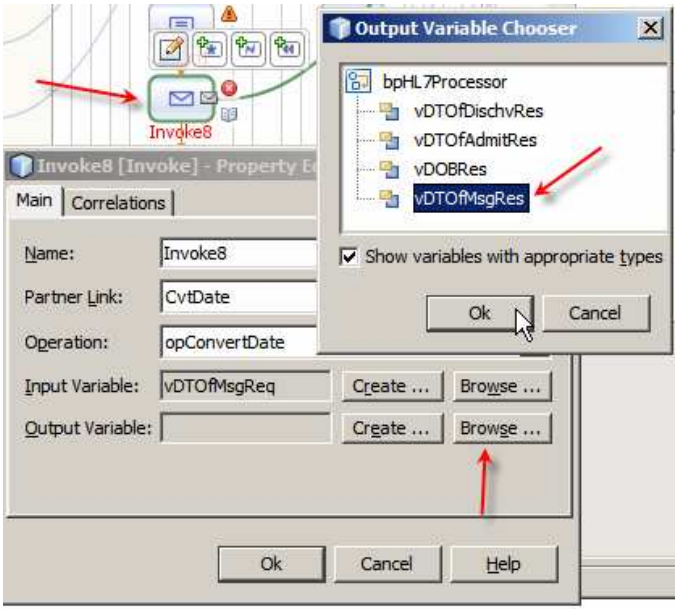


Figure 0-198 Choose existing variable vDTOfMsgRes as Output Variable

As you can gather Invoke8 will be used to convert MSH_7_DATE_TIME_OF_MESSAGE_ISO8601 and Invoke9 will be used to convert PID_7_DATE_TIM_OF_BIRTH.

Select Assign1, switch to Mapper view and map vA01In>part1>MSH>MSH.7>TS.1 to vDTOfMsgReq>msgRequest>sDateTimeIn. Assign literal string “yyyyMMddhhmmss” to vDTOfMsgReq>msgRequest>sDateTimeInFormat and “yyyy-MM-dd'T'hh:mm:ss” to vDTOfMsgReq>msgRequest>sDateTimeOutFormat.

Note the single quotes surrounding the capital letter T. This provides inputs to the WSSConvertDate.

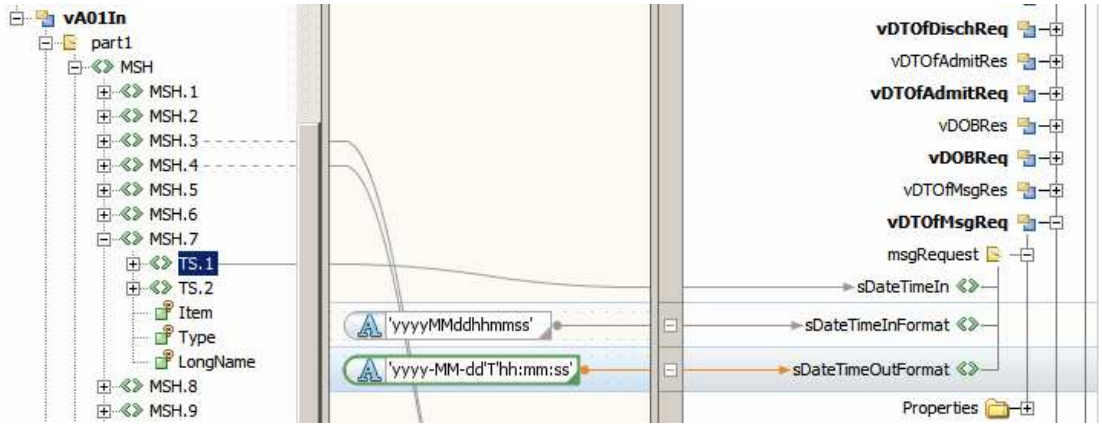


Figure 0-199 Map inputs to CvtDate for MSH-7- Date_Time_of_Message

Switch to Design view, select Assign8 and switch to Mapper view.

Let's first assign the output of the previous invocation of the WSSConvertDate service, vDTOfMsgRes>msgResponse>sDateTimeOut to the vMDMOut>part1>MSH>MSH_7_DATE_TIM_OF_MESSAGE field.

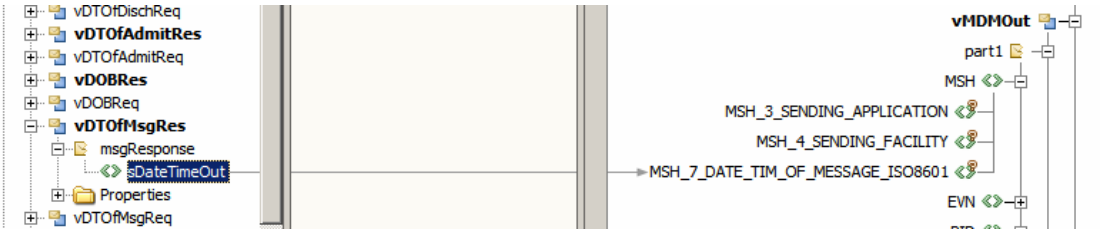


Figure 0-200 Assign converted date to _7_DATE_TIM_OF_MESSAGE

Still with Assign8 selected, in the Mapper, let's map the inputs for the vDOBReq. Assign vA01In>part1>PID>PID.7>TS.1 to vDOBReq>msgRequest>sDateTimeIn. Assign literal string "yyyyMMddhhmmss" to vDOBReq>msgRequest>sDateTimeInFormat and "yyyy-MM-dd'T'hh:mm:ss" to vDOBReq>msgRequest>sDateTimeOutFormat. Note the single quotes surrounding the capital letter T. This provides inputs to the WSSConvertDate.

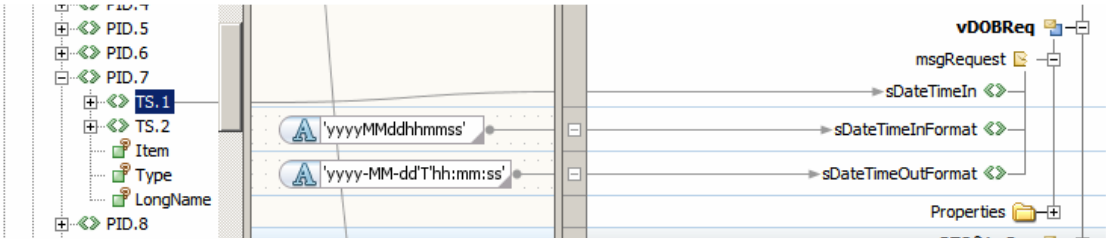


Figure 0-201 Provide inputs to convert Date of Birth

Switch to Design view, select the Assign9 activity, switch to Mapper view and map the output of the previous invocation of WSSConvertDate, vDOBRes>msgResponse>sDateTimeOut to vMDMOut>part1>PID>PID_7_DATE_TIME_OF_BIRTH.

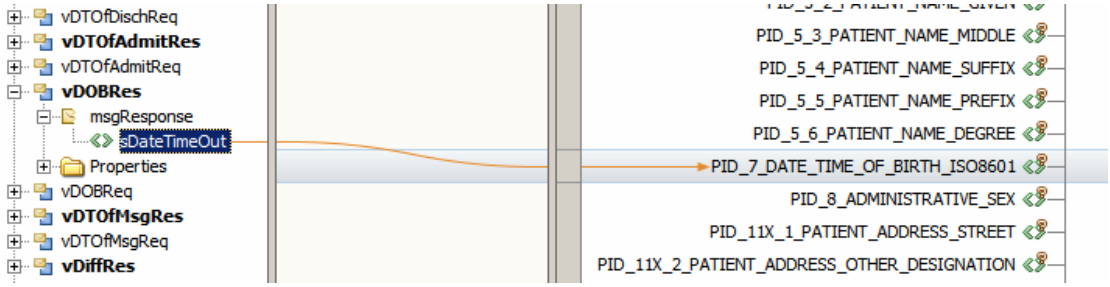


Figure 0-202 Populate vMDMOut>part1>PID>PID_7_DATE_TIME_OF_BIRTH

We are done developing BPEL logic. Let's build the process and build the composite application.

Notice, when the composite application is built, that there are no additional Consume endpoints in the BPEL Process in the CASA map, or additional SOAP Binding Components. We are reusing the same services.

Let's deploy the composite application and submit the same test file with two HL7 delimited transactions to see what the MDMCustomPatient message looks like. It should have the date/time fields populated. By exercising the process at this point we will be validating the work we have done just now.

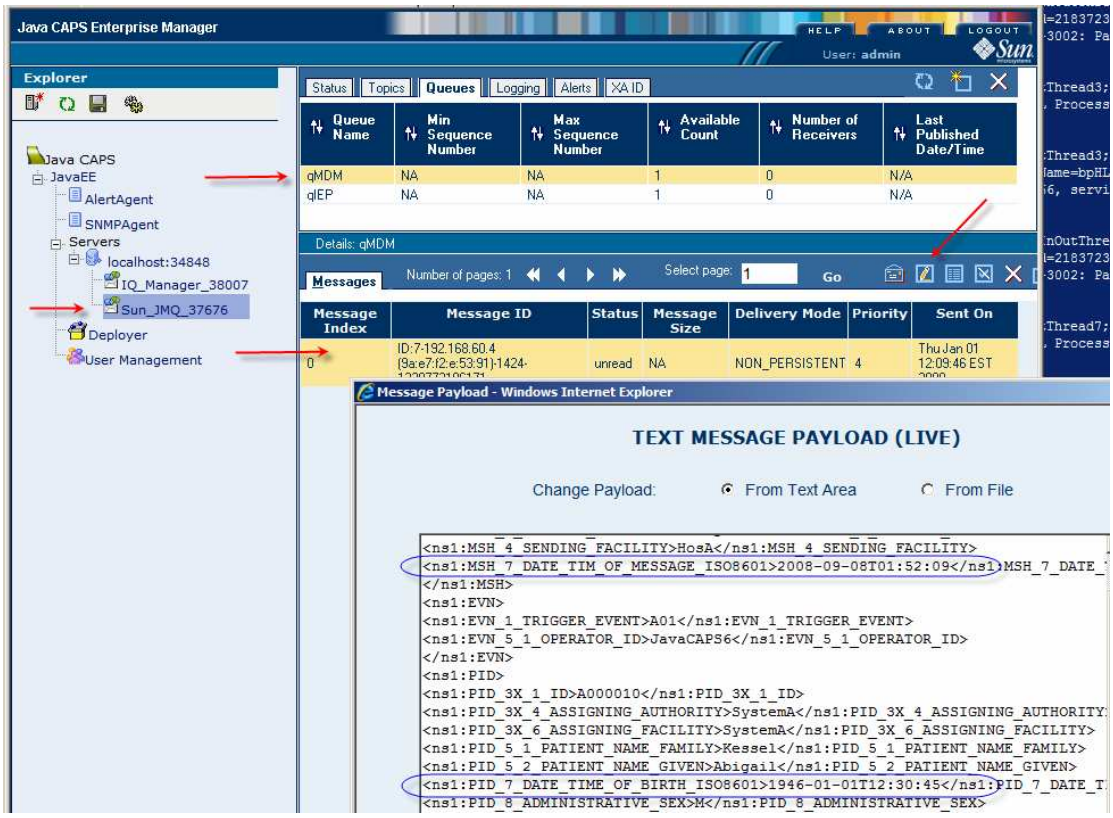


Figure 0-203 Dates, in IS)8601 format, in the MDMCustomPatient message

All is well. The HL7Processor solution is finished.

We could leave it at that, with the EJB-based web services being invoked from the BPEL process through the SOAP/HTTP BC using the SOAP over HTTP protocol. We can optimise this communication by including the EJBs in the Composite

Application Service Assembly, removing SOAP/HTTP BCs and taking advantage of the in-memory communication mechanism between the JBI container and the JEE container using the JavaEE Service Engine.

Let's open the HL7Processor_CA's Service Assembly and drag the WSSDateDiff and WSSConvertDate modules onto the JBI Modules part of the canvas.

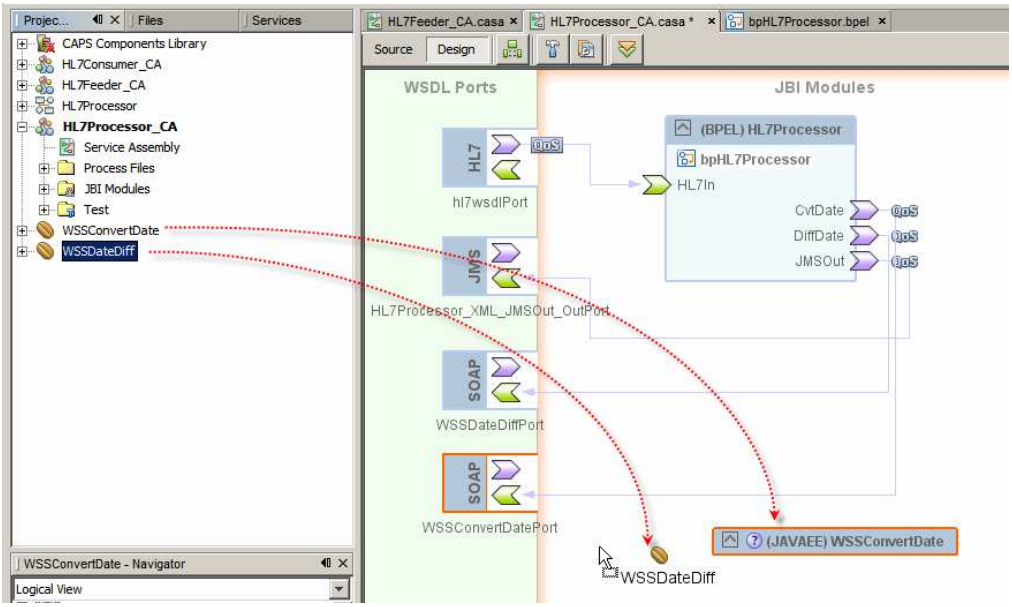


Figure 0-204 Add EJB web services to the Service Assmebly

Now build the Composite Application. Notice that the EJB Modules' Provide endpoints got connected to the corresponding Consume endpoints in the BPEL Process and that the SOAP/HTTP BCs got connected on both Consume and Provide sides.

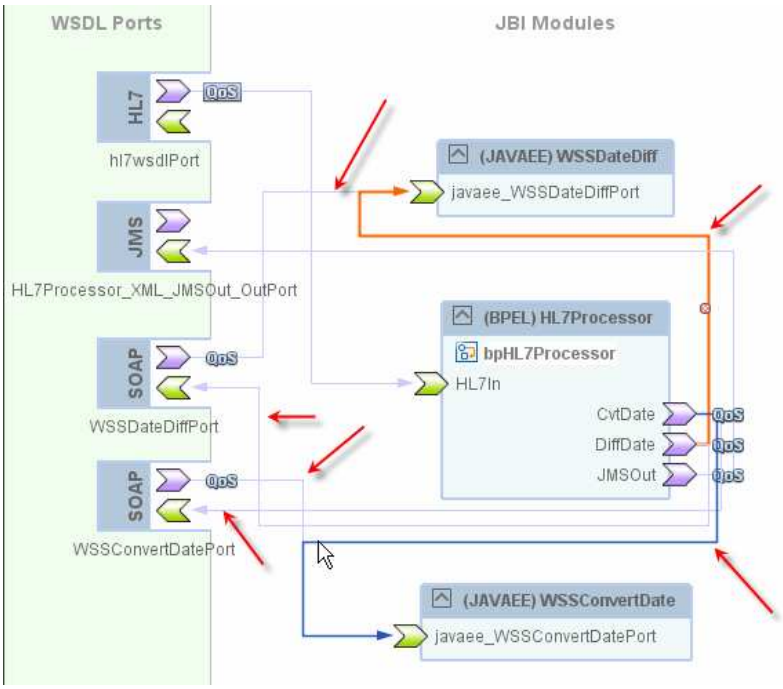


Figure 0-205 Service Assembly rebuilt after addition of EJB Modules

We can now delete the SOAP BCs from the canvas and build the Composite Application again. Select each SOAP BC in turn and press the Del key. Build and deploy the Composite Application when both are deleted.

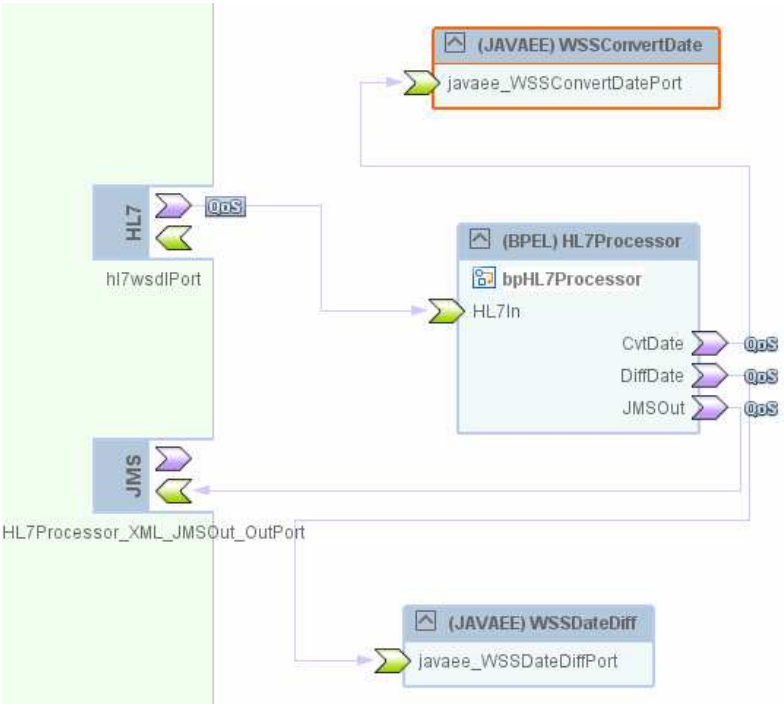


Figure 0-206 Final Composite Application with EJB-based Web Services using the JavaEE SE

Let's submit the test file again to verify that things are working as expected.

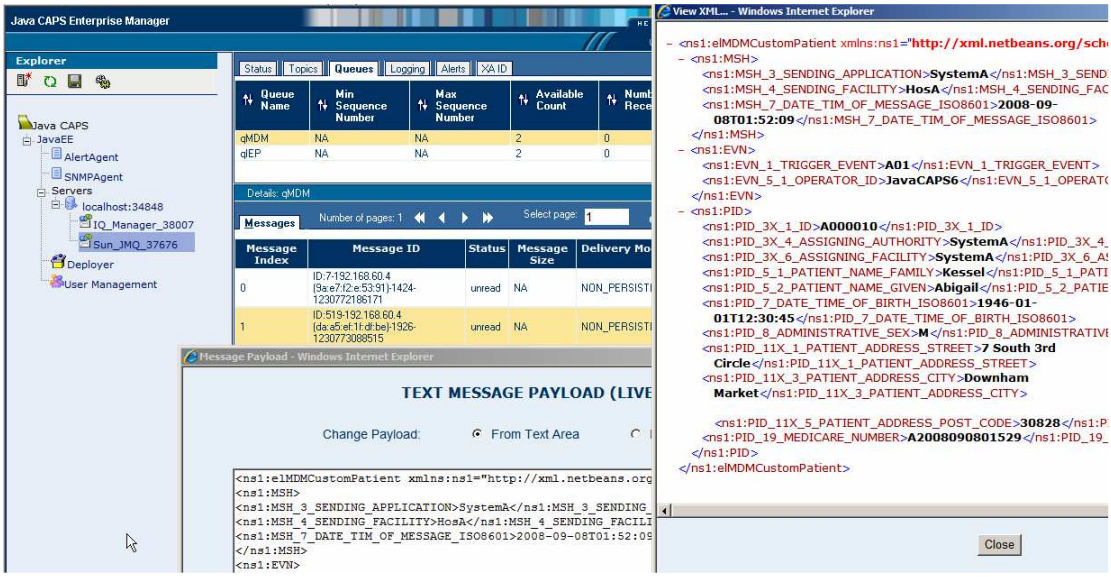


Figure 0-207 Two messages of each kind are available.

All done ☺

If you staid with me all this time and completed all projects – pat yourself on the back – you deserve it ☺

Summary

So, what have we seen and accomplished?

We used the HL7 Binding Component, the File Binding Component, the JMS Binding Component, the SOAP/HTTP Binding Component, the BPEL 2.0 Service Engine, the JavaEE Service Engine, the HL7 Encoder and EJB-based Web Services to solve a Healthcare-related business problem.

In the process we created XML Schema Documents (XSDs), Web Services Description Language Documents (WSDLs), a BPEL 2.0 Business Process, an EJB-based “Implementation First” web service, an EJB- and WSDL-based “Interface First” web service, a bunch of Composite Applications, BPLE 2.0 mapping, BPEL 2.0-based Web Service orchestration and on-the-fly conversion of HL7 version 2.3.1 delimited messages to their XML equivalents. We also got a pretty good exposure to what OpenESB and Java CAPS 6/JBI components look like, how they work and how they can be used to create real business solutions.