GlassFish ESB v 2.1
Creating a Healthcare Facility Web Service Provider

Michael.Czapski@sun.com
June 2009

Introduction

In some views SOA is represented as a series of 4 layers: Presentation Layer (SOA 1), Business Process Layer (SOA 2), Business Service Layer (SOA 3) and Technical Layer (SOA 4). Typically each layer higher up in the hierarchy consumes services exposed by the layer under it. So the Presentation Layer would consume services provided by the Business Process or Business Service Layers. Service interfaces are described using Web Services Description Language (WSDL), sheltering service consumers from details of service implementation. Web Services are seen as the technical means to implement the decoupled functional layers in a SOA development. Decoupling allows implementations of business functionality at different layers to be swapped in and out without disturbing other layers in the stack.

In this document I will walk through the process of developing a SOA Composite Application, exposed as a Web Service, which will make available simple business functionality using a multi-operation service. We will use the GalssFish ESB v2.1 infrastructure. The service will use the SOAP/HTTP Binding Component, the Database Binding Component and the BPEL 2.0 Service Engine. This simple service will introduce the components and discuss how a multi-operation web service can be constructed using the GalssFish ESB.

The business idea is that patients are looked after in various healthcare facilities. Frequently applications need to allow selection of a facility and to access facility details for display to human operators. A relational database is used to hold the details of facilities which are a part of the healthcare enterprise. To shelter application developers from the details of the data store facility list and details will be made available as a multi-operation web service. This web service will be used elsewhere to construct a portlet that can be used in an enterprise portal.

It is assumed that a GlassFish ESB v2.1-based infrastructure, supplemented by the Sun WebSpace Server 10 Portal functionality and a MySQL RDBMS instance, is available for development and deployment of the web service discussed in this paper. The instructions necessary to install this infrastructure are discussed in the blog entry "Adding Sun WebSpace Server 10 Portal Server functionality to the GlassFish ESB v2 .1 Installation" at http://blogs.sun.com/javacapsfieldtech/entry/adding_sun_webspace_server_10, supplemented by the material in blog entry "Making Web Space Server And Web Services Play Nicely In A Single Instance Of The Glassfish Application Server", at http://blogs.sun.com/javacapsfieldtech/entry/making_web_space_server_and.

Create MySQL objects

Our facility details will reside in a table in a relational database. Before the service can get access to that data we need to:

1. create the database
2. create the database table

3. populate the database table
4. create a NetBeans connection to the database and exercise it
5. create a runtime connection pool at the GalssFish App Server
6. create JNDI Reference to the connection pool

If you don't have the MySQL database installed you can obtain and install it following instructions in "MySQL Community Server and GUI Tools - Getting, Installing and Configuring", at http://blogs.sun.com/javacapsfieldtech/entry/mysql_community_server_and_gui.

The following text walks through the process of preparing data and connection pools for this writeup, a step at a time.

Start mysql command line client as the root user and execute the following commands:

```
drop user pblog;
drop database pblog;
create database pblog default character set utf8;
use pblog;
create user pblog;
grant all privileges on *.* to 'pblog'@'localhost' identified by 'pblog';
set password for 'pblog'@'localhost' = password('pblog');
```

This set of commands drops the pblog database and user, if they exist, and creates a new database and user for this document. Dropping the database destroys all the objects it may contain. One assumes you don't have a database pblog used for a different purpose.

Execute the following commands to create the ui_facility table:

```
DROP TABLE IF EXISTS `pblog`.`ui_facility`;
CREATE TABLE `pblog`.`ui_facility` (
  `facility_code` VARCHAR(5) BINARY NOT NULL,
  `description` VARCHAR(30) BINARY NULL,
  `status` VARCHAR(5) BINARY NULL,
  `addr1` VARCHAR(30) BINARY NULL,
  `addr2` VARCHAR(30) BINARY NULL,
  `city` VARCHAR(30) BINARY NULL,
  `state` VARCHAR(5) BINARY NULL,
  `post_code` VARCHAR(7) BINARY NULL,
  `country` VARCHAR(20) BINARY NULL,
  PRIMARY KEY (`facility_code`)
)
ENGINE = INNODB;
SET FOREIGN_KEY_CHECKS=1;
```

Execute the following commands to add sample data to the table:

```
-- Disable foreign key checks
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;

INSERT INTO `pblog`.`ui_facility`
    (`facility_code`
, `description`
, `status`
, `addr1`
, `addr2`
, `city`
, `state`
, `post_code`
, `country`)
VALUES
    ('STC', 'SYDNEY TECHNICAL HOSPITAL', 'A', '404 HUNTINGTON DRIVE', NULL,
'MONROVIA', 'CA', '91016', 'USA'),
  ('D210', 'Nepean Hospital', 'A', NULL, NULL, NULL, NULL, NULL, NULL),
  ('D230', 'Tresillian', 'A', NULL, NULL, NULL, NULL, NULL, NULL),
  ('D204', 'Blue Mountains Hospital', 'A', NULL, NULL, NULL, NULL, NULL, NULL),
  ('D754', 'Governor Phillip', 'A', NULL, NULL, NULL, NULL, NULL, NULL),
```

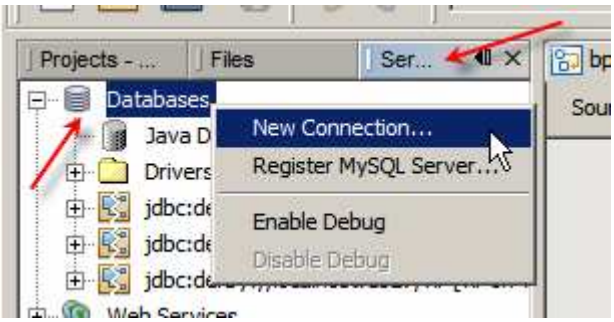
```
  ('D214', 'Springwood Hospital', 'A', NULL, NULL, NULL, NULL, NULL, NULL),
  ('ICPMR', 'ICPMR', 'A', NULL, NULL, NULL, NULL, NULL, NULL),
  ('BIGH', 'A BIG HOSPITAL', 'A', NULL, NULL, NULL, 'NS', '2000', NULL),
  ('ARMC', 'A RED MEDICAL CENTRE', 'A', 'SOMEWHERE ST', NULL, 'SOMEWHERE CITY',
'NSW', '2060', 'AU'),
  ('TBIGH', 'THE BIG HOSPITAL', 'A', 'WHOKNOWES WHERE ST', NULL, 'ANYWHERE
CITY', 'NSW', '2132', 'AU'),
  ('NSMC', 'NORTH SYDNEY MEDICAL CENTRE', 'A', '100 WALKER ST', NULL, 'NORTH
SYDNEY', 'NW', '2000', NULL);

-- Re-enable foreign key checks
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
```

Execute the following select statement to make sure the data is there:

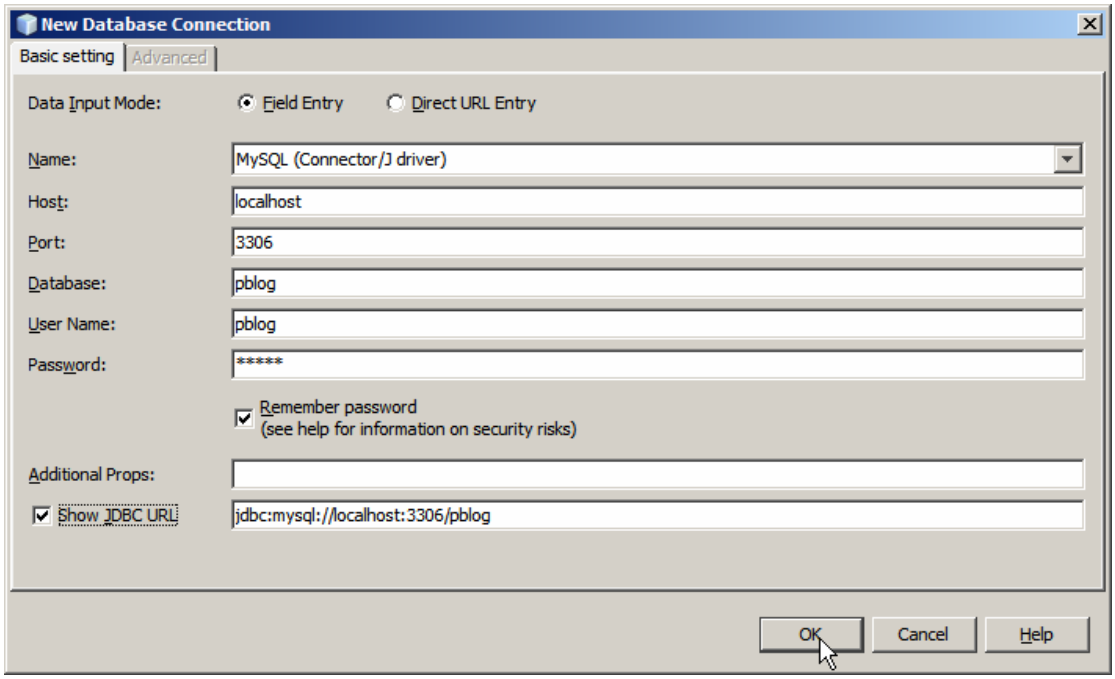
```
select * from ui_facility;
```

Now that we have the database objects in the database we need to “introduce” the database to NetBeans. This will make it simpler for us to use the database in the project.

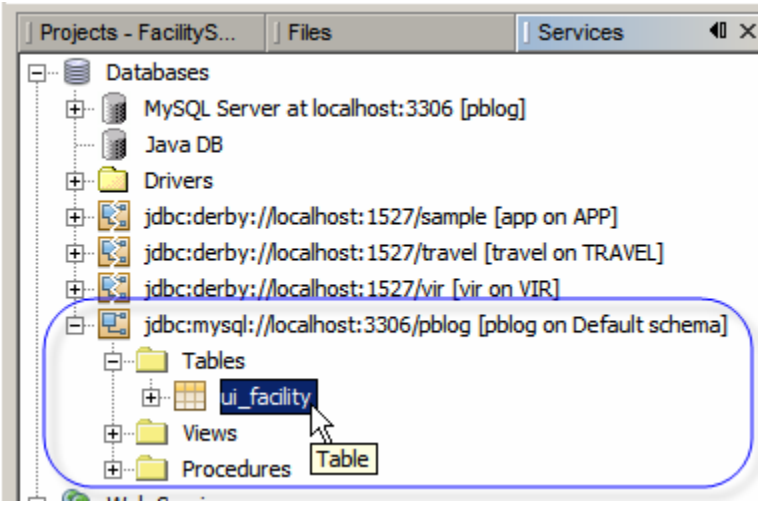
Switch to the Services Tab, right-click Databases and choose “New Connection …”.



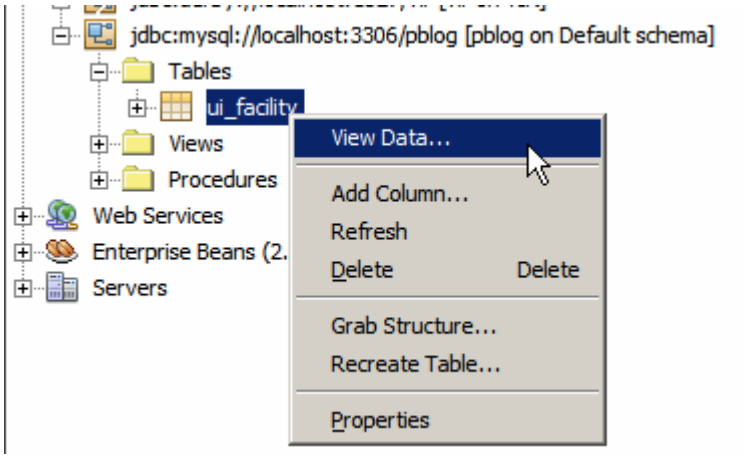
Populate fields of the connection wizard and click OK.



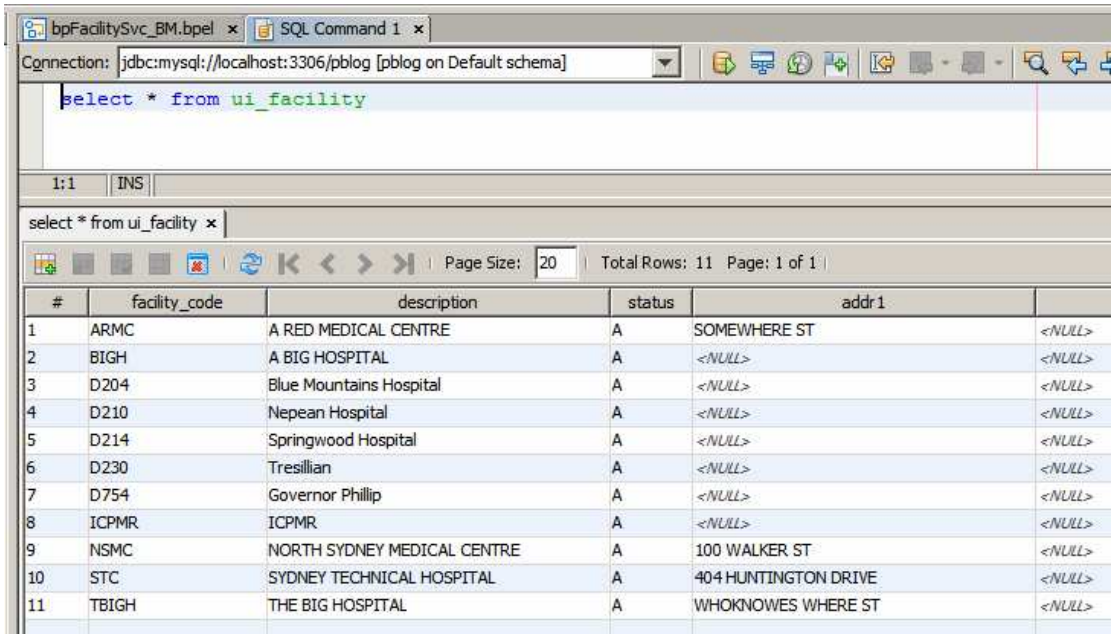
If all parameters are correct a new database connection entry will appear.



Right-click the name of the table and choose “View Data …”



This will produce display similar to that shown below.



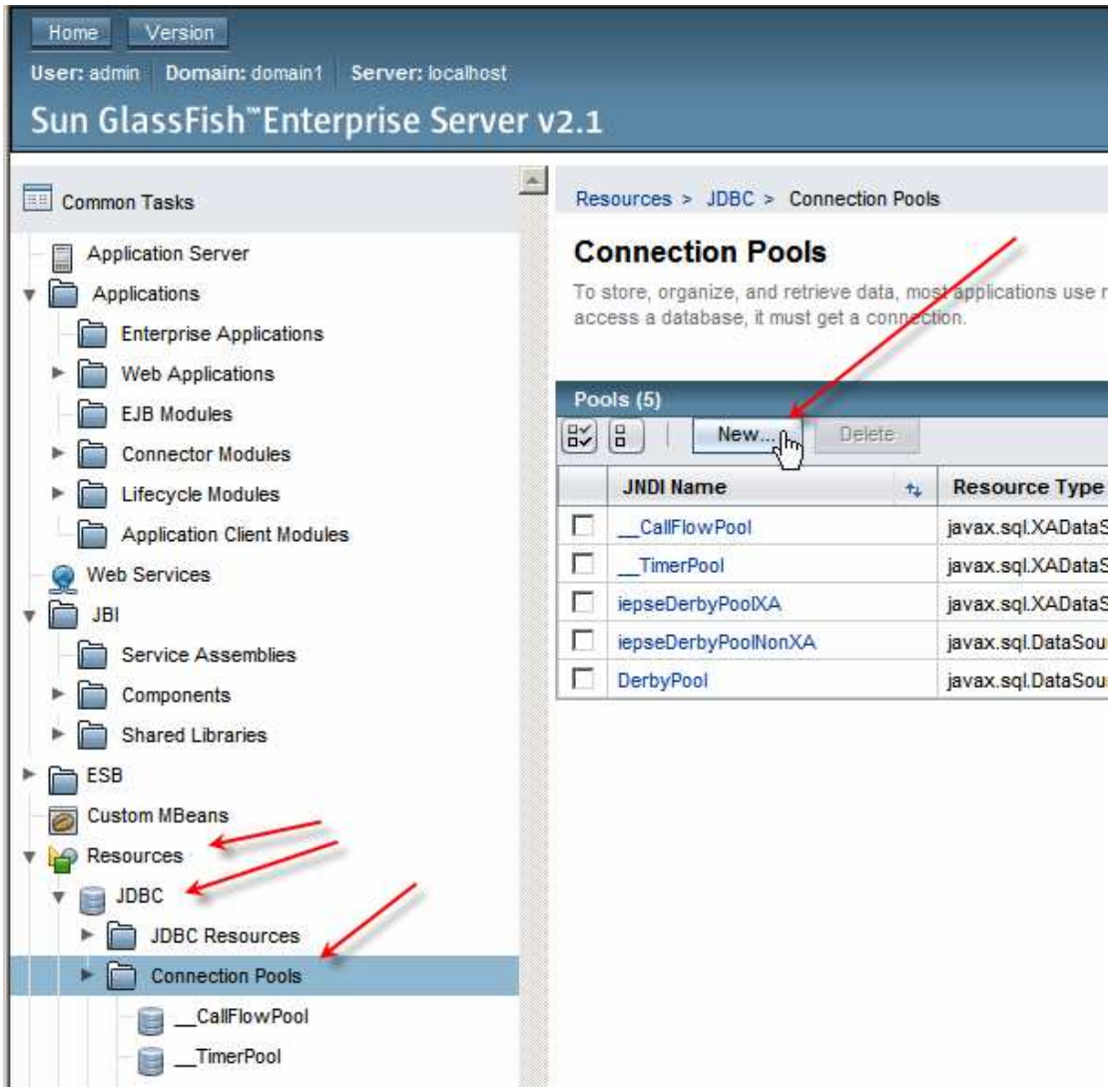
#	facility_code	description	status	addr1	
1	ARMC	A RED MEDICAL CENTRE	A	SOMEWHERE ST	<NULL>
2	BIGH	A BIG HOSPITAL	A	<NULL>	<NULL>
3	D204	Blue Mountains Hospital	A	<NULL>	<NULL>
4	D210	Nepean Hospital	A	<NULL>	<NULL>
5	D214	Springwood Hospital	A	<NULL>	<NULL>
6	D230	Tresillian	A	<NULL>	<NULL>
7	D754	Governor Phillip	A	<NULL>	<NULL>
8	ICPMR	ICPMR	A	<NULL>	<NULL>
9	NSMC	NORTH SYDNEY MEDICAL CENTRE	A	100 WALKER ST	<NULL>
10	STC	SYDNEY TECHNICAL HOSPITAL	A	404 HUNTINGTON DRIVE	<NULL>
11	TBIGH	THE BIG HOSPITAL	A	WHOKNOWES WHERE ST	<NULL>

The database connection is available and correctly configured for NetBeans. Now we need to create a connection pool to be used at runtime.

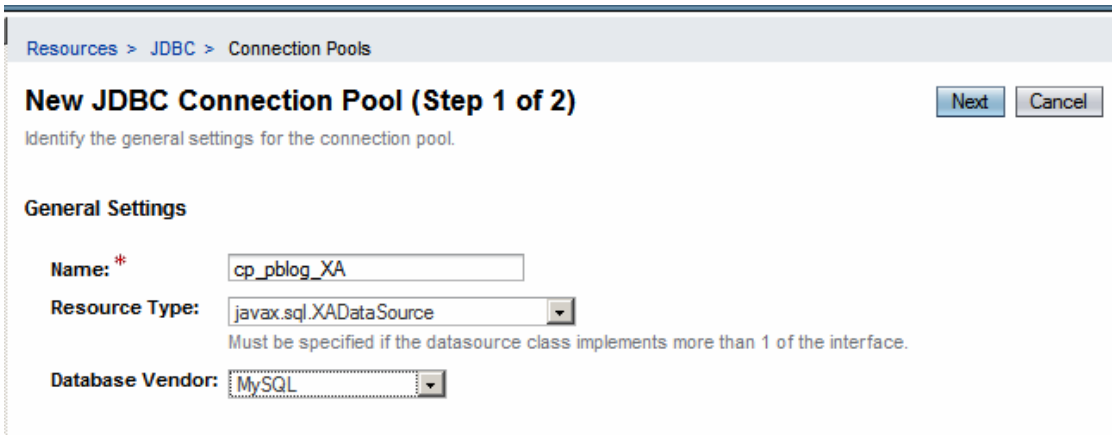
Make sure that the GlassFish Application Server has access to the appropriate MySQL JDBC Driver JAR. The MySQL JDBC driver, mysql-connector-java-5.1.6-bin.jar, is distributed as part of the GlassFishESB installation and is located in {GlassFishESBv21_install_rtoot}/netbeans/ide10/modules/ext. Copy the driver JAR file to {GlassFishESBv21_install_rtoot}/glassfish/domains/domain1/lib/ext and restart the application server before continuing.

Start the GlassFish Admin Console (http://localhost:4848 by default – use your own Admin Port if you didn't use defaults at install time) and log in as user admin.

Navigate to Resources -> JDBC -> Connection Pools



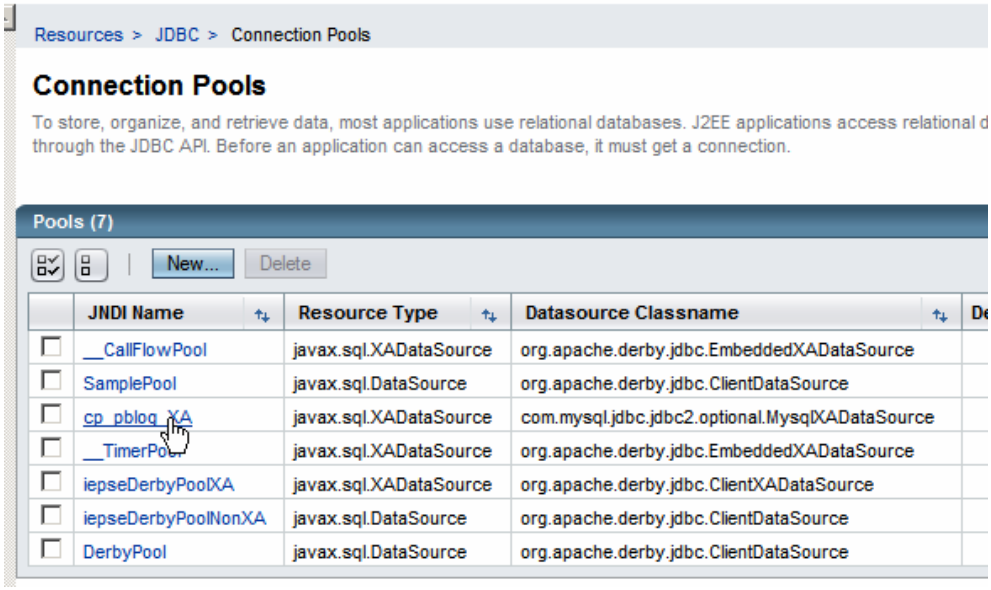
Name the new pool "cp_pblog_XA", select the javax.sql.XADateSource type and MySQL Database Vendor, then click Next.



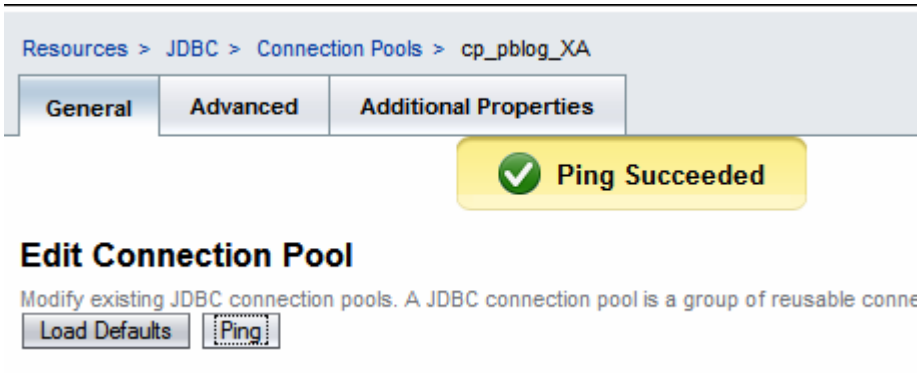
Scroll to Additional Properties and configure the following properties and click Finish:

DatabaseName	pblog
Password	pblog
ServerName	localhost
URL	jdbc:mysql://localhost:3306/pblog
Url	jdbc:mysql://localhost:3306/pblog
User	pblog

Click the name of the new Data Source in the list and click Ping.

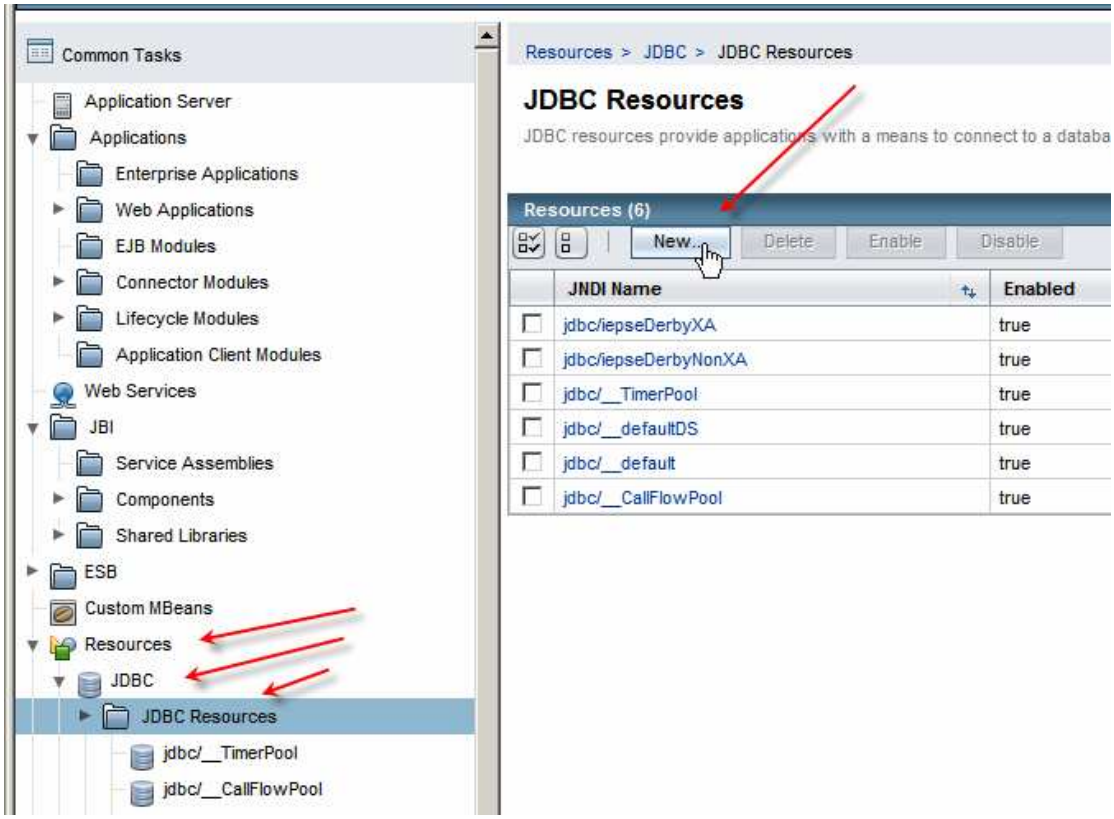


If all is configured correctly a success message should appear.



Finally, let's create the JNDI reference to use in referring to this connection pool.

Expand Resources -> JDBC -> JDBC Resources and click New



Name the resource reference "jdbc/ cp_pblog_XA" and choose the correct pool, cp_pblog_XA, as the pool name, then click OK

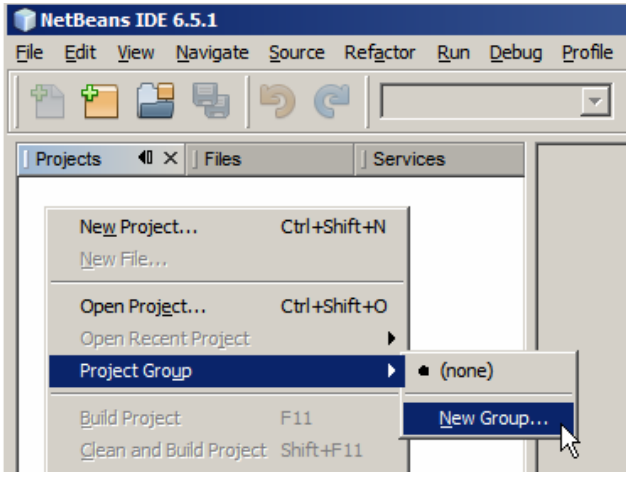
This completes database configuration.

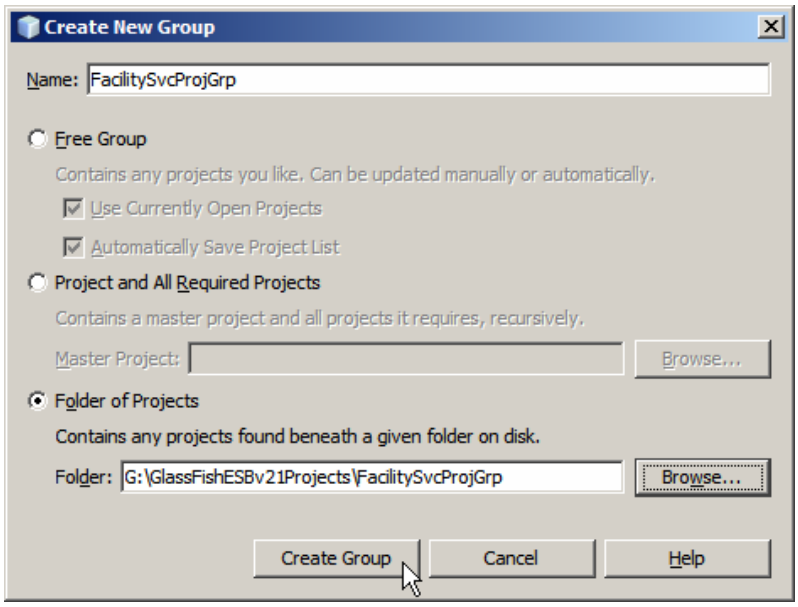
The connection pool created above will be good for anything that needs a XA connection pool for the MySQL database pblog.

Implementing the service

Create a Project Group

Let's start by creating a NetBeans Project Group, FacilitySvcProjGrp, in as a Folder of Projects in a convenient location in the file system.





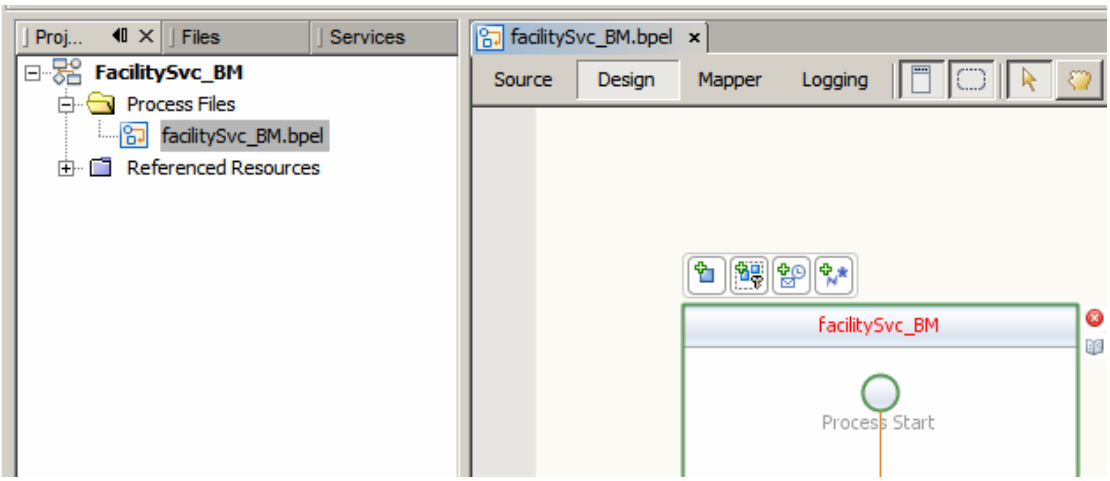
Create BPEL-based Service Implementation

The process will be invoked as a web service, using the WSDL FacilitySvc we will create shortly, and will invoke a Database service encapsulating access to the ui_facility table in the MySQL database.

To complete this section we will:

1. create XML Schema definition for input and output messages
2. create a web service interface WSDL definition
3. create a Database BC WSDL for a service which will select a code and description for all facilities
4. create a Database BC WSDL for a service that will select all information about a specific facility
5. implement two operations, each of which will orchestrate one of the two Database BC services
6. create a composite application and deploy it
7. create a test case and test the application

Let's create a new BPEL Module Project, FacilitySvc_BM. Once done, a skeleton BPEL module, facilitySvc_BM, will have been created. Rename it to bpFacilitySvc_BM. You don't really have to. I just don't like names other people picked for stuff so I tend to rename things more to my liking.

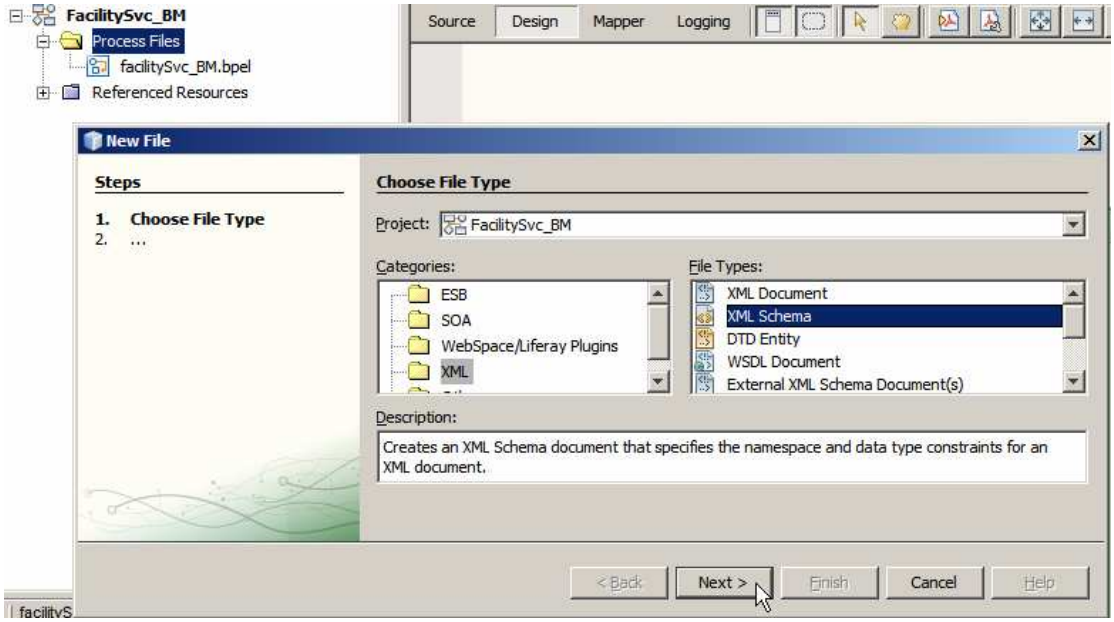


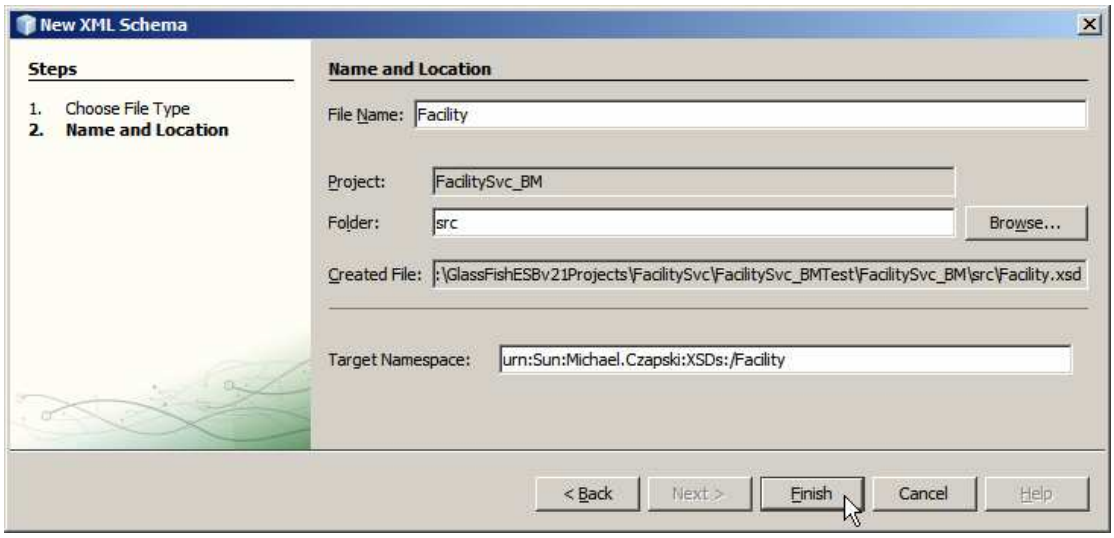
The service implementation will provide a list of all facilities and, given a facility code, all details available for the specific facility. A web service with two operations will be implemented.

The List operation will accept a dummy string, which it will ignore, and will return a list of facility_code and description pairs.

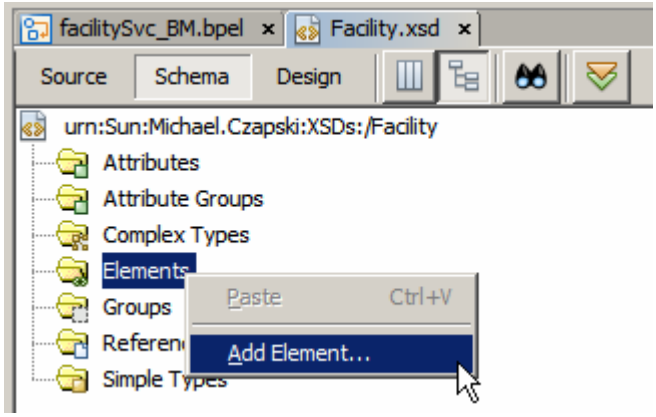
The Details operation will accept a facility_code and will return all facility details for the nominated facility.

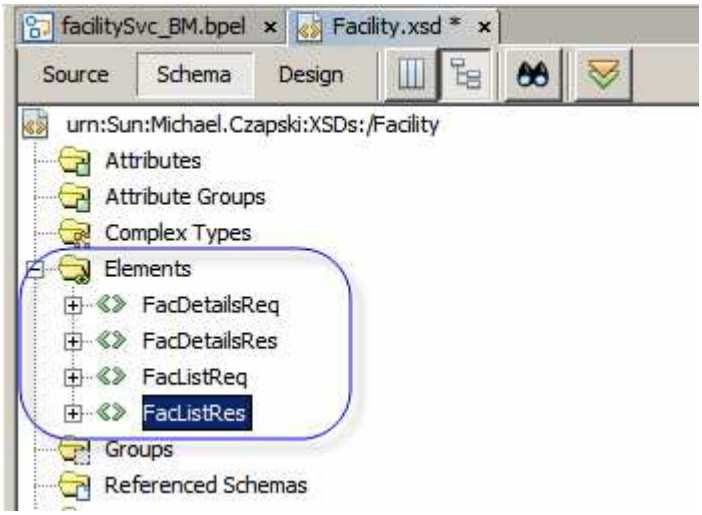
Let's create a New -> XML Schema, Facility, with target namespace "`urn:Sun:Michael.Czapski:XSDs:/Facility`".



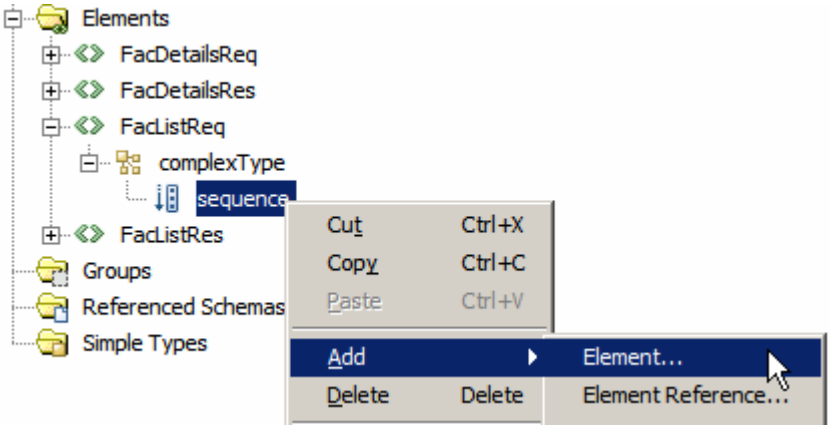


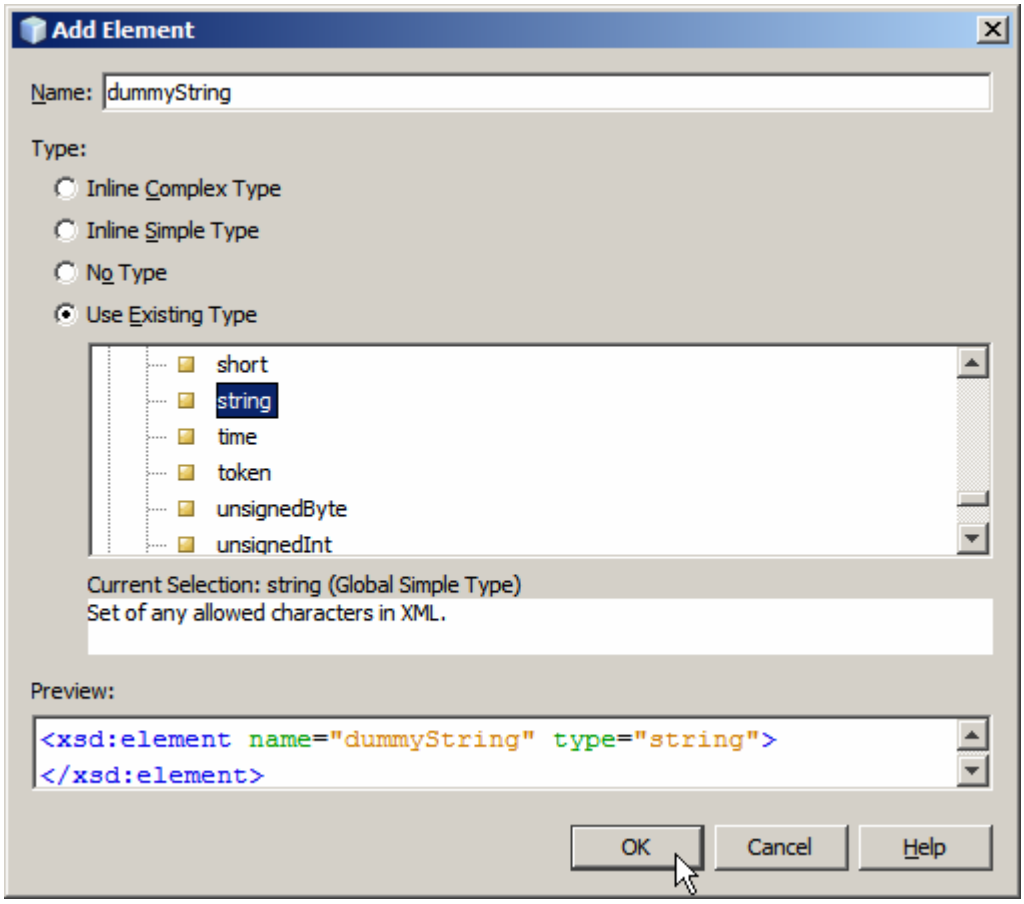
Add 4 elements at the root level: FacListReq, FacListRes, FacDetailsReq and FacDetailsRes.





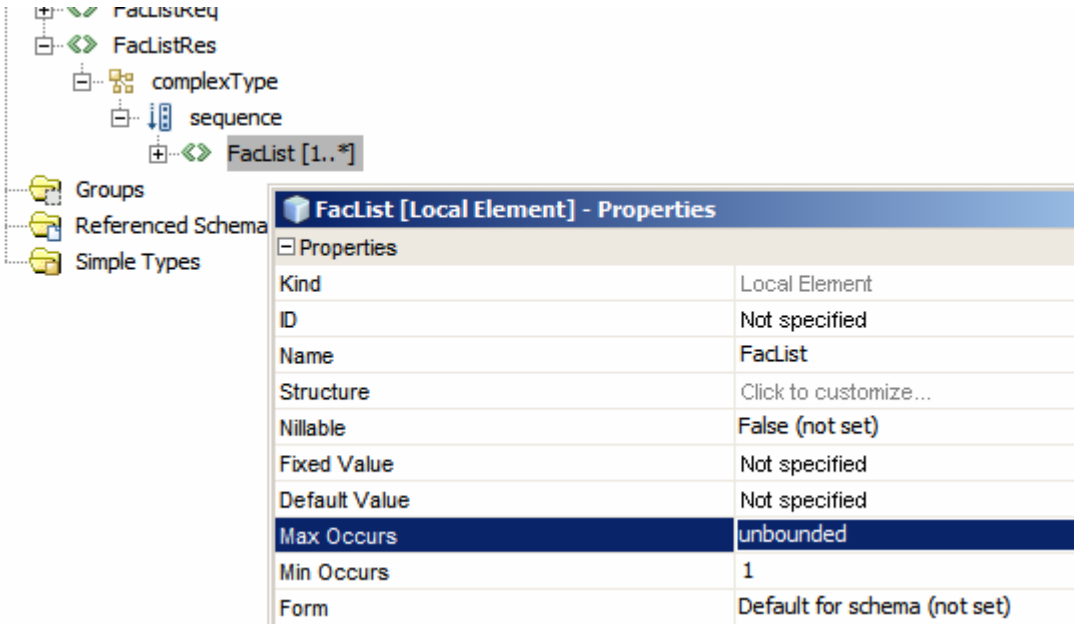
To FacListReq add a leaf element, dummyString of “existing, built-in” type string and minOccurence of 0.



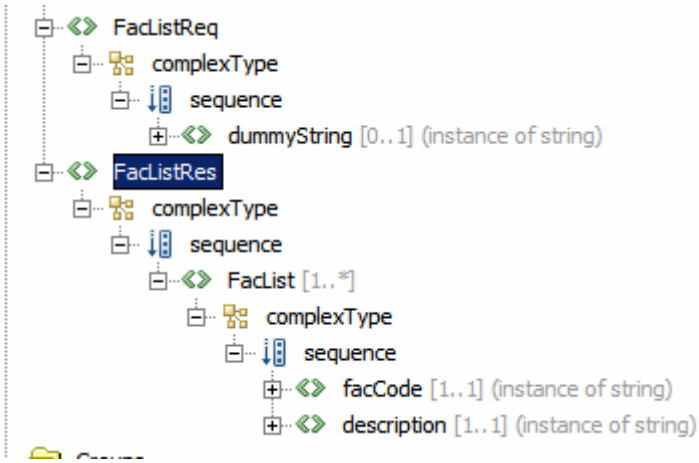


Use properties pane to change min occurrence for dummyString to 0.

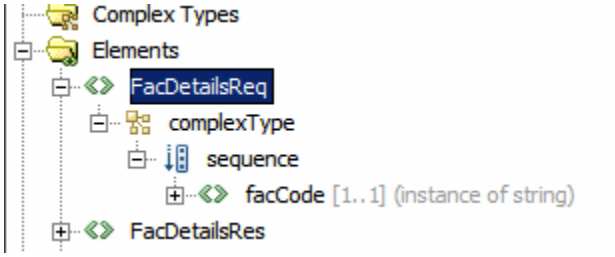
To FacListRes add an Element FacList with maxOccurs of “unbounded”.



Under FacList add two leaf elements, facCode and description, both of string type.

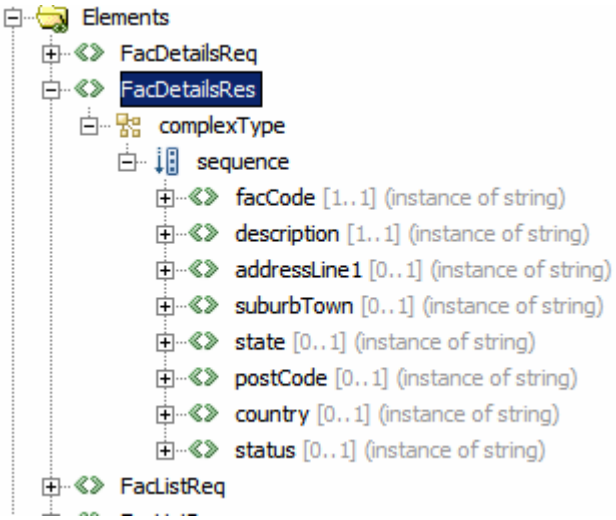


To FacDetailsReq add a string element facCode.



To FacDetailsRes add the following elements, each of string type and of nominated optionality:

Name	Type	Min Occurence	Max Occurrence
facCode	string	1	1
description	string	1	1
addressLine1	string	0	1
suburbTown	string	0	1
state	string	0	1
postCode	string	0	1
country	string	0	1
status	string	0	1

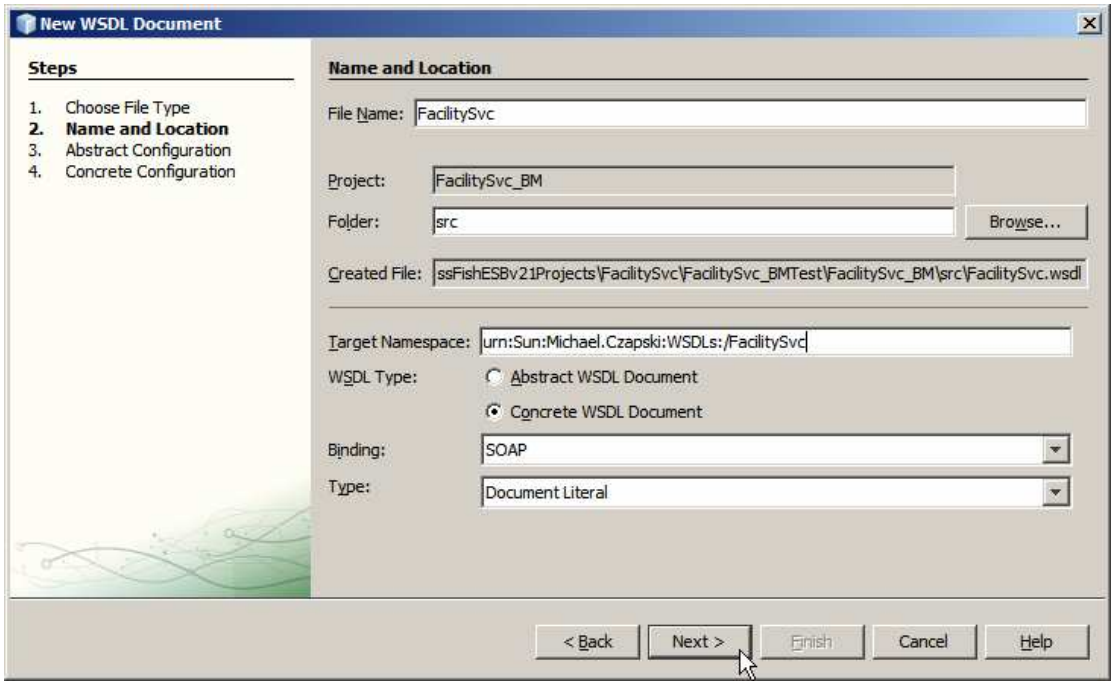


FacilityListReq and FacilityListRes are messages that will be associated with the operation opGetFacilityList , which will return the cafility code and description pairs for all facilities in the database table.

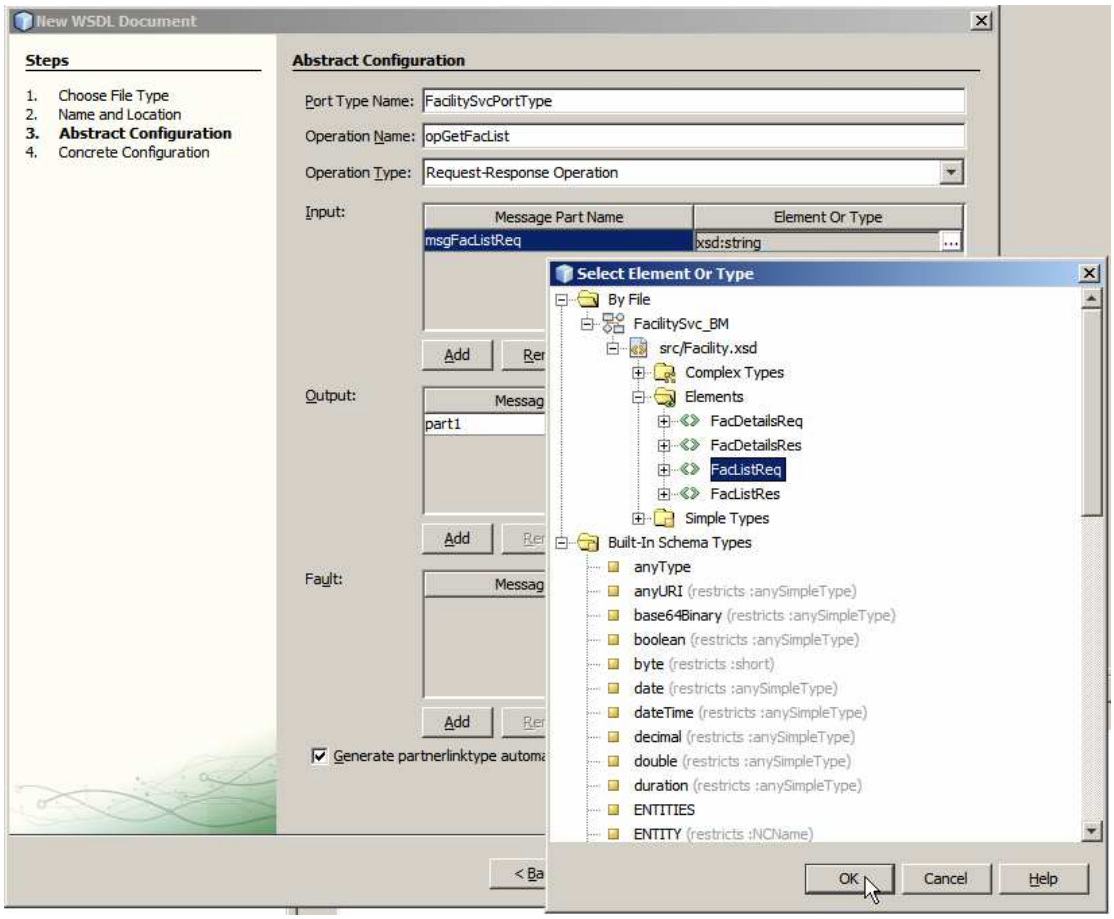
FacilityDetailReq and FacilityDetailRes are messages that will be associated with the operation opGetFacilityDetail, which will return all available details for a specific facility.

Note that the opGetFacilityList input message consists of a single optional string. The content of the message, if any, will be ignored since the service implementation will return the complete list of facilities every time the operation is invoked.

Let's now create a New -> WSDL Document, of WSDL Type Concrete, Binding: SOAP and Type Document/Literal, named FacilitySvc, with target namespace of "`urn:Sun:Michael.Czapski:WSDLs:/FacilitySvc`", which uses the schema elements.

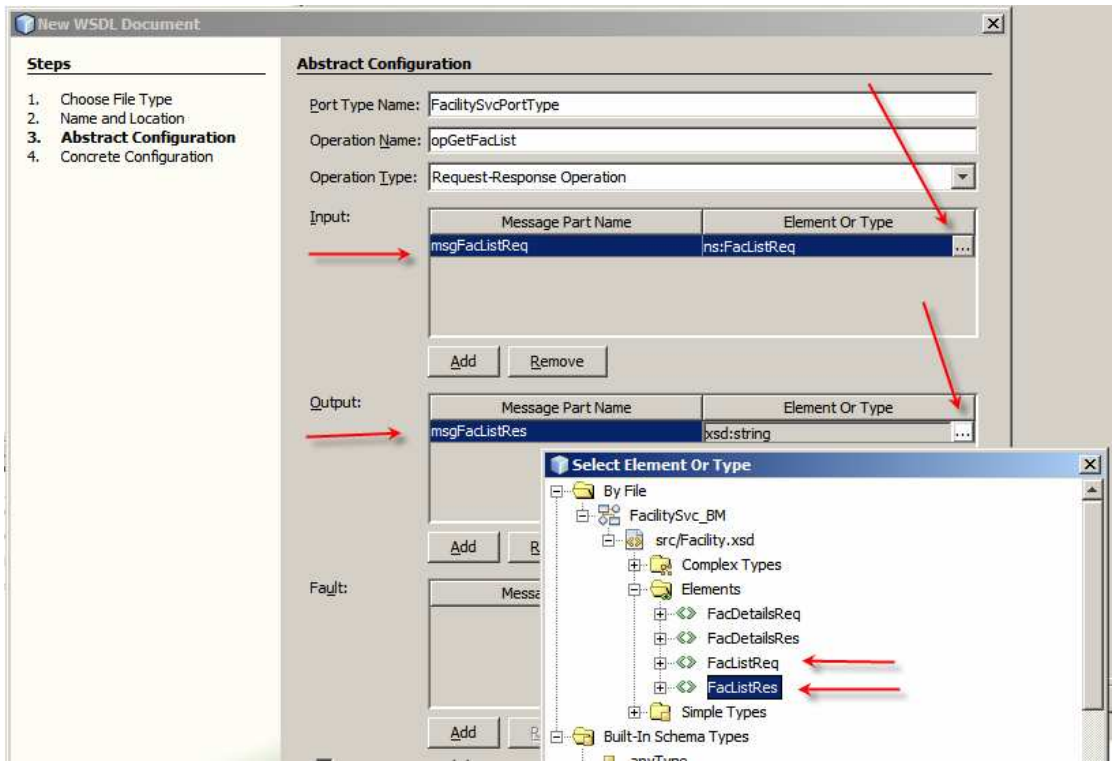


Rename operation to opGetFacList and start naming and typing messages.



Input: msgFacListReq of type FacListReq.

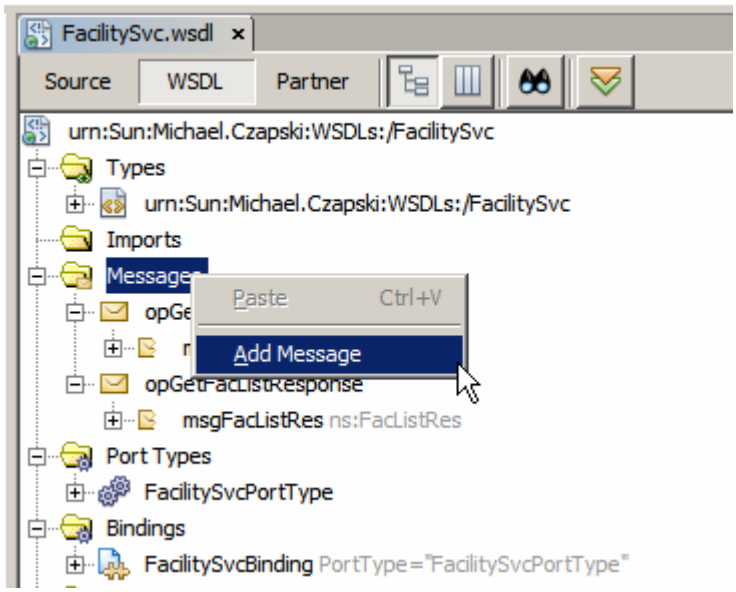
Output: msgFacListRes of type FacListRes.

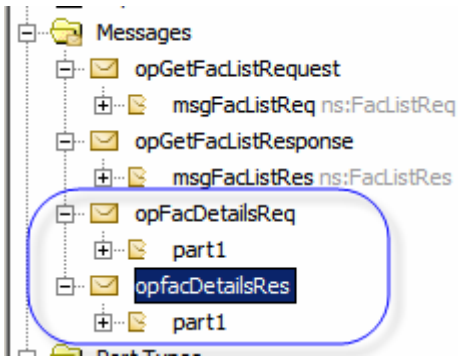


Finish the wizard.

This gives us a WSDL interface definition for a service with a single operation, opGetFacList. We now need to add another operation, opGetFacDetails, which uses FacDetailsReq and FacDetailsRes messages for input and output respectively.

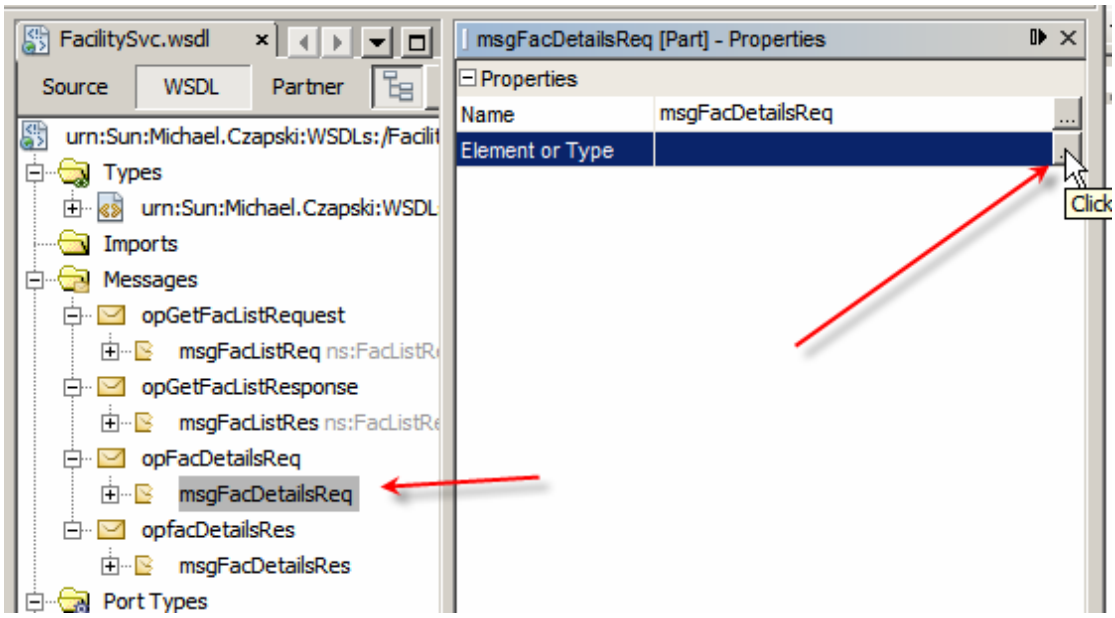
Let's add two new messages, opFacDetailsReq and opFacDetailsRes.

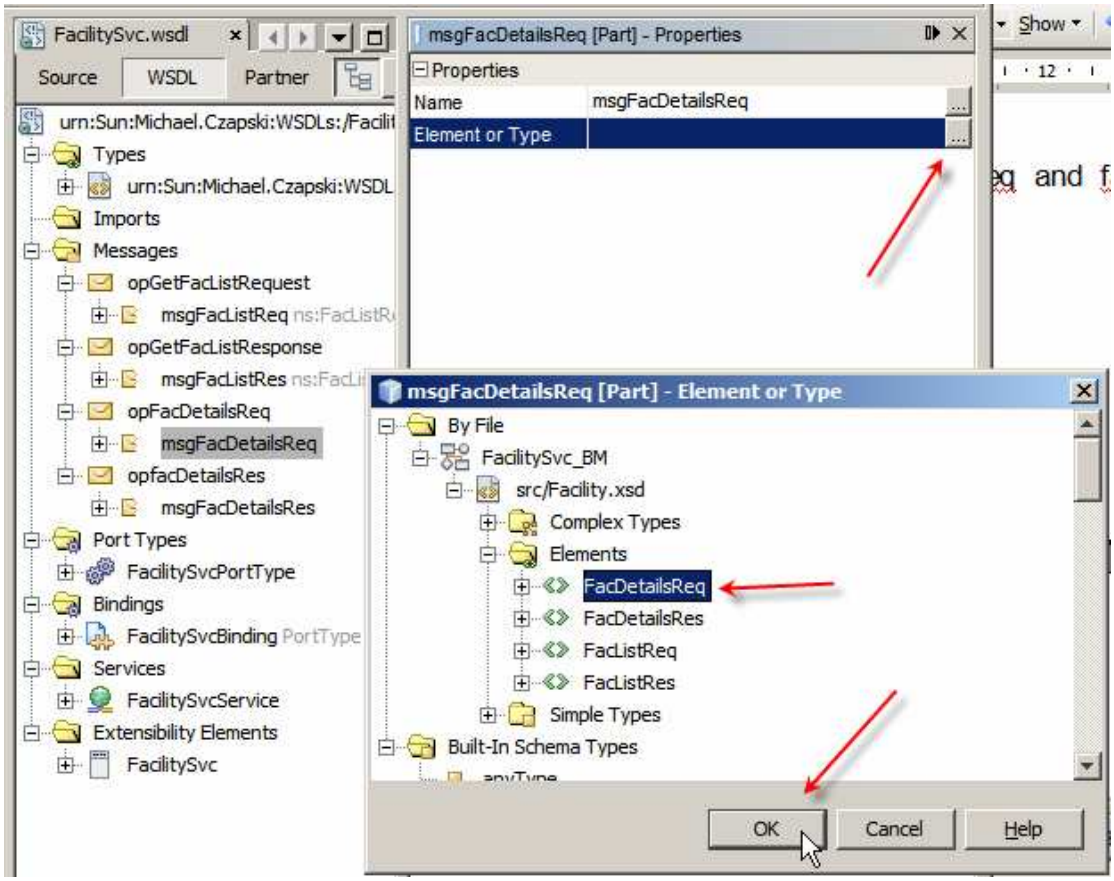


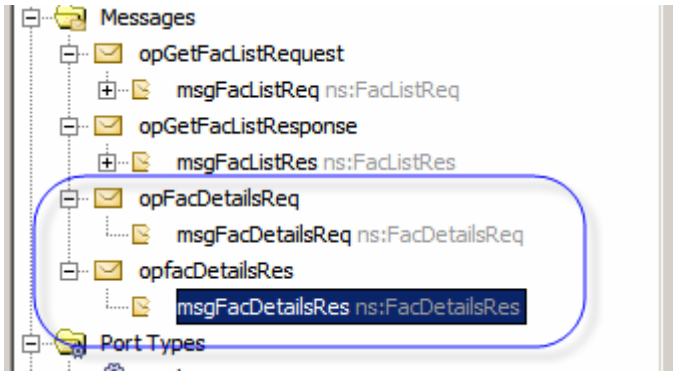


Let's now rename parts to msgFacDetailsReq and msgFacDetailsRes.

Let's now set the data types for these parts: FacDetaislReq and facDetailsRes respectively.

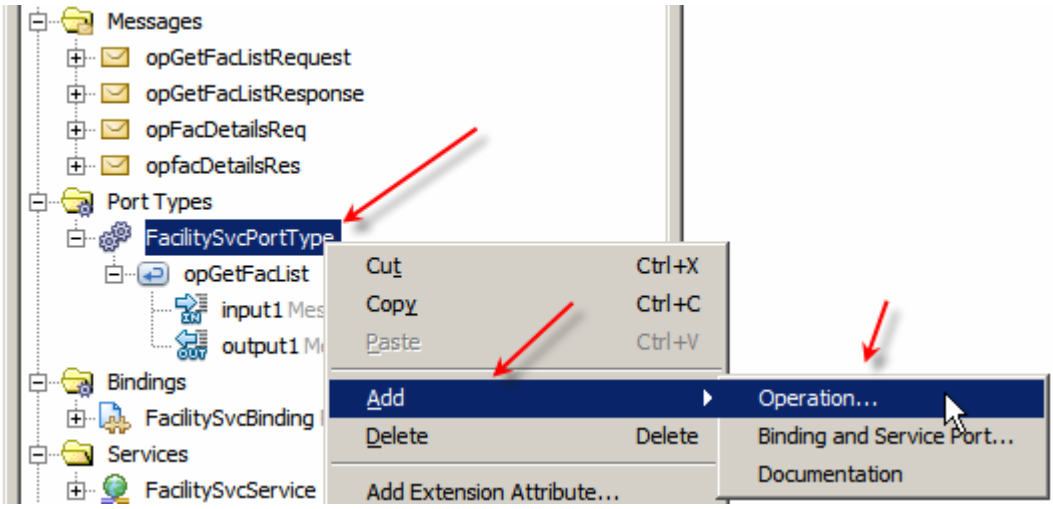

FacilitySvc.wsdl
Source
WSDL
Partner
urn:Sun:Michael.Czapski:WSDLs:/Facilit
Types
urn:Sun:Michael.Czapski:WSDL
Imports
Messages
opGetFacListRequest
msgFacListReq ns:FacListR
opGetFacListResponse
msgFacListRes ns:FacListR
opFacDetailsReq
msgFacDetailsReq
opfacDetailsRes
msgFacDetailsRes
Port Types
msgFacDetailsReq [Part] - Properties
Properties
Name
msgFacDetailsReq
Element or Type
Click


FacilitySvc.wsdl
Source
WSDL
Partner
urn:Sun:Michael.Czapski:WSDLs:/Facilit
Types
urn:Sun:Michael.Czapski:WSDL
Imports
Messages
opGetFacListRequest
msgFacListReq ns:FacListR
opGetFacListResponse
msgFacListRes ns:FacLi
opFacDetailsReq
msgFacDetailsReq
opfacDetailsRes
msgFacDetailsRes
Port Types
FacilitySvcPortType
Bindings
FacilitySvcBinding PortType
Services
FacilitySvcService
Extensibility Elements
FacilitySvc
msgFacDetailsReq [Part] - Properties
Properties
Name
msgFacDetailsReq
Element or Type
Show
msgFacDetailsReq [Part] - Element or Type
By File
FacilitySvc_BM
src/Facility.xsd
Complex Types
Elements
FacDetailsReq
FacDetailsRes
FacListReq
FacListRes
Simple Types
Built-In Schema Types
OK
Cancel
Help

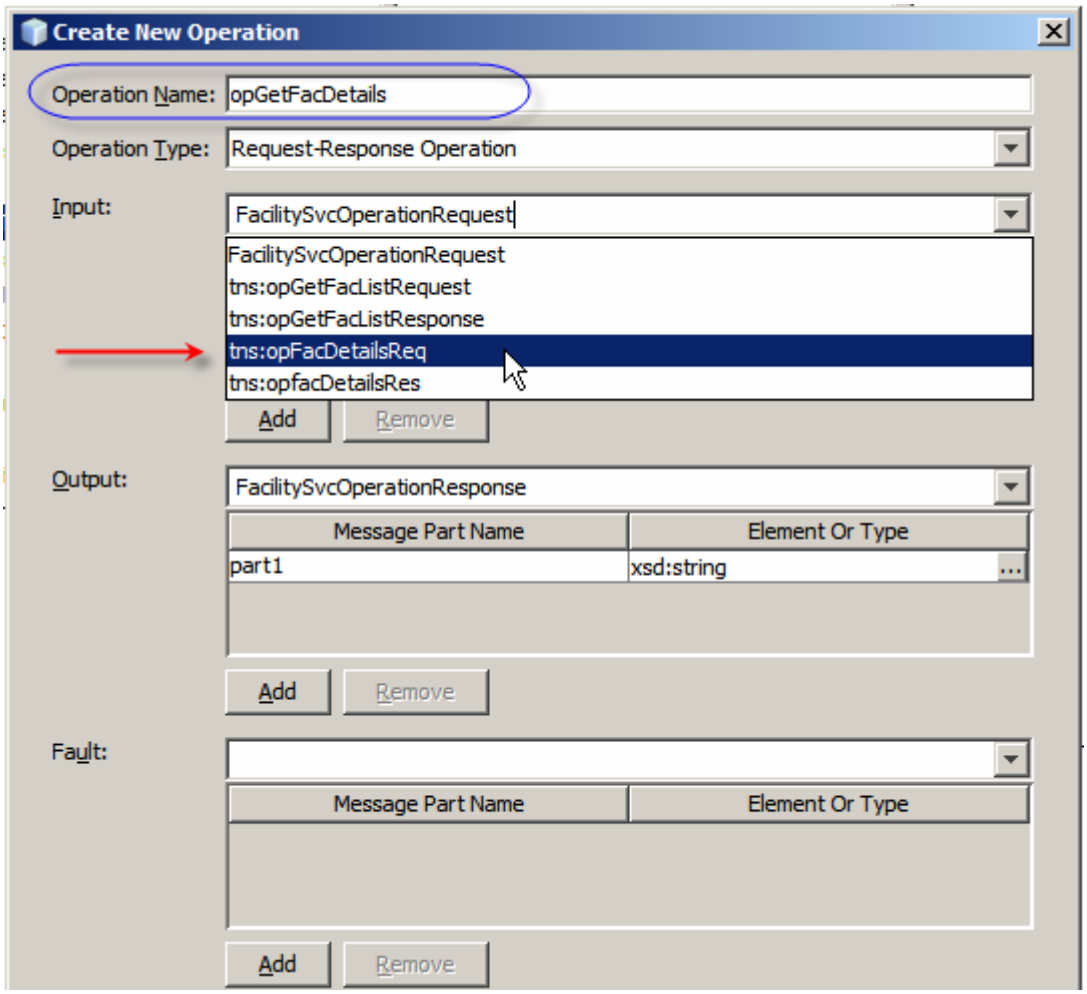


This gives us two new messages for the new operation.

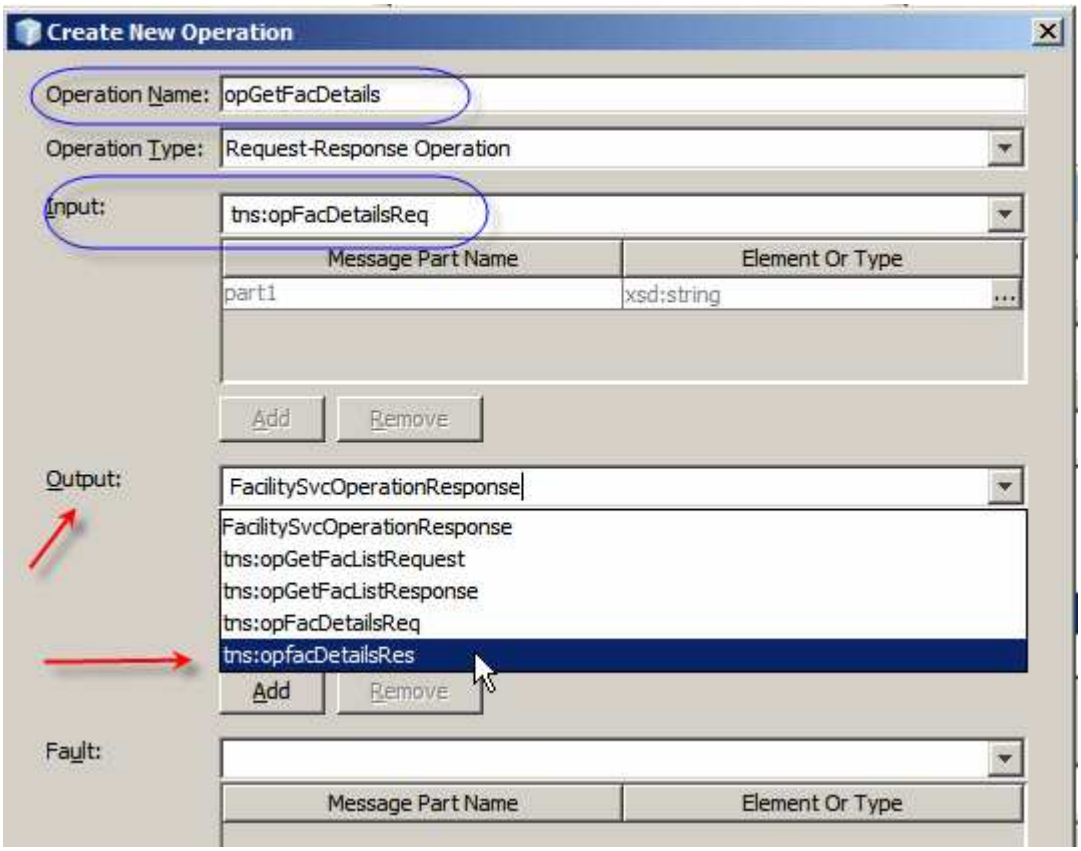
Let's right-click the FacilitySvcPortType node, choose Add and choose Operation.



Rename operation to opGetFacDetails and choose opFacDetailsReq message type for Input.



Choose opFacDetailsRes message type for Output.

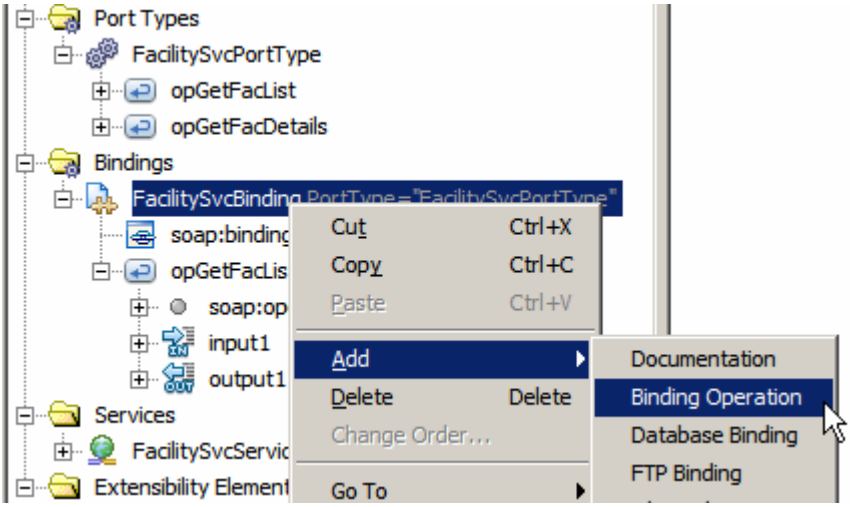


Complete the wizard.

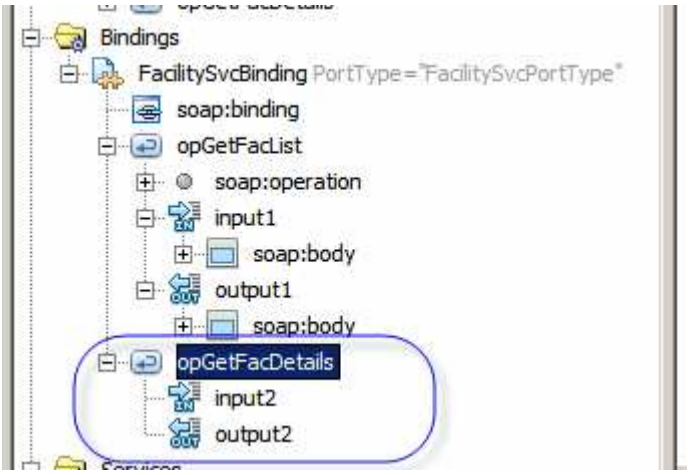
The abstract operation, opGetFacDetails is now configured.

We need to add a new binding.

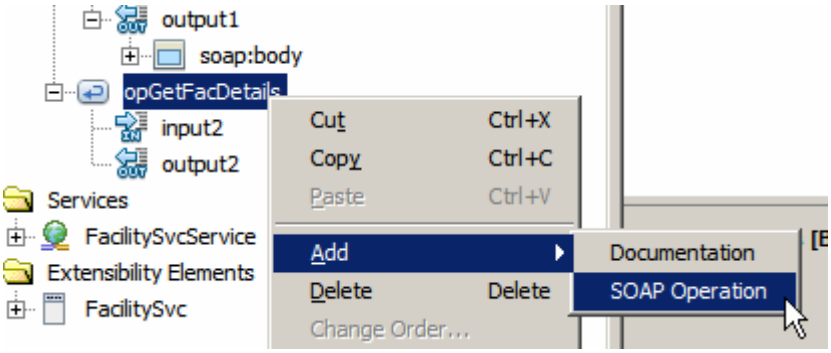
Right-click the FacilitySvcBinding, choose Add and choose Binding Operation.



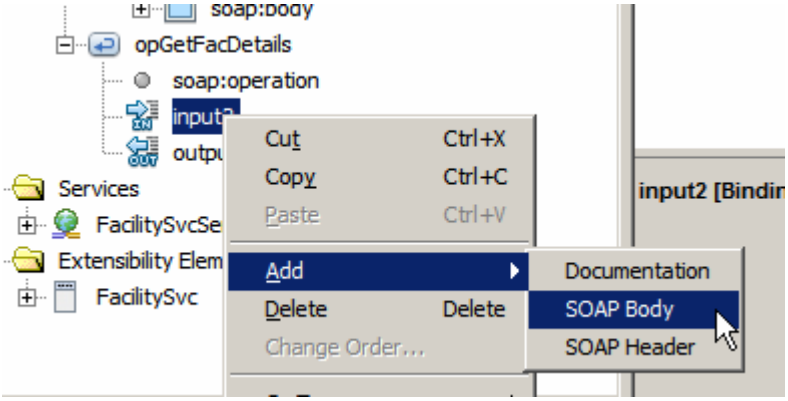
New operation gets added. We must now configure it.

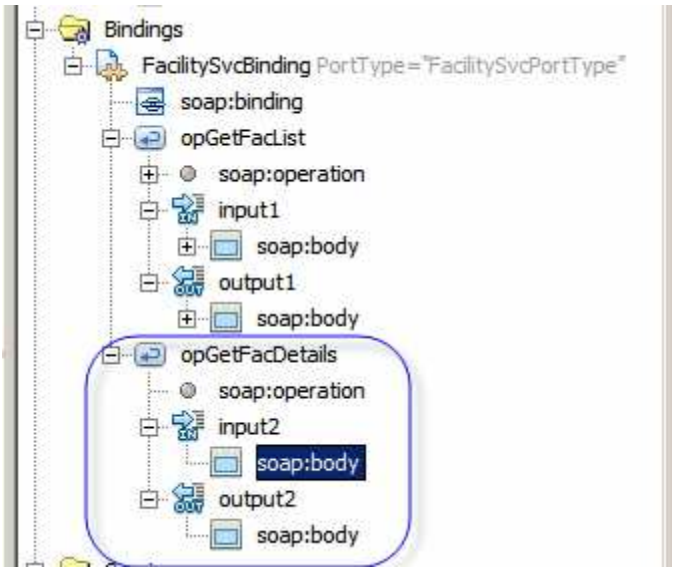


Right-click opGetFacDetails and choose Add -> SOAP Operation.

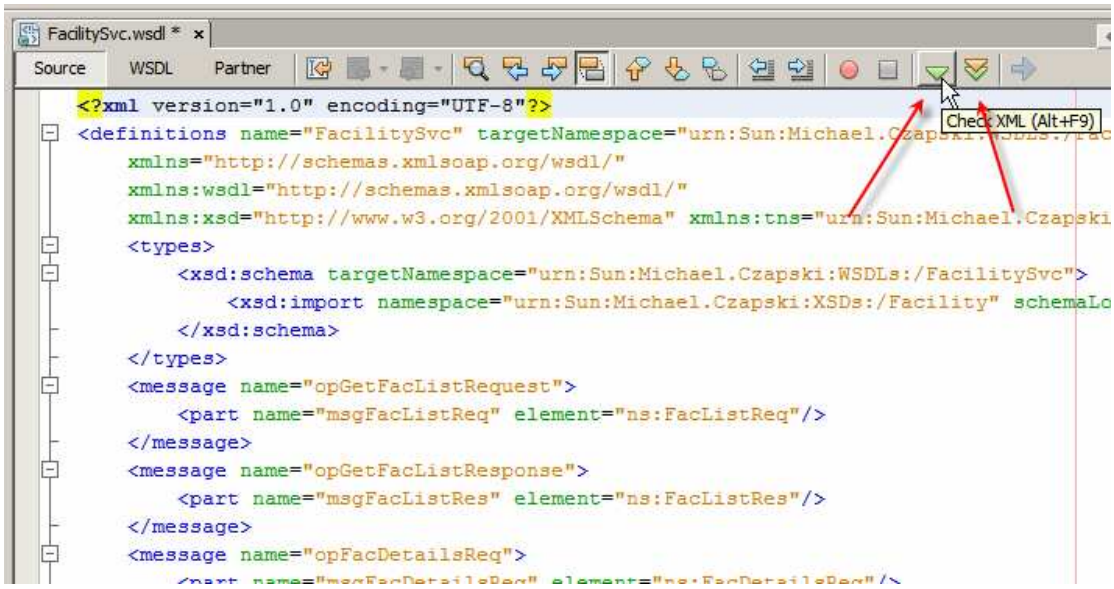


Right-click in Input2 and Output2 in turn and choose Add -> SOAP Body





This completes the WSDL definition. Switch to Source mode and Validate and Check XML.



Now we need to prepare interfaces definitions for Database-related services.

Each of the two service operations will execute a different SQL statement to get different data from the database.

With GlassFish ESB and the Database Binding Components there are a couple of ways in which a database service can be configured. I chose to use the SQL File method. I create a SQL File containing the prepared statement then use this SQL File to construct a WSDL for the Database BC.

The first operation, opGetFaclList, will return a list of facility code and description pairs. The SQL statement will be:

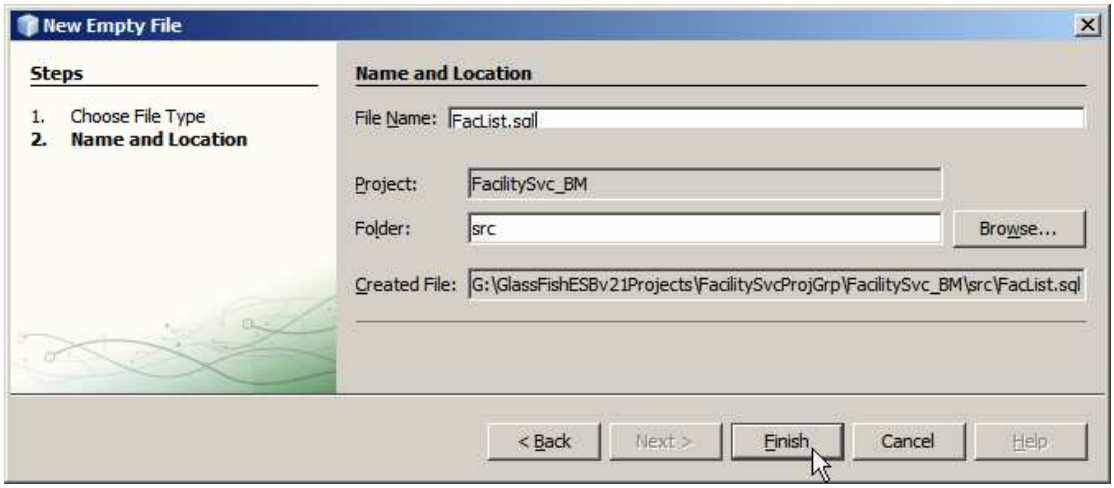
```
select facility_code, description from ui_facility;
```

The second operation, opGetFacDetails, willl return all there is to know about a specific facility. The SQL statement will be:

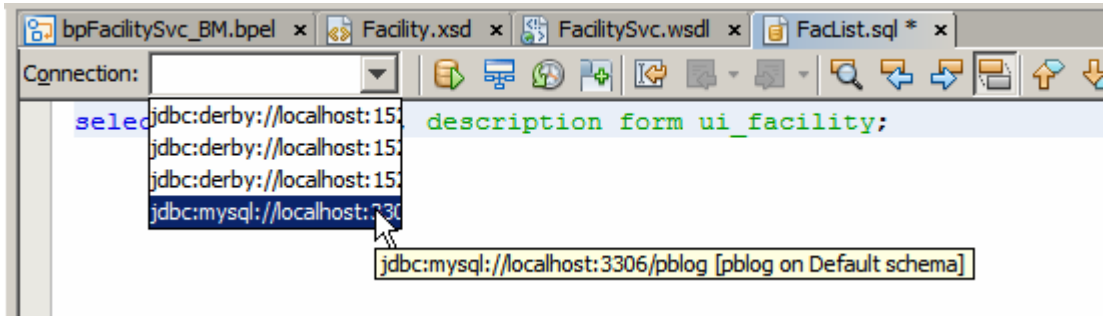
```
select * from ui_facility where facility_code = ?;
```

Let's construct the SQL Files and Database BC WSDLs one at a time.

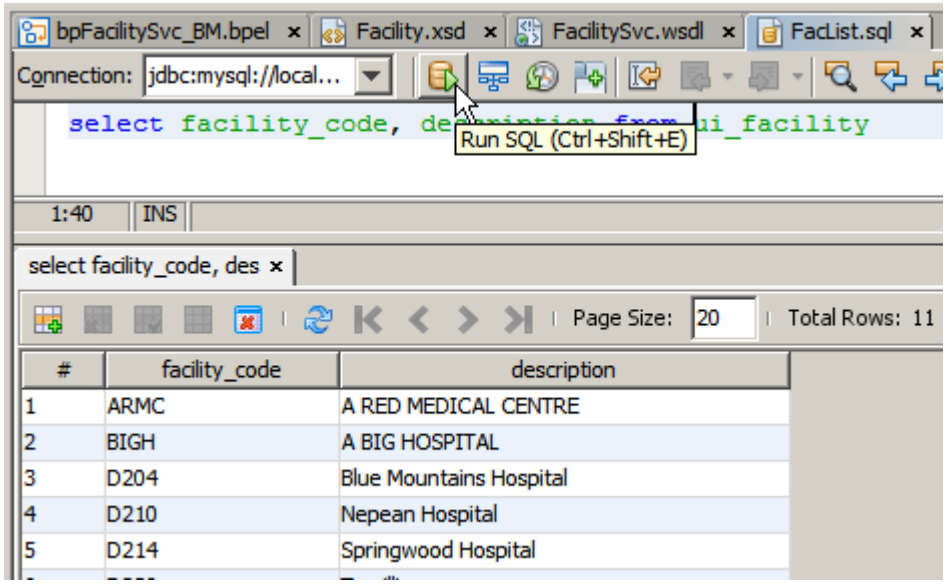
Let's create a New -> Other -> Empty File, named FacListDB.sql.



Let's enter "select facility_code, description from ui_facility;" into the editor window and choose the MySQL connection we created earlier as the connection to use for this SQL statement.

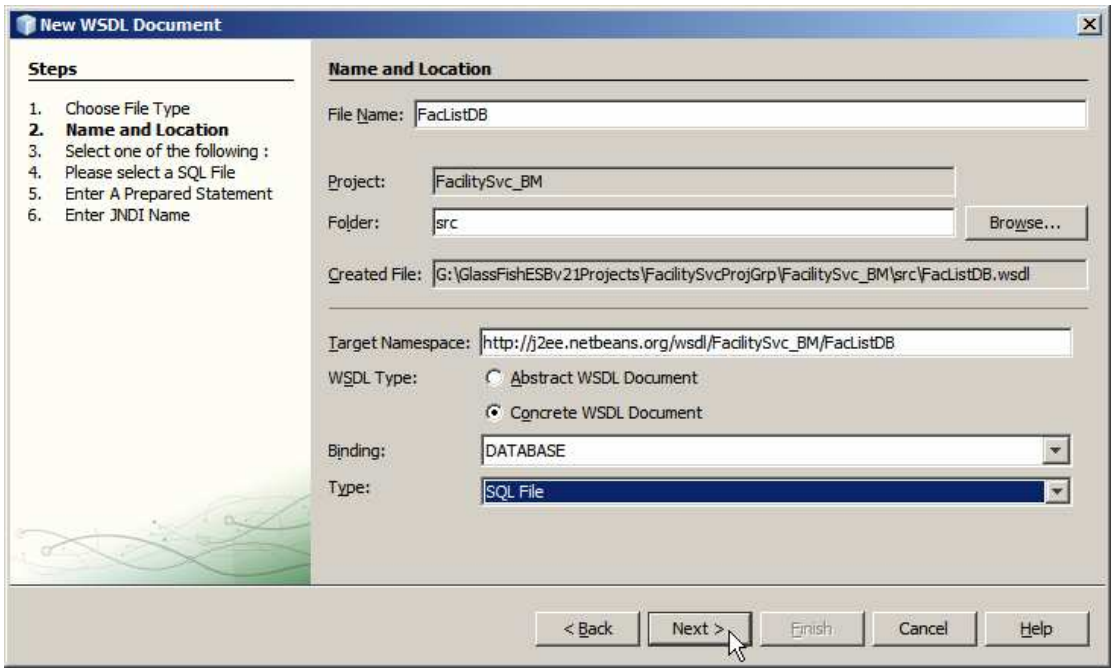


To verify that the connection and the statement work let's click the "Run SQL" button.

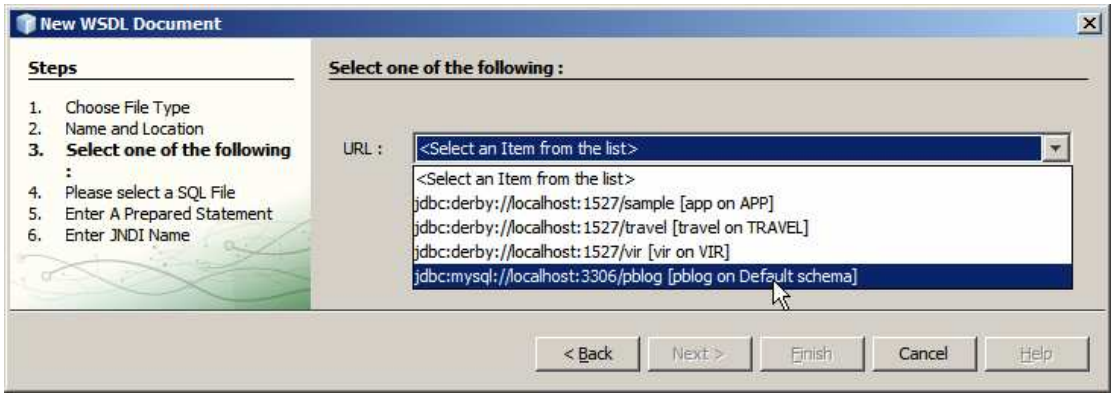


The connection and the statement work as expected.

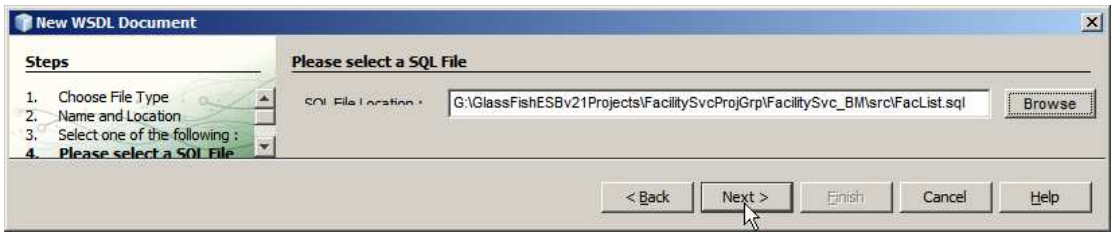
Let's now create a new WSDL Document, FacListDB.wsdl, a Concrete WSDL, using Database Binding of type SQL File.



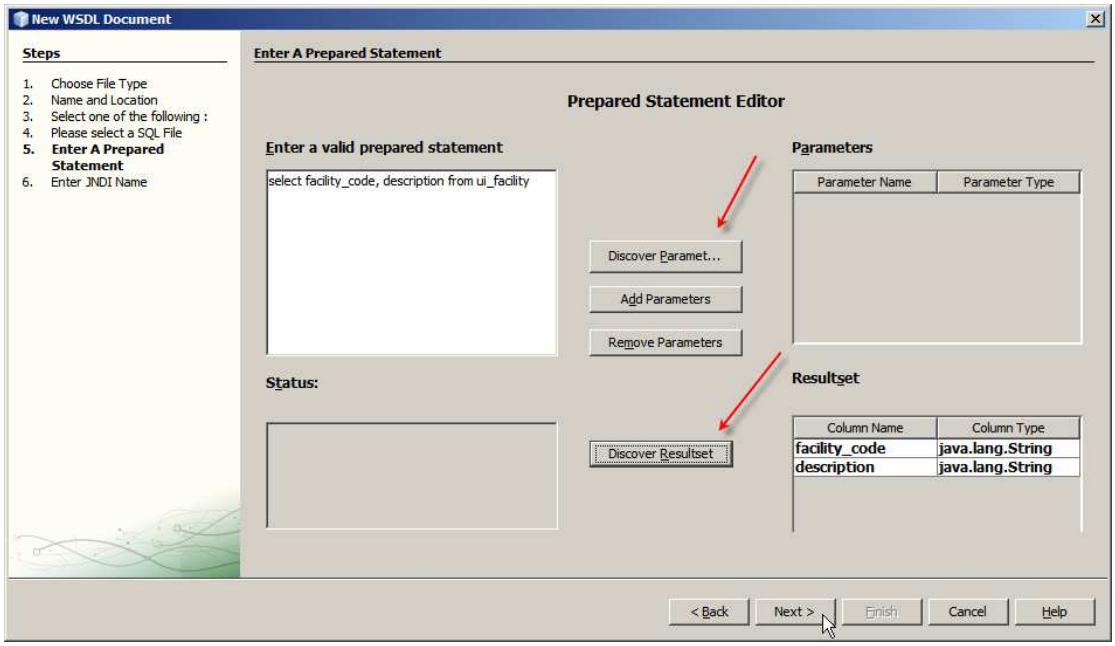
Choose the MySQL connection we created earlier.



Choose the FacList.sql SQL file we created earlier.

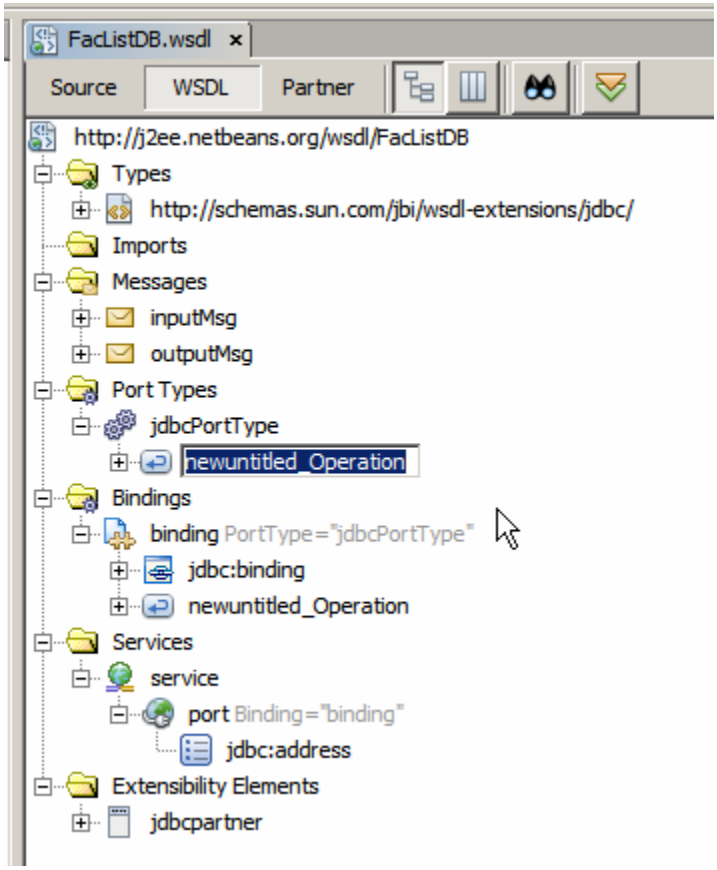


Click the “Discover Parameters” button (there will be non in this case) and the “Discover Resultset” button, then click Next.

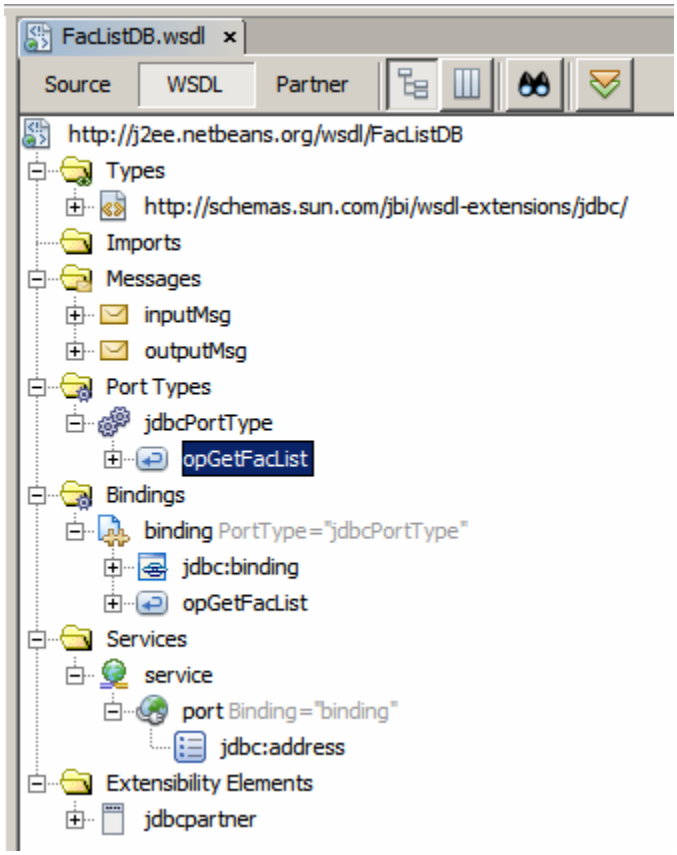


Specify the name of the pool we created earlier, “jdbc/cp_pblog_XA”, and click Finish.

Rename the operation under Port Types -> jdbcPortType from newUntitled_Operation ot opGetFacList.



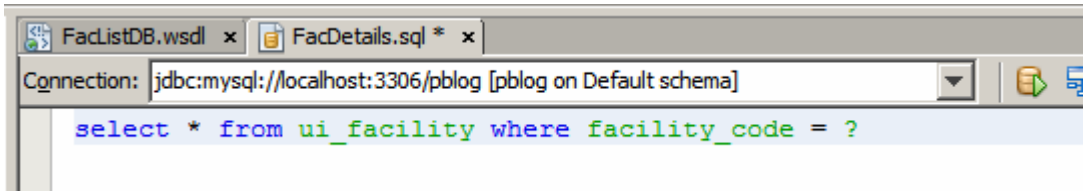
Note that Bindings -> Binding Operation gets renamed as well.



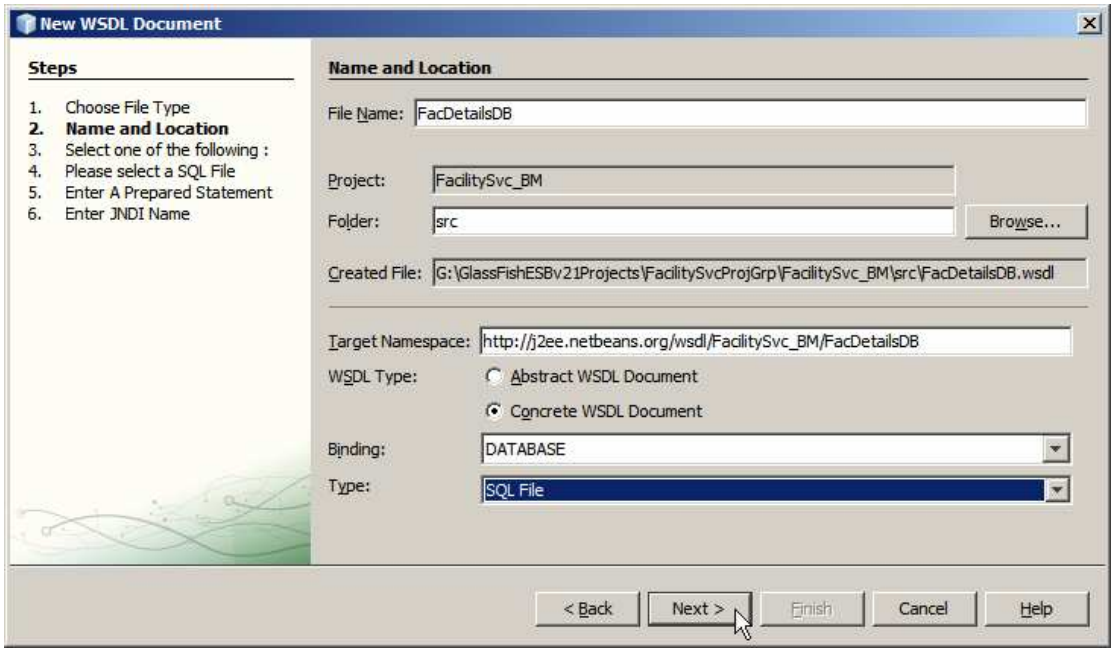
This completes the Database BC WSDL for the SQL statement that will return the list of facility codes and descriptions.

Let's now create the SQL File and the Database BC WSDL for the statement "select * from ui_facility where facility_code = ?".

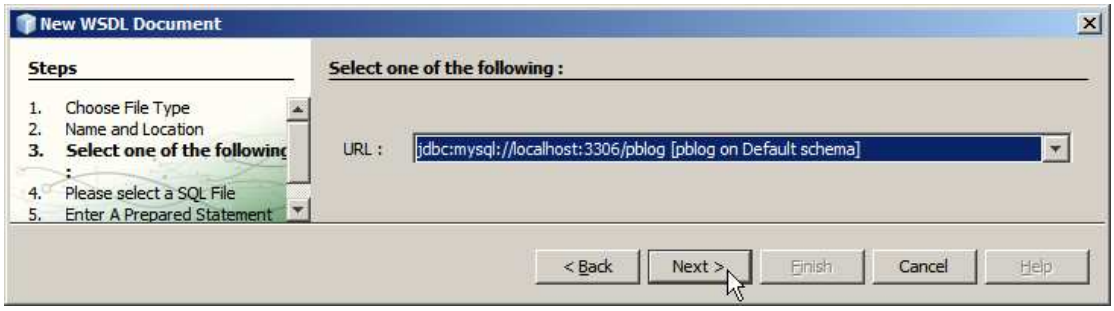
Let's create New -> Other -> Empty File, named FacDetails.sql, using the MySQL connection created earlier.



Let's now create a New -> WSDL Document, named FacDetailsDB.wsdl, a Concrete WSDL using Database Binding of type SQL File.

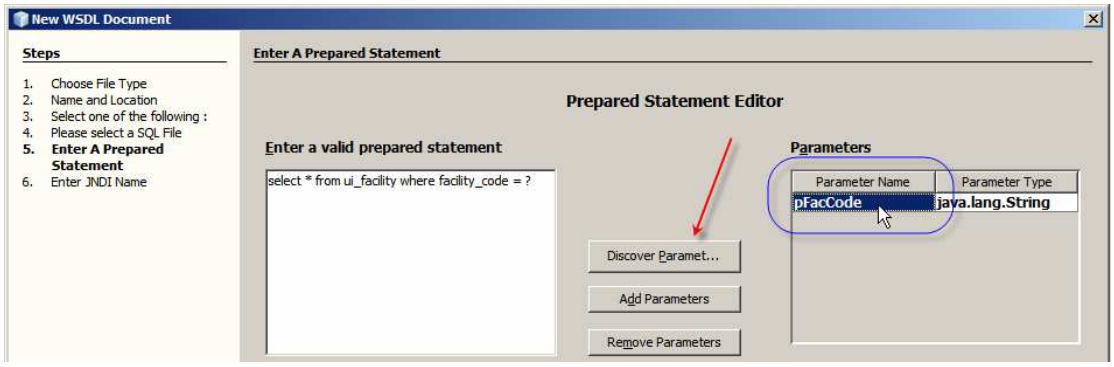


Let it use the MySQL connection we created earlier.

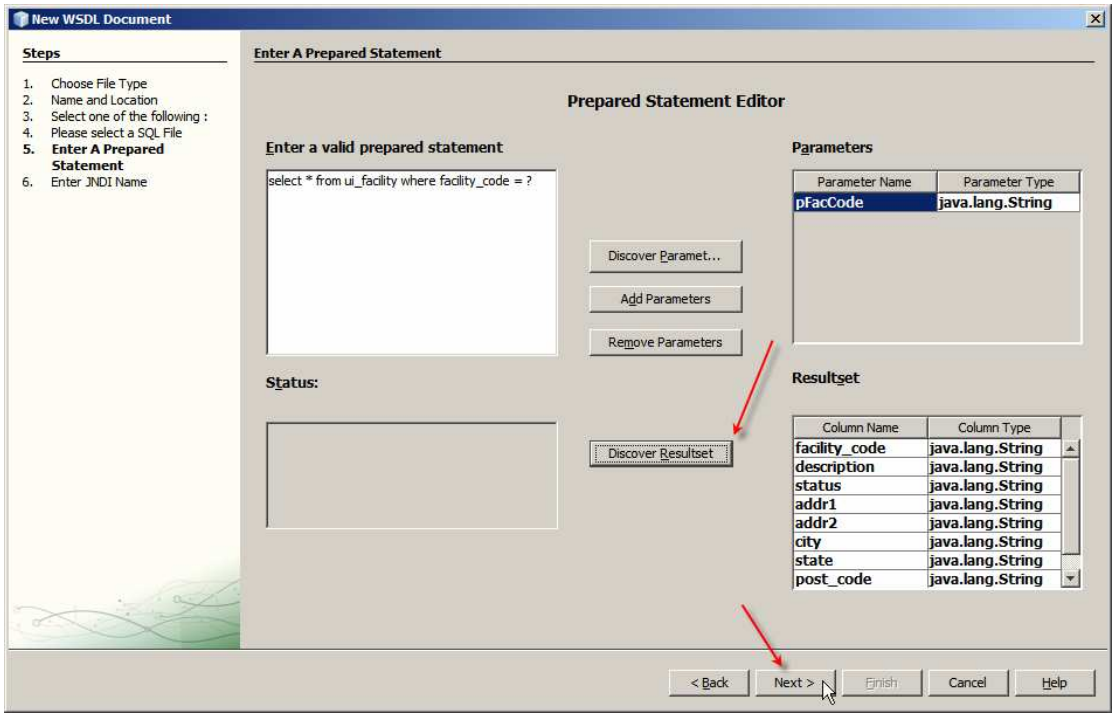


Let it use the SQL File, FacDetails.sql, we created earlier.

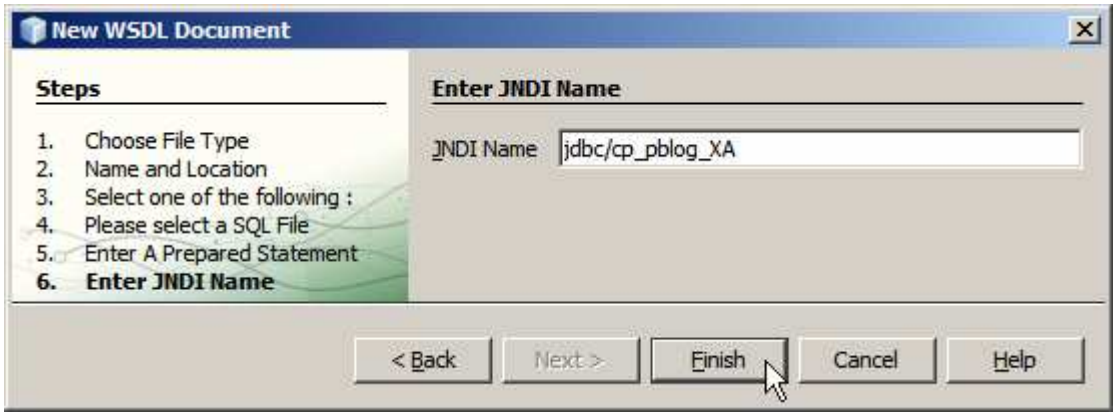
Click the “Discover Parameters” button and change the name of the parameter to “pFacCode”.



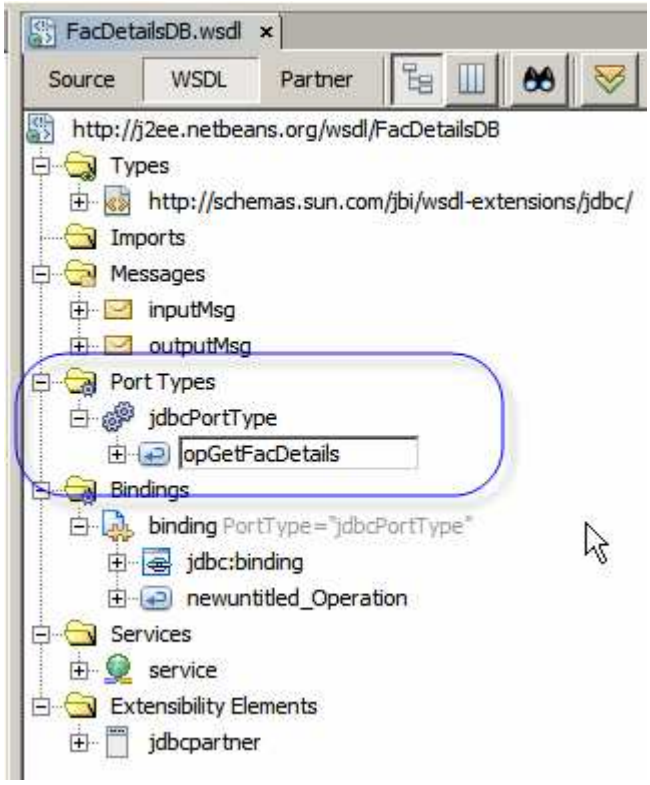
Click the “Discover Resultset” button and click Next.



Specify the connection pool to use, “jdbc/cp_pblog_XA” and Finish.

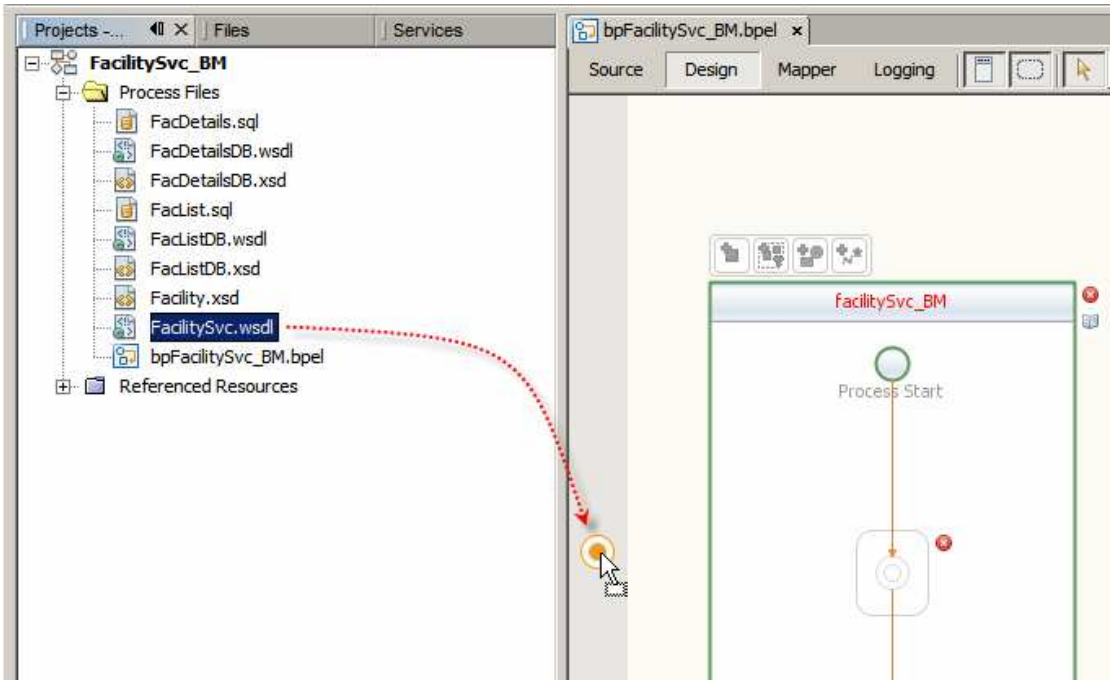


Rename “newuntitled_Operation” to opGetFacDetails.

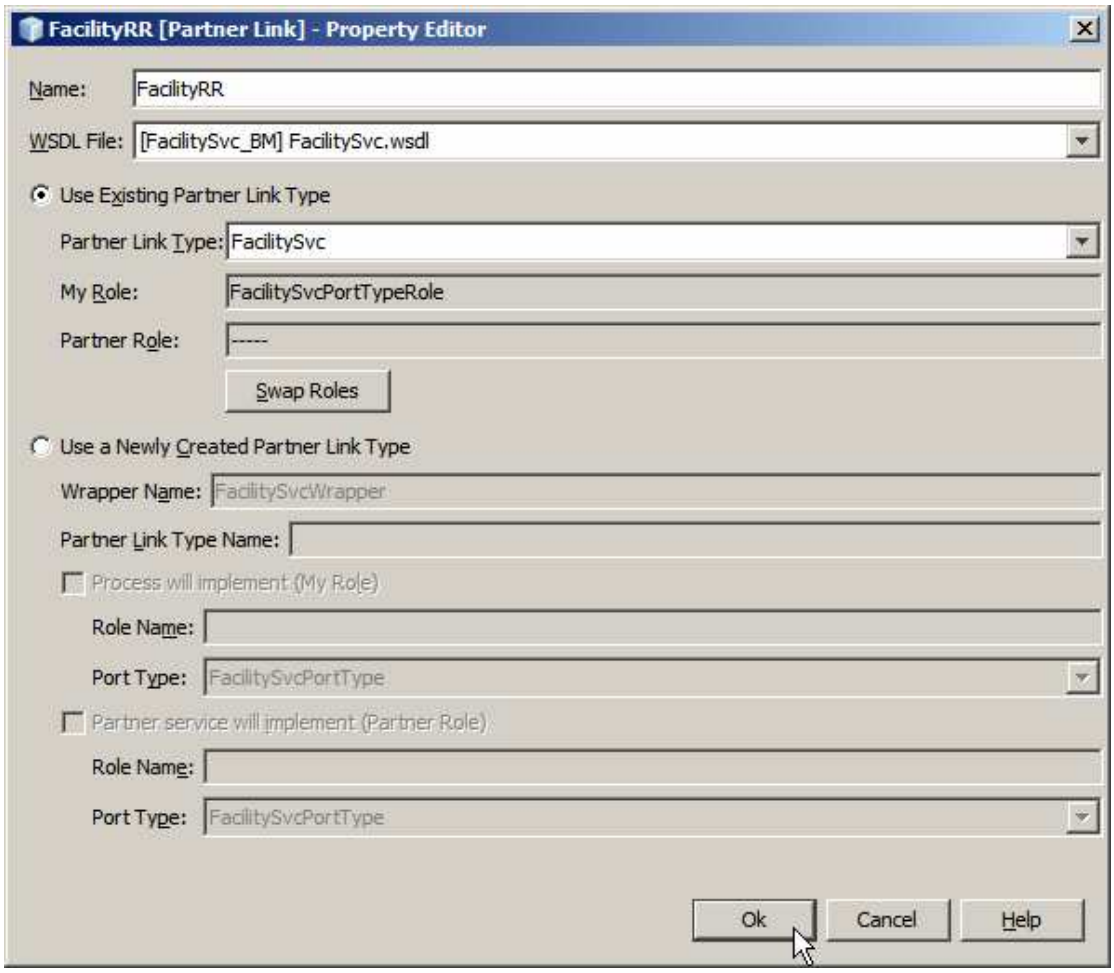


Both Database BC WSDLs are now ready. The service interface, FacilitySvc WSDL is also ready. We can now construct the business process to orchestrate the two Database BC services.

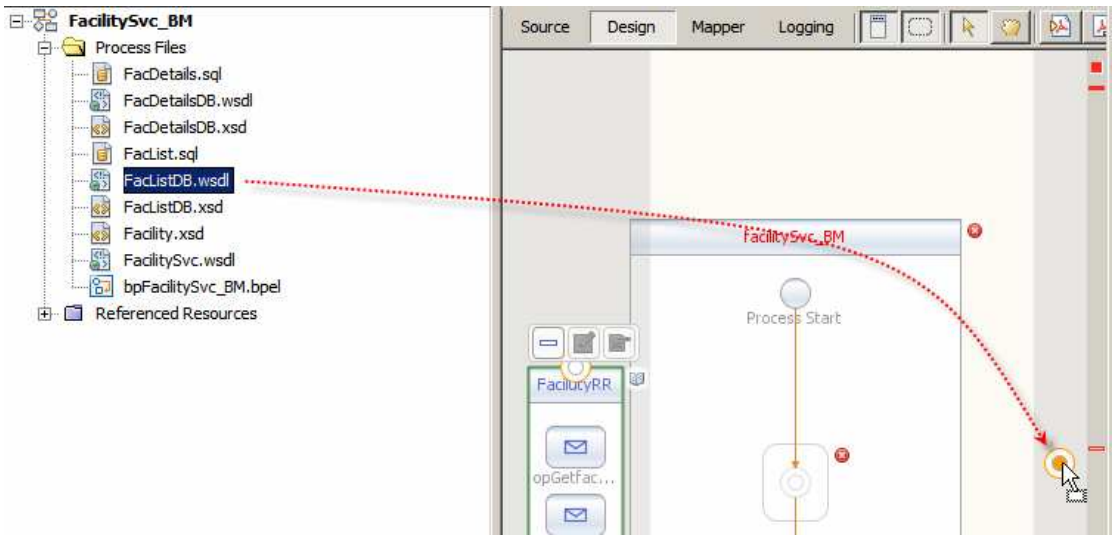
Let's drag the FacilitySvc WSDL onto the left-hand swim line of the bpFacilitySvc_BM process and release over the target marker.



Double-click the object to open the editor panel, change the name for FacilityRR and click OK.

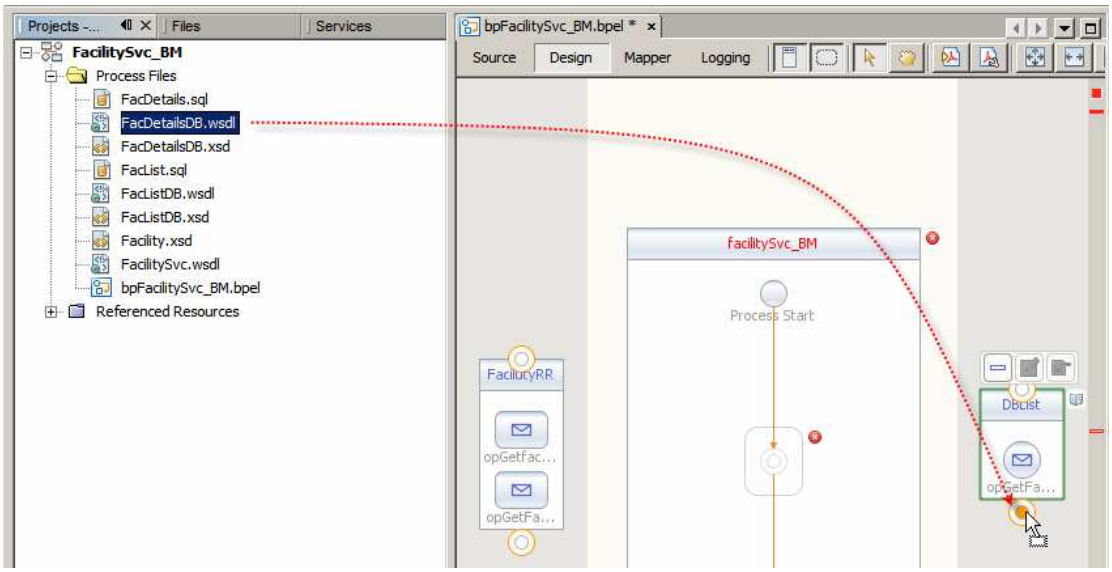


Let's now drag the FacListDB WSDL and drop it onto the target marker at the right-hand swim line.



Rename the partner link to DBList.

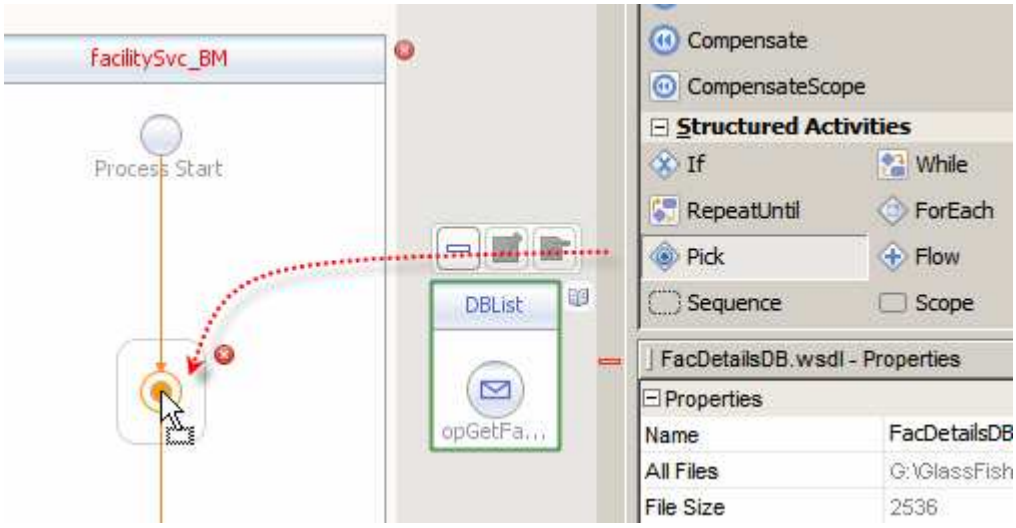
Let's drag the FacDetailsDB WSDL onto the right-hand swim line and release. Rename the partner link to DBDetails.



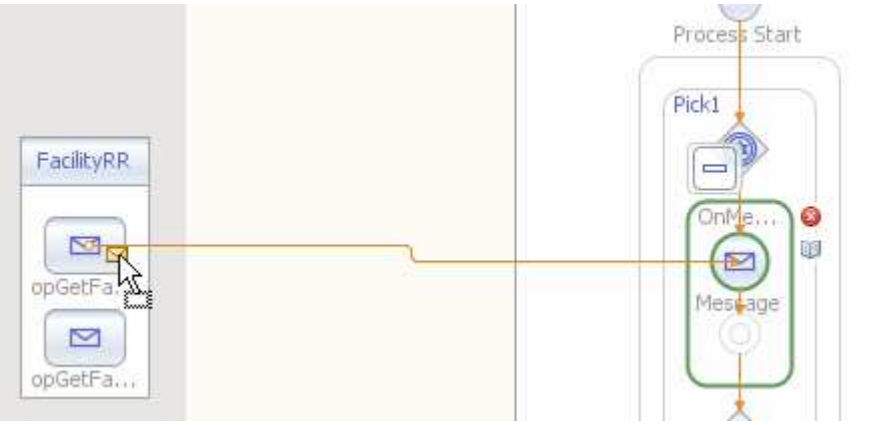
The FacilitySvc WSDL (FacilityRR partner link) has two operations – opGetFacilityList and opGetFacilityDetails. We will implement each as a separate stream of activities starting with the Pick activity.

Pick Activity is a 'either or' selection. The service will be invoked using either the opGetFacList operation or the opGetFacDetails operation. Implementation of the logic required by each operation is different. The mechanism that allows the BPEL Process to distinguish between the operations required for the particular invocation is the Pick Activity.

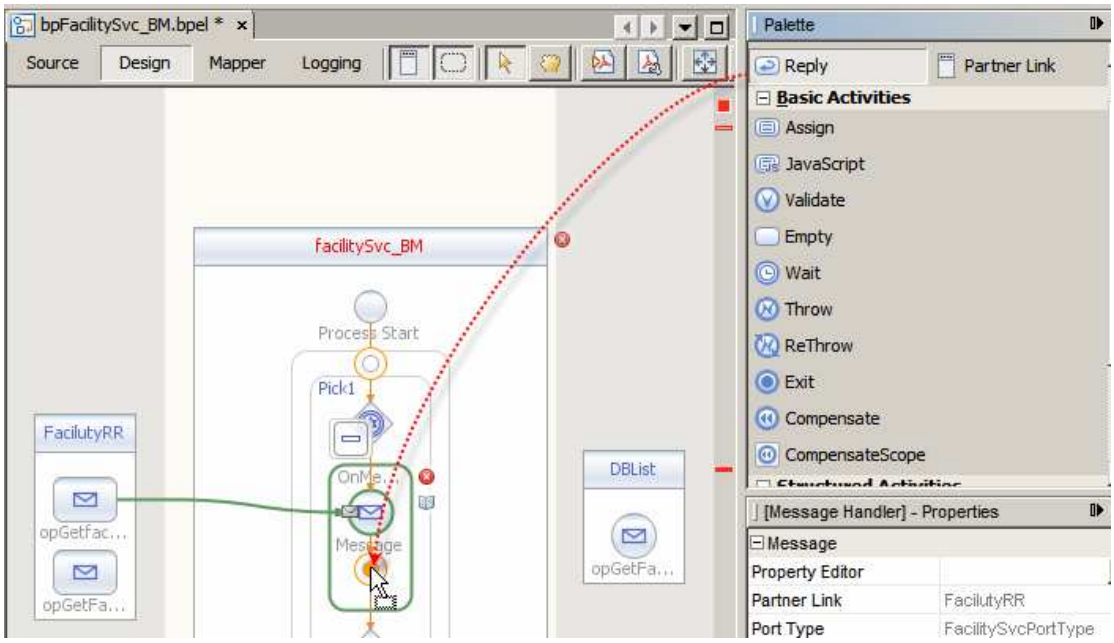
Let's drag the Pick activity from the palette to the canvas and drop it at the target marker in the process scope.



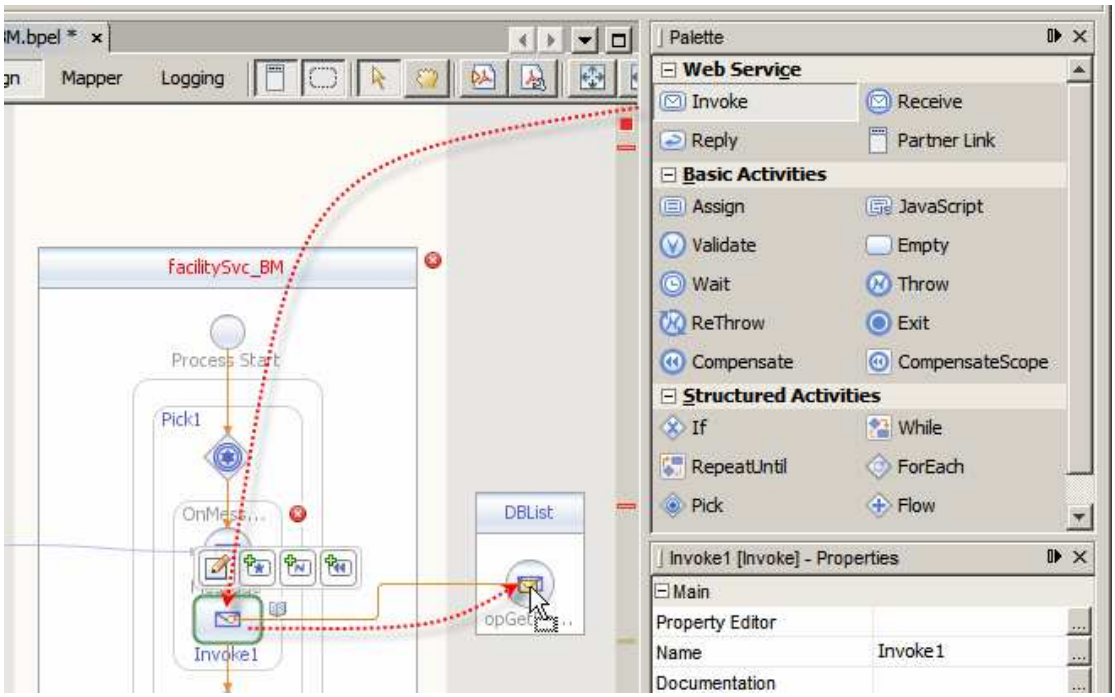
Connect the "OnMessage" Activity to the first operation of the FacilityRR partner link.



The service is a request/reply service. OnMessage acts as a receive activity for the specific operation. The Reply Activity will provide the reply on completion. Let's drag the Reply Activity onto the target marker and release.

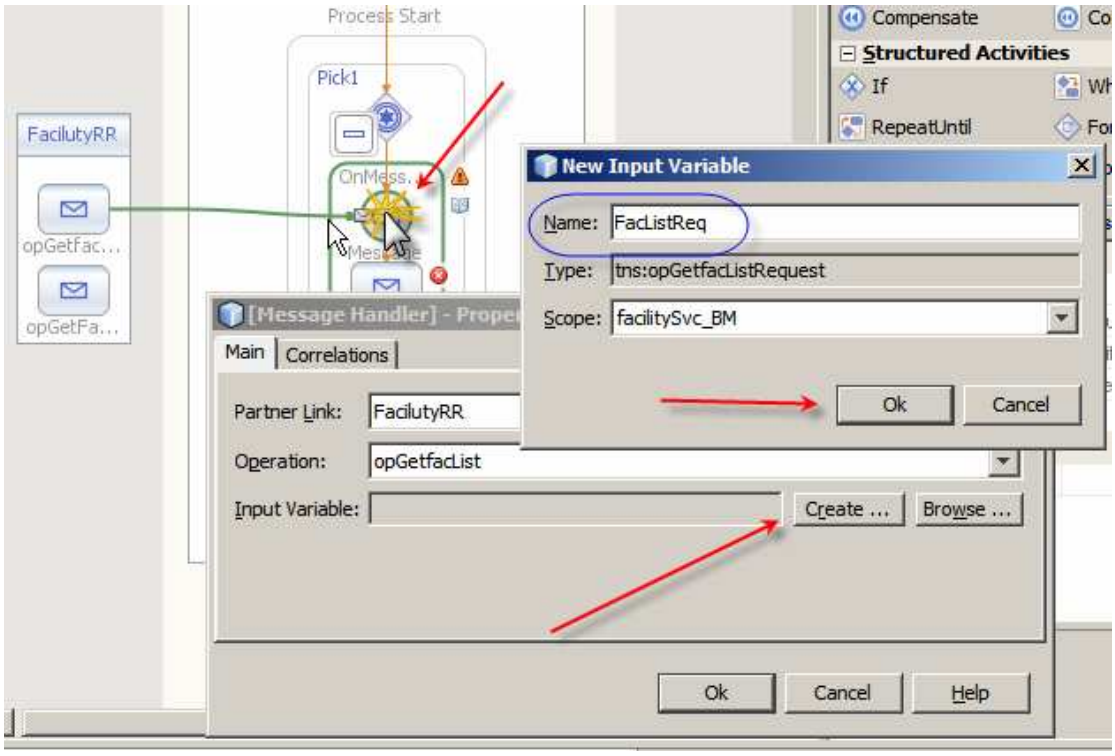


Between receiving a trigger message and replying, the service must access the database and receive the list of facilities. Let's drag the Invoke Activity, which will cause invocation of the DBList service, onto the canvas and connect it to the DBList service's operation.

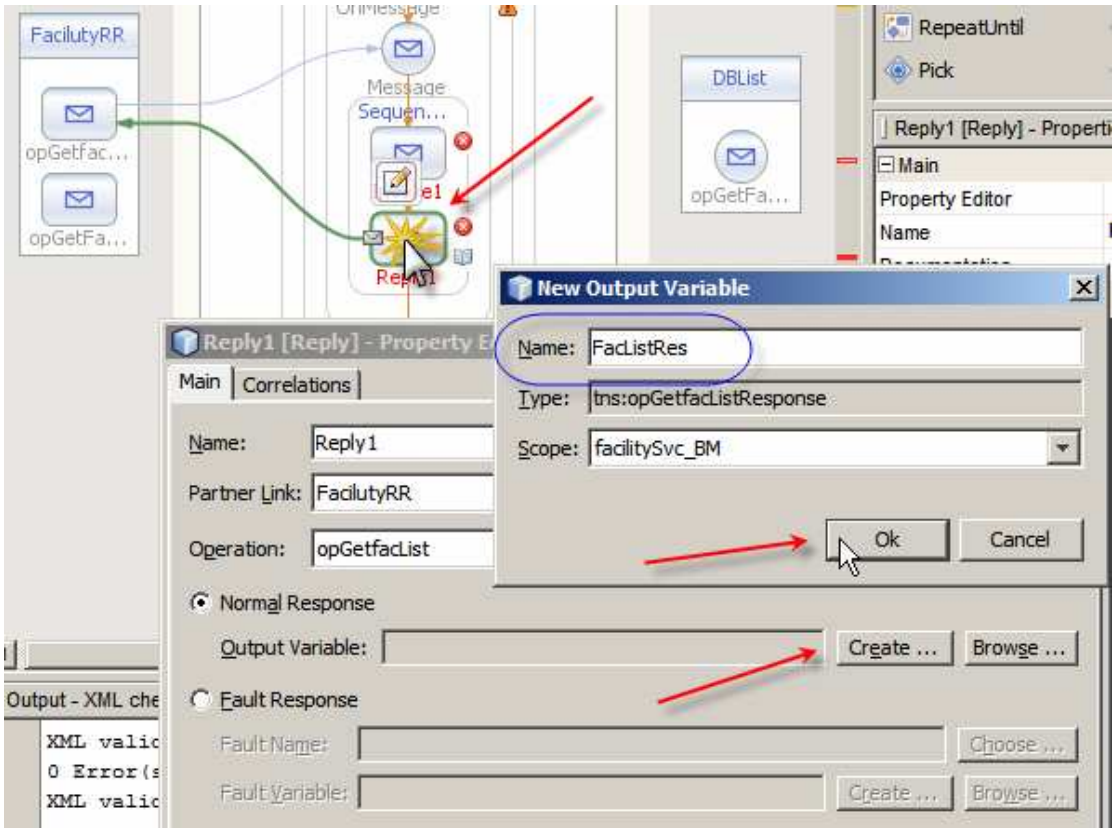


Let's create an input variable for the FacilityRR trigger message – FacListReq.

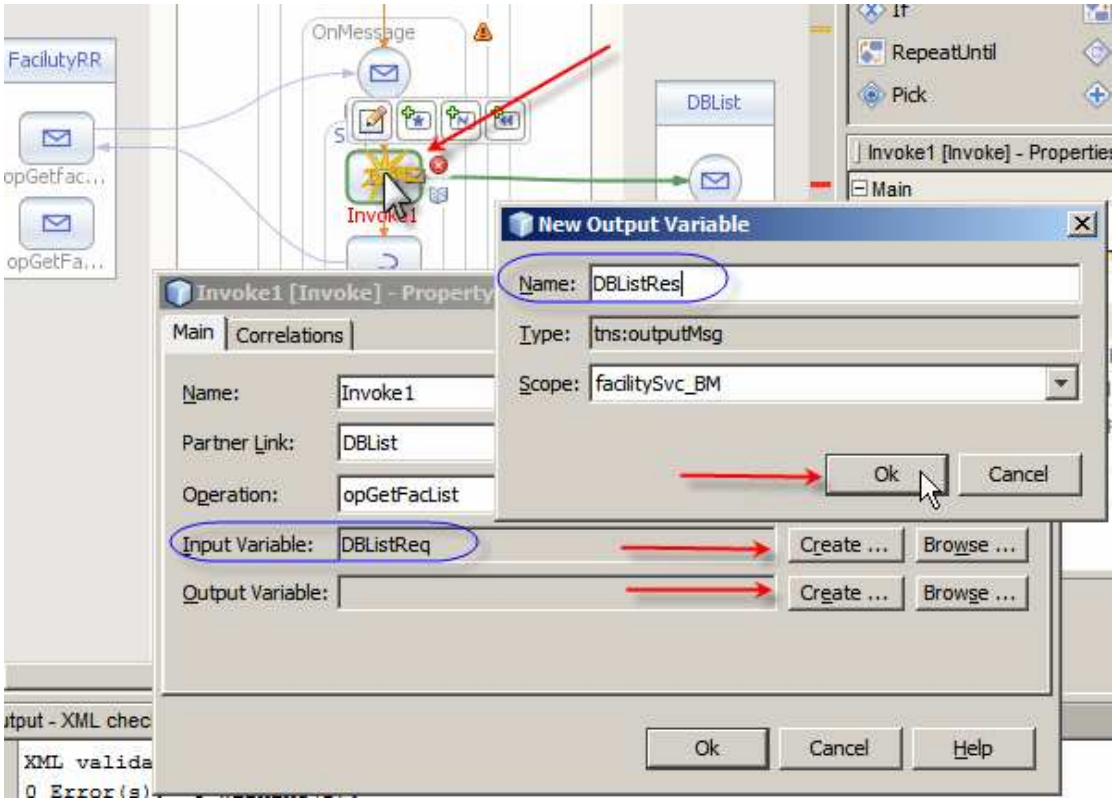
Double-click the OnMessage activity, click the Create button next to the Input message, change the name and complete.



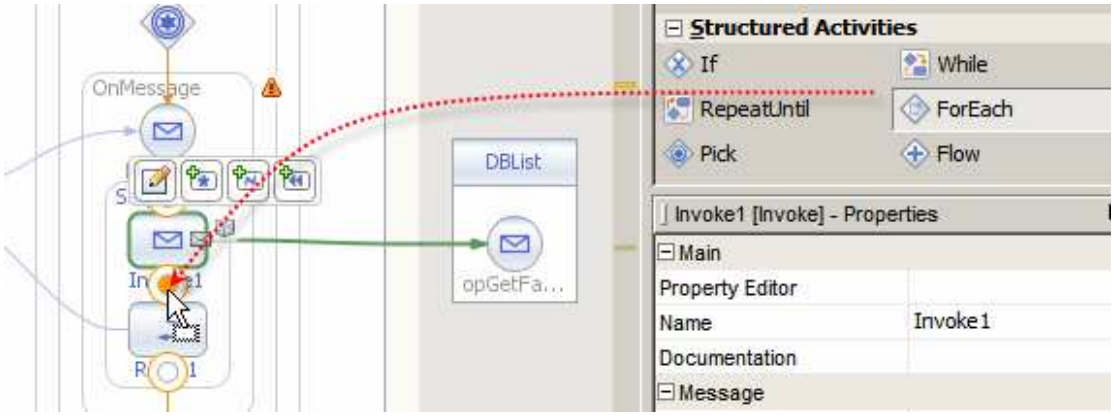
Let's connect the Reply1 activity to the FacRR opGetFacList operation and create the output variable for the Reply1 activity – FacListRes.



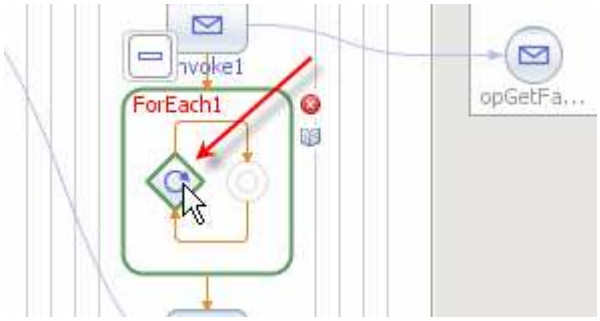
The input and output variables will contain the request and response messages. Invocation of the DBList partner will result in a set of records being returned. We need to connect the Invoke1 activity to the DBList partner and provide the input and output variables for this service. Double-click the Invoke1 activity and create input variable, DBListReq, and output variable, DBListRes, to hold the messages.



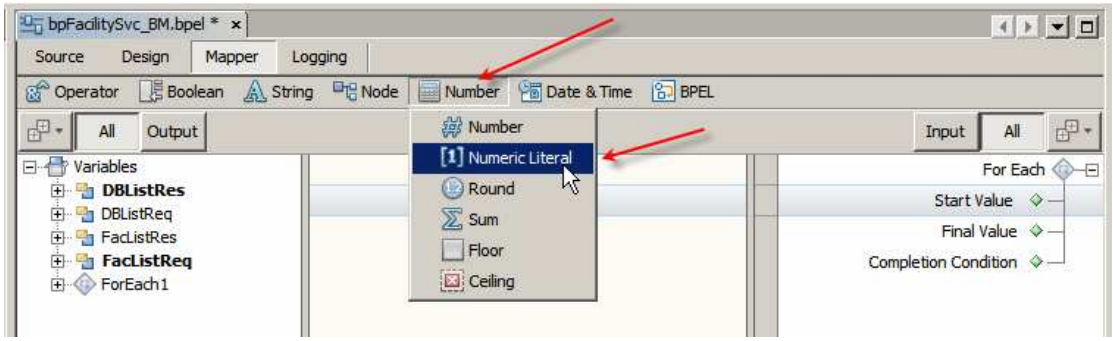
The output message is expected to contain the list of facility codes and descriptions. To allow our BPEL service to return this list to the caller we must map the list from the output message of the DBList service to the response message of the BPEL process. Since the structure is a repeating structure we need to add a ForEach loop in order to map each iteration from the DBListRes to FacListRes. Let's drag the ForEach activity to the target marker between Invoke1 and Reply1.



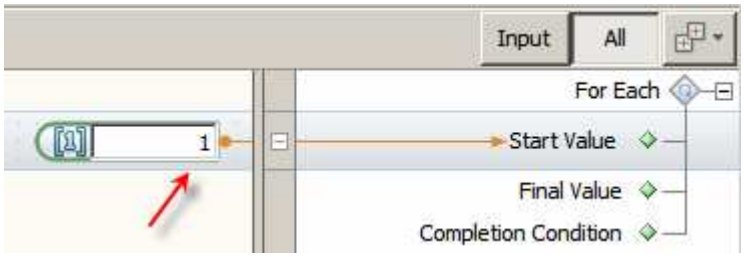
Let's configure the loop conditional to iterate over all elements of the list starting at 1 and continuing for as long as there are elements in the list. Double-click the ForEach icon.



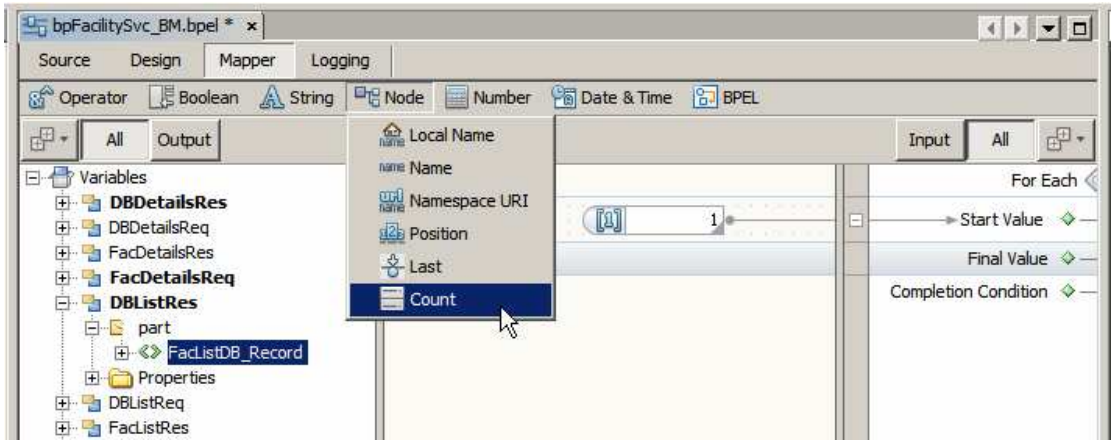
Select "Start Value", drop down the Number functiod list and choose Numeric Literal.



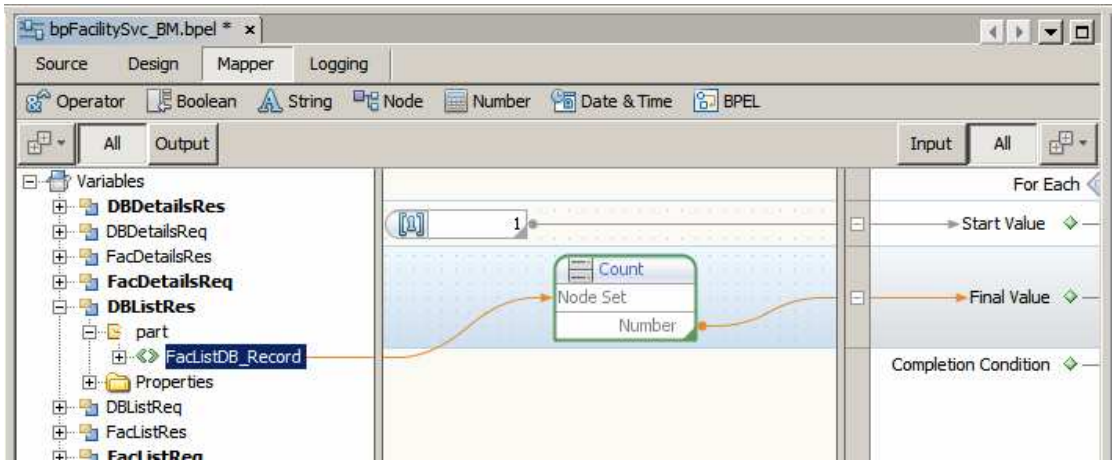
Change the value to 1, which is the BPEL's base index, and connect to the Start Value node.



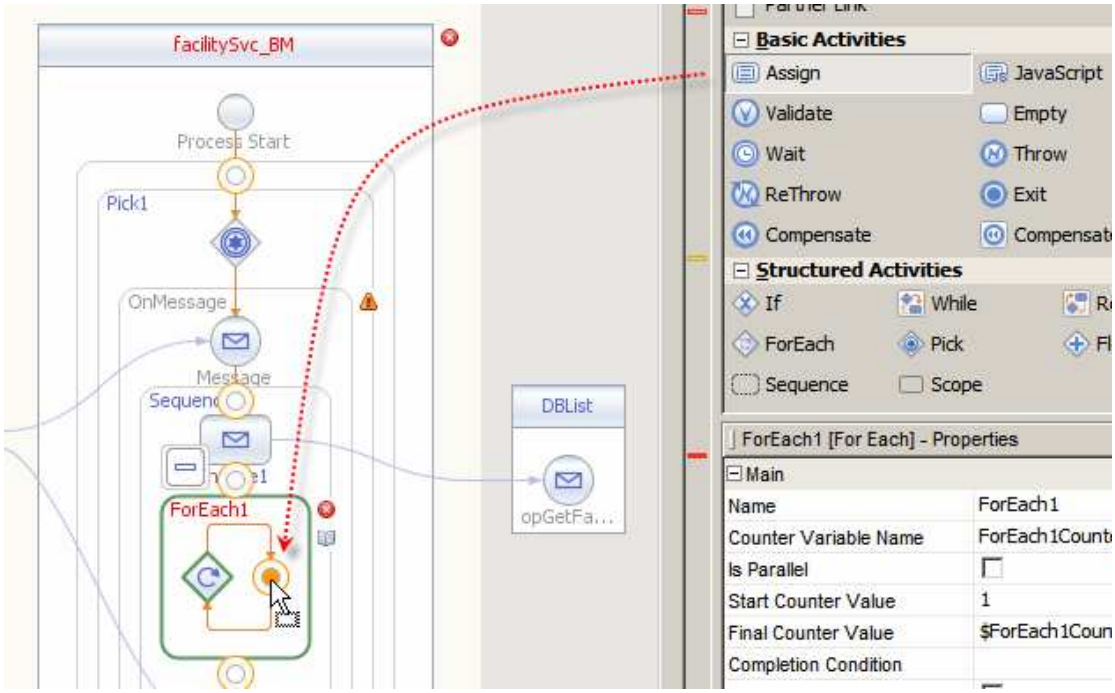
The Final Value node will be the count of items in the list. There is a Count functoid that will be used to count the number of items. Lets select Final Value, drop down the Node functiod list and click the Count functoid.



Now connect the FacListDB_Record node to the Count and the Count to the Final Value.

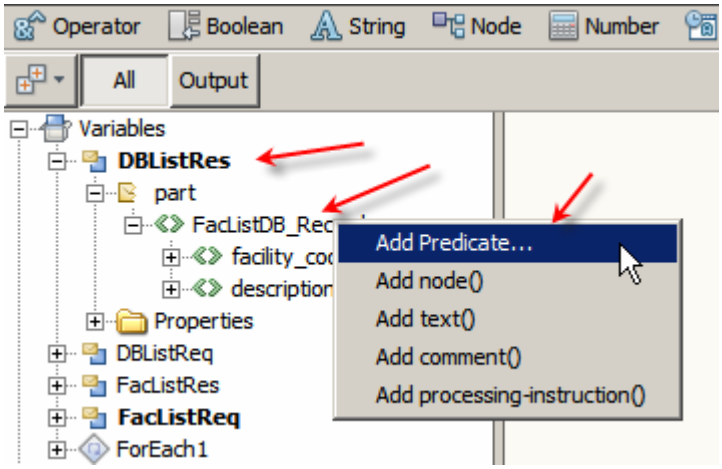


Let's now add the Assign Activity to the ForEach loop to perform the actual mapping.

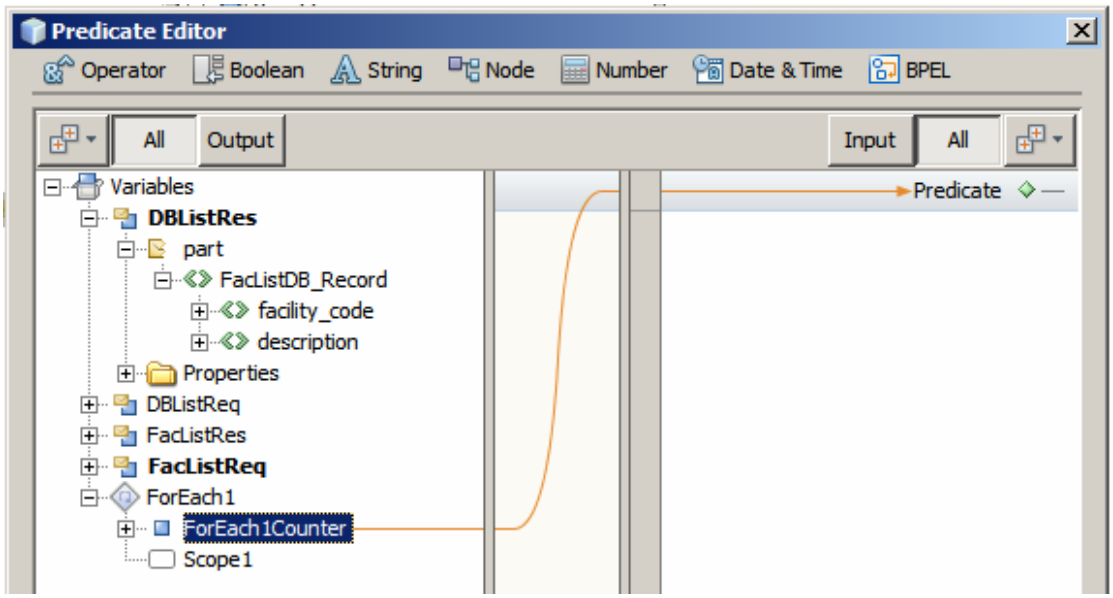


Double-click Assign1 to switch to the Mapper editor and map the nodes of the DBListRes structure to the corresponding nodes of the FacListRes structure. This will be done in a few stages, for clarity.

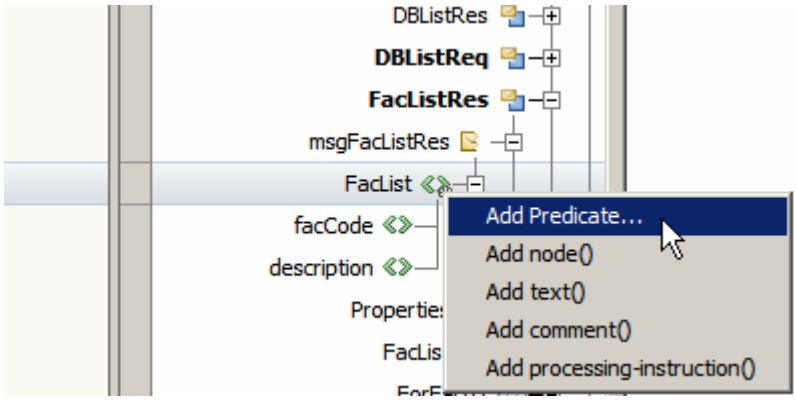
Let's first add a predicate to the DBListRes at the FacListDB_Record node level.



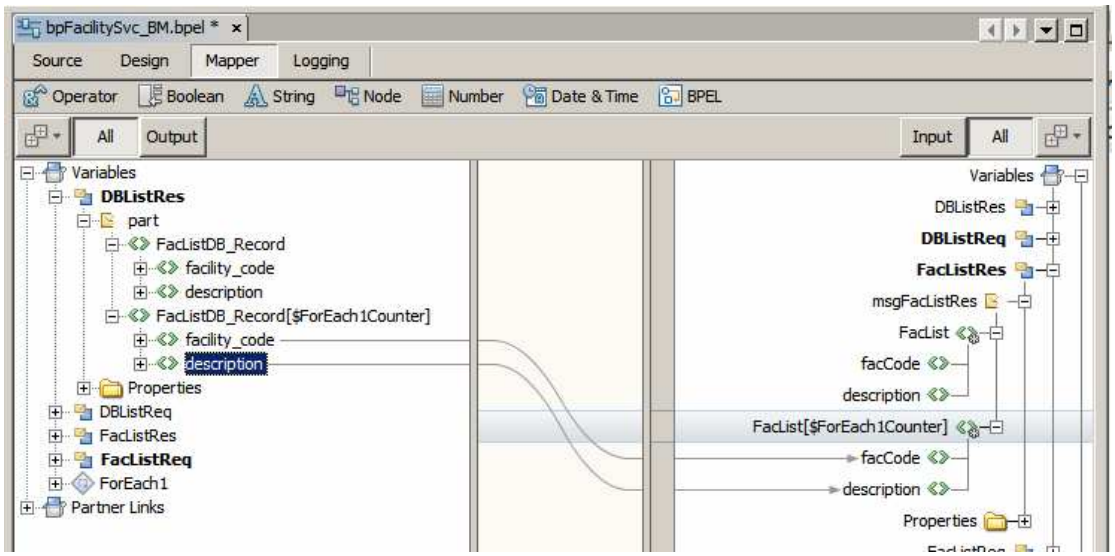
The predicate will be based on the current value of the ForEach1Counter node.



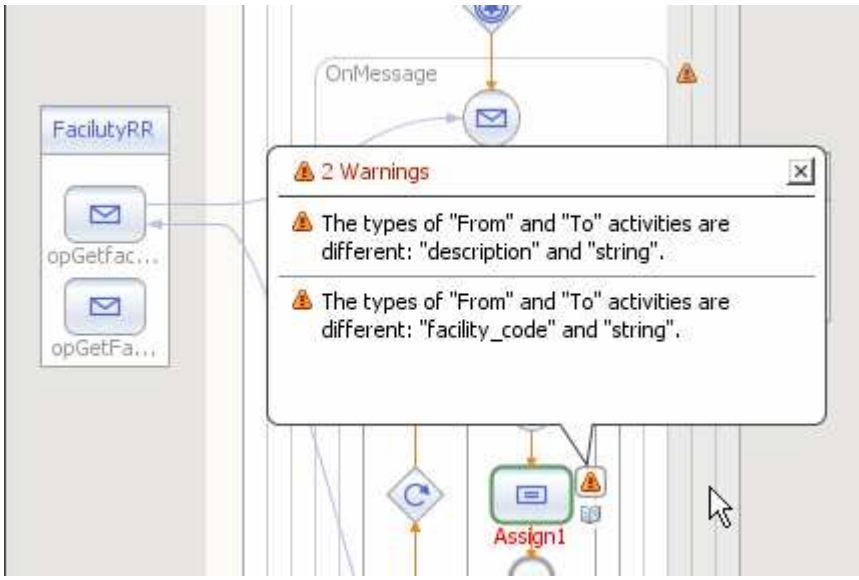
Let's add a predicate for the target, FacListRes at the FacList repeating node.



Set the predicate to the current value of the ForEach1Counter node as well. This way the assignment, which comes next, will use corresponding entries in the source and target lists.

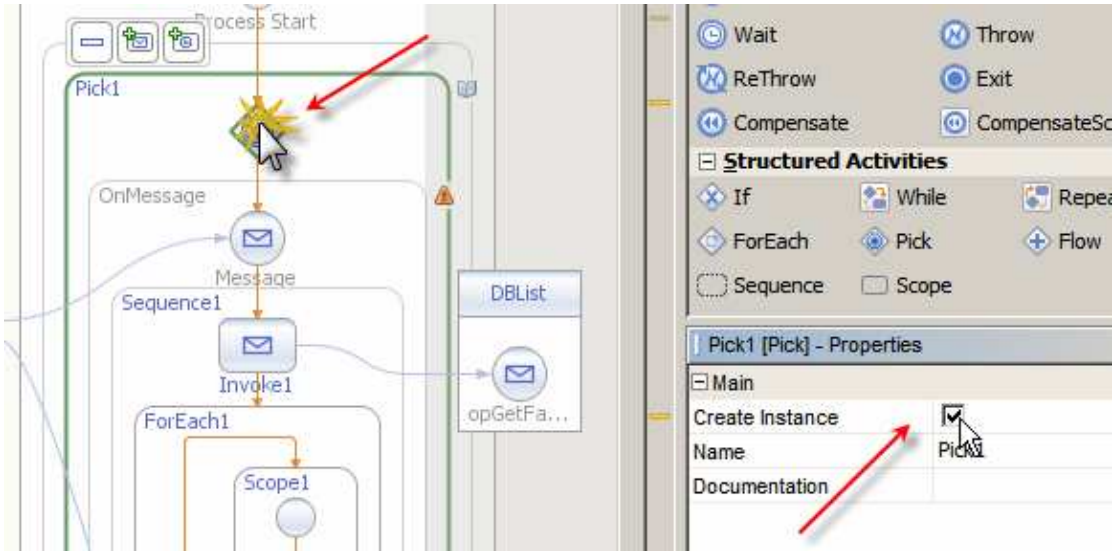


Switch to Design view, click the Warning icon next to the Assign1 Activity and note the warnings.

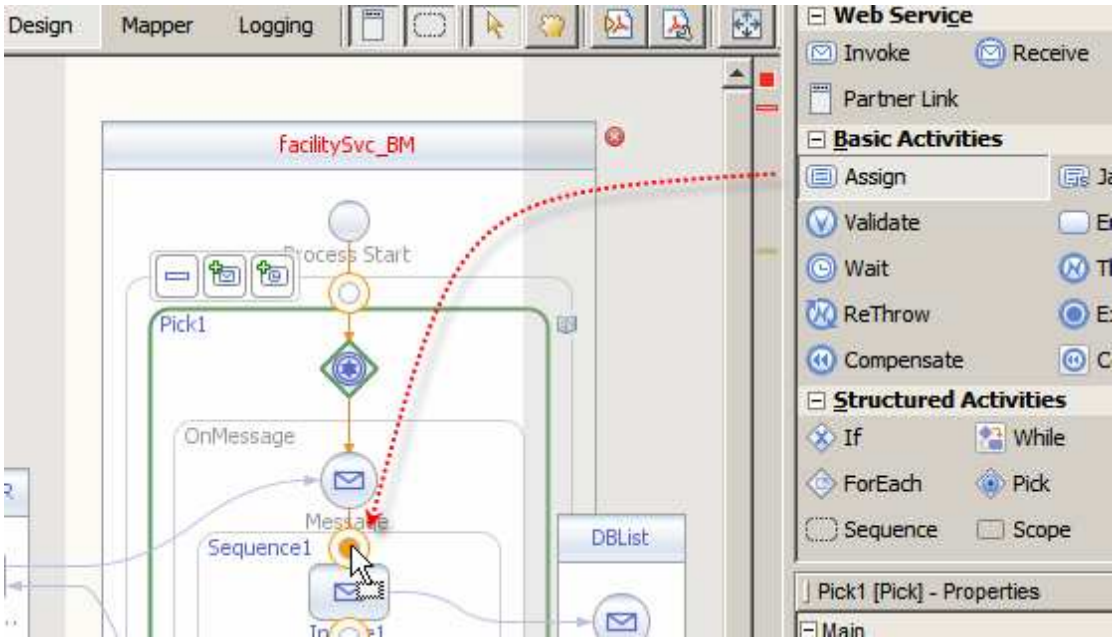


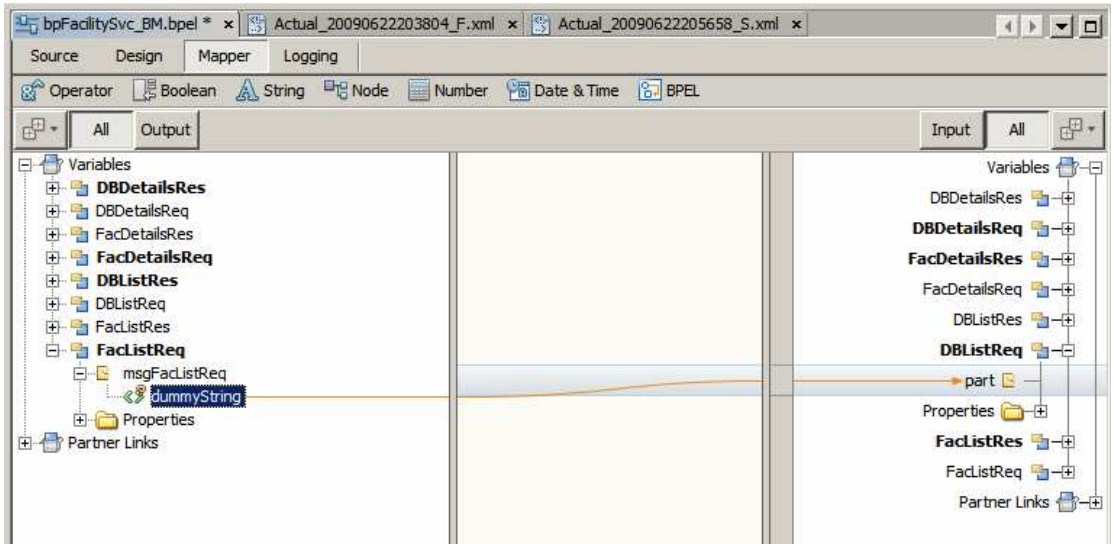
The BPEL editor complaints that the nodes don't have the same data types. Ignore the warning since datatype in both cases are actually compatible.

Click on the Pick symbol and make sure to check the "Create Instance" property.



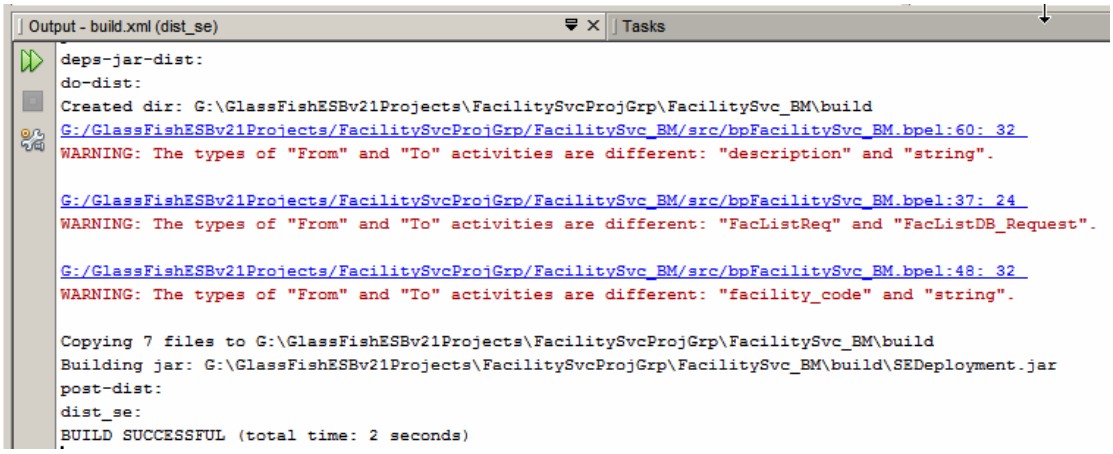
While the DBList service does not require input the BPEL process will not be able to be built if the input message to this service is not initialized and if the input message to the process itself is not used. Let's add an Assign activity and map the FacListReq ->dummyString message node to the DBLisgReq message node.





Right-click the name of the project and choose Build.

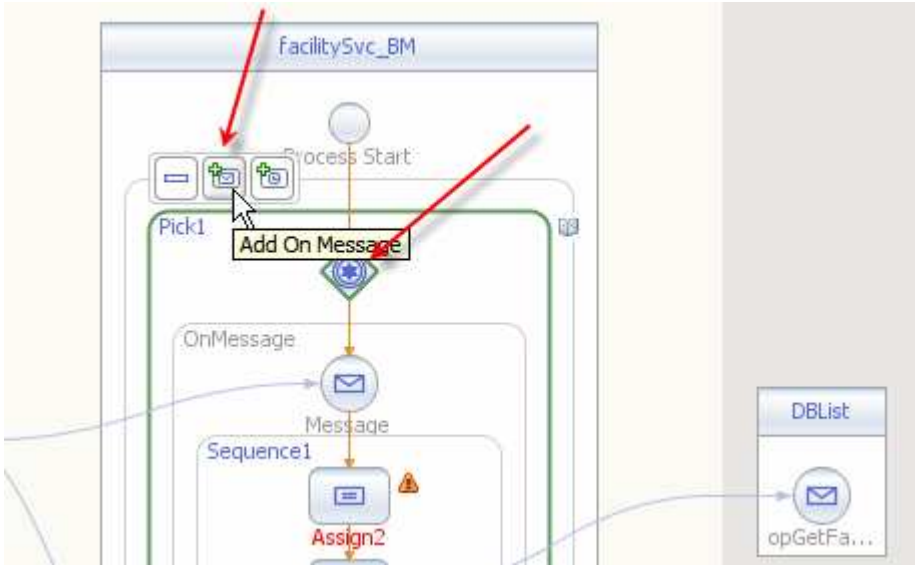
The process will be built successfully with warnings. Note the warnings – review them and ignore them for this process.



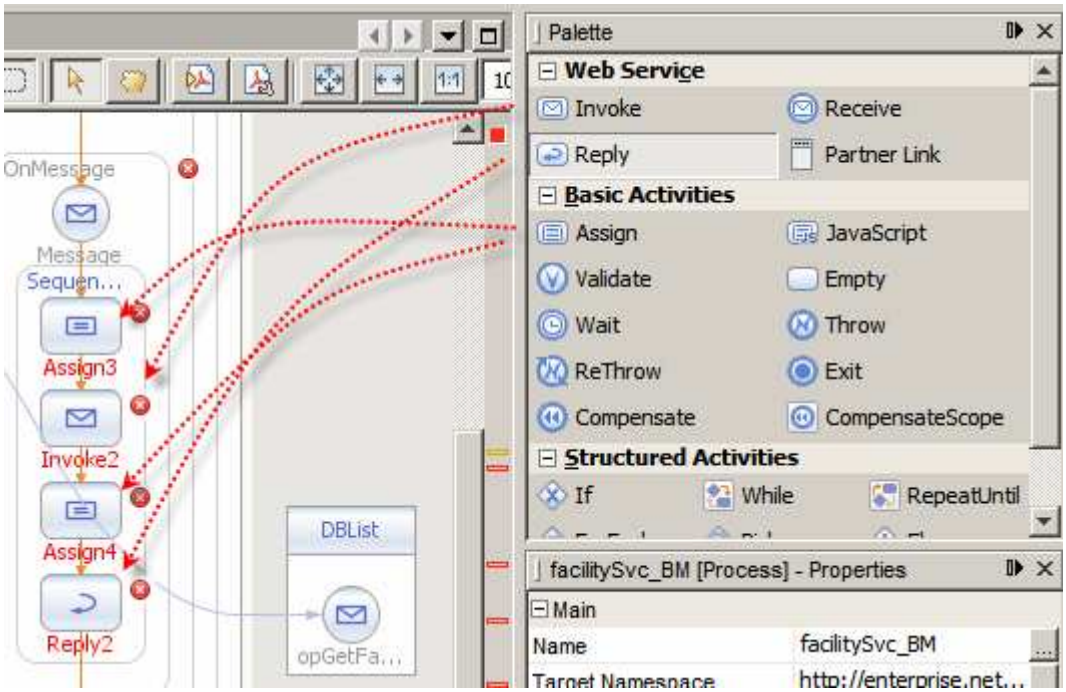
This completes the opGetFacList operation implementation.

Now we will implement the opGetFacDetails operation. Here the input message will contain the facility code used to look up the facility whose details are to be returned.

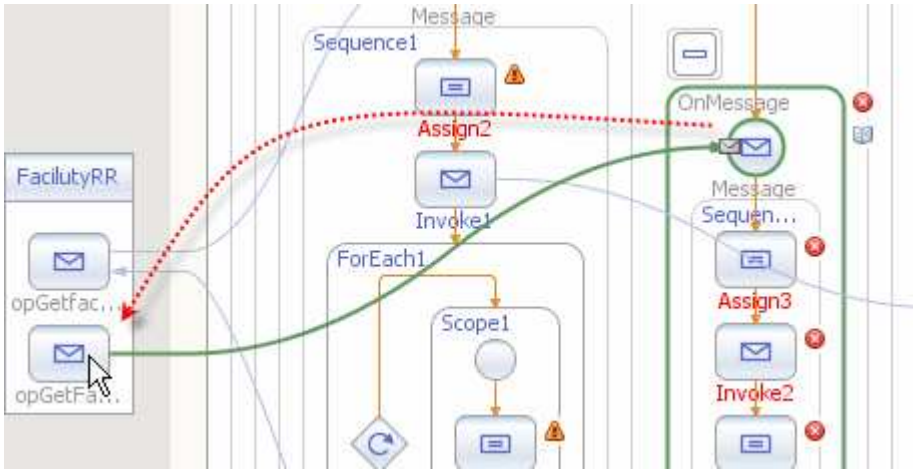
Click the Pick symbol and the Add OnMessage icon.



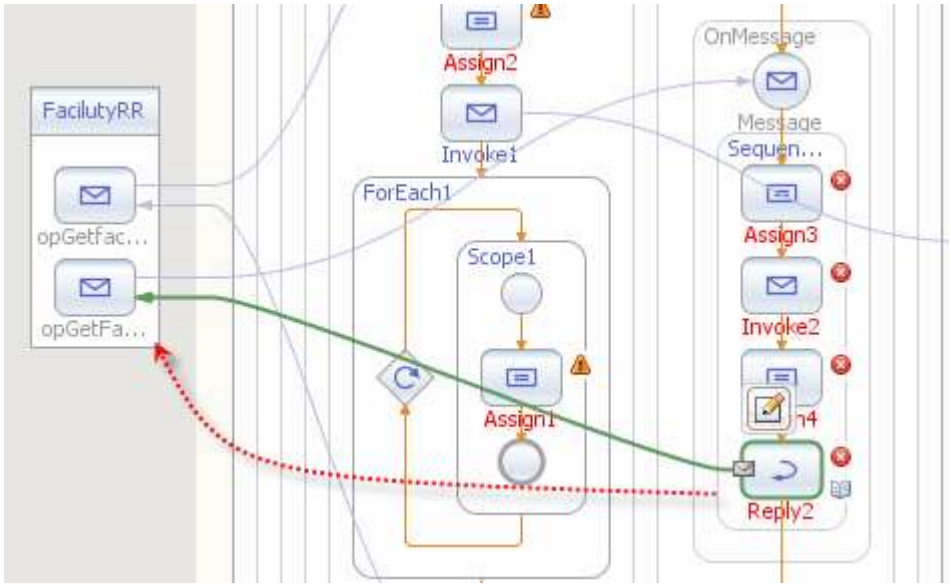
Add Assign, Invoke, Assign and Reply activities.



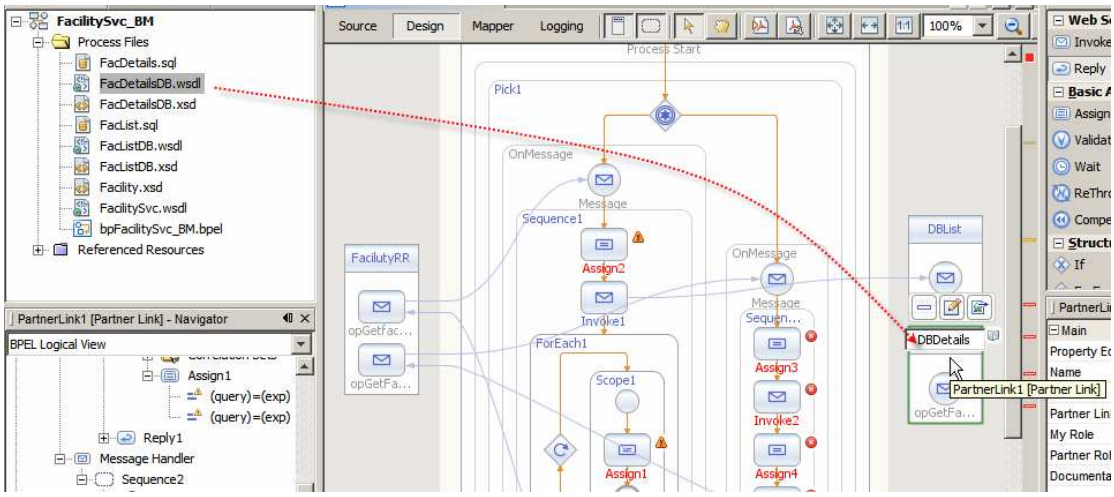
Connect the new OnMessage activity to the opGetFacDetails operation.



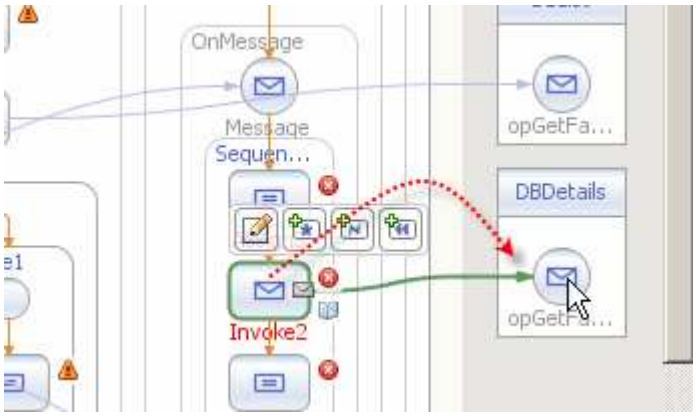
Connect Reply2 to the opGetFacDetails operation.



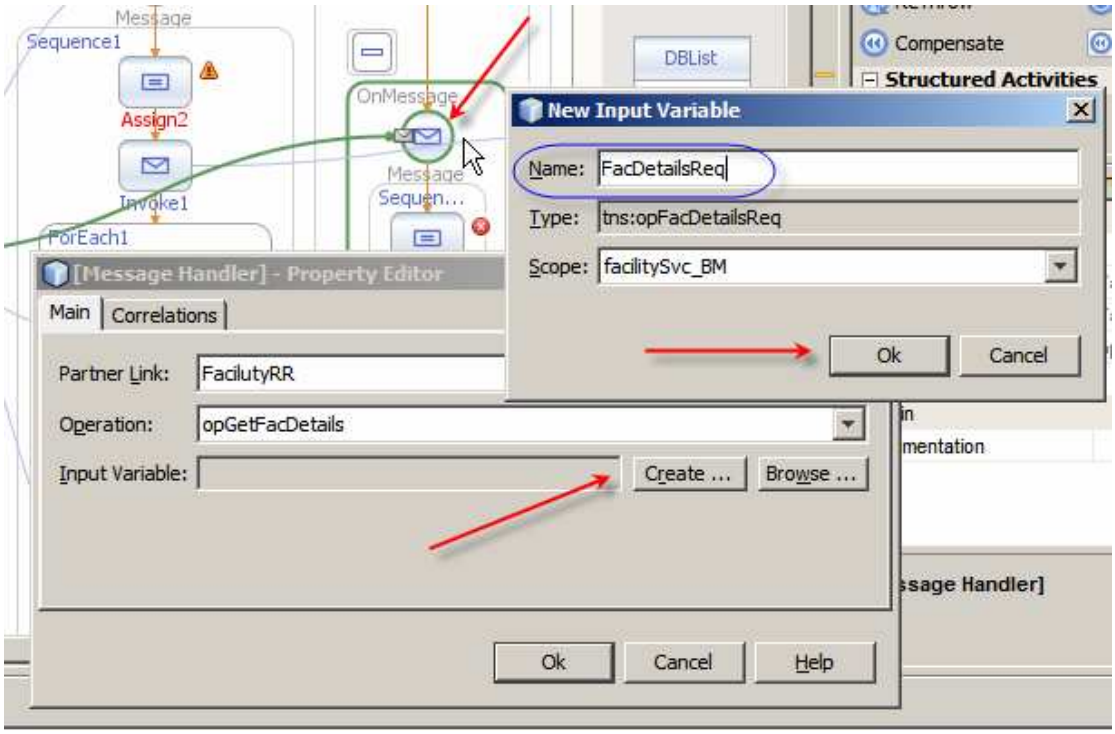
Drag the DBDetails.wsdl onto the target market at the right hand swim line and rename it to DBDetails.



Connect Invoke2 to the DBDetails partner operation.

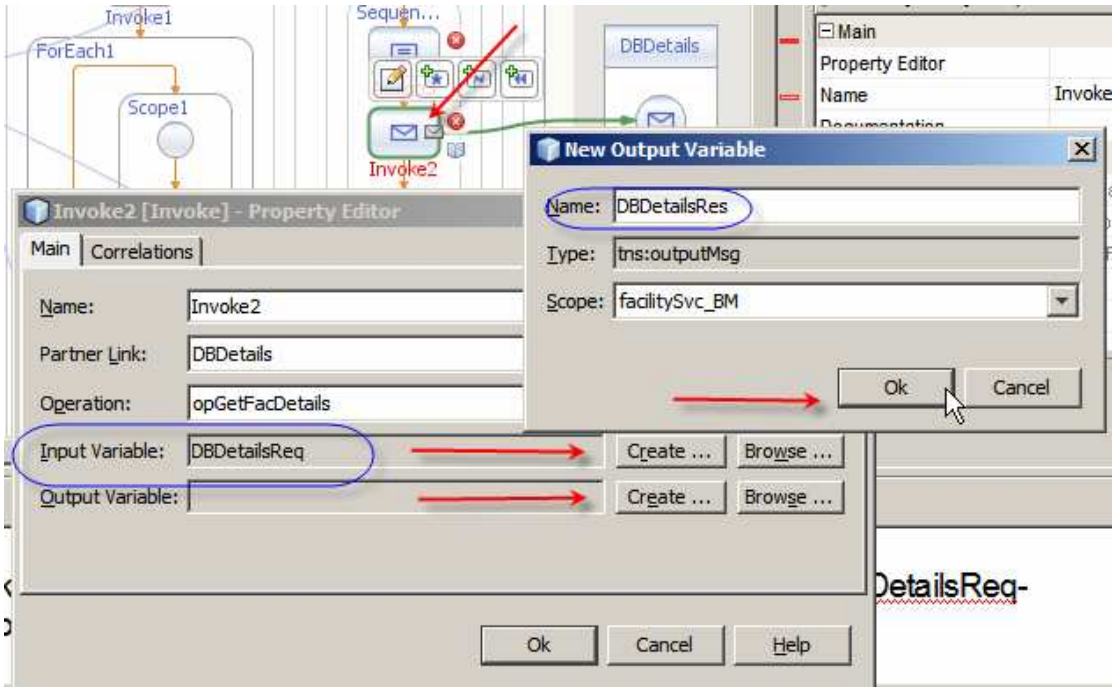


Double-click the new OnMesage activity and create a new Input Variable FacDetailsReq.

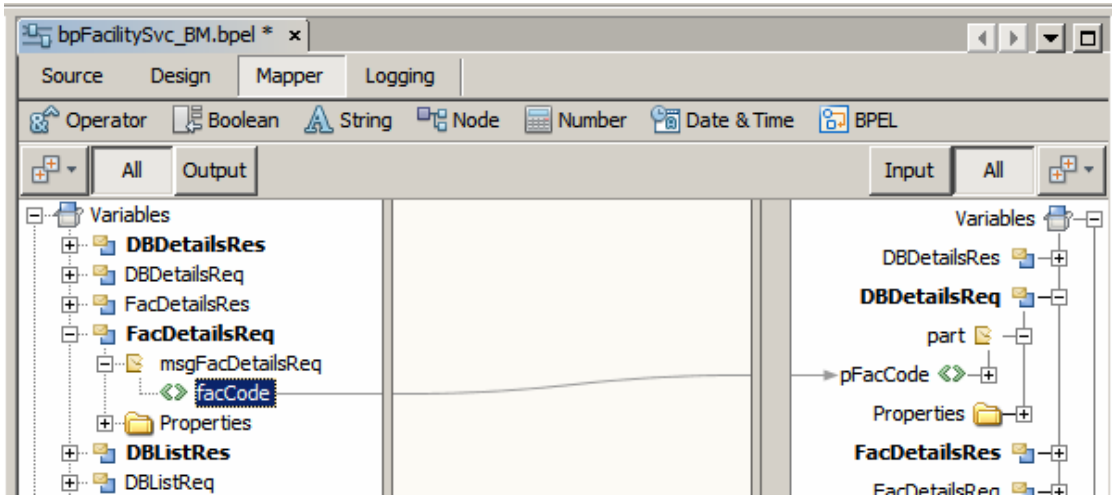


Double-click the Reply2 Activity and create a new Output Variable FacDetailsRes.

Double-click the Invoke2 Activity and create two new variables, Input DBDetailsReq and Output DBDetailsRes.

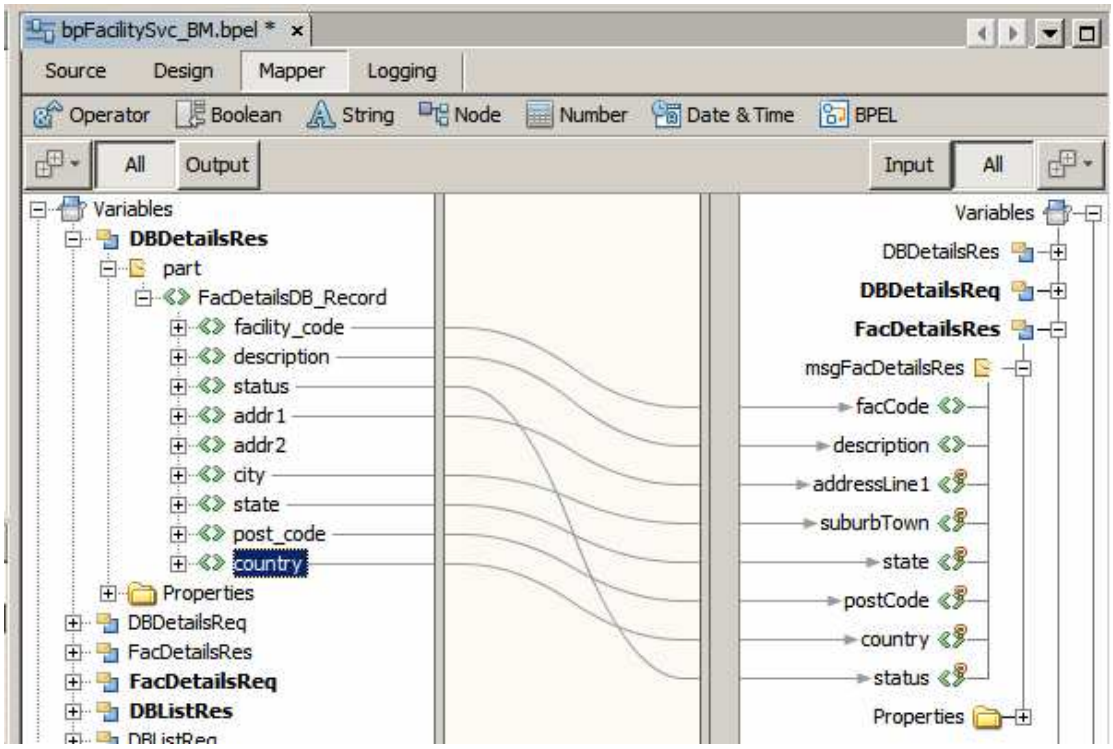


Double-click Assign3 and add mapping from FacDetailsReq->msgFacDetailsReq->facCode to DBDetailsReq->part->pFacCode.

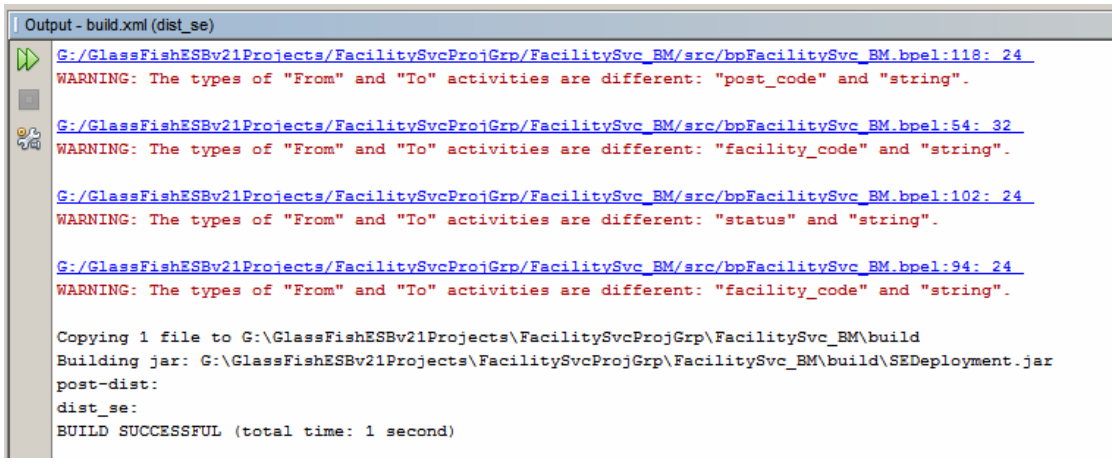


Switch back to Design view.

Double-click Assign4 activity and map nodes of the DBDetailsRes structure to the corresponding nodes of the FacDetailsRes structure.

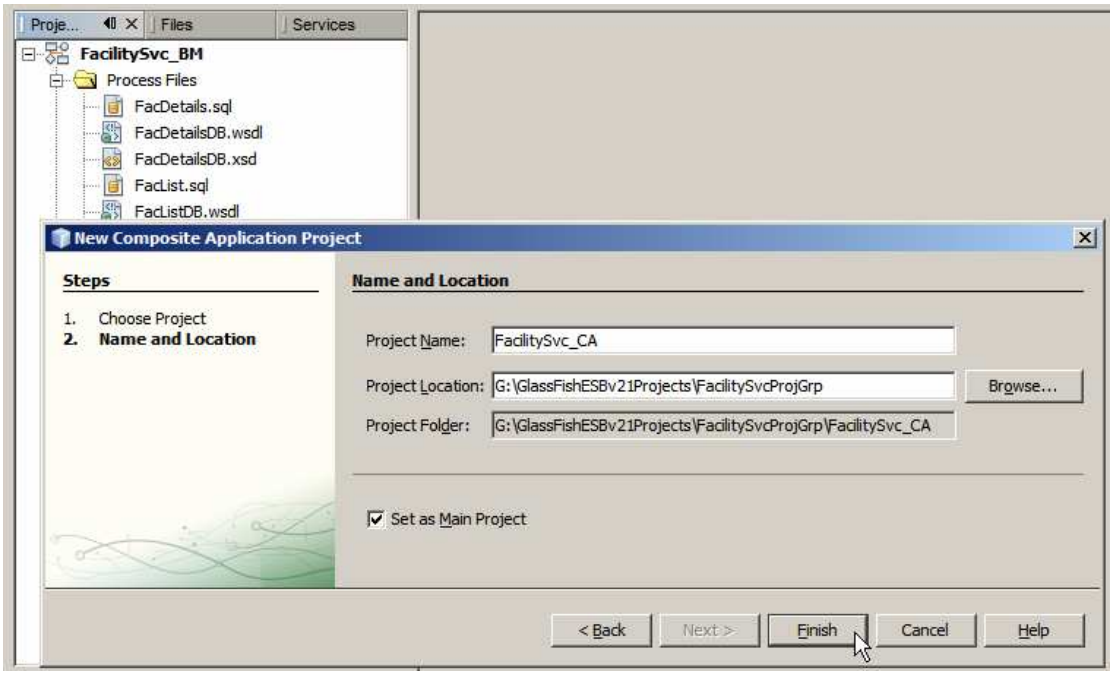


Switch to Design view, right-click project name and choose Build. The build should complete successfully with warnings.

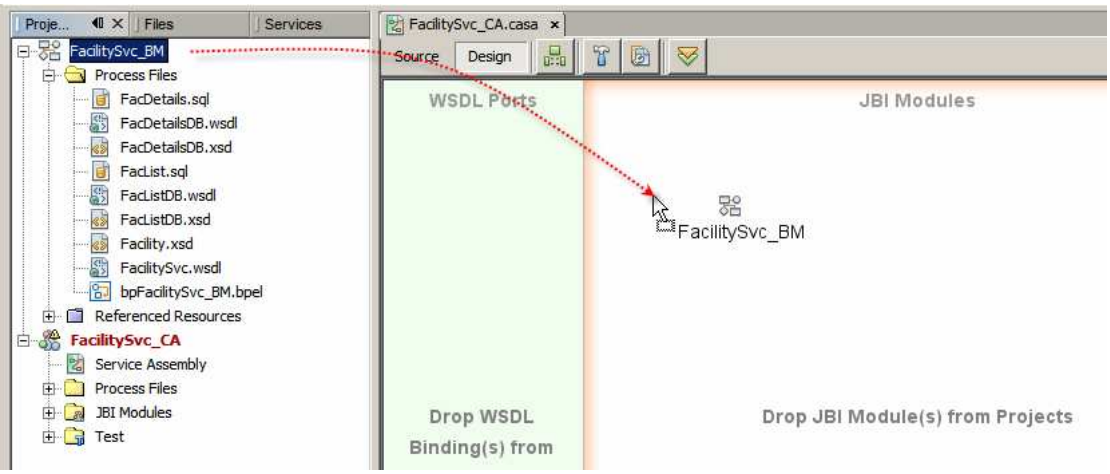


The process is complete.

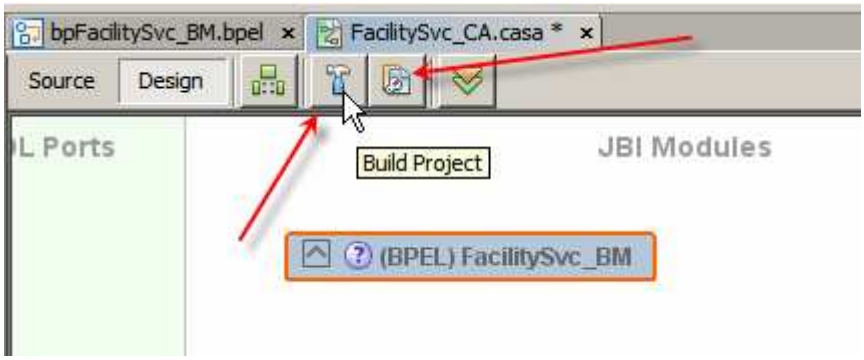
To deploy this piece of logic to the runtime we need to create a Composite Application Project. Let's call it FacilitySvc_CA.



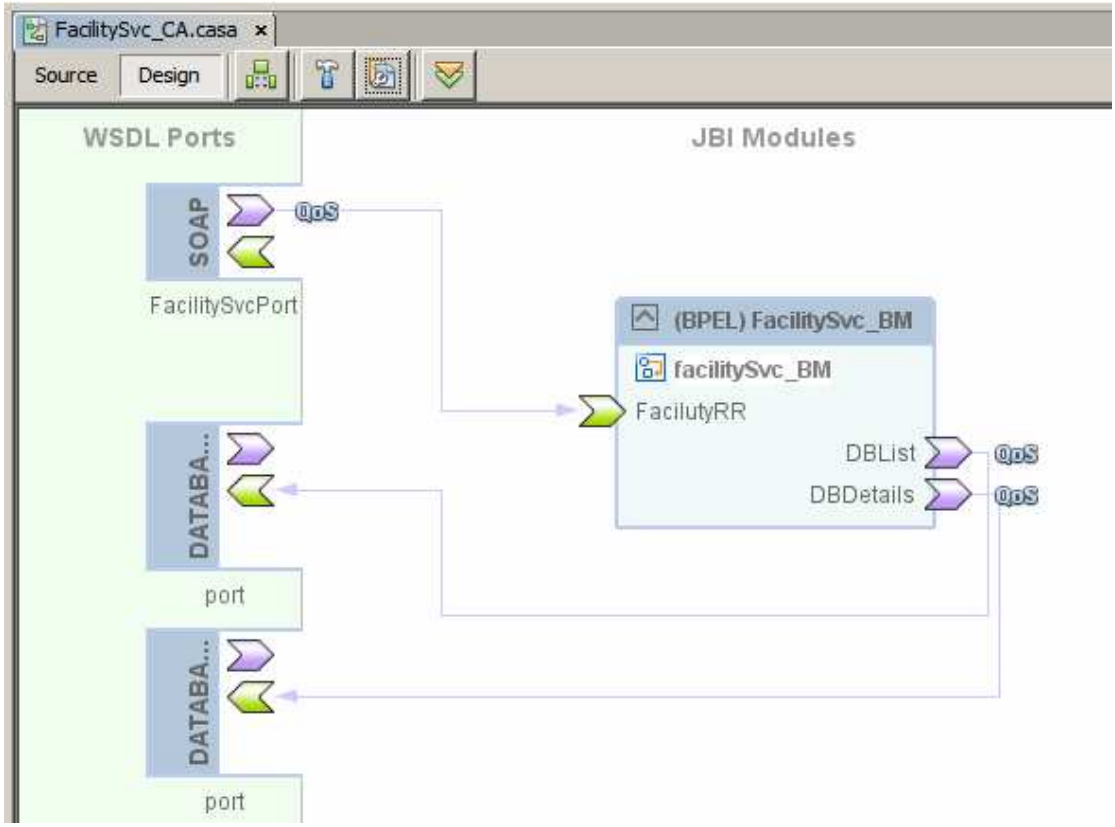
Drag the project FacilitySvc_BM, by project name node, onto the CASA Editor canvas.



Click Build, and when completed, Deploy buttons.

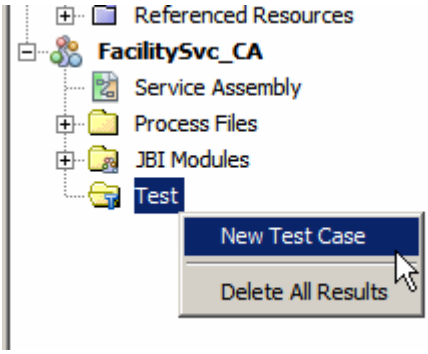


The complete CASA map will look similar to that shown below.

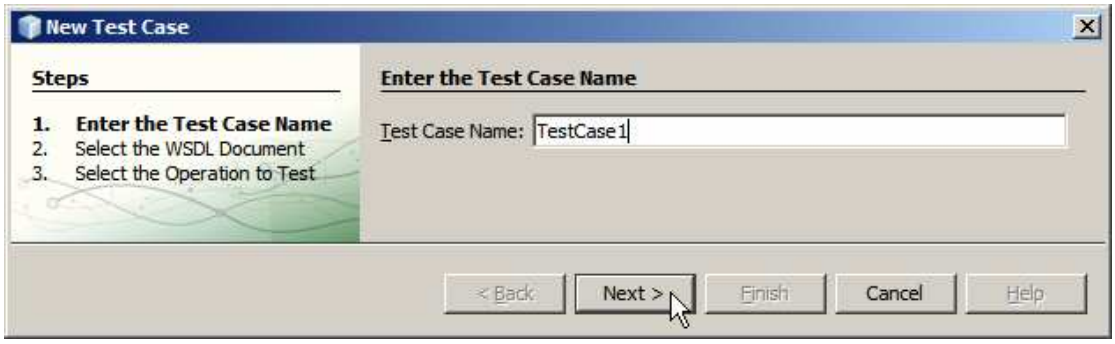


Let's now test the service using the built-in testing facility.

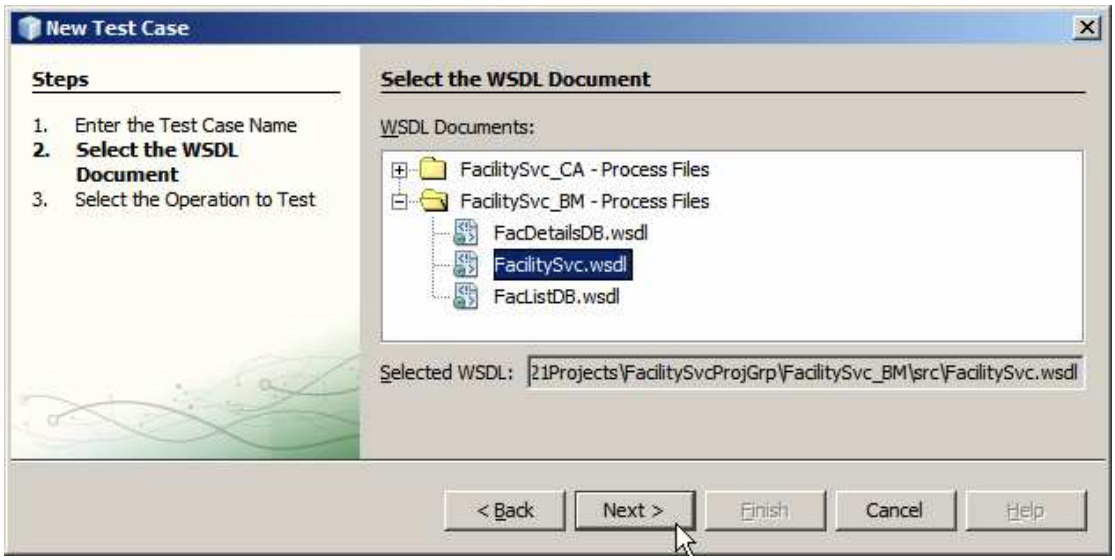
Right-click the Test node and choose New Test Case.



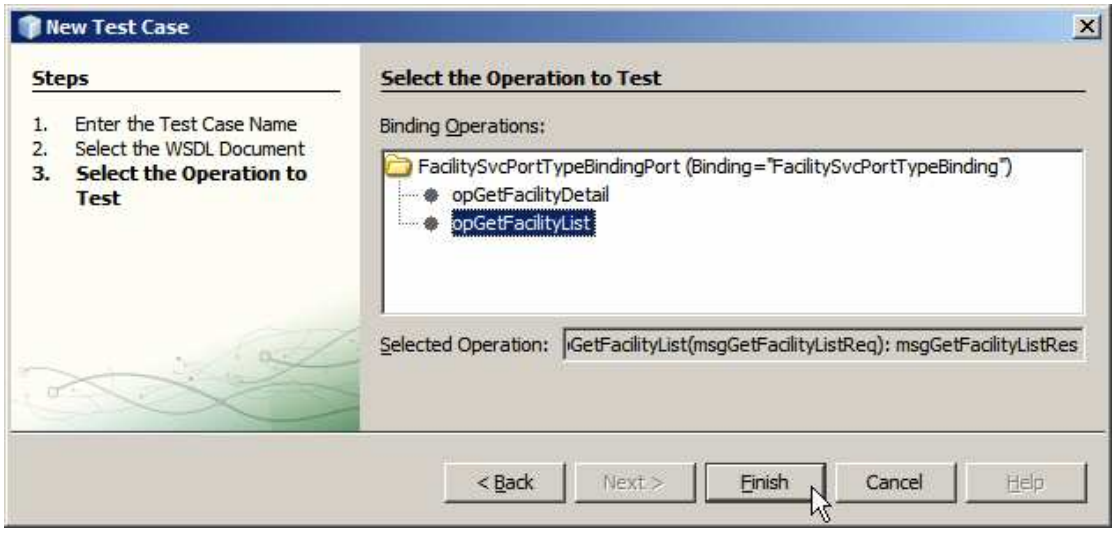
Accept default name.



Choose SOAP WSDL.

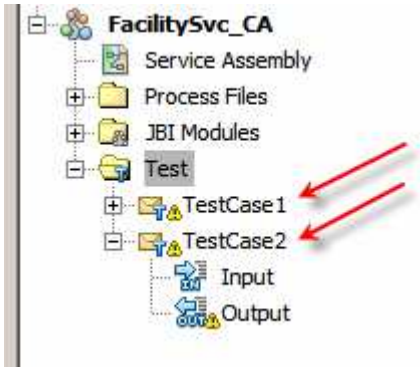


Choose opGetFacList operation and click Finish.



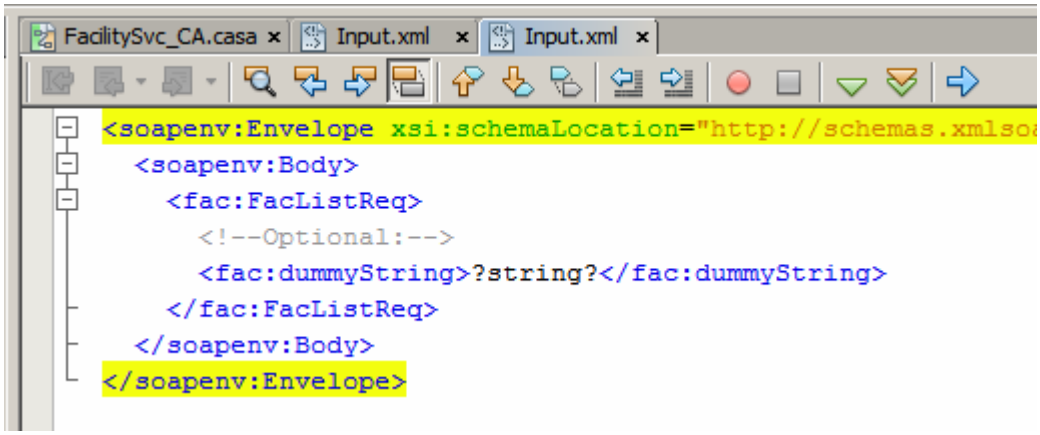
It may be that the test case will not appear. There appears to be a bit of an issue with NetBeans at this point. Let's go ahead and create TestCase2 for testing the other operation. Right-click the Test node, choose New test case, accept the default name testCase2, choose FacilitySvc.wsdl in the FacilitySvc_BM project's process files, choose opGetFacDetails operation and click Finish.

Note the two test cases now appear in NetBeans.

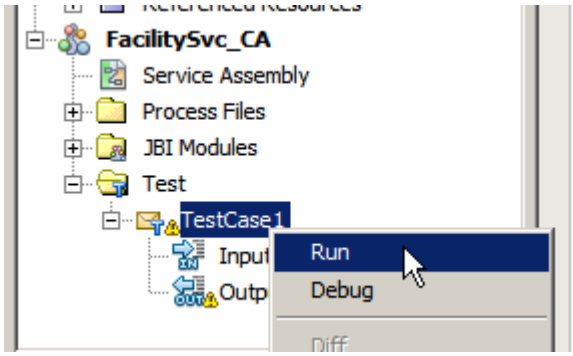


Expand the TestCase1 and look at the input message.

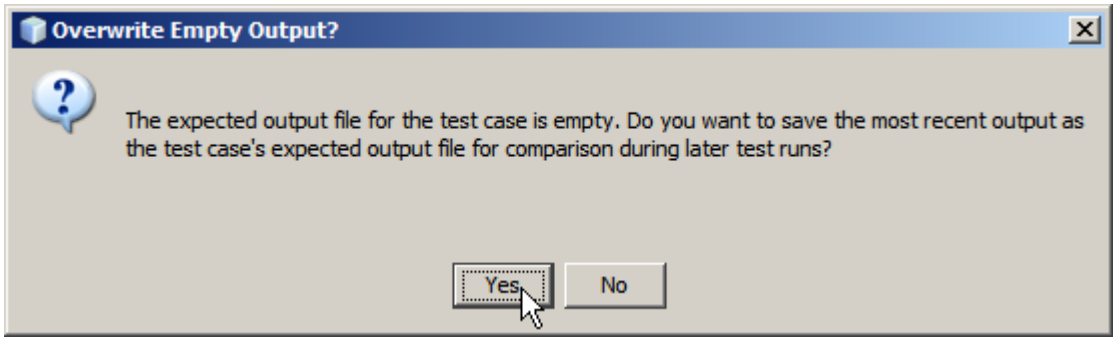
Leave the request as is. Recall that the List operation requires no parameters.



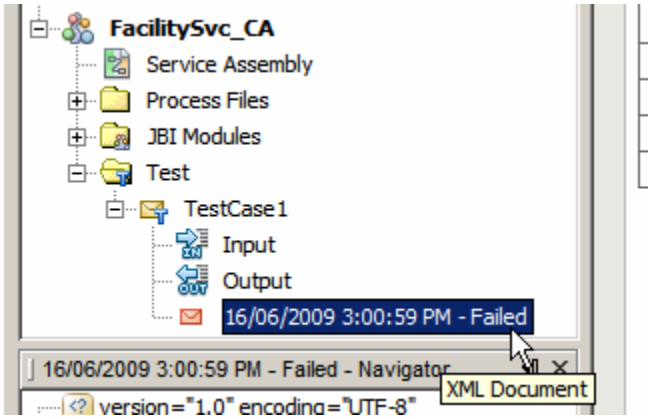
Right-click TestCase1 and choose Run.



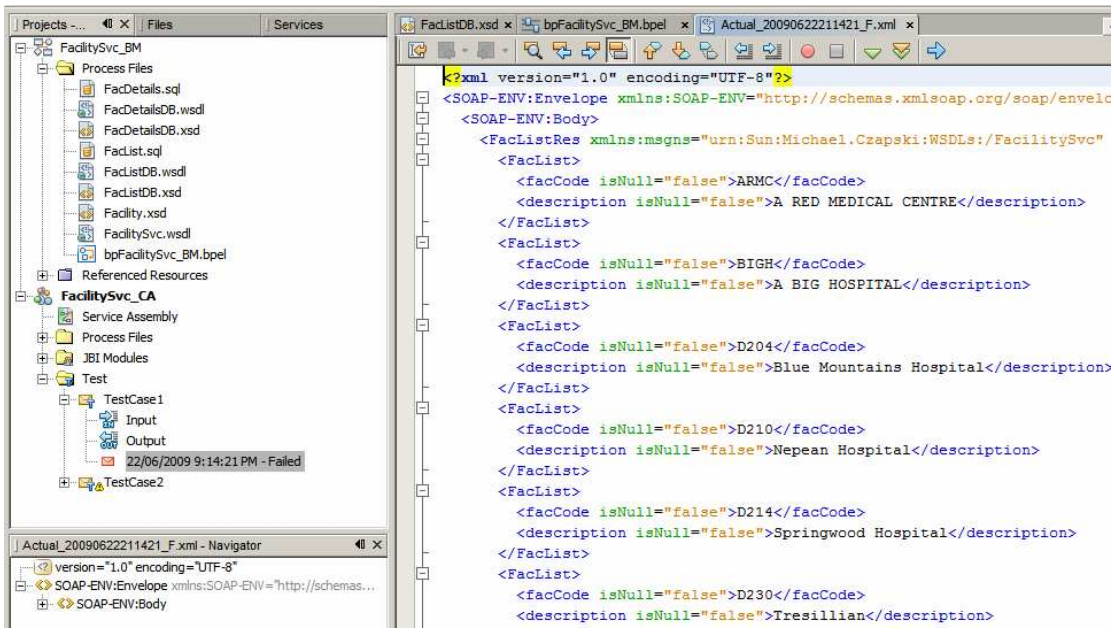
Clcik Yes to create a new output file.



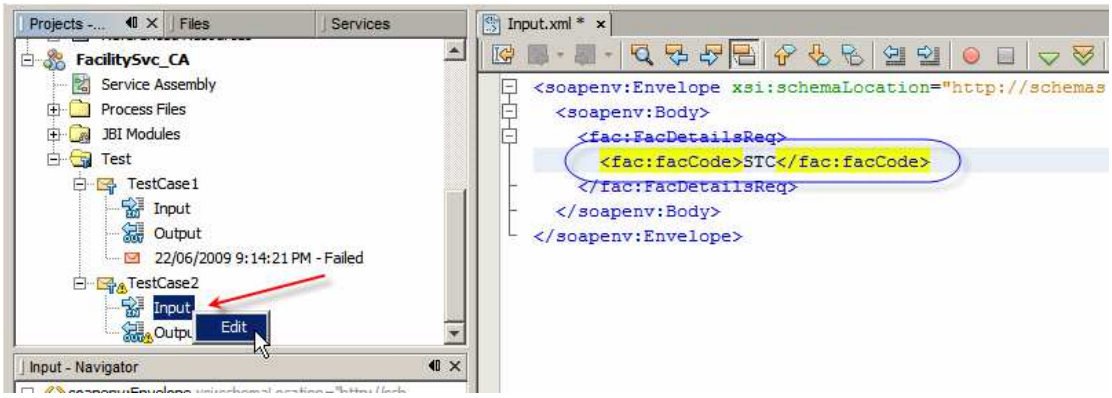
Double-click the output document and inspect it.



The test output shows a XML instance document with a list of faciliuty and description pairs – what we expect to see.

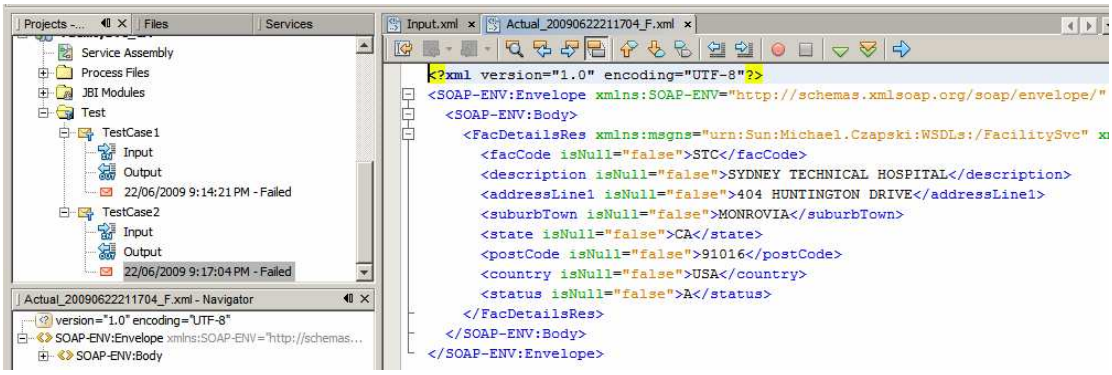


Let's now run TestCase2. Before we do. Lets modifythe input focument to provide a facility code STC.



As before, let's accept the output file and look at it.

Indeed, the XML instance document contains information about facility whose code was STC.

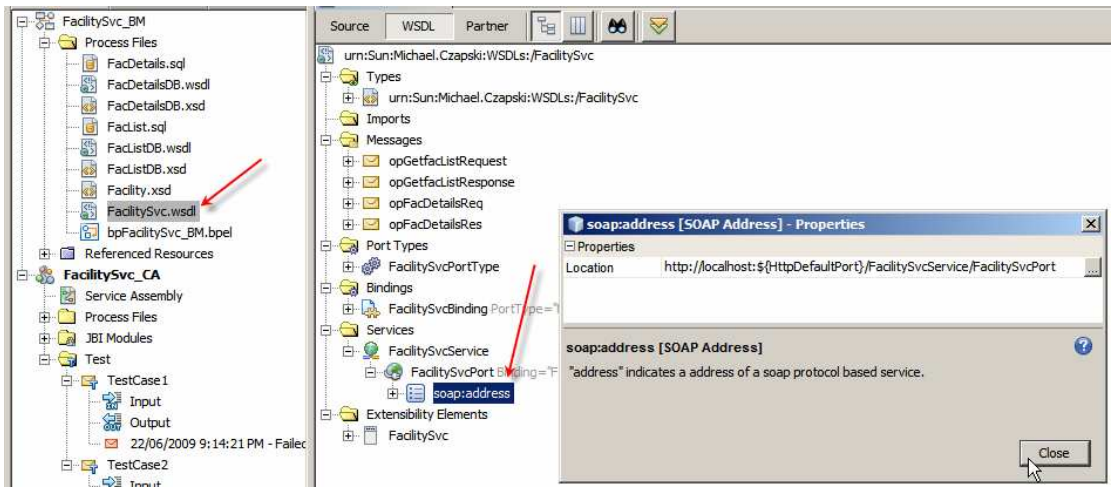


Test with Soap UI

The service is implemented and, for all appearances, works.

To use the service in a composite application or a web application we need to be abgle to get at the service WSDL and at the service at runtime. That information is available if one knows where to look. Let's use Soap UI plugin to emulate a client invoking the service. This will require us to deal with the location of the WSDL and the location of the service.

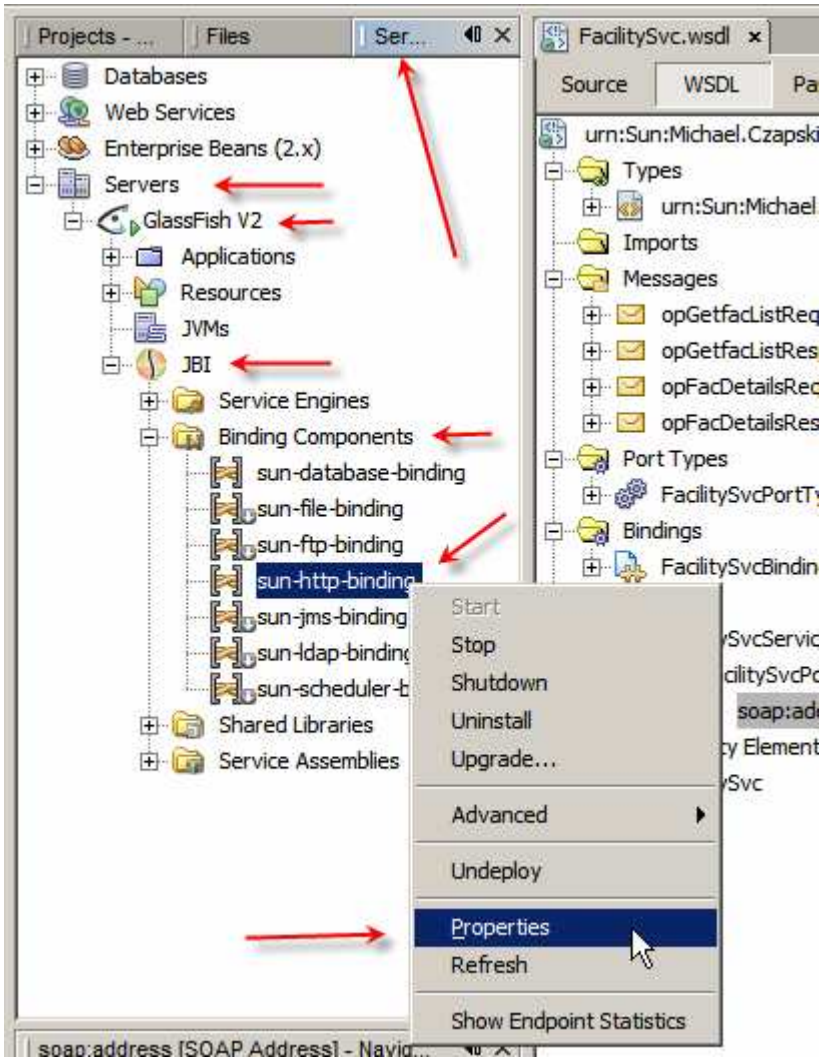
Let's open the facilitySvc.wsdl document and inspect the properties of the soap:address node under the FacilitySvcService node.



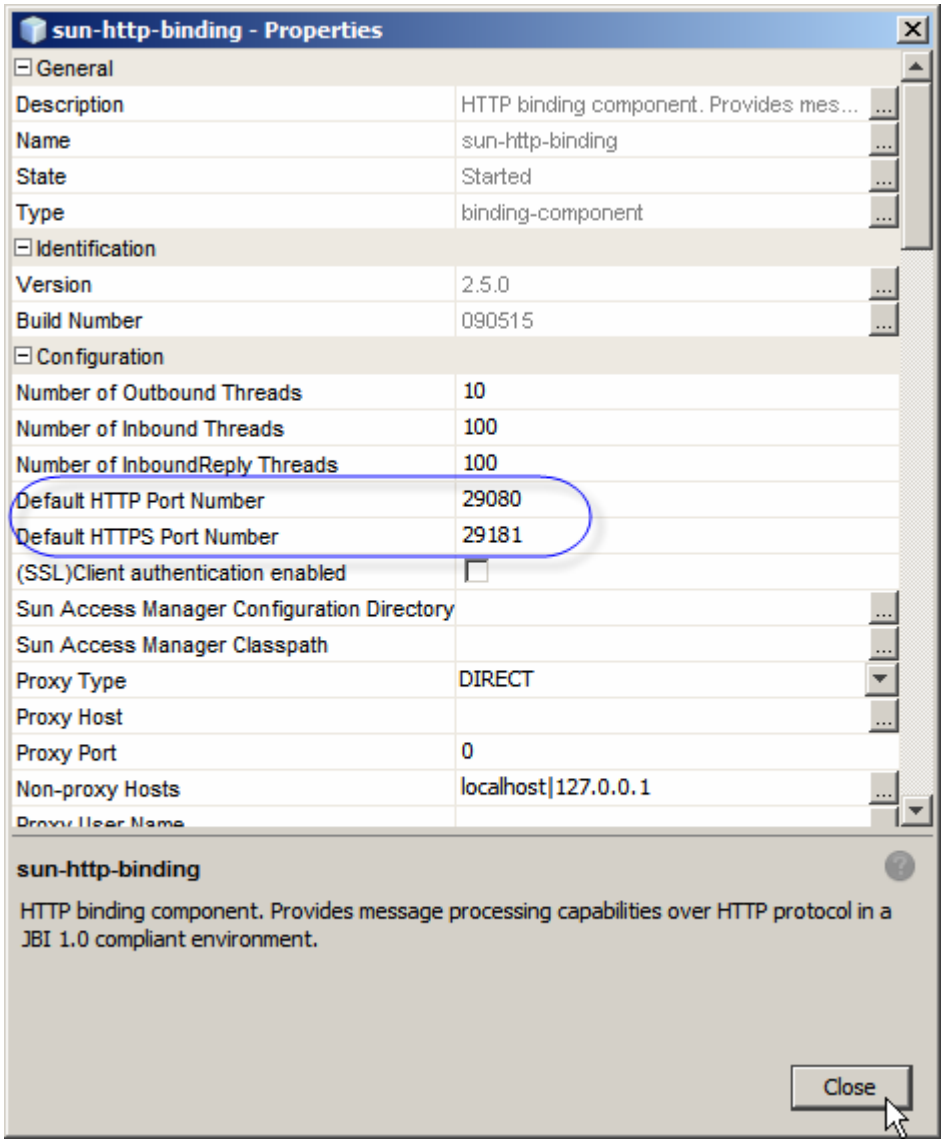
Note the Location property value:

`http://localhost:${HttpDefaultPort}/FacilitySvcService/FacilitySvcPort`

The HttpDefaultPort is the port which SOAP/HTTP BCs use. At CA deployment time this variable gets replaced with the actual port. To find out what this port is let's switch to the Services tab in Netbeans, expand Servers, expand JBI, expand Binding Components, right-click sun-http-binding and choose Properties.



Observe the Default HTTP Port Number property value. For my installation this will be 29080. For a default installation it will be 9080. It can be changed.



So, the final service URL, from the soap:address Location property earlier, will be:

`http://localhost:`**`29080`**`/FacilitySvcService/FacilitySvcPort`

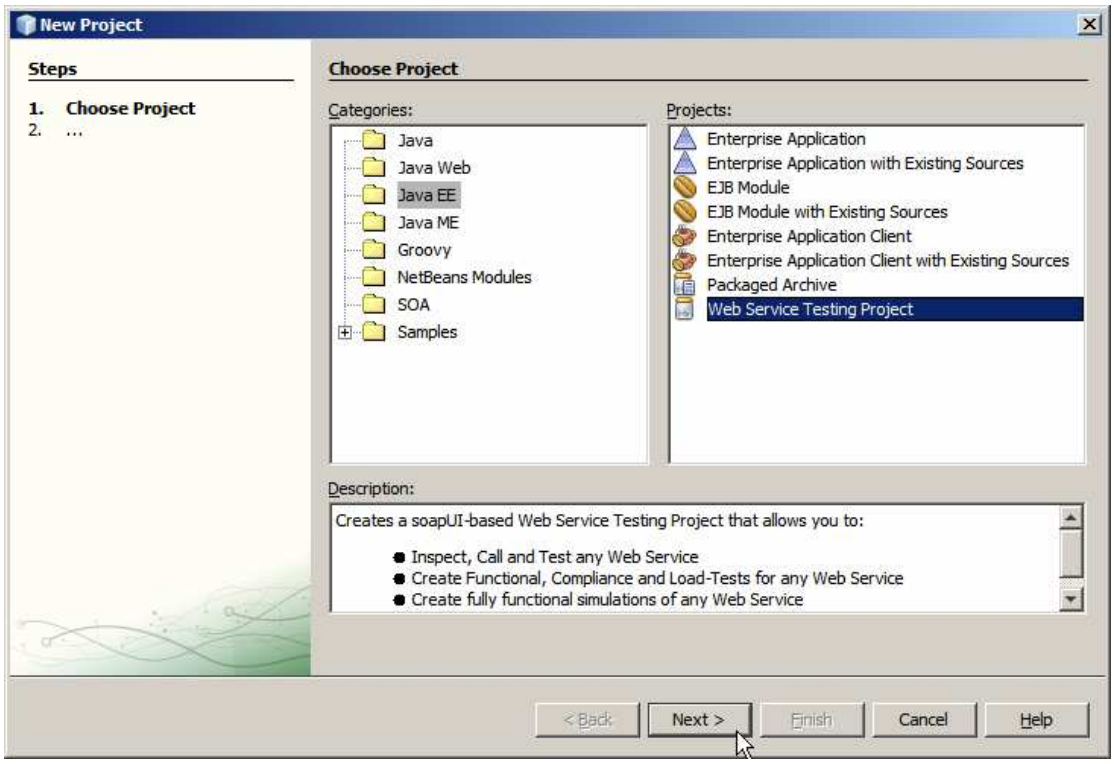
This URL is the service location.

The WSDL for this service can be accessed, using the regular convention, at:

`http://localhost:`**`29080`**`/FacilitySvcService/FacilitySvcPort`**`?WSDL`**

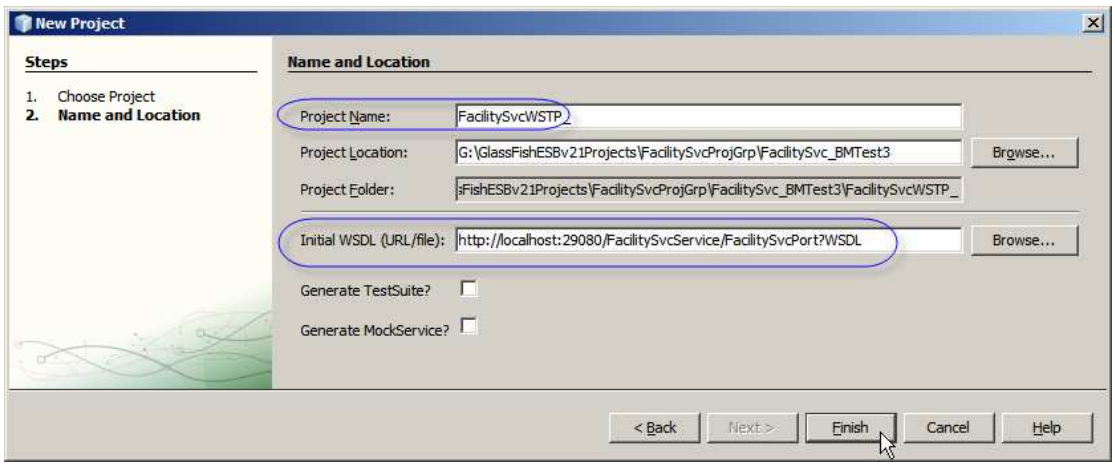
With this knowledge let's create a Web Services testing Project.

Create a New -> Java EE -> Web Services testing Project (this assumes you installed the Soap UI Plugin – if not this project type will not be available).

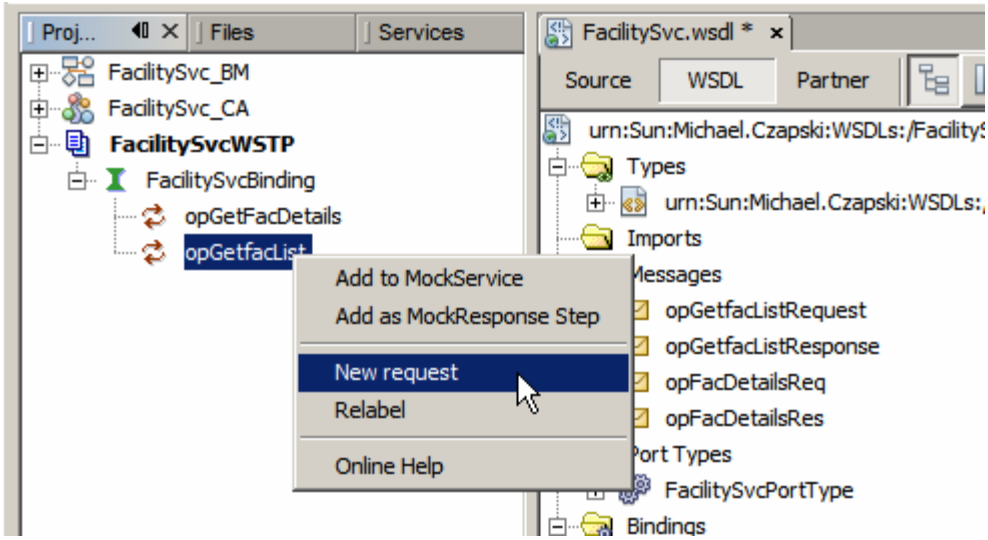


Name the project FacilitySvcWSTP and provide the WSDL WRL which you derived a little while ago :

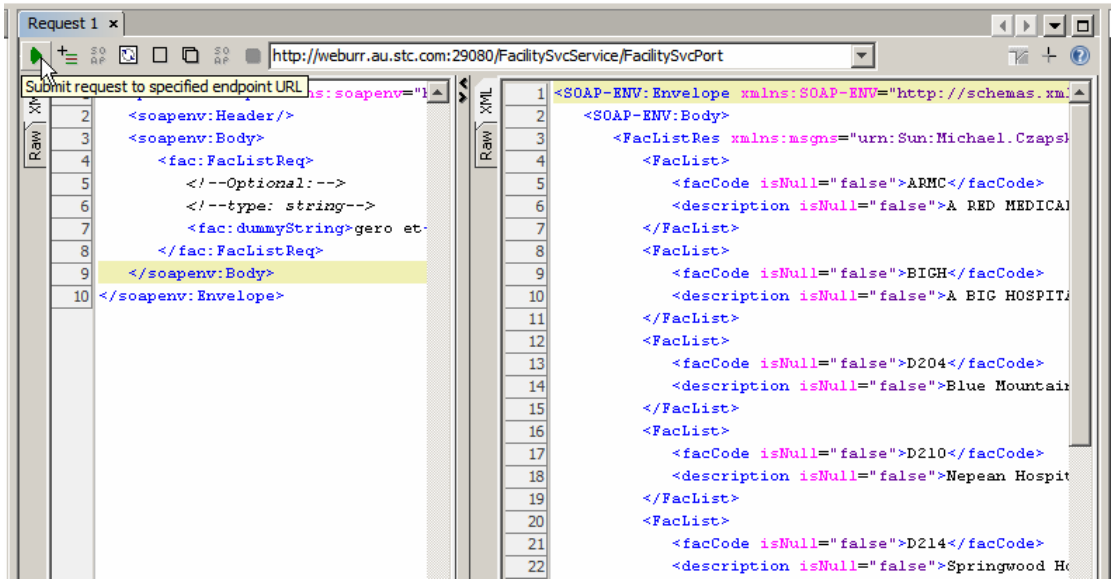
`http://localhost:`**`29080`**`/FacilitySvcService/FacilitySvcPort`**`?WSDL`**



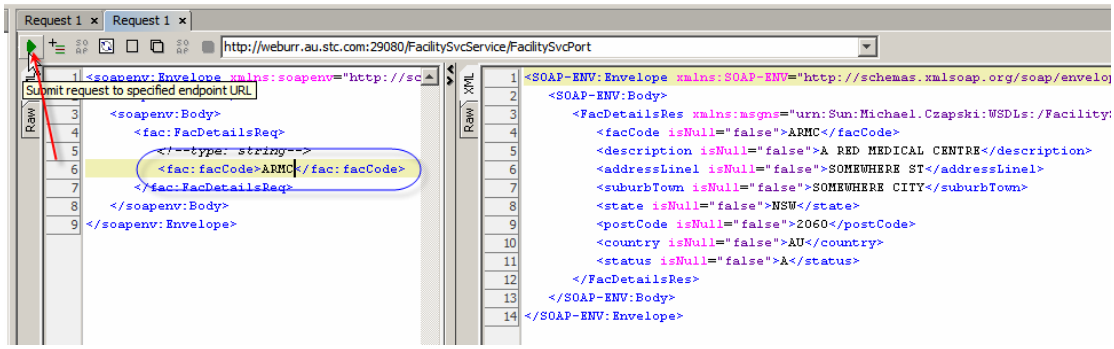
Right-click on the opGetfacList and choose New Request.



Accept default name. Submit the request and observe the result.



Right-click on opGetfacDetails, modify the request to use facility code of “ARMC”, submit the request and observe the result.



This is it. The service has been built, deployed and exercised.

Summary

In this document we created and exercised a multi-operation web service that provided a list of Healthcare Facilities and details of a specific Facility.

We used the GlassFish ESB v 2.1 infrastructure. In particular, the BPEL 2.0 Service Engine was used to orchestrate access to the relational database and to expose the logic as a SOAP over HTTP Web Service. We used the SOAP/HTTP Binding Component to expose the process as a service. We used the NetBeans tooling to create SQL Files containing prepared statements and to create Database BC service interfaces for these statements. We used the built-in JUnit testing facility to test both service operatins and, independently, a Soap UI plugin-provided Web Service testing Project type to test the service.

Building a multi-operation web service that accesses a relational database is fairly straight-forward when using good tooling like that provided by the GalssFish ESB v2.1.

The service developed in this document will be used as a data provider for the Visual Web JSF Web Application and a Visual Web JSP Portlet in subsequent documents. At the end of the process we will have a SOA 1, Presentation Layer, as well as SOA 3, Business Service (this service) and SOA 4, Technical Layer (the Database BC services) artifacts that are the basis for a part of a healthcare SOA solution.