



Pymma consulting

OpenESB training

Design a calculator with OpenESB

ABOUT PYMMA CONSULTING

Pymma Consulting is a technical architect bureau. Pymma provides expertise in the design and implementation of high-performance Java EE and JBI (Open-ESB) architectures, with a high capacity of integration. Pymma was founded in 1999 and is a European company based in London (*headquarters*), with regional offices in Amsterdam and Paris. (contact@pymma.com or www.pymma.com)

Contents

Introduction	4
Create a BPEL project.....	5
Create an XML Schema	6
Create Complex types	6
Create Elements	10
Create a WSDL	12
Add operations.....	16
Add an operation with fault.....	17
Create a BPEL	21
Define BPEL services	22
Build the process.....	23
Pick activity	23
Link onMessage with operation.....	25
Assign activities.....	27
Reply activities	28
Variable mapping with assign activity	31
Special case division.....	33
Add a fault as reply	38
Create Composite Application	43
Add SOAP action	46
Clean and build the project.....	47
Test the calculator	49
Summary	53

Introduction

This document aims to provide the reader with a first non-obvious exercise with OpenESB. This exercise introduces basic concepts like Schema, WSDL, BPEL activities, Fault, Composite application and Test.

Often during training, delegates face two difficulties in parallel, functional requirement and technical development. We tried to avoid it by setting up an exercise where functional requirements are very easy to understand:

- “Do a calculator with the four basic operations: addition subtraction, multiplication and division.

To make the exercise a bit more relevant, we ask the reader to support the case “Division by zero” and return a fault if this case occurs.

The target for this exercise is the OpenESB beginners, so we try to detail the development process as much as we can and consequently the document is longer than usual.

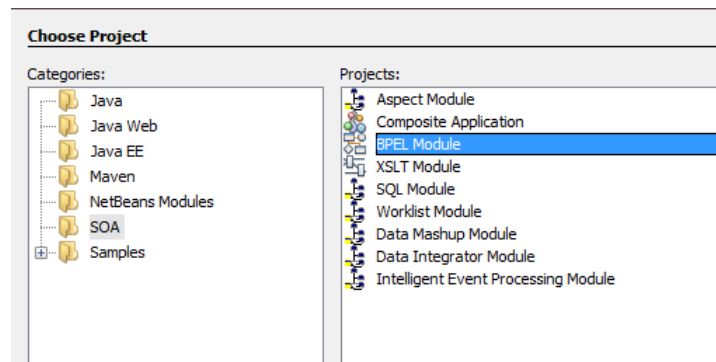
We hope you will enjoy it.

Create a BPEL project

The chapter objective is to create a BPEL project. A project at the design time defines an OpenESB component named Service Unit (SU) at the runtime. It is up to you to define SU granularity and its content. Note that a service unit is not a deployable entity. It must be embedded in a Service Assembly (or CASA) to be deployed. We will create a service assembly later in this exercise.

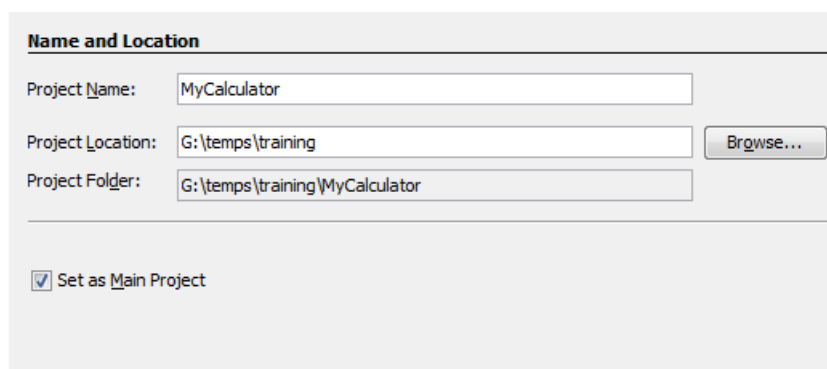
Let's go:

From the main menu select File→Project. The following window appears.



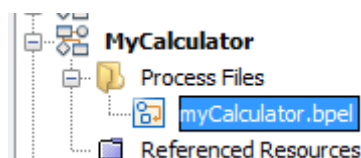
Select SOA category and BPEL project. Click Next.

Enter Project Name “MyCalculator” and Project Location (you can use the location per default).

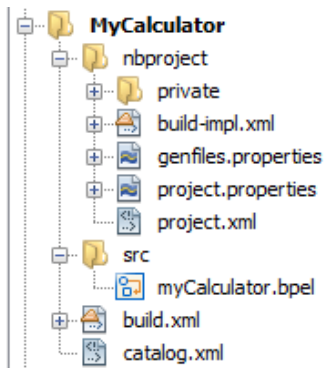


Click Finish

The new project appears in the project hierarchy the left part of the screen.



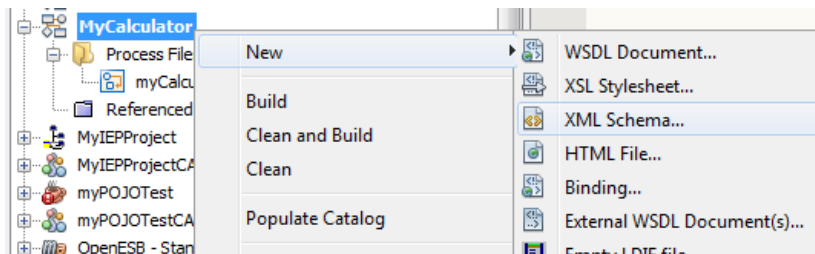
Select File tab and note a more complex project structure. Many files have been added by Netbeans to define a project. Understanding files content is beyond the scope of this exercise.



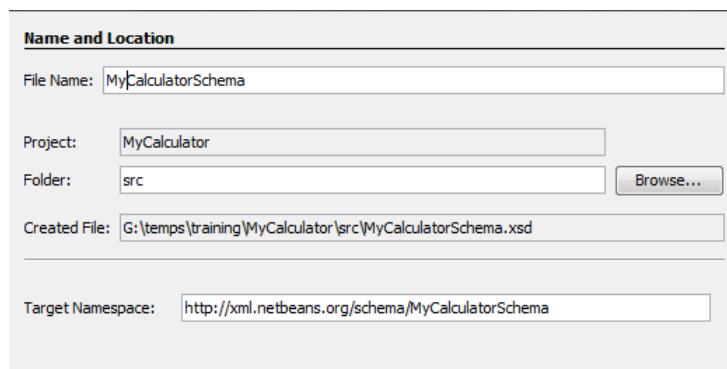
Create an XML Schema

Let's start by creating a Schema which defines message structures exchanged between service consumer and producer.

Go back to the project tab. Click right on MyCalculator Node. Select New → XML Schema



Give the name MyCalculatorSchema



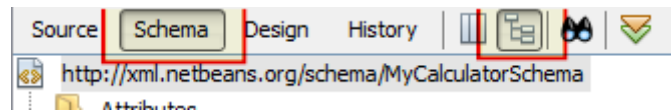
Click button finish

Create Complex types

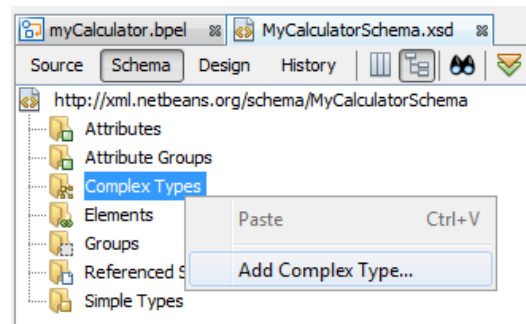
We create 3 complex types for our project: InputComplexType, OutputComplexType and OutputFaultComplexType.

- InputComplexType contains two integers to add, subtract, multiply or divide.
- OutputComplexType contains the operation result
- OutputFaultComplexType contains a string used as error message.

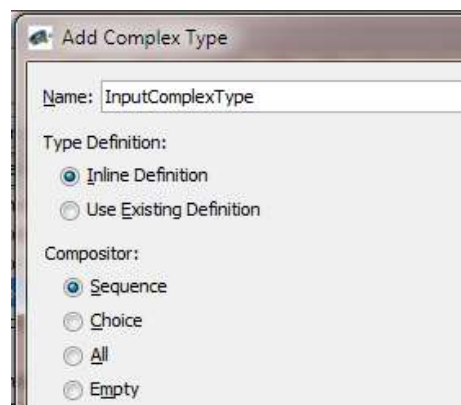
In Schema editor, select Schema view and Tree view



Click right on ComplexType → Add ComplexType.

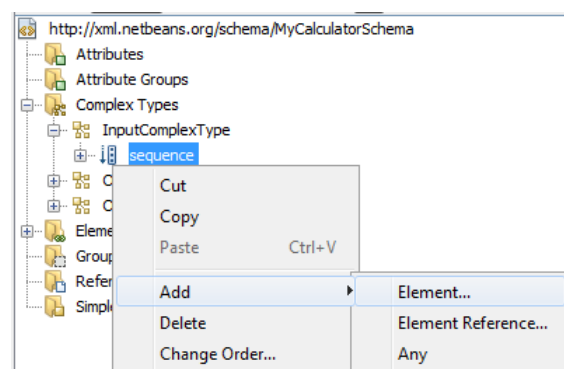


Setup the new complex type as follows: Name=InputComplexType

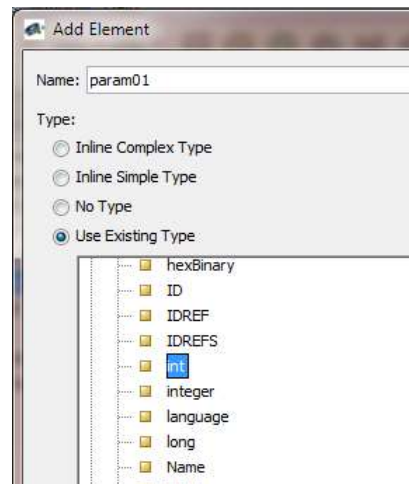
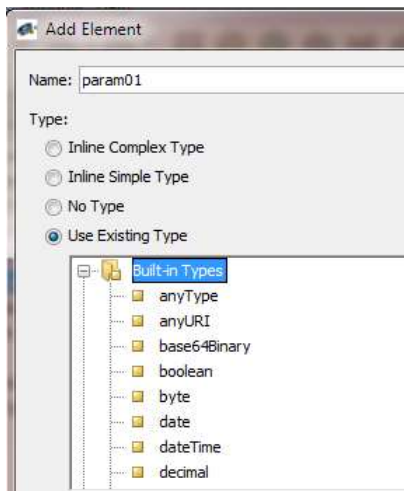


Click OK.

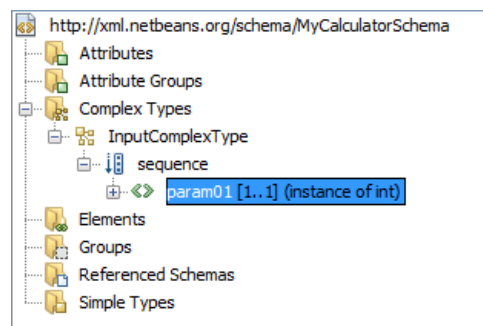
Expand InputComplexType node Select sequence. Click right on sequence → Add Element



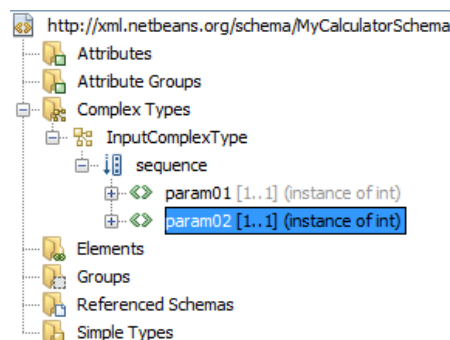
Create the first parameter for the operations. Name it param01, select Use Existing Type → Build-in Type and type "int". Select "int" as the element type.



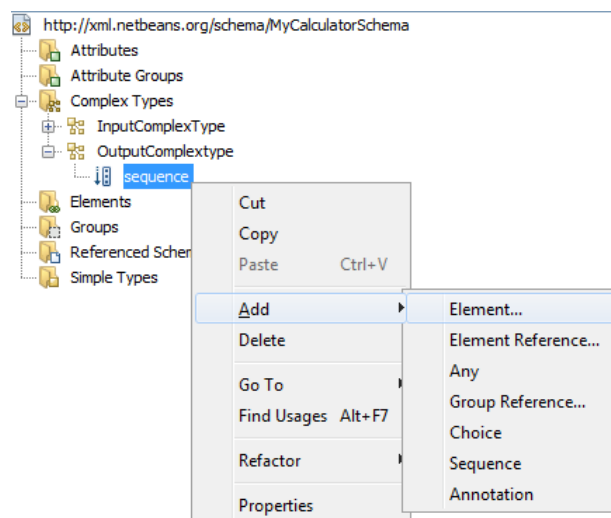
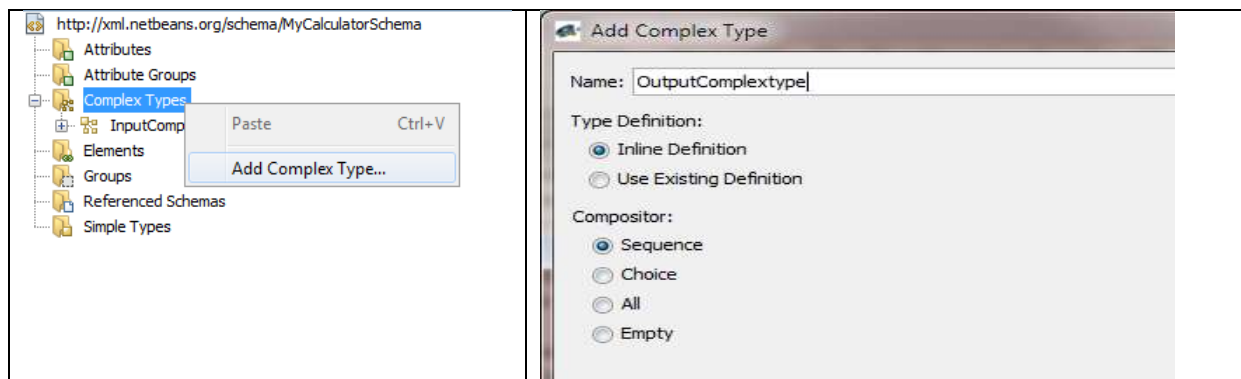
Schema structure is as follows:



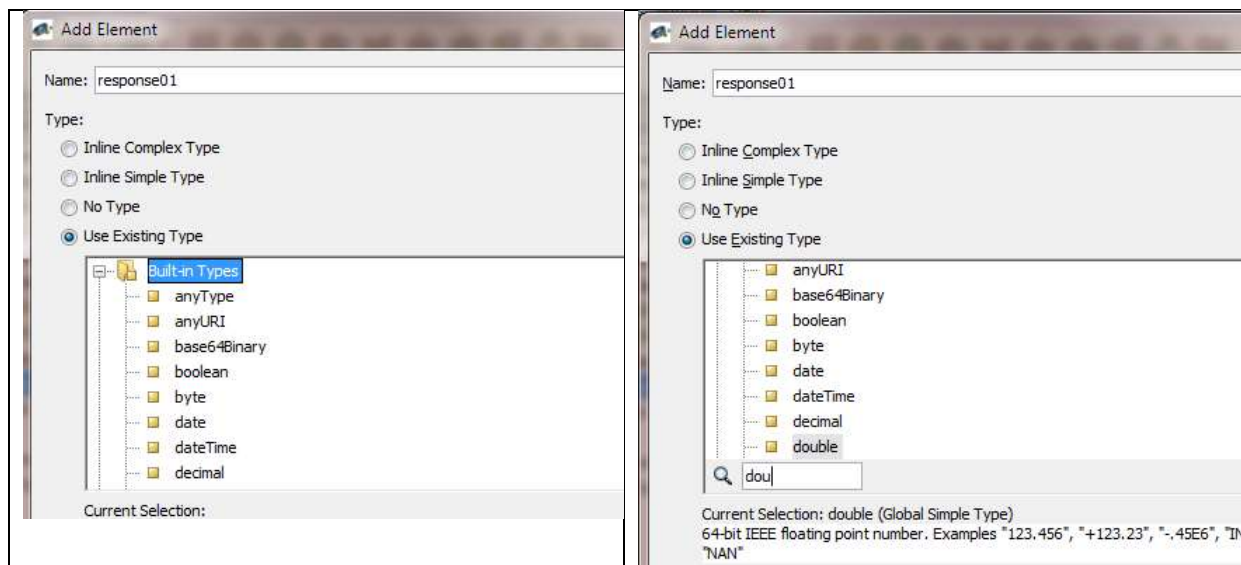
Redo the same task for param02



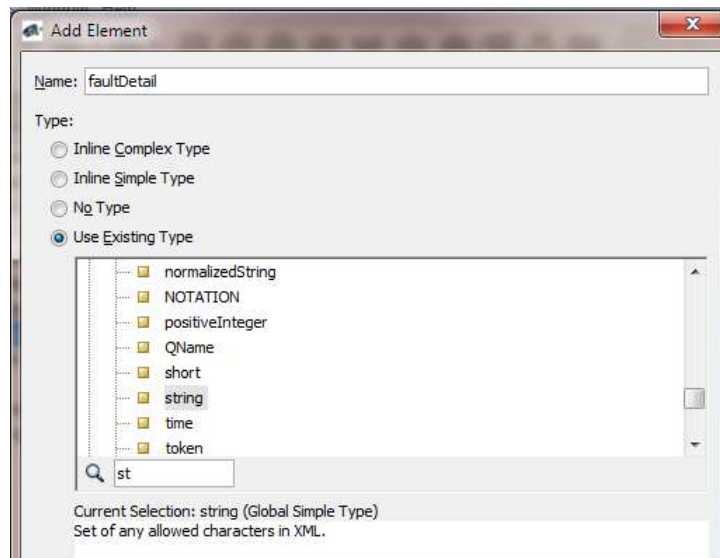
Now let's Create 2 other complex types for the answer: OutputComplexType and OutputFaultComplexType.



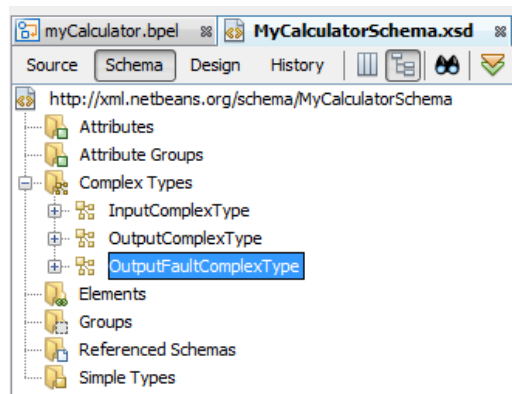
Add the element response01 (double)



Create the third complex type OutputFaultComplexType, add an element named "faultDetail" Type String.

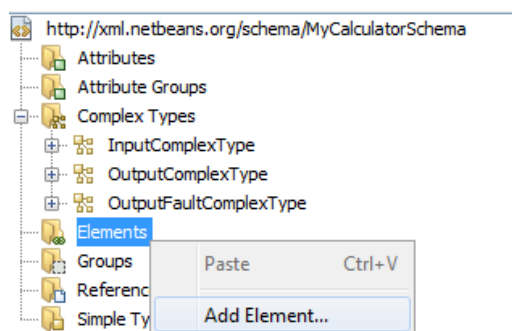


Now, you have three complex types defined.

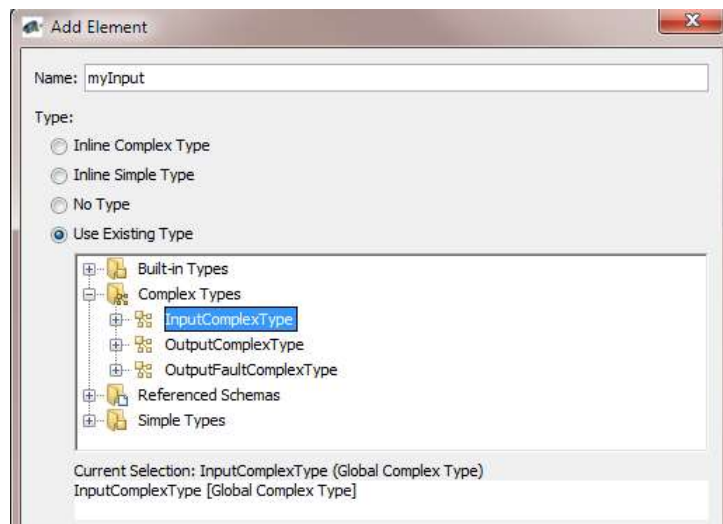


Create Elements

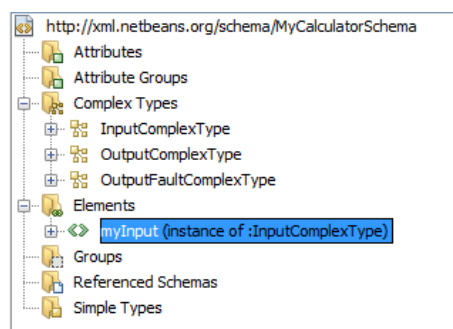
In the schema hierarchy, click right on Element → Create Element



Select "Use Existing Type" → ComplexType → InputComplexType.

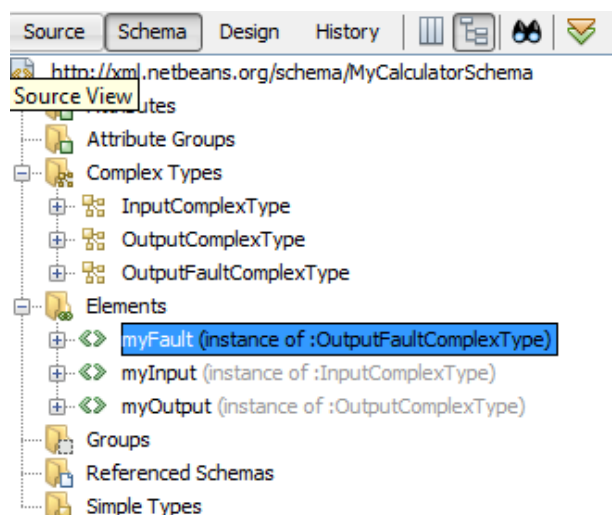


Now, schema hierarchy is as follows:

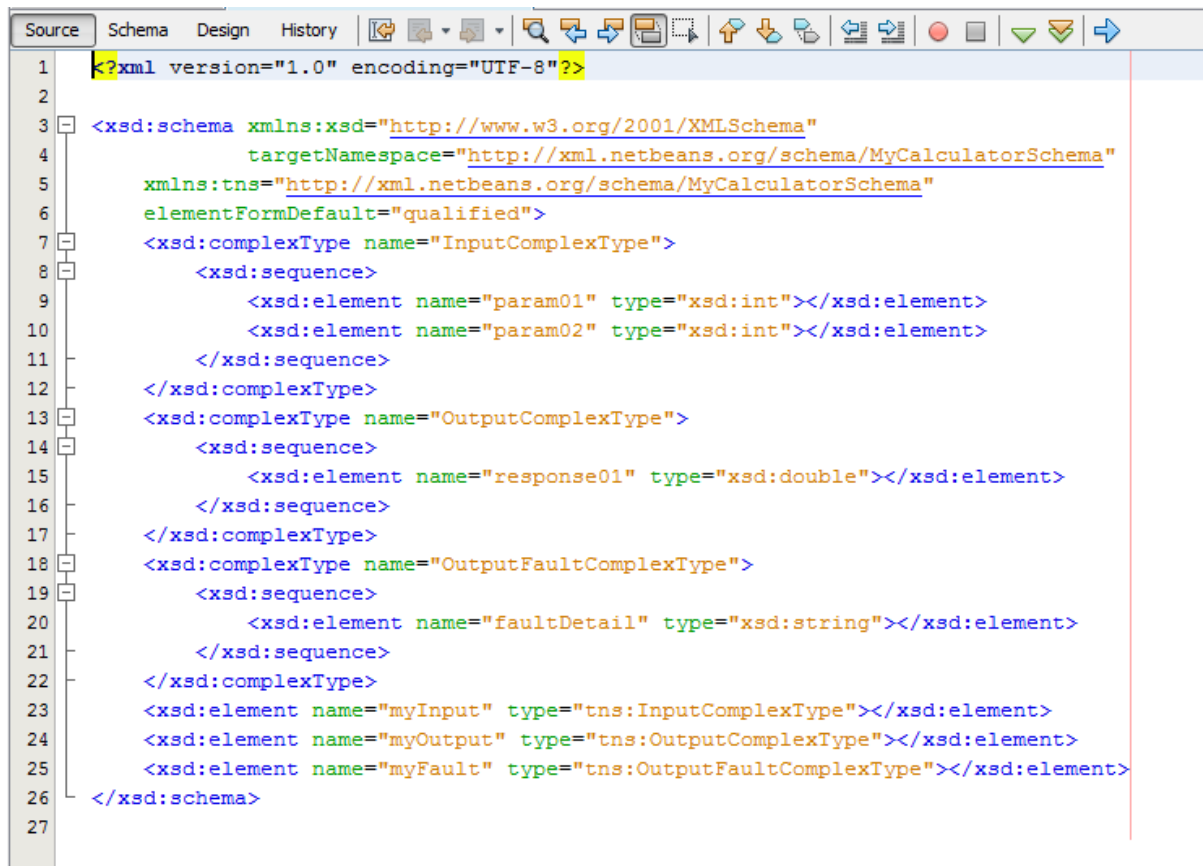


In the same way, create the myOutput and myFault element.

The complete schema is as follows



For your information have a look at the XML code

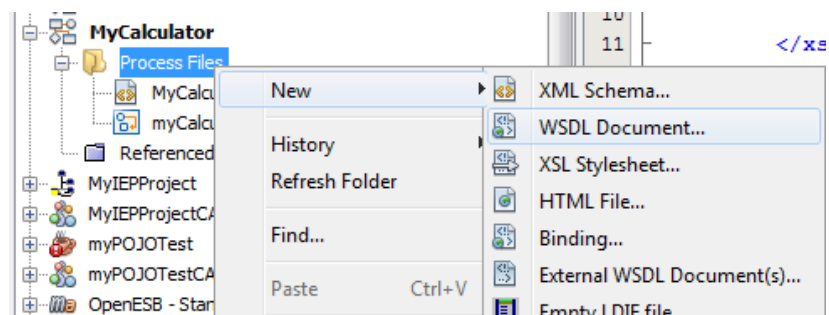


Save your work (Ctrl+Shift+S)

Create a WSDL

The second step in our process is a WSDL document creation. WSDL describes messages and operations proposed by a service. WSDL design is a critical task that impacts the rest of services oriented developments deeply. This chapter has been designed to give a flavour of WSDL design.

In MyCalculator project click right on Process File → New → WSDL Document



Name it "MyCalculator" check if the target namespace is the same as shown in the picture below. Select "Abstract WSDL Document" as WSDL Type.

Name and Location

File Name:

Project:

Folder:

Created File:

Target Namespace:

WSDL Type: ☒ Abstract WSDL Document
☐ Concrete WSDL Document

Click Next

Change Operation name to additionOperation

Abstract Configuration

Port Type Name:

Operation Name:

Operation Type:

Input:

Message Part Name	Element Or Type
part1	xsd:string


Output:

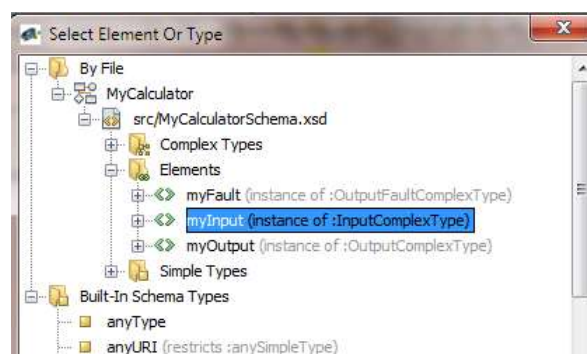
Message Part Name	Element Or Type
part1	xsd:string

Fault:


Message Part Name	Element Or Type
-------------------	-----------------

☒ Generate partnerlinktype automatically.

Input→Element or Type→ click on the button  and set part1 with the type “myInput” element in the popup windows



Click OK

Output→Element or Type→click on the button  and select “myOutput” element in the popup windows

Abstract Configuration

Port Type Name: myCalculatorPortType

Operation Name: additionOperation

Operation Type: Request-Response Operation

Input:

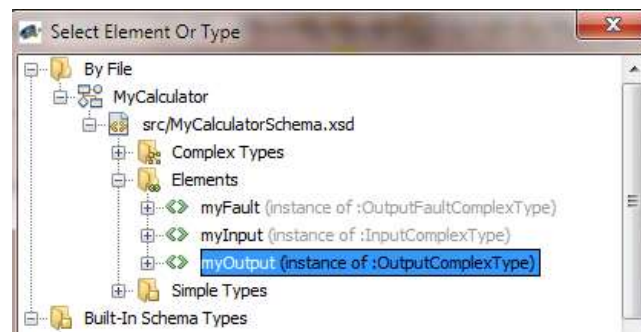
Message Part Name	Element Or Type
part1	ns:myInput

Add Remove

Output:

Message Part Name	Element Or Type
part1	xsd:string

Add Remove



Click Ok

If the abstract windows if as follows:

Abstract Configuration

Port Type Name: myCalculatorPortType

Operation Name: additionOperation

Operation Type: Request-Response Operation

Input:

Message Part Name	Element Or Type
part1	ns:myInput

Add Remove

Output:

Message Part Name	Element Or Type
part1	ns:myOutput

Add Remove

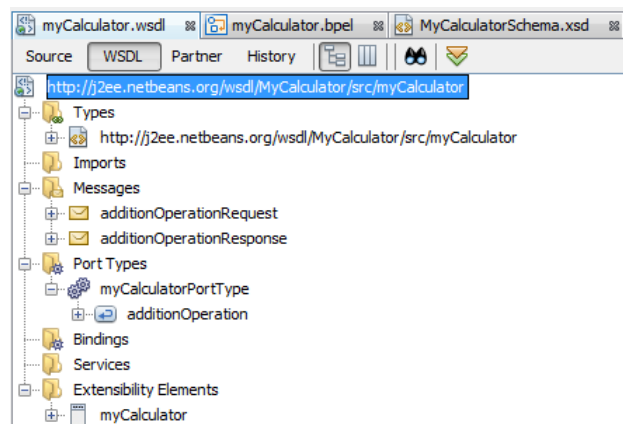
Fault:

Message Part Name	Element Or Type
-------------------	-----------------

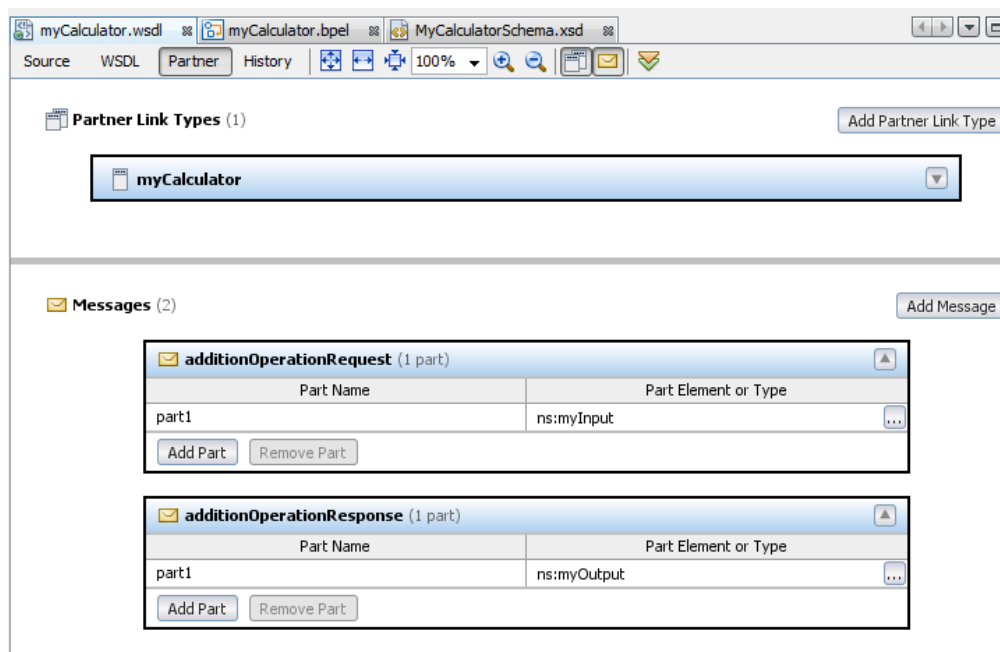
Add Remove

☒ Generate partnerlinktype automatically.

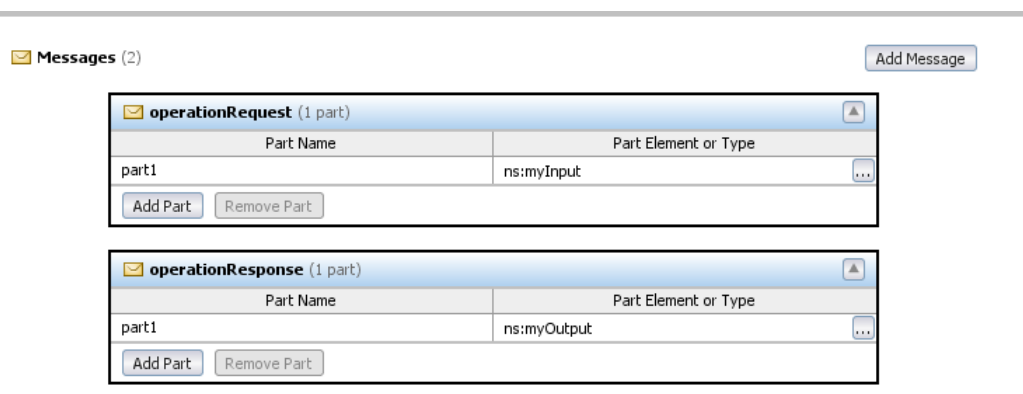
Click OK. WSDL hierarchy is as follows:



To set up the additional operations in the WSDL it is easier to use the Partner view

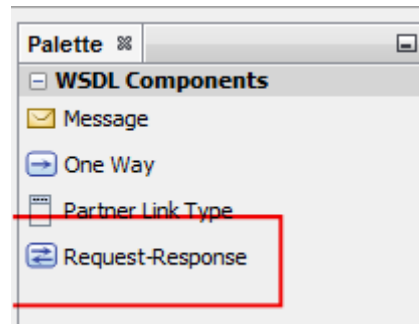


Since we reuse messages for the four operations, let's rename message name to "operationRequest" and "operationResponse". Double click on message name and change the name

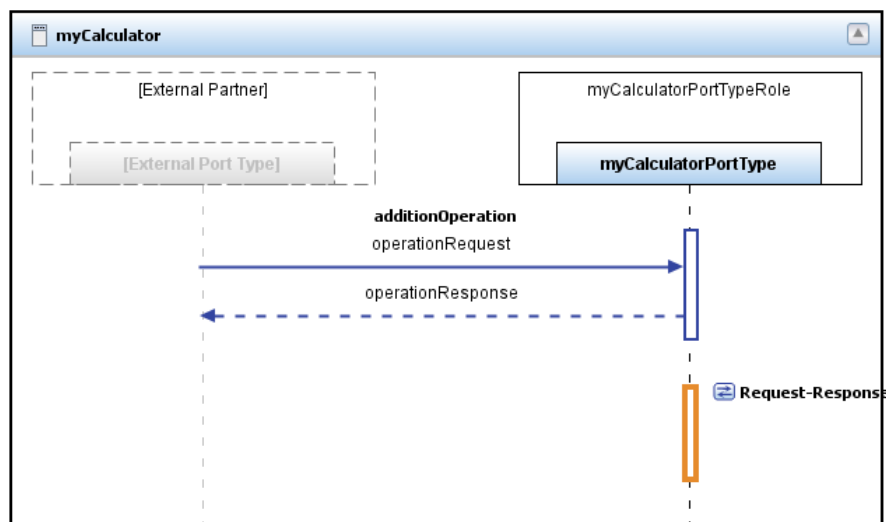


Add operations

On the right side of WSDL editor, there is palette with four icons

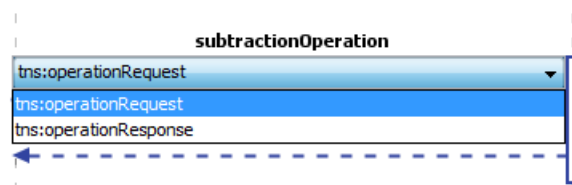


Select "Request-Response icon, drag it and drop it on myCalculatorPortType line as shown in the picture below:

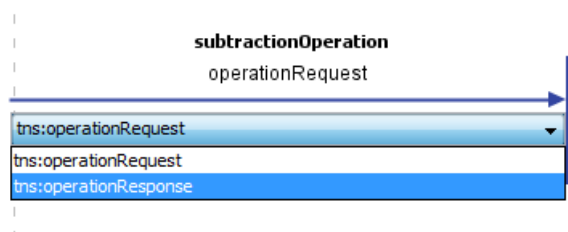


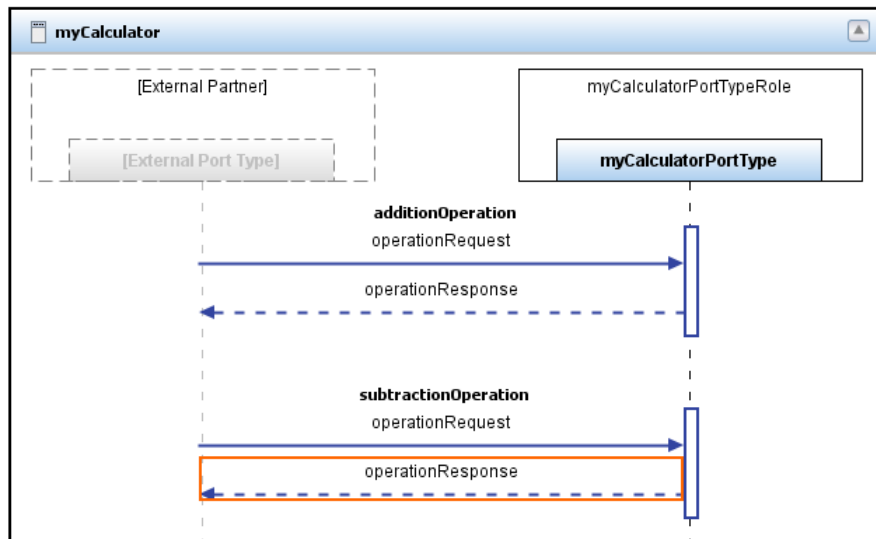
Set the operation name "subtractionOperation".

Above the inbound arrow, double click on <No Message Selected> and select "operationRequest"

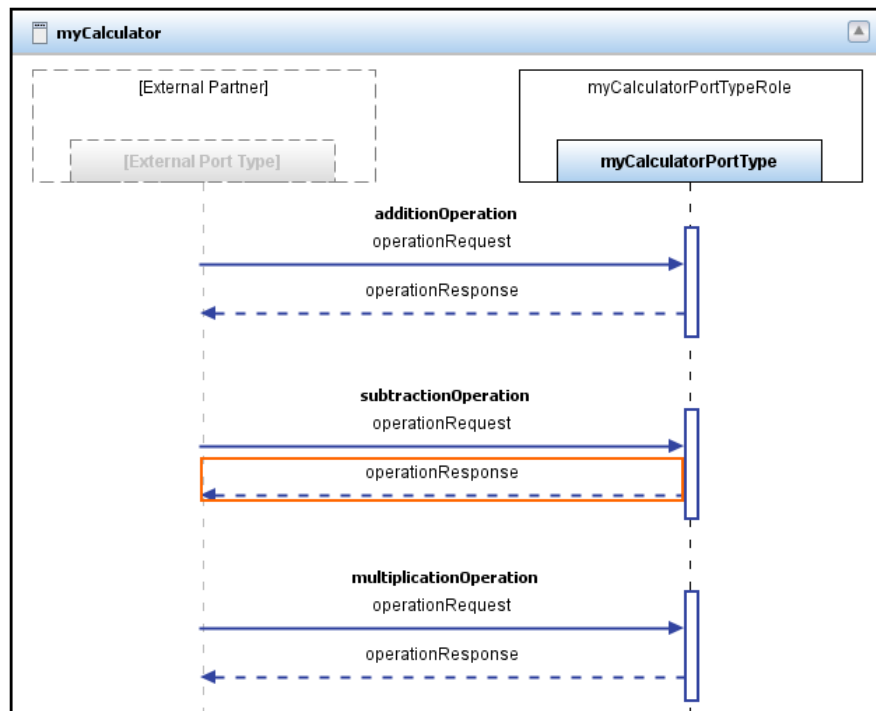


Above the outbound arrow, double click on <No Message Selected> and select "operationResponse"





Redo the same process to create “multiplicationOperation” and its messages



Add an operation with fault

The division operation is a bit different from the other operations since it supports a fault as a response. A fault is returned when the second parameter (Denominator) equals zero. In the WSDL, this point must be specified.

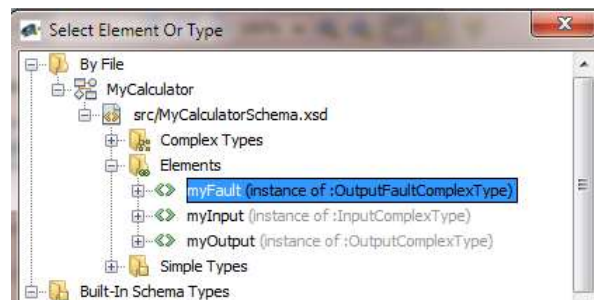
Create a new message and name it “operationFault”

✉ **operationFault** (1 part)

Part Name	Part Element or Type
part1	<Undefined>

Add Part Remove Part

Double click on <Undefined> and select myFault as element



✉ Messages (3)

Add Message

✉ **operationRequest** (1 part)

Part Name	Part Element or Type
part1	ns:myInput

Add Part Remove Part

✉ **operationResponse** (1 part)

Part Name	Part Element or Type
part1	ns:myOutput

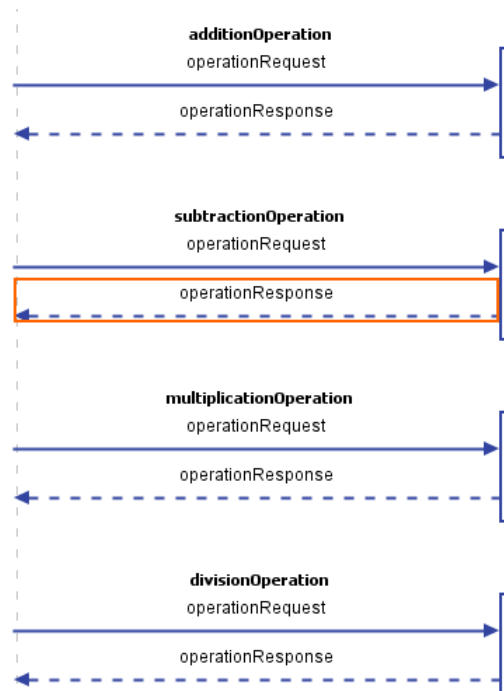
Add Part Remove Part

✉ **operationFault** (1 part)

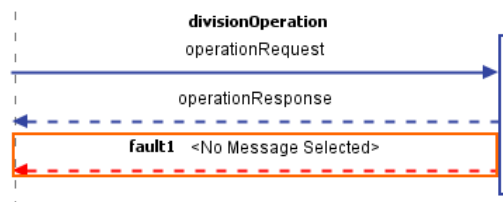
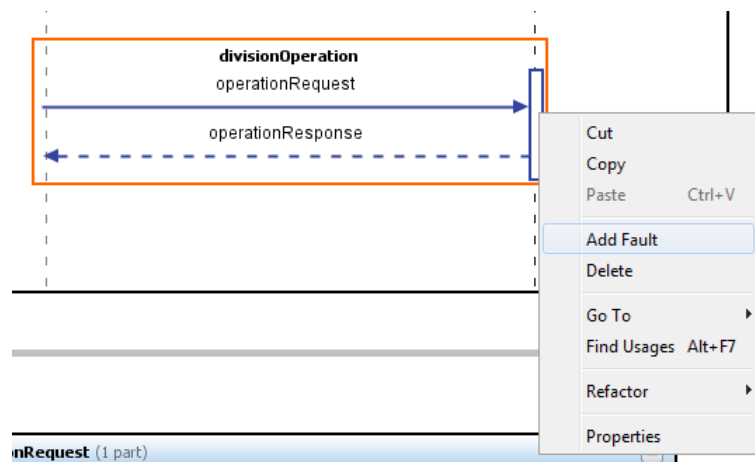
Part Name	Part Element or Type
part1	ns:myFault

Add Part Remove Part

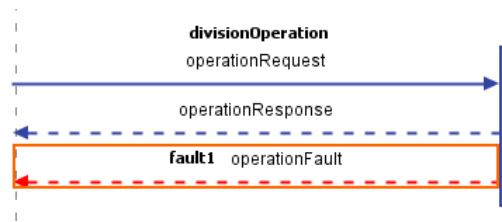
Create a new operation and name it divisionOperation.



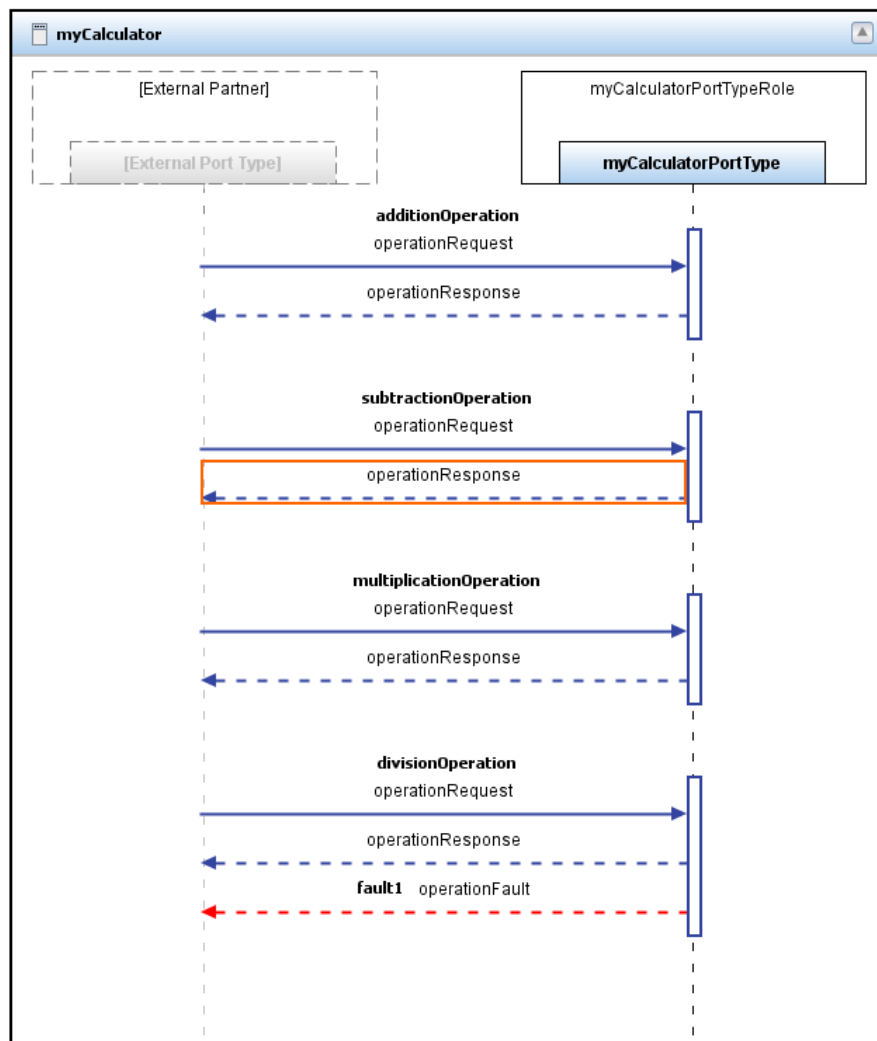
Click right on divisionOperation→Add Fault



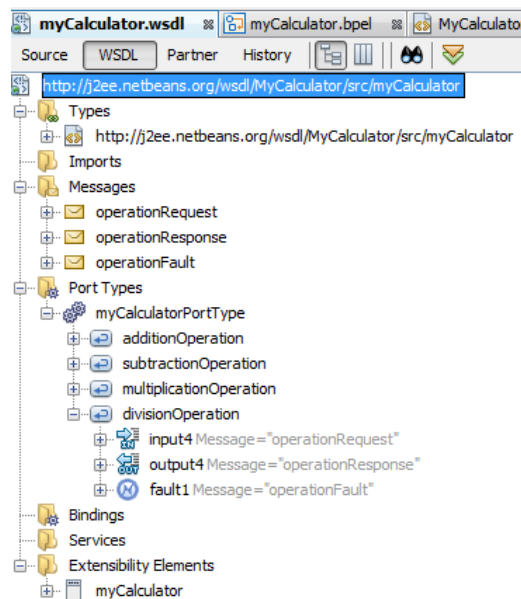
Double click on <No Message Selected> and select operationFault message



The complete WSDL is as follows:



Back to the WSDL view, you can see the operations and messages in the WSDL hierarchy.

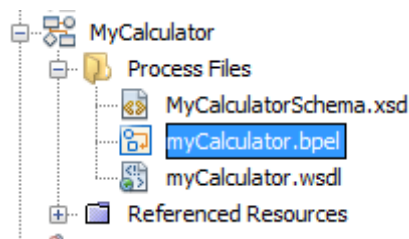


Save your work (Ctrl-Shift-S) and close Schema and WSDL editor

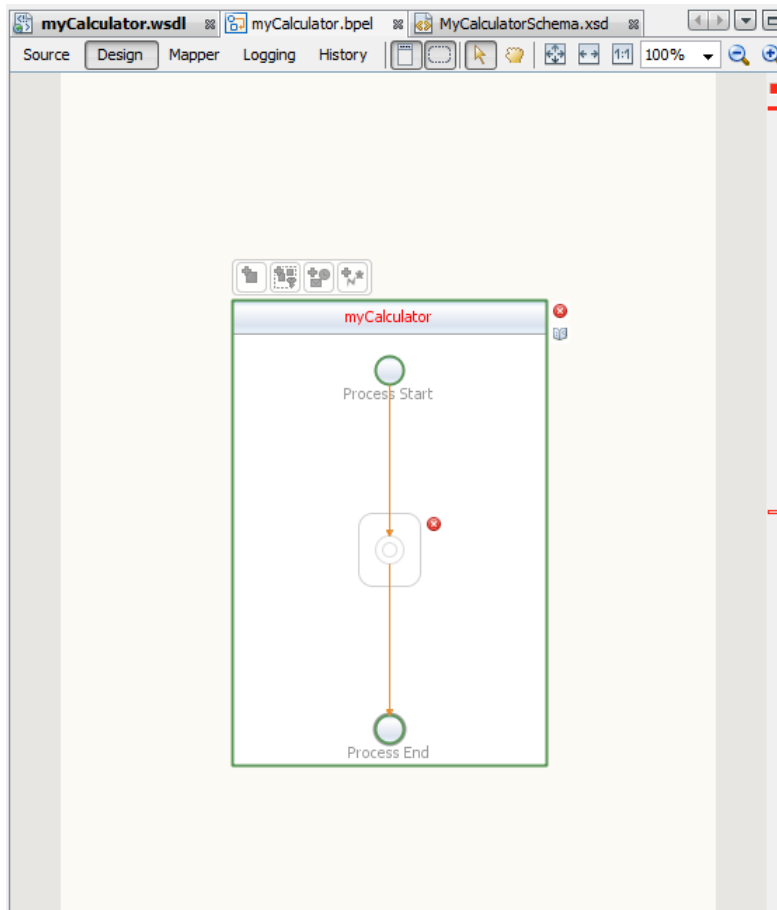
Create a BPEL

The third step in OpenESB development is BPEL design. BPEL represents the business process to run. When Netbeans creates a BPEL project by default, it creates a BPEL document associated with the project.

Open My Calculator.bpel

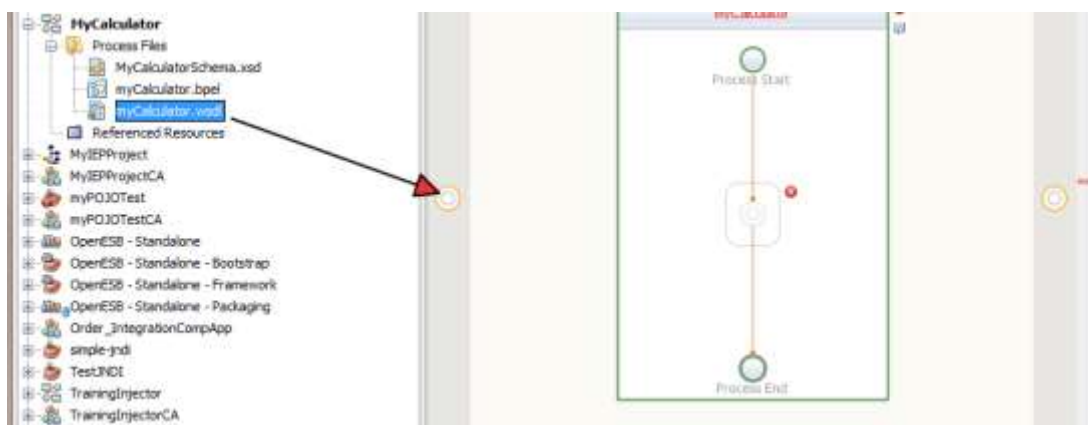


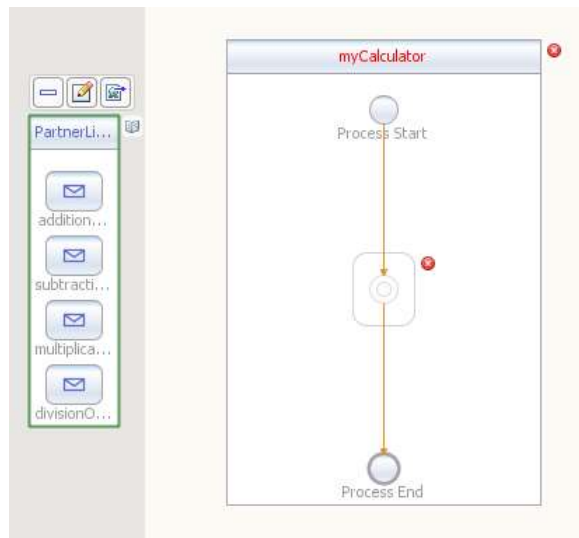
BPEL editor is made up of three lanes. The large one in the middle is used to define the process. The thin grey lane on the left defines the services proposed by the BPEL process. The right grey lane defines the services invoked by the BPEL.



Define BPEL services

BPEL's services propose four operations addition, subtraction, multiplication and division. These operations are detailed in the WSDL document you created above. To link the WSDL and the BPEL, drag myCalculator.wSDL and drop it on the orange circle that appears when you move the pointer on left grey lane





The four operations defined in the WSDL appear as BPEL inputs

Build the process

Building a process is setting up a chain of BPEL Activity that looks like an algorithm. In this paragraph, we create this chain of activity.

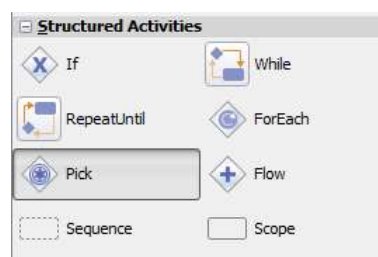
On the right side of BPEL Editor, there is a pallet with many icons representing BPEL activities.

Click right on the palette and select “Show Big Icons.”

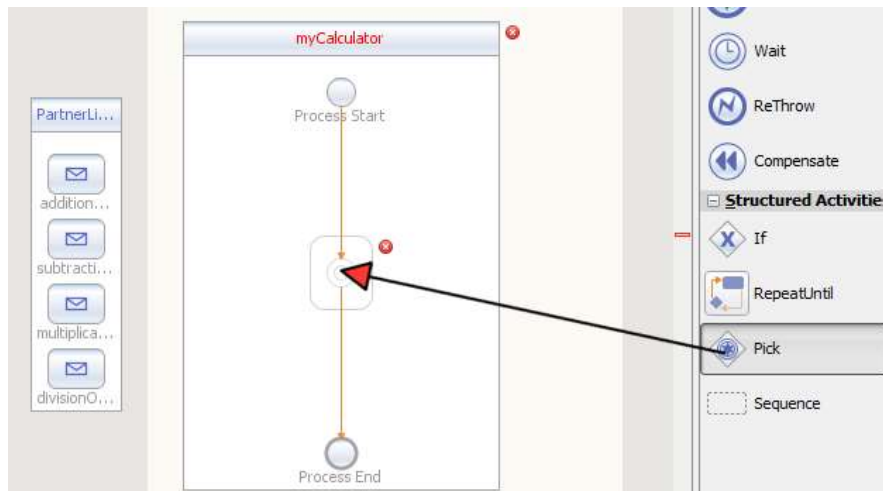
Ok now let’s start:

Pick activity

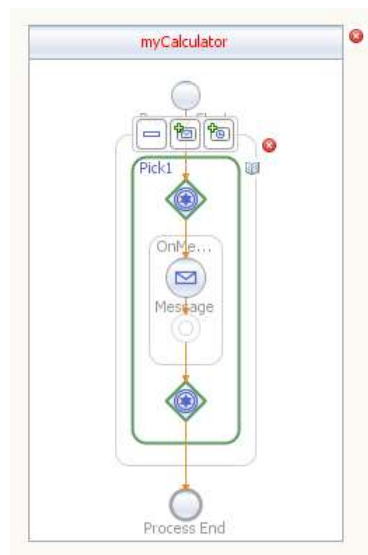
Select Pick activity



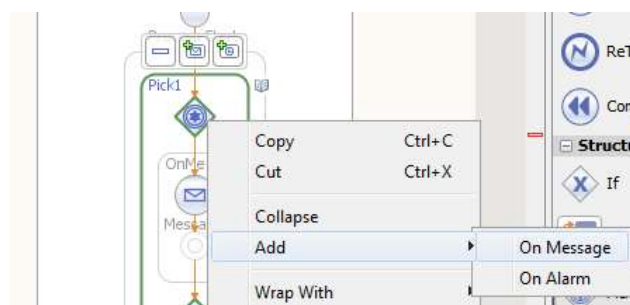
Drag and drop the icon as shown in the picture below:

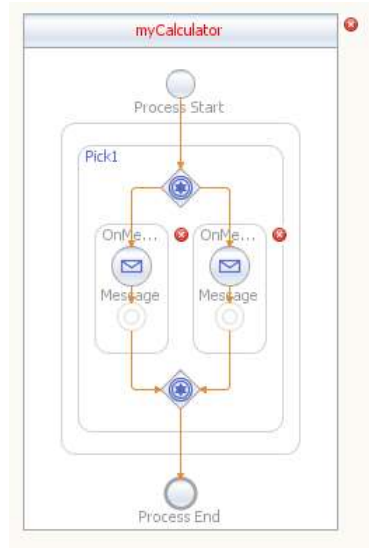


After dropping the icon, BPEL process is as follows:

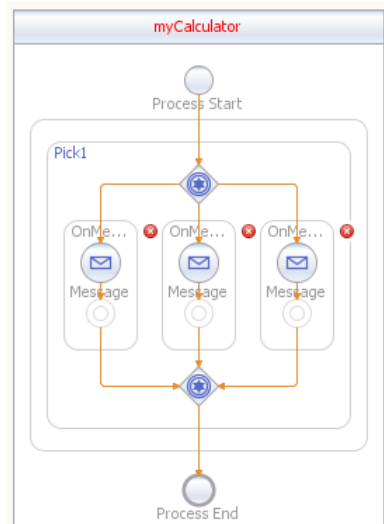


Click right on Pick Add → onMessage





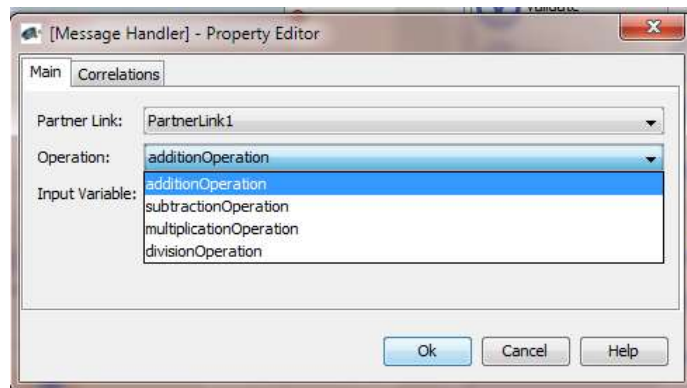
Add a new onMessage; each onMessage corresponds to an operation. Just create three onMessage. The last one for the division will be added later.



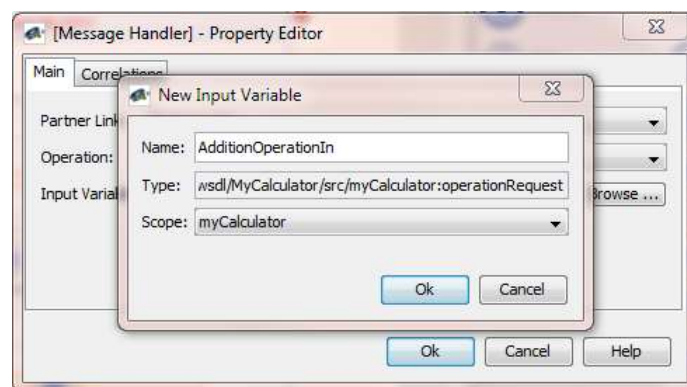
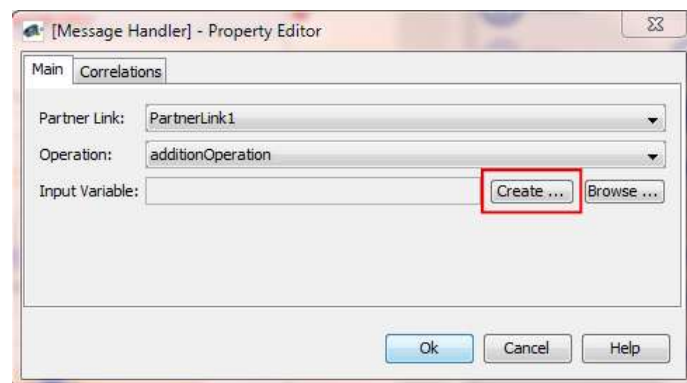
Link onMessage with operation

Double click on the first OnMessage icon to open Property Editor

Select additionOperation as operation

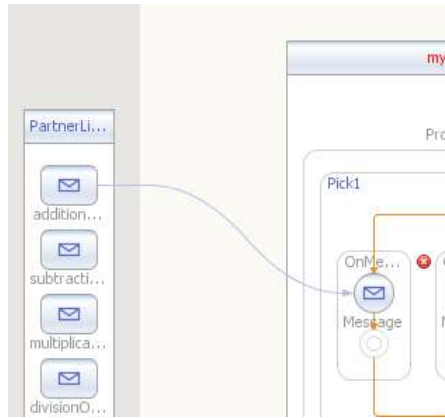


Click Create to create the Input variable.



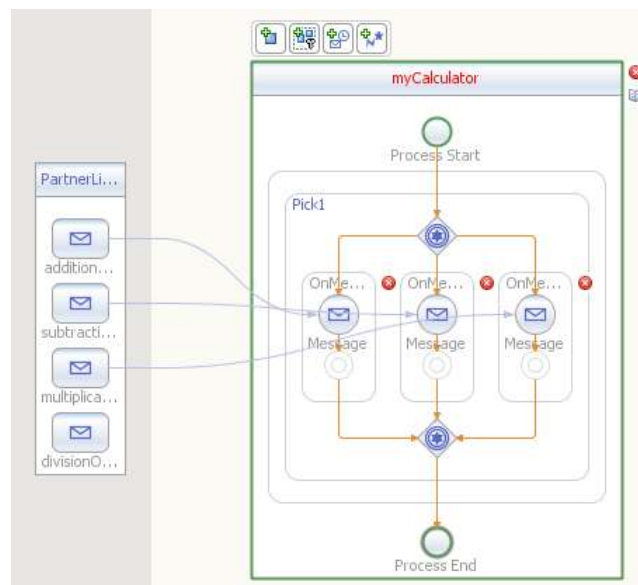
Keep the default values proposed by Netbeans. Click OK to create the variable

Click Ok to set the Message properties.



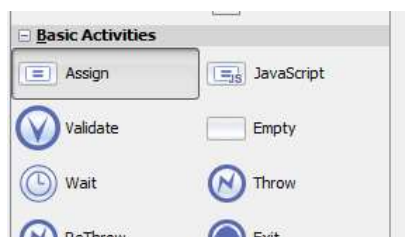
A link has been created between addition and the first onMessage activity.

Do the same thing for subtraction and multiplication.

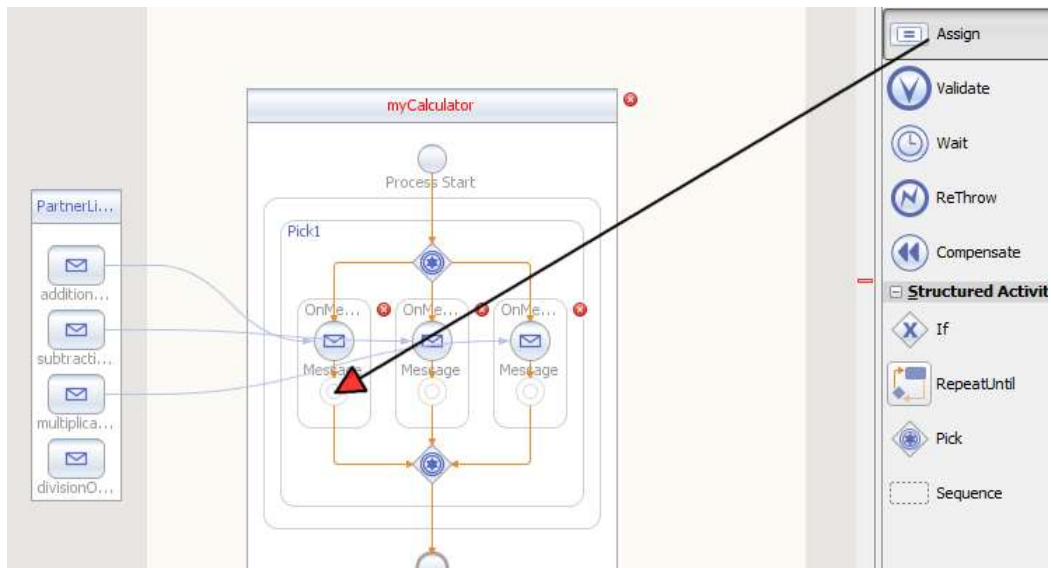


Assign activities

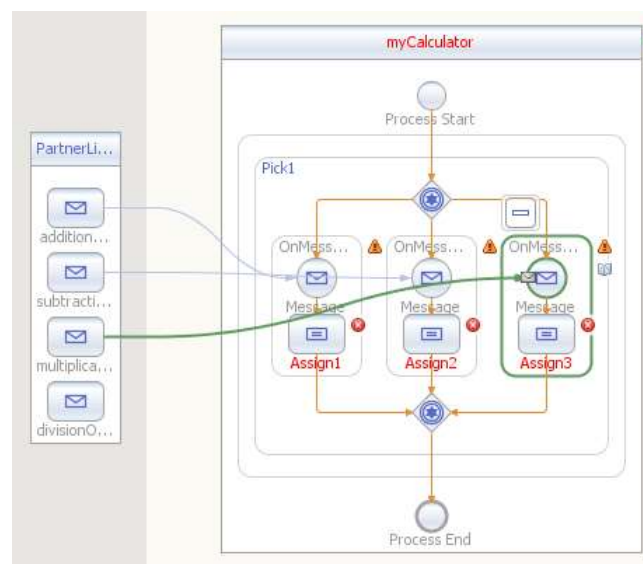
Assign activities support the arithmetic operations. They define the mapping between input and output variables.



Drag Assign activity as shown in the picture below:

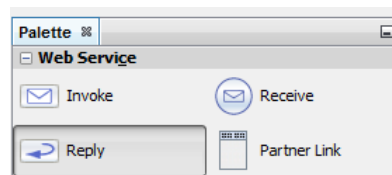


Add Assign activities for all Pick branches.

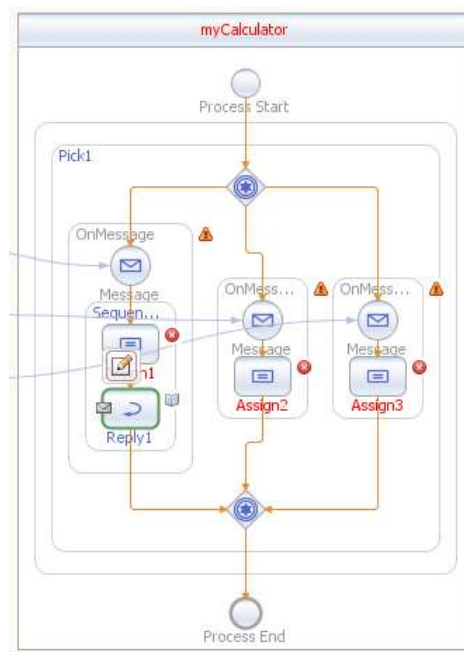
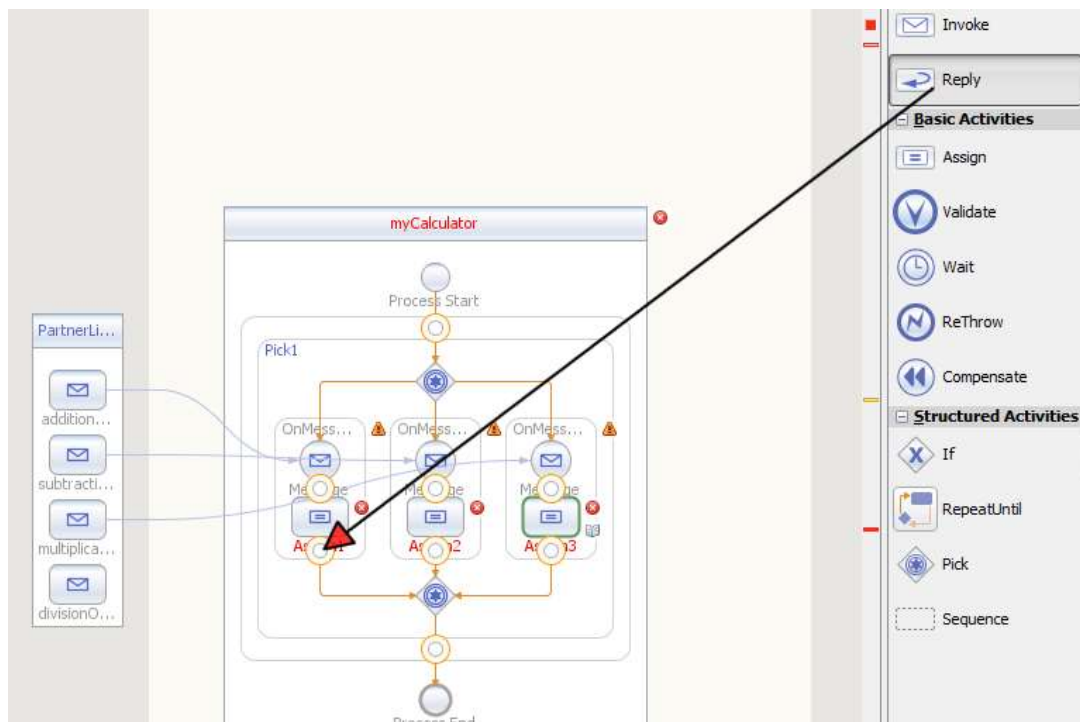


Reply activities

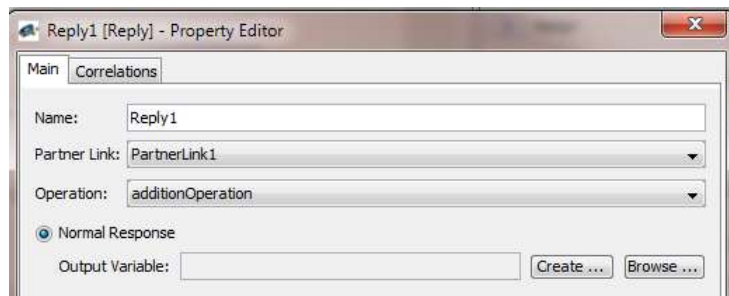
Reply is used to return a response to the service consumer. It can equally return a normal message or a fault.



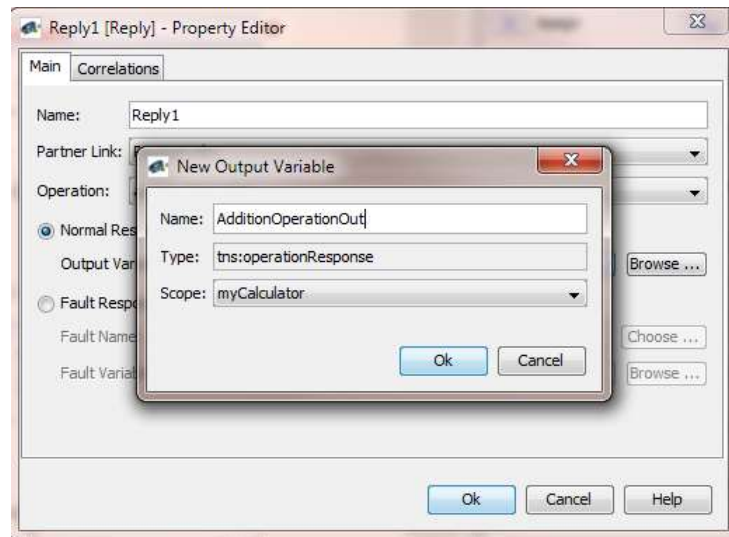
Drag and drop Reply activity as explained below:



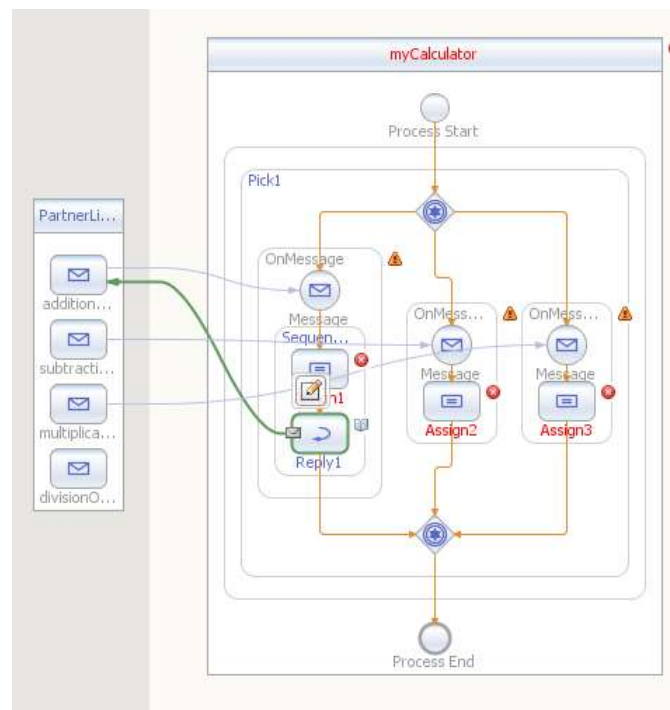
Double click on Reply1 and set the Reply properties as follows:



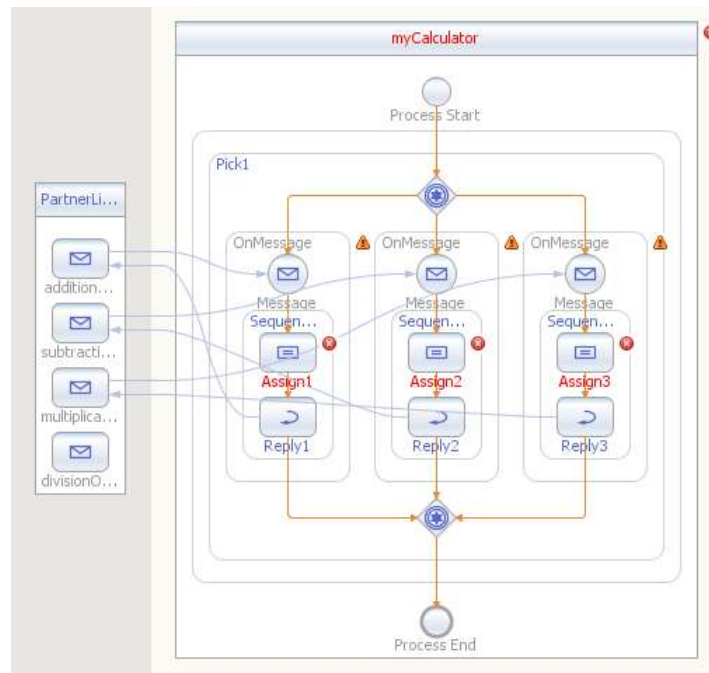
Click Create to Create an output variable. Keep value by default proposed by Netbeans



Click OK to create the output variable and Click OK to set up the properties.



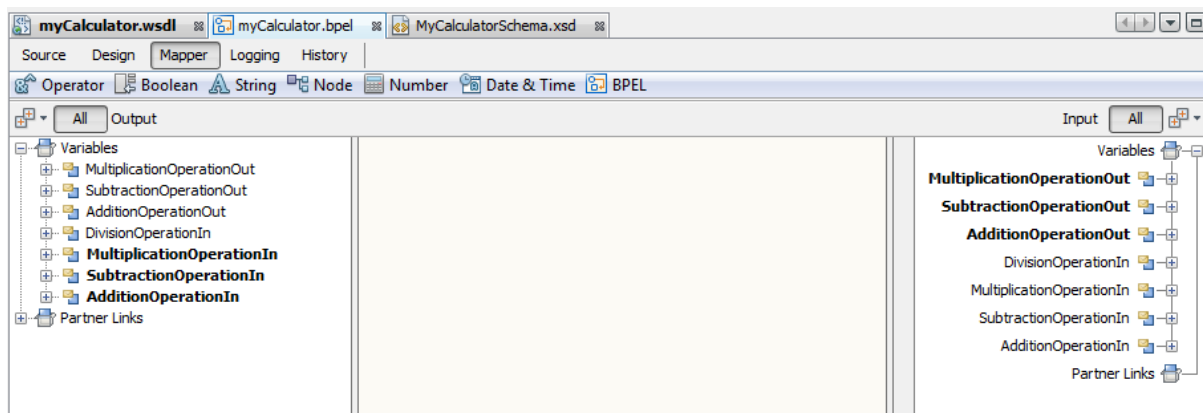
Redo it for subtraction and multiplication.



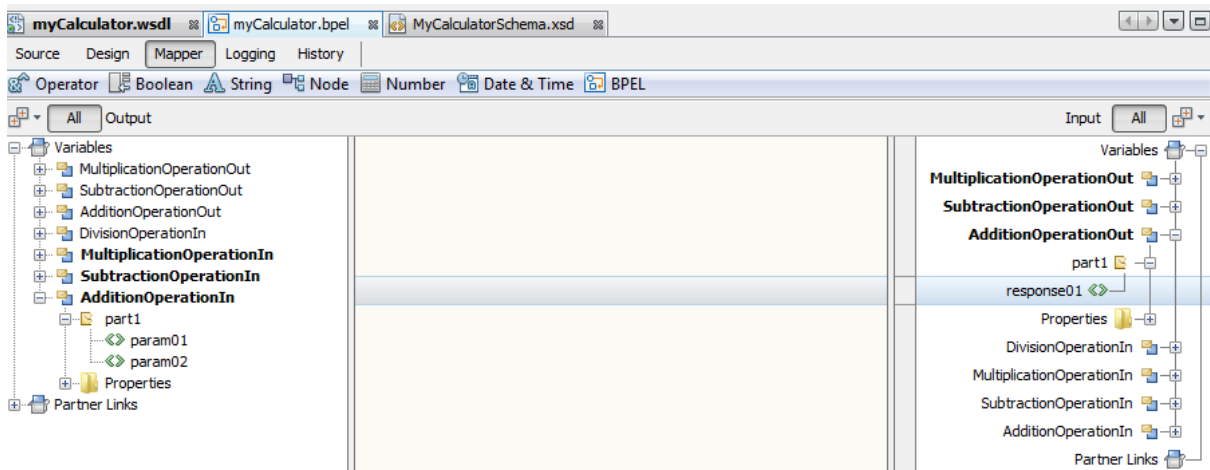
Variable mapping with assign activity

We setup Input and output variables. It is time to map these variables.

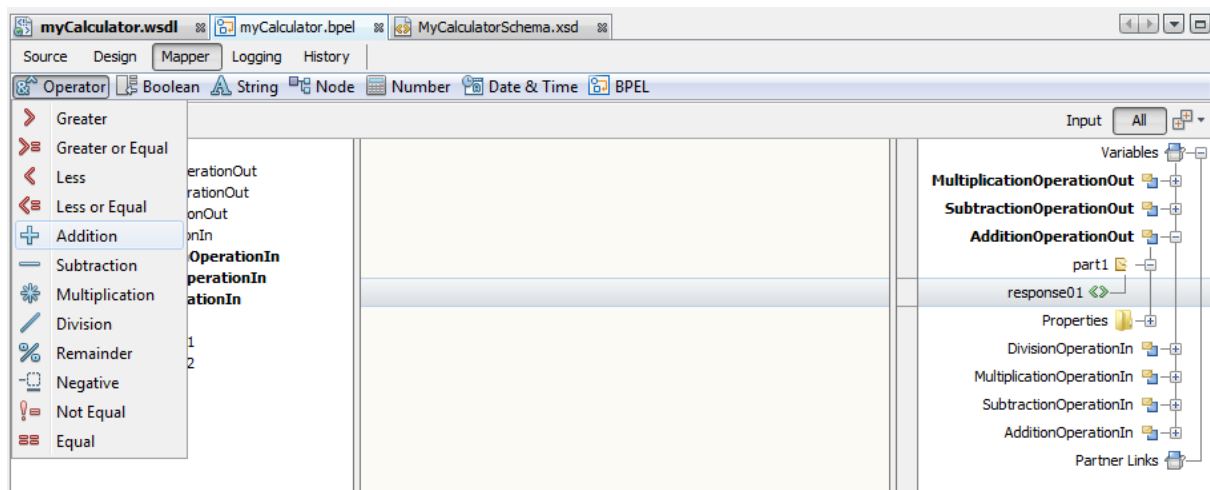
Double click on Assign1 Activity to open the mapper editor. Mapper is used to define a mapping between input and output variable.



On the right lane expand "AdditionOperation" node and select response1. A blue line appears.



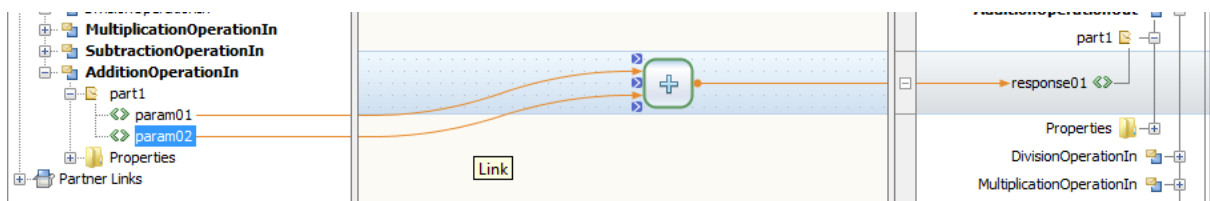
On the mapper menu select Operator→Addition.



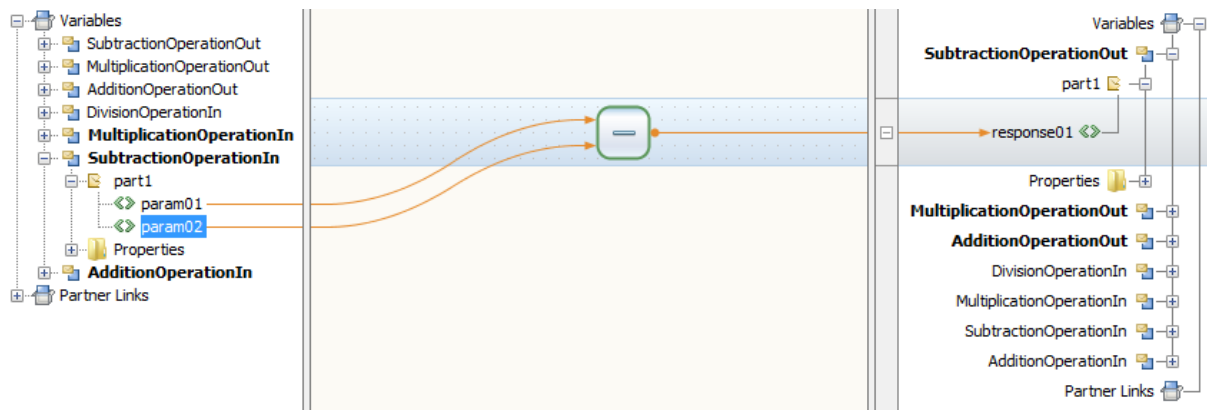
The icon addition appears on the blue line.



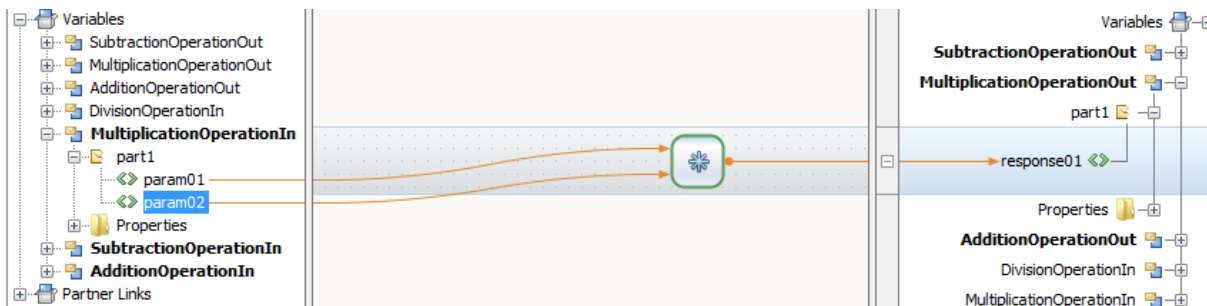
Set param01 and param02 as input and response01 as output. Drag and drop a variable to link it to the icon.



Mapping for Subtraction (Assign2)



Mapping for Multiplication (Assign3)

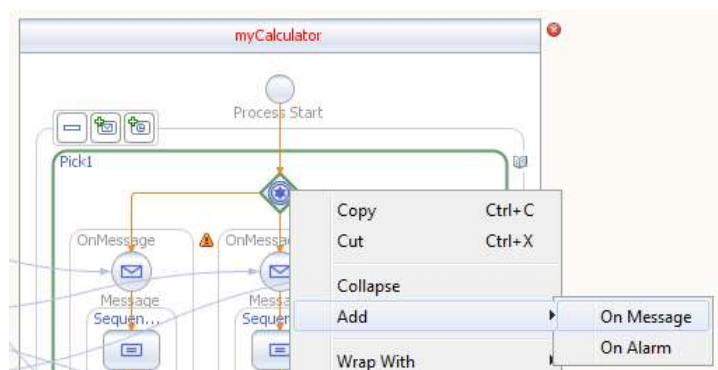


Go back to Design tab and save your work (ctrl+shift+s)

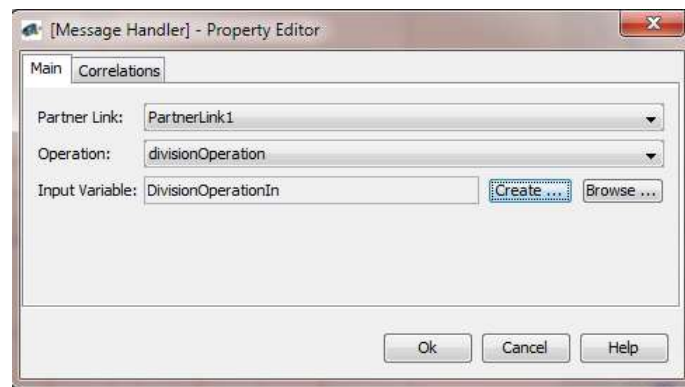
Special case division

There is an issue with the division. The denominator cannot be equal to zero. So, we check if it is equal to zero, we have to return a fault.

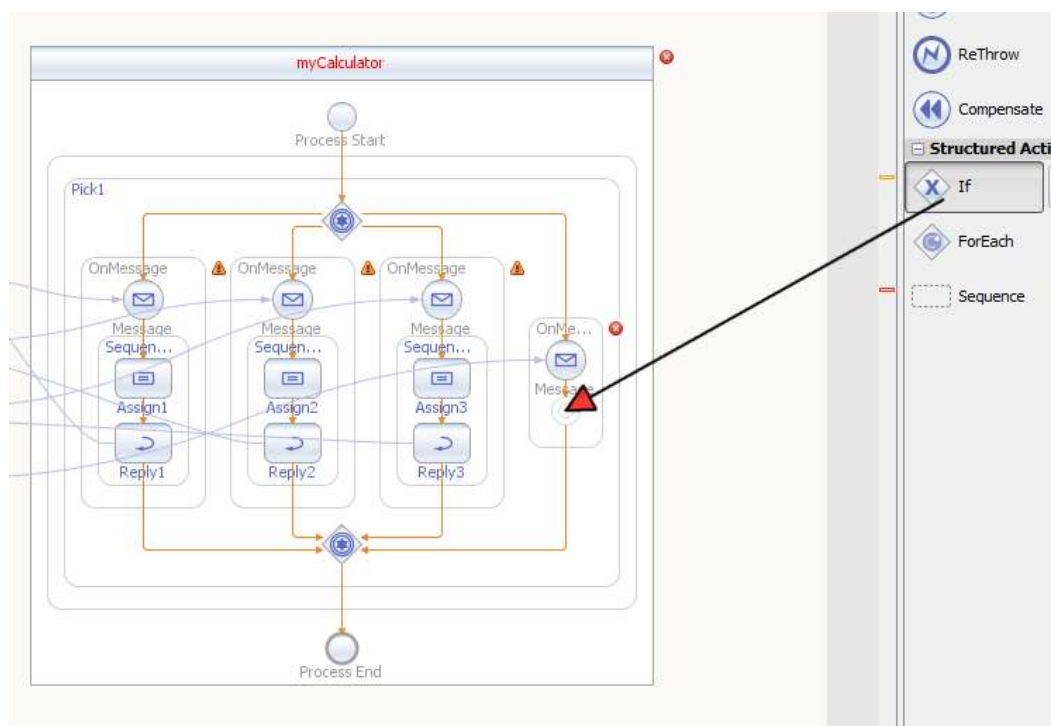
Create a new pick branch for the division.



Set onMessage properties as follows:

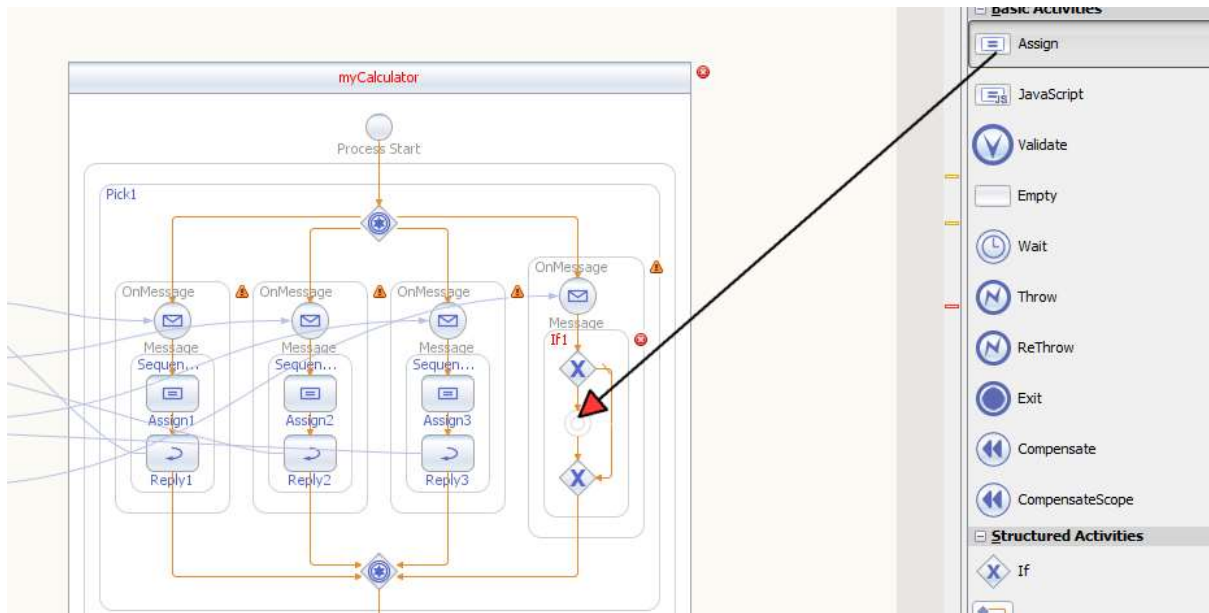


When a division service is invoked, we have to check if the denominator equals zero or not. To do it we add an “if” activity that will validate this condition.

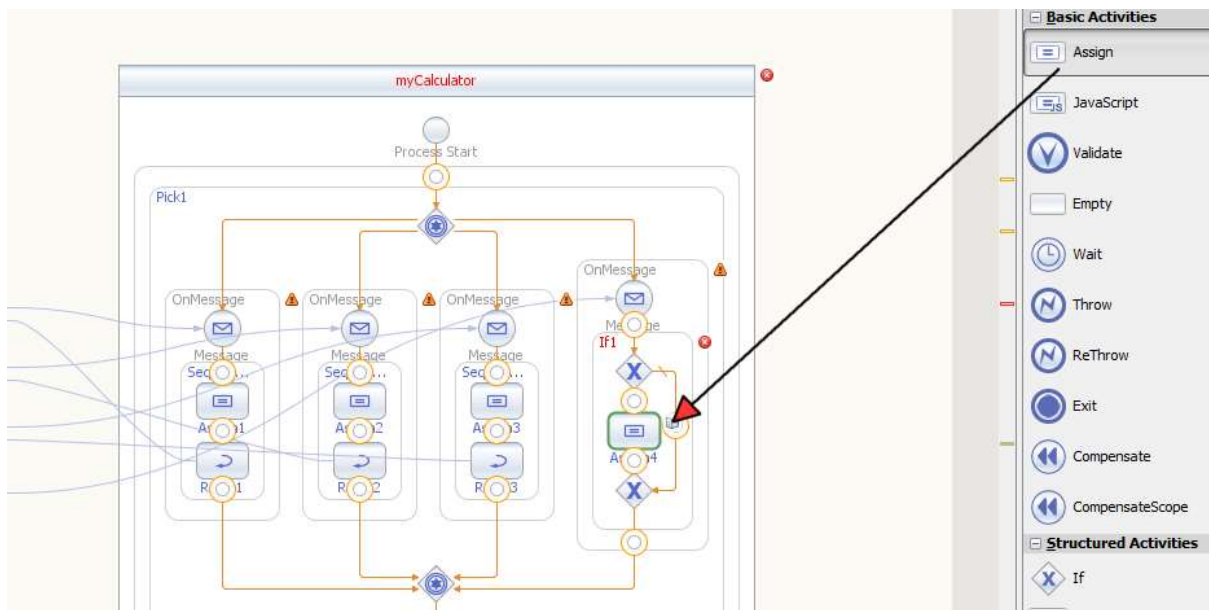


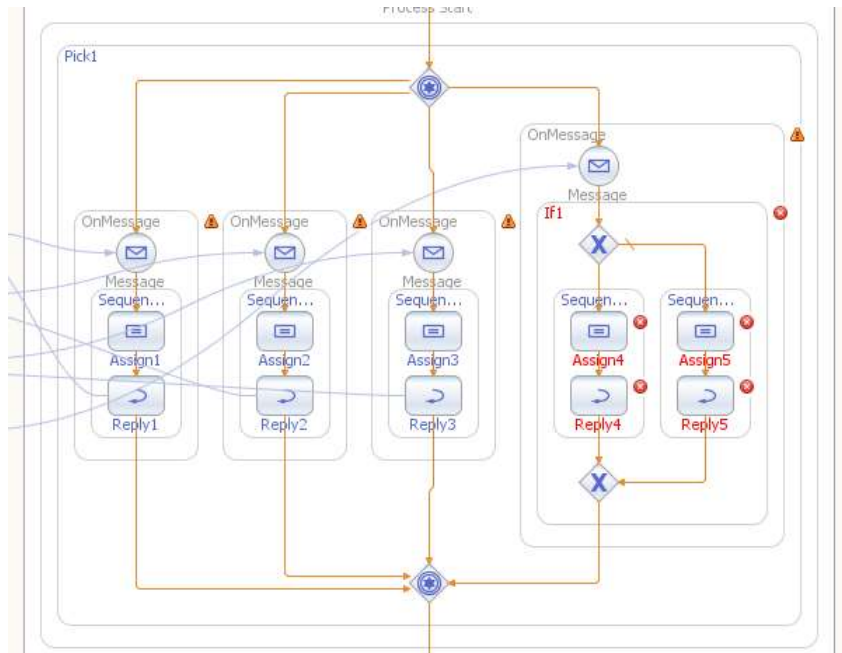
“If” activity creates two branches. The left one is used when the Boolean condition is true and the right one when the Boolean condition is false.

Drag an “Assign” activity and drop it on the left branch of the “if”.

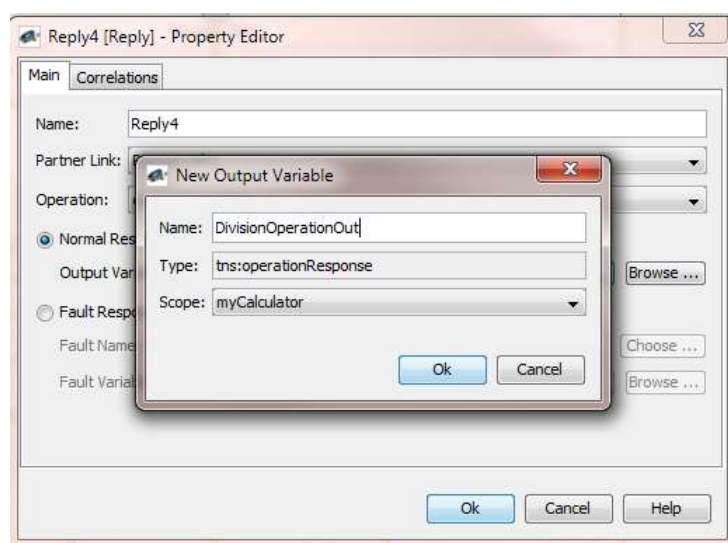
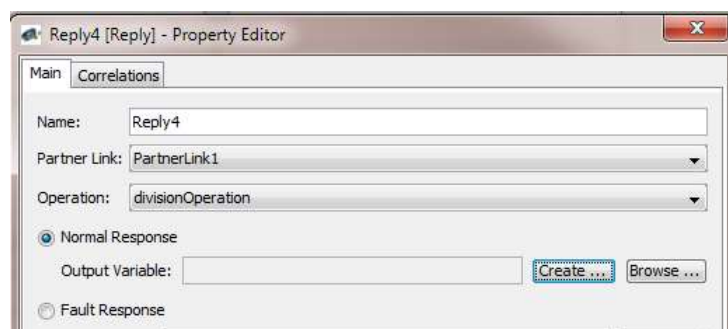


Drag an “Assign” activity and drop it on the left branch of the “if”.





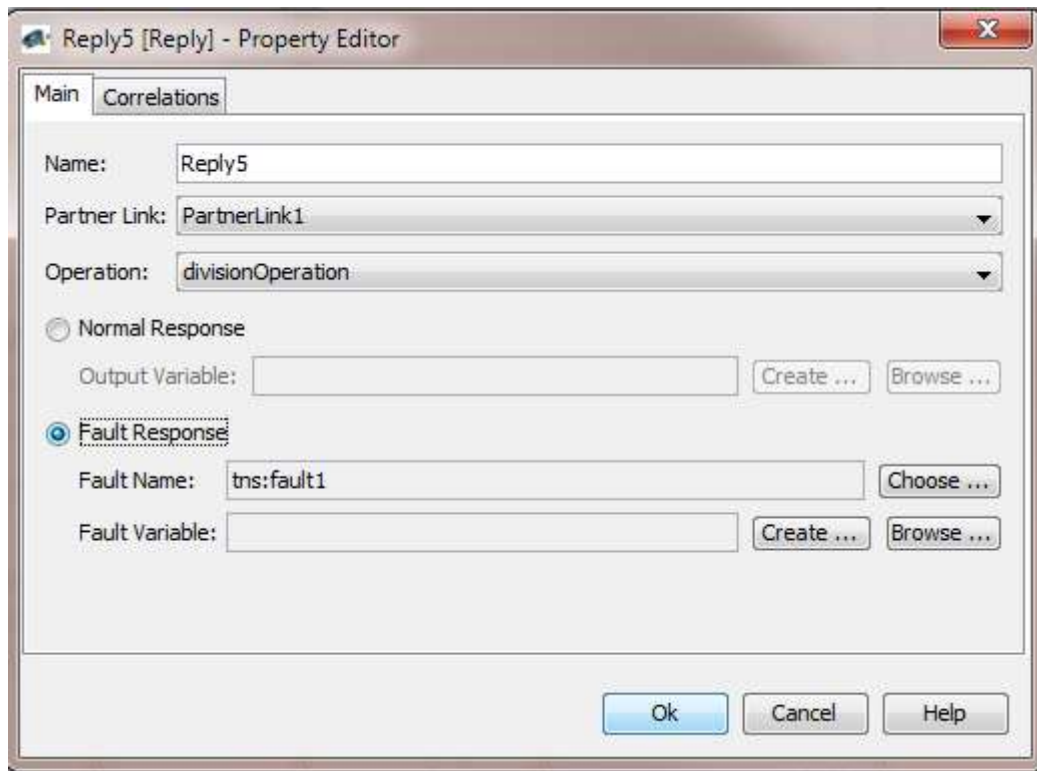
Double click on Reply4 and set up the properties



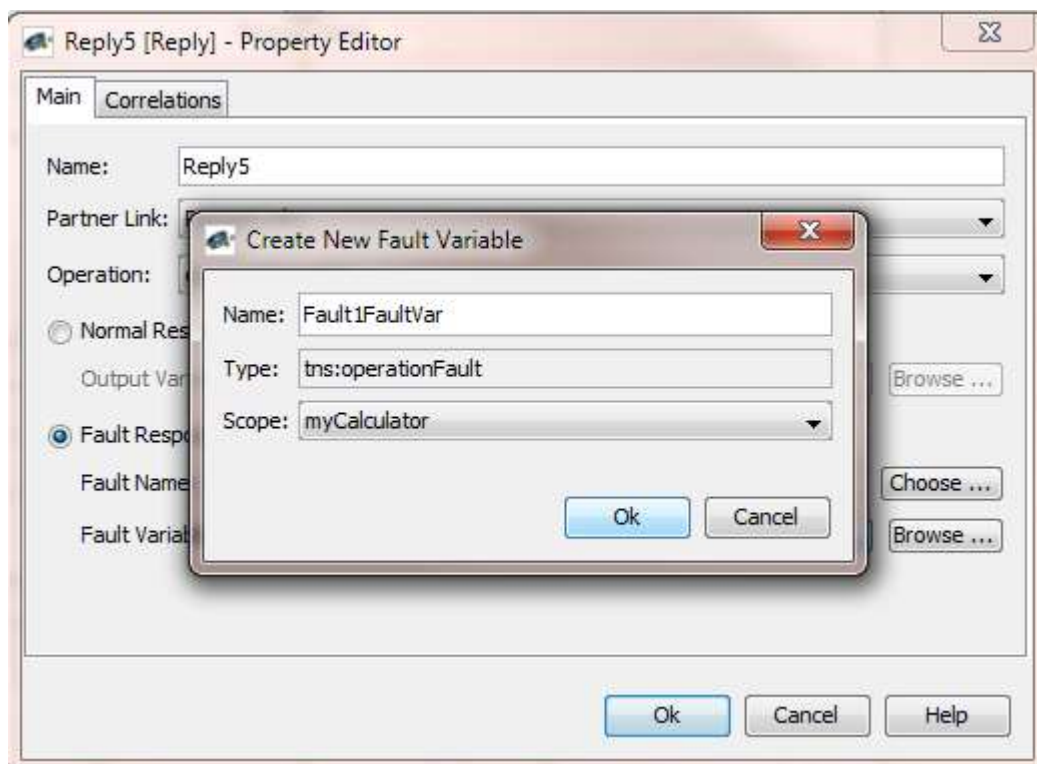
Click OK to create the output variable and OK setup Reply4.

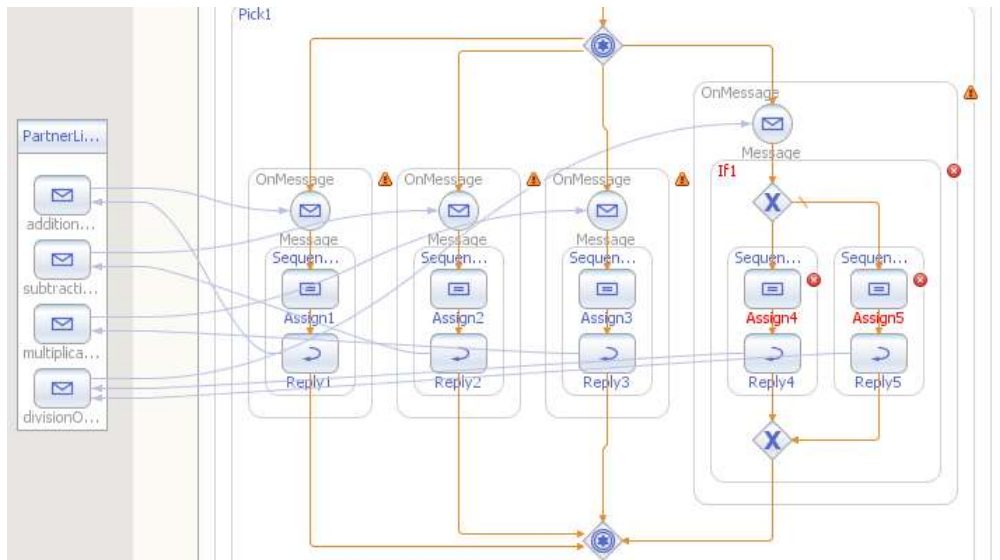
Add a fault as reply

Double click on Reply5 and set up the properties as follows. Note, you setup a response as a “Fault Response” and not a “Normal Response”.



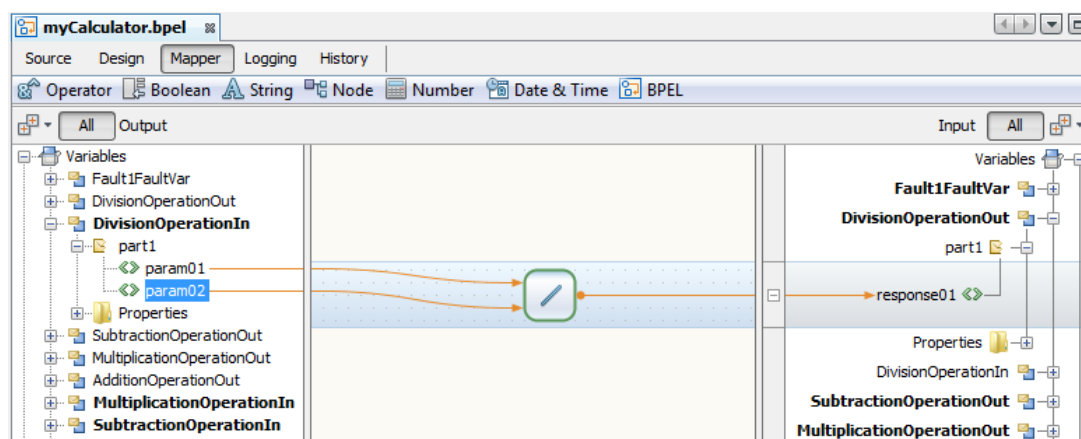
Create a fault variable



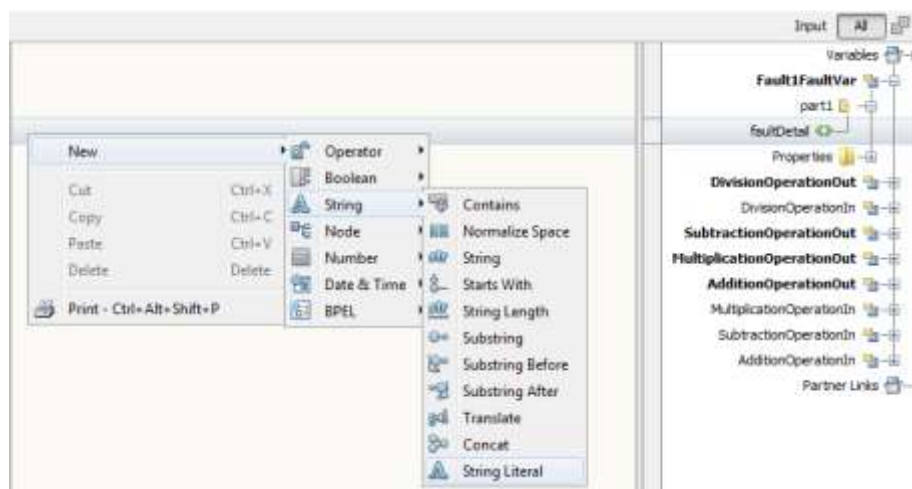


Again, note there is no difference between replying a “Normal Response” or a “Fault Response”

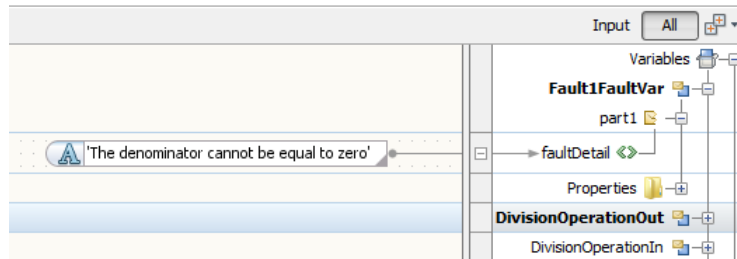
Double click on “Assign4” and set up the mapping for the division.



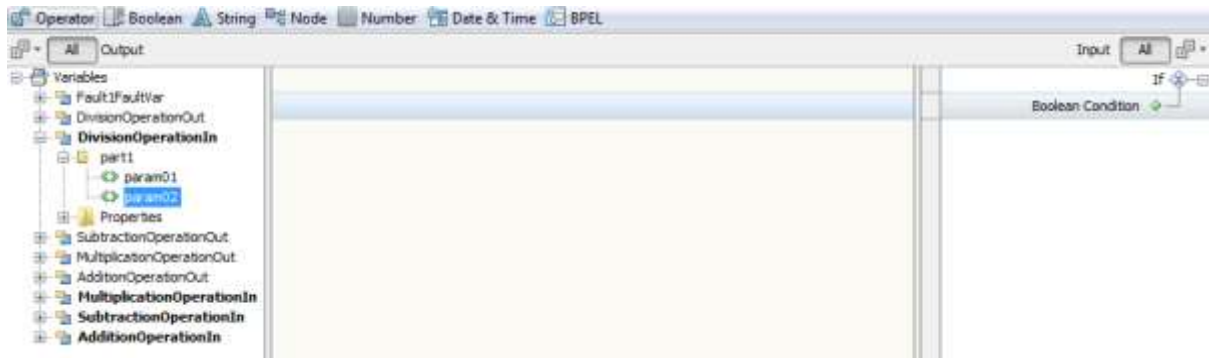
Double click on “Assign 5”, ExpandFault1Faultvar and select faultDetail on the right side. If you work on Windows, click right on the blue line New→String→String Literal. Else use the menu.



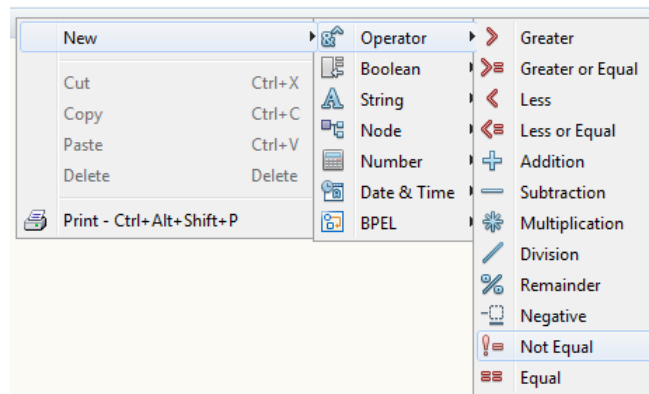
Set String Literal with “The denominator cannot be equals to zero.”



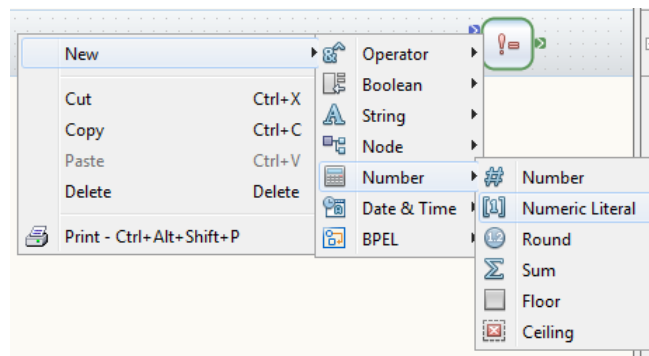
Double click on the if node, to set up the Boolean condition. It looks like the regular mapper. Click on Boolean condition.



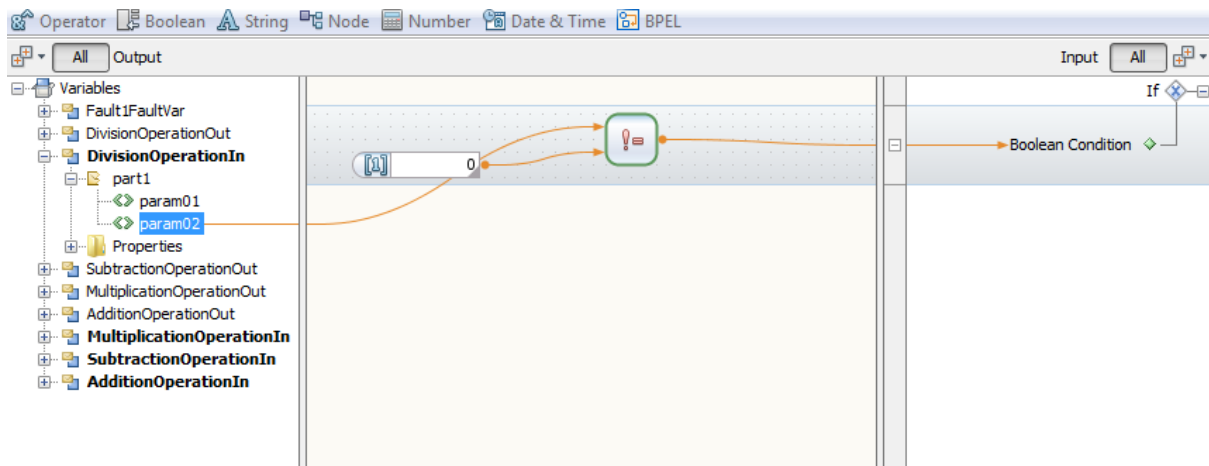
Add the operator “Not Equal”



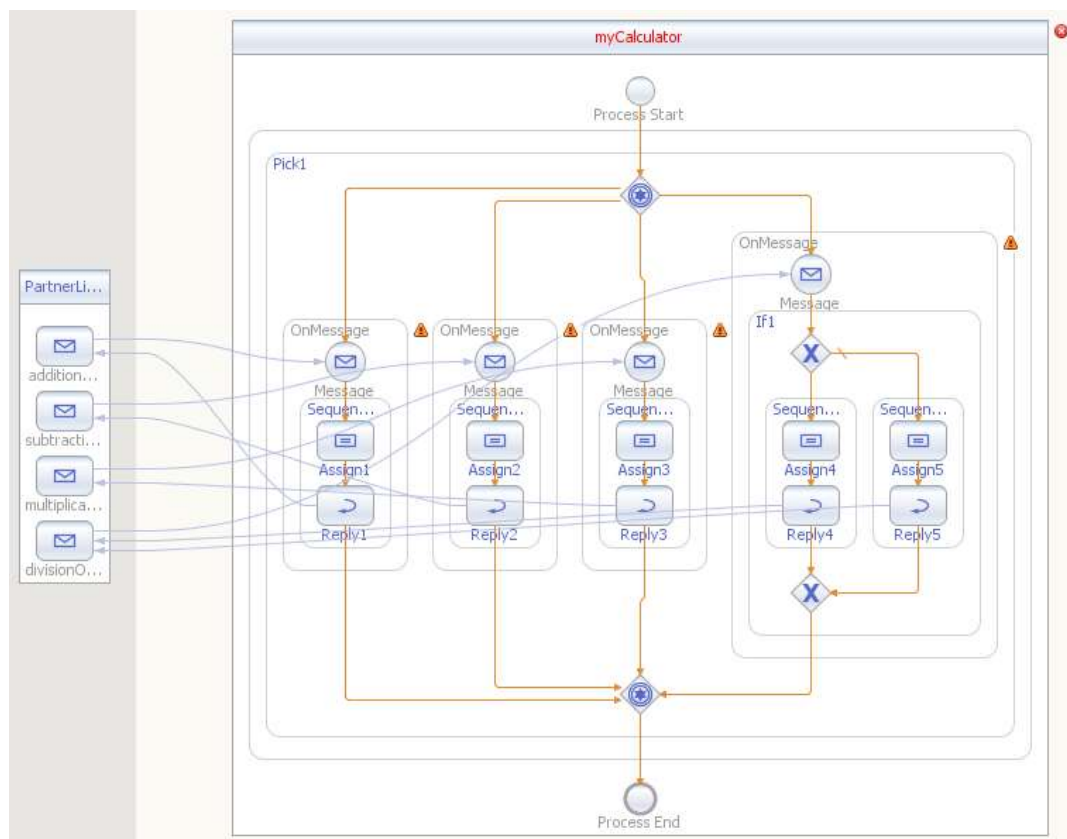
Add a Numeric Literal



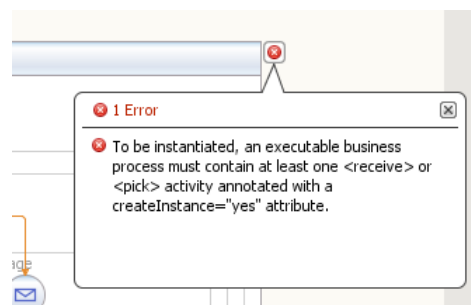
Arrange the mapping as follows:



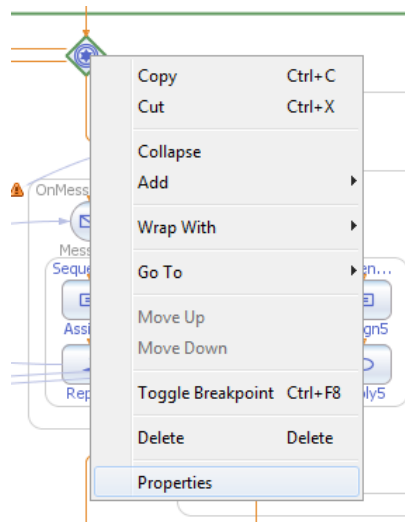
Back to the Design tab the BPEL looks like that”



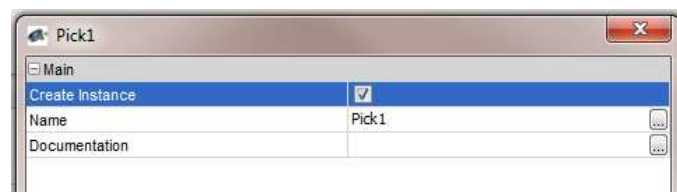
You can note in the upper right corner a red icon. It means that a validation or warning message has been issued by the validation daemon. If you click, you get



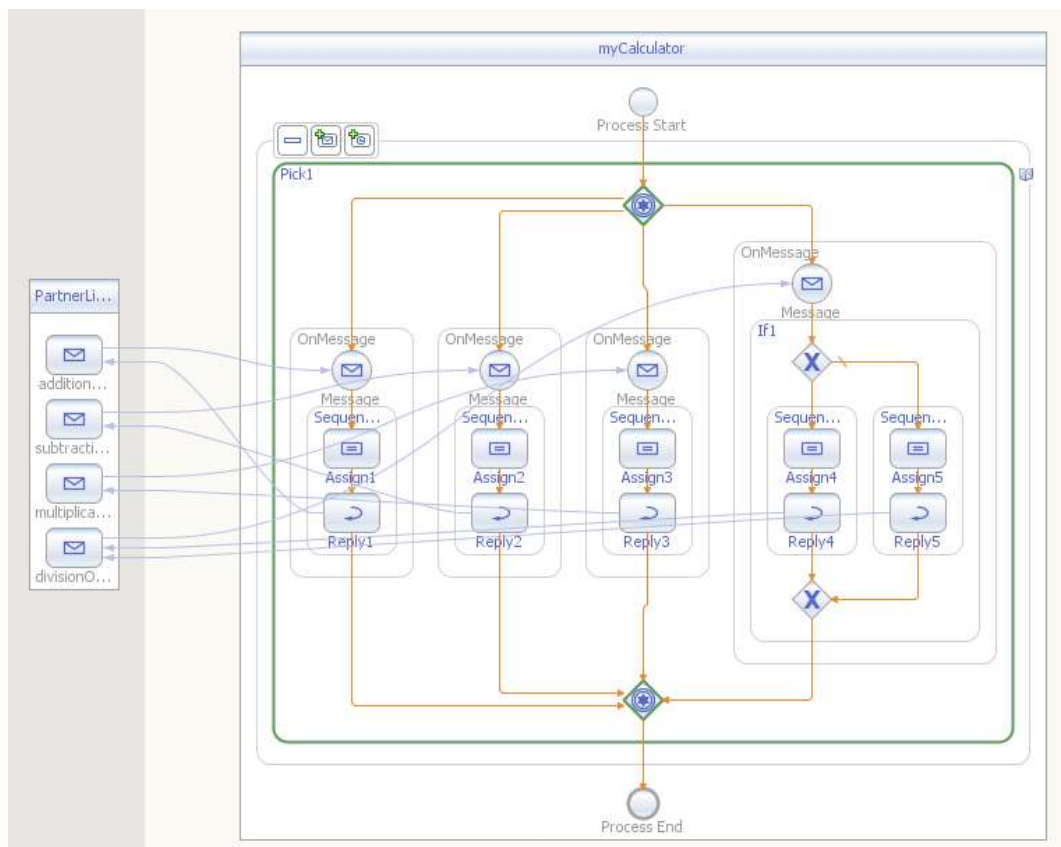
To fix this Error, you have to set up one of the pick properties. Click right on Pick1 → Properties



Check “Create instance” property



Click OK and note the red icon disappears.

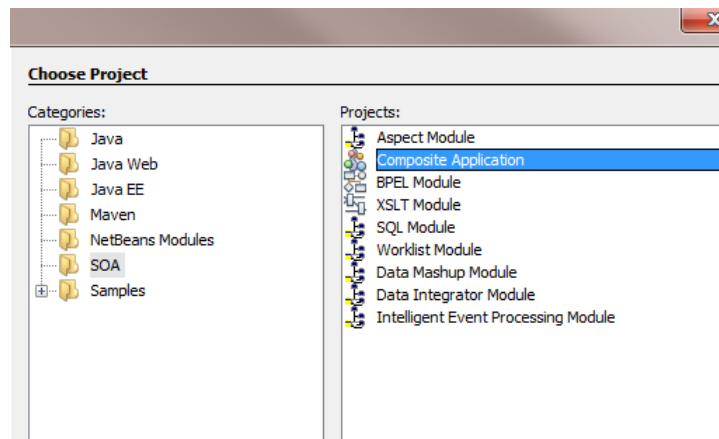


Save your work (Ctrl-Shift-S).

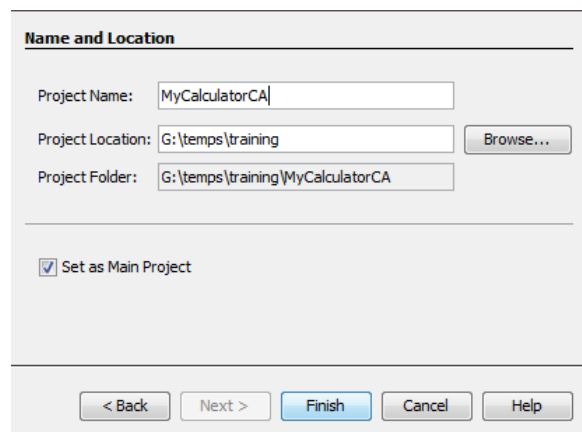
Congratulations, you finish your first complex service unit. It is time for you to create a service assembly to Deploy and test your work.

Create Composite Application

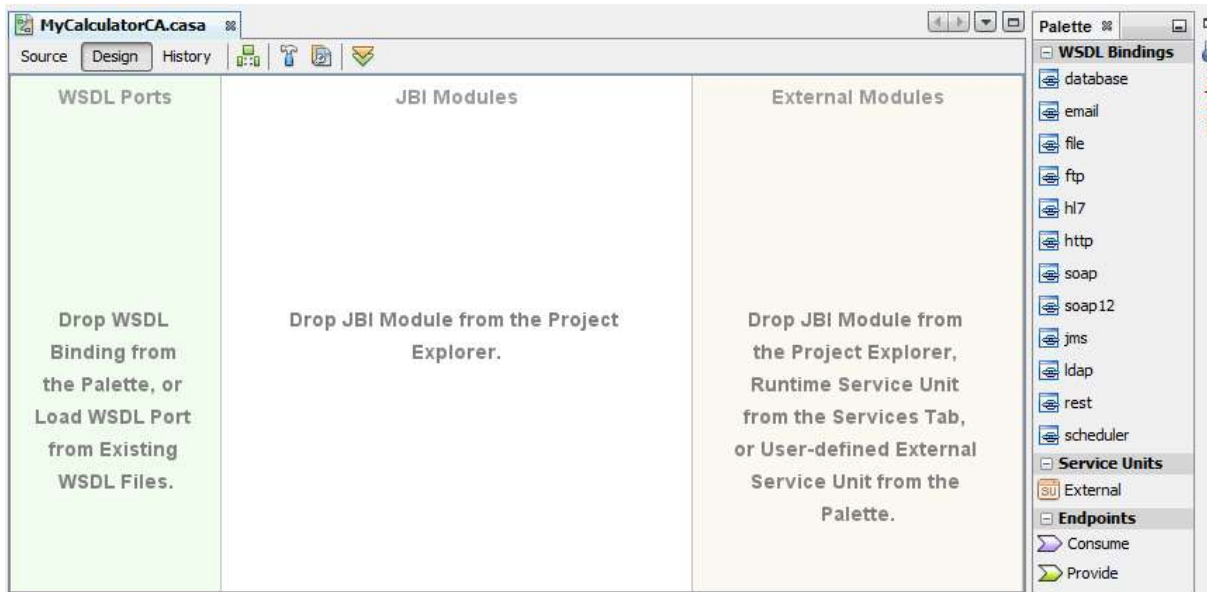
Create a composite application project



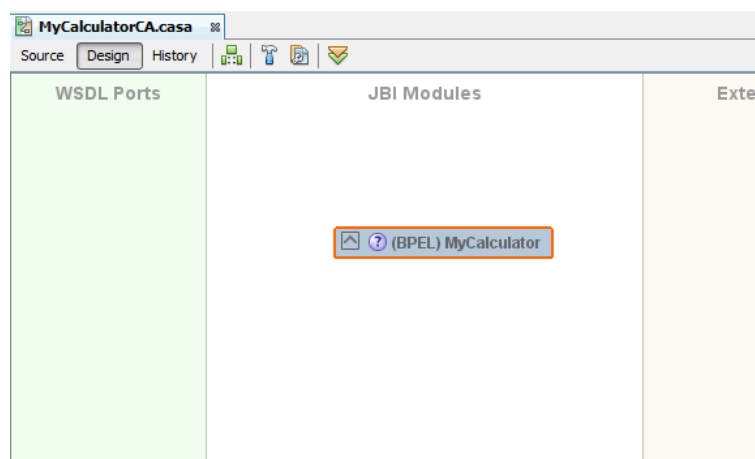
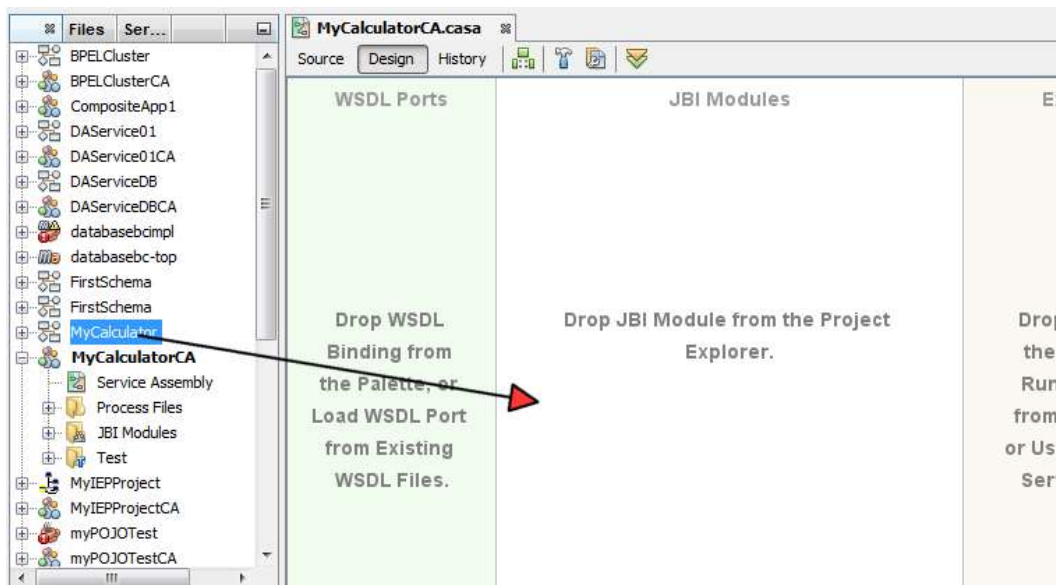
Name it MycalculatorCA




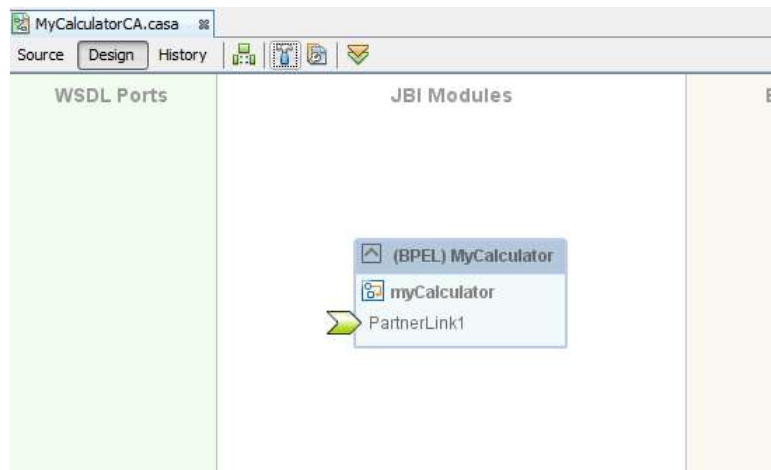
The CASA editor opens. There are three lanes in the CASA editor. The left lane is dedicated to the WSDL Binding ports. These are the channel where messages are received and sent. The central lane is used to show the Service Units embedded in the composite application. We use it to define communication between SUs. In the right lane we set up the JBI Module (or SUs) not embedded in the composite application but used by the composite application.



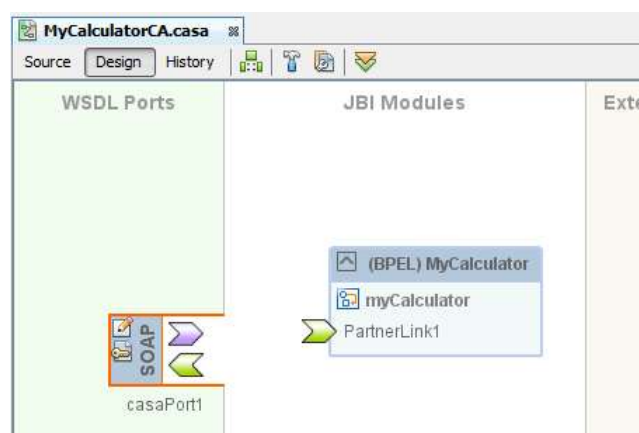
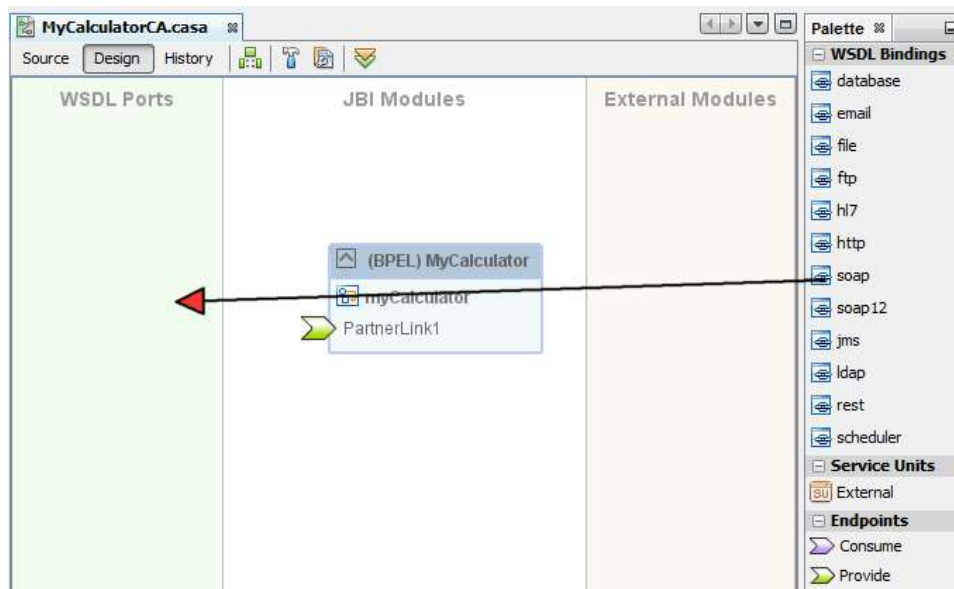
Drag MyCalculator project and drop it in the central lane.



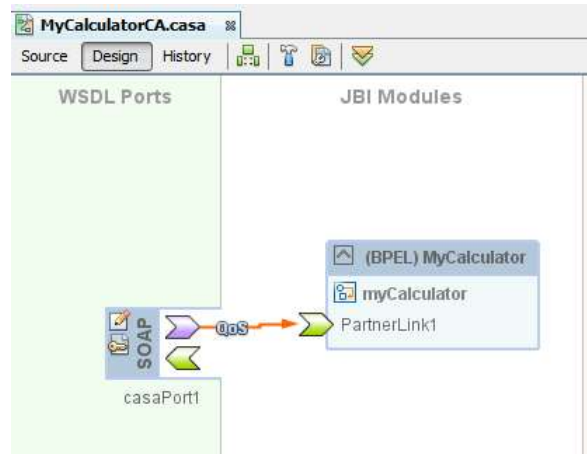
Click on the build button  and wait few seconds. Then the JBI Module port types appears



Drag SOAP icon from the palette and drop it on the left lane.



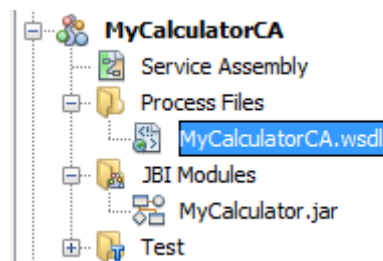
Drag the purple arrow from SOAP and drop it on the green arrow from my Calculator. An orange link between SOAP and MyCalculator appears. So, my calculator can be accessed through a SOAP message.



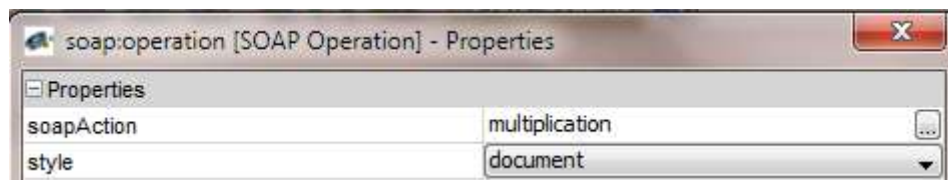
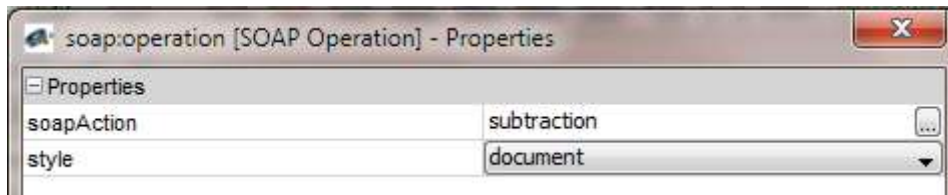
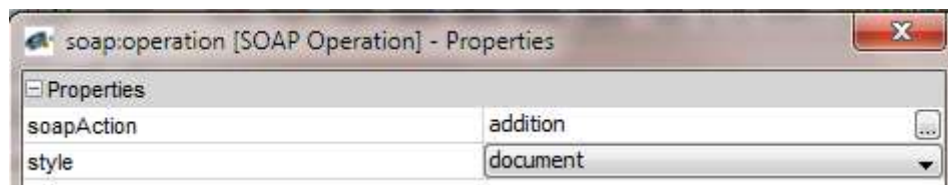
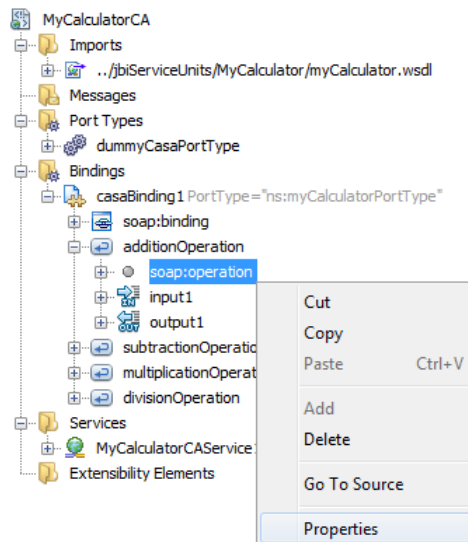
Add SOAP action

For technical reason, a parameter name SOAP Action must be set up. It helps the SOAP port to understand which operation you invoke. SOAP Action is added in the WSDL associated to the CASA (or Composite Application).

Add SOAP Action in the WSDL binding.



Open MyCalculatorCA.wSDL and expand the Binding Node. Click right on Bindings/casaBinding1/additionOperation/soapOperation → Properties. Set up soapACTION with the value addition. Save you setting and redo it for the remaining operations

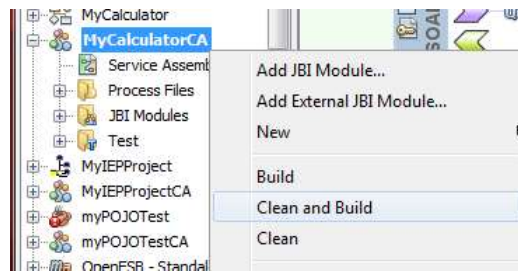


Save your work (ctrl+Shif+S)

Clean and build the project

When you build a project, you create a zip file organised regarding JBI Specifications. Don't care about it for the moment. Just ask for a build, Netbeans does the job for you.

Click right on MyCalculatorCA → Clean and build



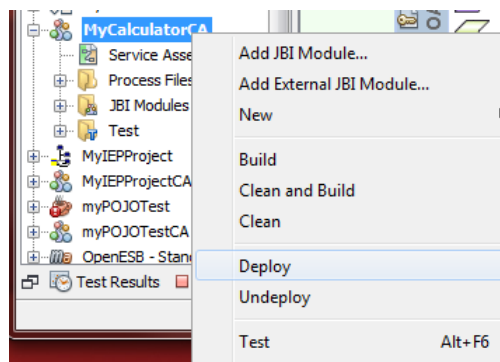
In the output window, you can read the following message

```

Creating/Updating CASA...
Filtering Java EE Endpoints...
Deleting: G:\temps\training\MyCalculatorCA\build\BCDeployment.jar
Created dir: G:\temps\training\MyCalculatorCA\dist
Building jar: G:\temps\training\MyCalculatorCA\dist\MyCalculatorCA.zip
jbi-clean-build:
BUILD SUCCESSFUL (total time: 20 seconds)

```

Then you have to deploy the project into the application server



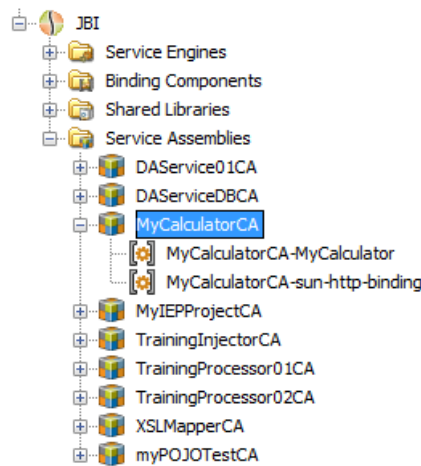
In the output window, you can read the following message

```

Building jar: G:\temps\training\MyCalculatorCA\dist\MyCalculatorCA.zip
run-jbi-deploy:
[deploy-service-assembly]
    Deploying a service assembly...
        host=localhost
        port=4848
        file=G:\temps\training\MyCalculatorCA\dist\MyCalculatorCA.zip
[start-service-assembly]
    Starting a service assembly...
        host=localhost
        port=4848
        name=MyCalculatorCA
run:
BUILD SUCCESSFUL (total time: 4 seconds)

```

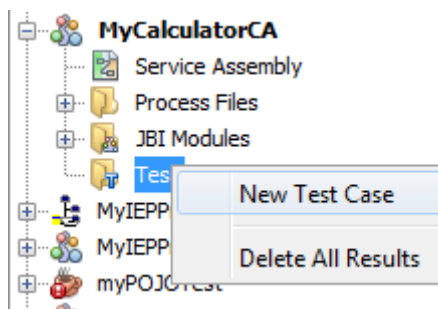
Let's have a look now at the service tab. Expand "Servers" → OpenESB Standalone X.X → JBI → Service Assemblies.



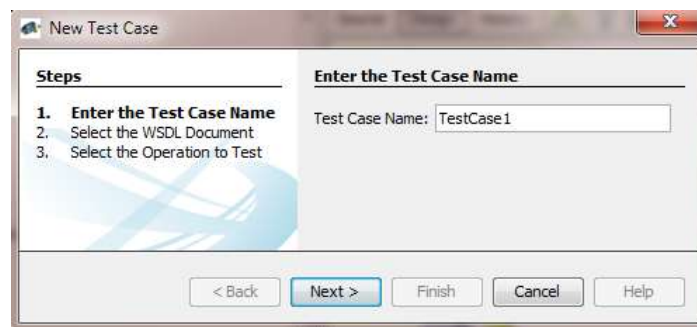
My calculator has been added as a JBI application. Now it is time to test our work

Test the calculator

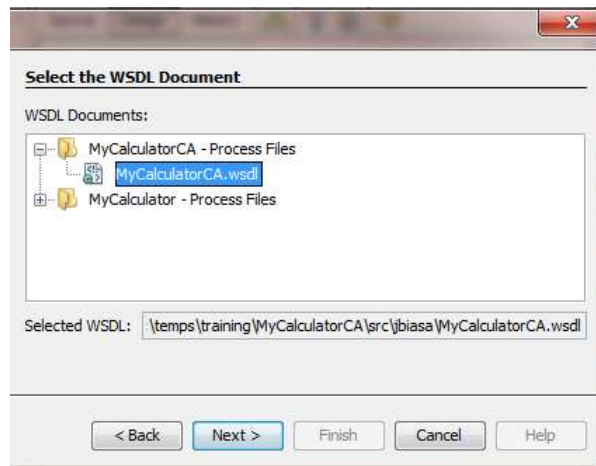
Netbeans proposes a simple but efficient test tool. Back to the Project tab, expand MyCalculatorCA, click right on "Test" → New Test Case.



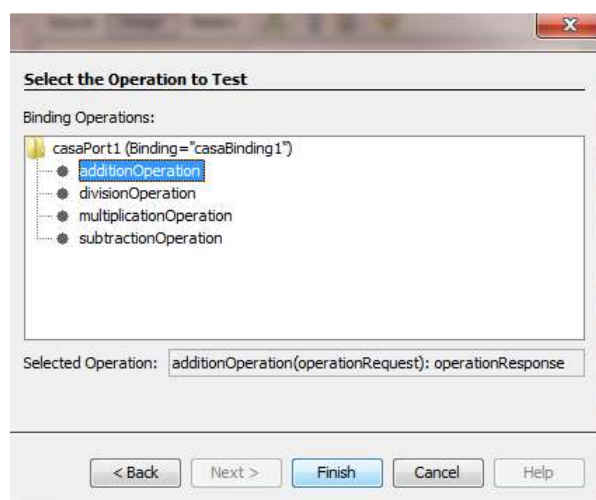
Name the test: TestCase1 and click Next



Expand MyCalculatorCA Process Files Select "MyCalculatorCA.wsdl" then click "Next"

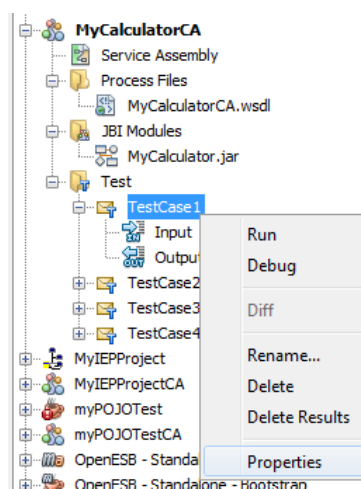


Select “additionOperation” then click Finish.

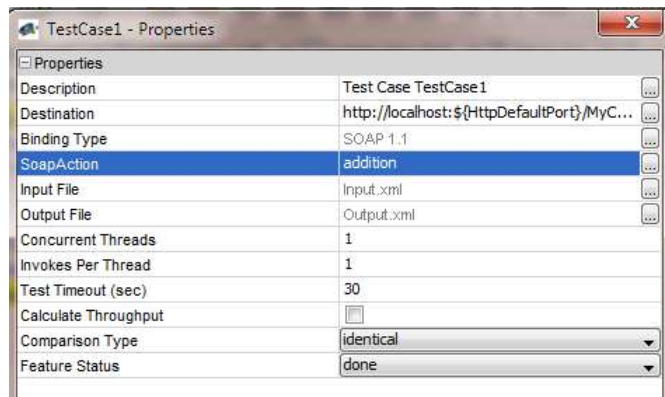


Since the same port type accepts many operations, we have to set up the SOAP Action parameter.

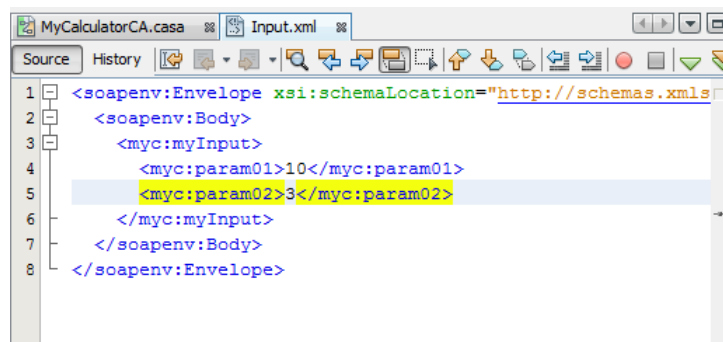
Click right on TestCase1 select Properties.



Setup SOAP Action with “addition”

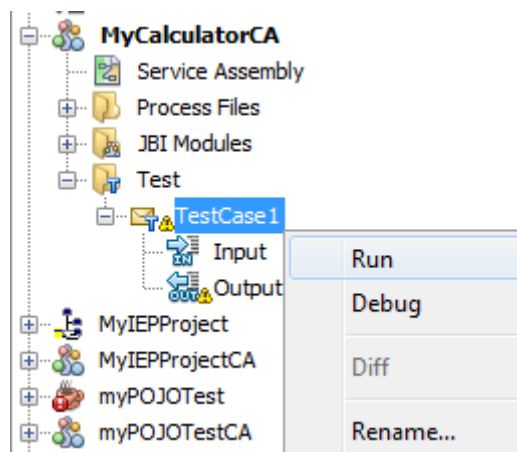


Setup the input file as follows:

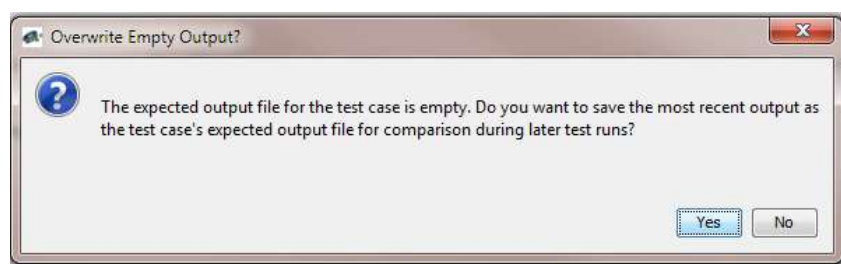


Save the file (Ctrl-Shift-S)

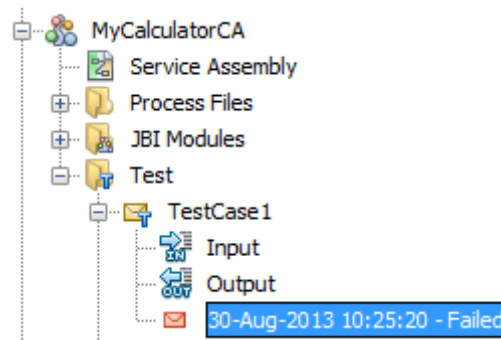
Run the test



You get a popup window then click Yes.



A new result file appears in the test hierarchy. (Don't take into account "Failed")



Open the result

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/ http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <myOutput xmlns="http://xml.netbeans.org/schema/MyCalculatorCA">
      <response01>13.0</response01>
    </myOutput>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Redo the same work for subtraction, multiplication and division.

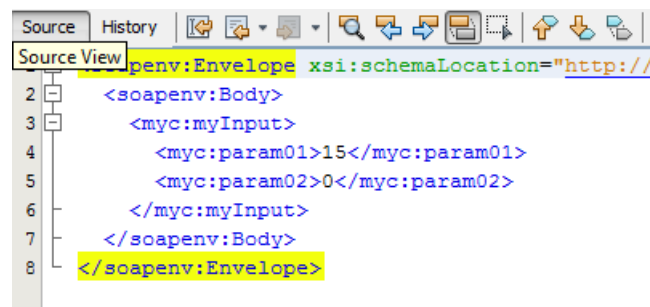
For division operation set up the input text as follows:

```
1 <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
2   <soapenv:Body>
3     <myc:myInput>
4       <myc:param01>15</myc:param01>
5       <myc:param02>5</myc:param02>
6     </myc:myInput>
7   </soapenv:Body>
8 </soapenv:Envelope>
```

The result is

```
1 <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2 <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/ http://schemas.xmlsoap.org/soap/envelope/">
5   <SOAP-ENV:Body>
6     <myOutput xmlns="http://xml.netbeans.org/schema/MyCalculatorCA">
7       <response01>3.0</response01>
8     </myOutput>
9   </SOAP-ENV:Body>
10 </SOAP-ENV:Envelope>
```

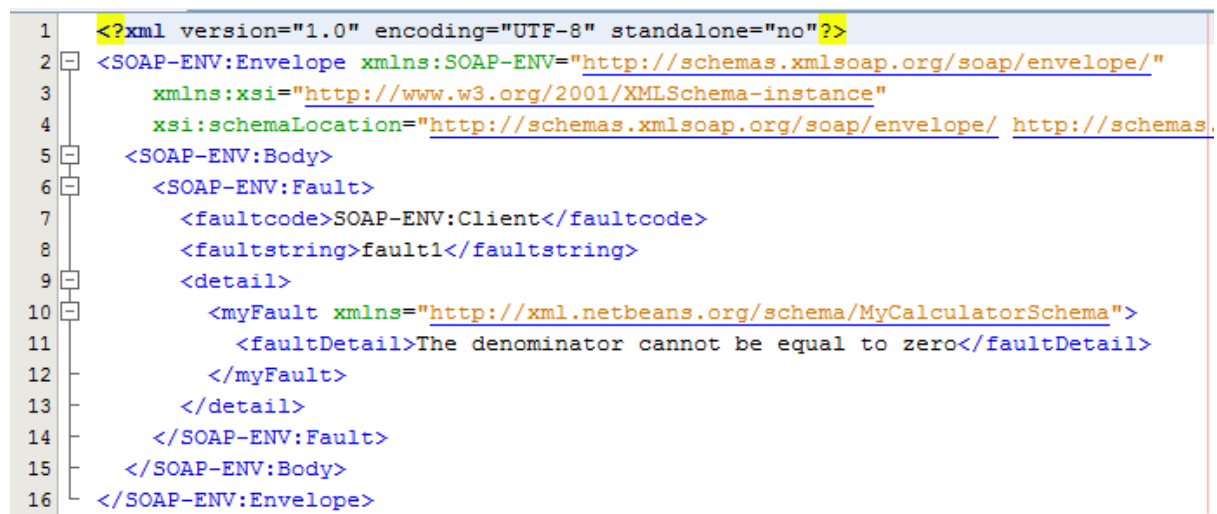
Now change the input as follows:



The screenshot shows a 'Source View' window with a toolbar at the top. The XML content is as follows:

```
1 <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2 <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/ http://schemas.
5 <SOAP-ENV:Body>
6   <SOAP-ENV:Fault>
7     <faultcode>SOAP-ENV:Client</faultcode>
8     <faultstring>fault1</faultstring>
9     <detail>
10      <myFault xmlns="http://xml.netbeans.org/schema/MyCalculatorSchema">
11        <faultDetail>The denominator cannot be equal to zero</faultDetail>
12      </myFault>
13    </detail>
14  </SOAP-ENV:Fault>
15 </SOAP-ENV:Body>
16 </SOAP-ENV:Envelope>
```

The result will be



The screenshot shows a 'Source View' window with a toolbar at the top. The XML content is as follows:

```
1 <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2 <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/ http://schemas.
5 <SOAP-ENV:Body>
6   <SOAP-ENV:Fault>
7     <faultcode>SOAP-ENV:Client</faultcode>
8     <faultstring>fault1</faultstring>
9     <detail>
10      <myFault xmlns="http://xml.netbeans.org/schema/MyCalculatorSchema">
11        <faultDetail>The denominator cannot be equal to zero</faultDetail>
12      </myFault>
13    </detail>
14  </SOAP-ENV:Fault>
15 </SOAP-ENV:Body>
16 </SOAP-ENV:Envelope>
```

The calculator sends back a Fault instead a Normal message as designed in the BPEL.

Congratulation you success!!!

Summary

We covered a lot of ground in this paper. Here are the highlights.

- We detail OpenESB development process based on Schema, WSDL, BPEL and Composite Application.
- We create a Schema with complex type and element.
- We design a WSDL with many operations and Fault
- We create a complex BPEL with Pick and condition
- We designed and deployed a composite application with a SOAP binding
- We test the application

More information on OpenESB at www.open-esb.net or www.pymma.com