



## Tutorial 6: Combining Multiple Services

---

### CONTENTS

Setting up and importing services

Invoking imported services

Passing input to the imported services

Creating a variable to store the sum

Storing the sum of purchases

Using If and Else activities to determine the Output message

Changing the Output message

Deploying and testing

Using debug mode to check values

Reference image of the completed process

This tutorial assumes that you have completed the previous tutorials and builds upon them.

This tutorial provides an example of combining results from more than one service to reach a conclusion. It takes an account number as input, then uses that number to get data from two different services. From one, it retrieves an account balance and charge limit; from another, it retrieves the prices of a number of potential purchases. The process finds the sum of all potential purchases, combines it with the account balance and tests whether the charge limit would be exceeded if the purchases are made.

The tutorial ends with a brief introduction to using debug mode to inspect the values of variables while a process is running.

Prerequisites:

- Micro Focus Verastream Process Design Studio
- An installed and running Micro Focus Verastream Process Server
- Internet browser
- Experience using the XPath and Copy Rule Editors from previous tutorials
- Some familiarity with XML Schema, WSDL, XPath, BPEL, and Web service standards

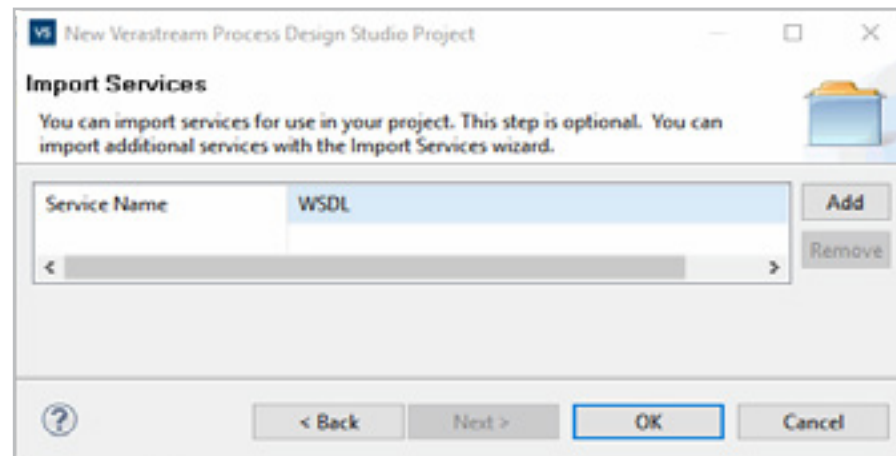
Let's get started.

## Setting up and importing services

You can import a service at any time by selecting **File > Import Service**. You will see the **Import Service** dialog, shown in step 6 of this lesson.

The process you are creating will receive an account number input as a string. The output message will also be stored in a string. The default types for **Input** and **Output** are both string, so you do not have to change them.

1. Start by selecting **File > New Project...**
2. Name the new project **MultipleServices** and click **Next**.
3. On the **Import Services** dialog, click **Add**.
4. On **Import Service**, click **Import a service from the Internet**, then copy this location into the URL field:  
`http://localhost:9999/PurchasesDemoService?wsdl`  
Click **Next**.
5. In the **Name** field, **PurchasesDemoService** has been added. You can change this name to anything you want, but in this tutorial we will use **PurchasesDemoService**. Click **Finish**.
6. To add the second service, click **Add**.
7. On **Import Service**, click **Import a service from the Internet**, then copy this location into the URL field:  
`http://localhost:9999/CICSAcctsDemoService?wsdl`  
Click **Next**.
8. The service is named **CICSAcctsDemo**. Click **Finish**.
9. On the **Import Services** dialog, click **OK**.
10. In the **BPEL Editor**, delete the **DoSomethingHere** activity (and the **AssignValue** it contains) from your default project. Save your project.



## Invoking imported services

Invoking a service is the process of executing a web service operation. Typically, this involves passing it input and accepting output from it. Each input variable that belongs to an imported service is automatically initialized by the **Process Design Studio**, but you must pass a value to it before you can use it in an **Invoke** activity.

1. Open the **Service Explorer** tab, expand **PurchasesDemoService**, and drag **GetPurchasesByAcctNum** into the **BPEL Editor**. Place it between **ReceiveInput** and **ReplyWithOutput**.
2. Expand **CICSAcctsDemo** and drag **GetAccountDetail** into the **BPEL Editor**. Place it between **GetPurchasesByAcctNum** and **ReplyWithOutput**.

*For variables to be initialized automatically in the **Process Design Studio**, the setting **Initialize Variables Automatically**, accessible from the **Preferences** menu, **BPEL Properties** panel, must be selected. This option is selected by default.*

*When you place **GetPurchasesByAcctNum** and **GetAccountDetail** in the **BPEL Editor**, **Invoke** activities are automatically created for them.*

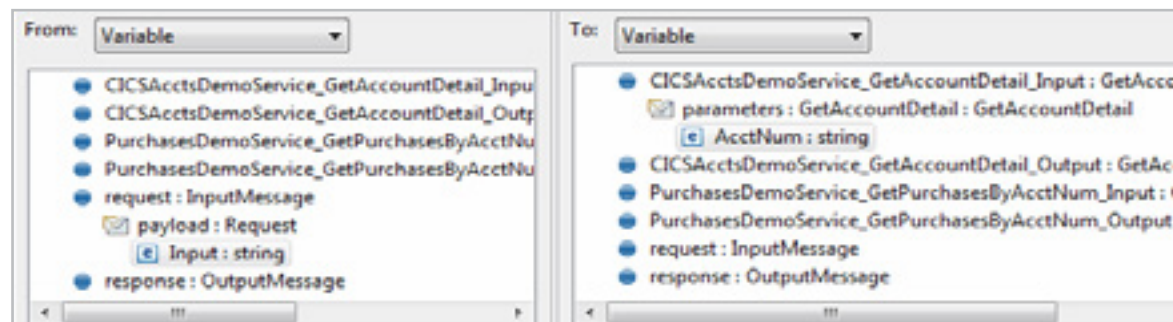
## Passing input to the imported services

At this point, your process only gets an account number as input. You need to add an Assign activity to copy the account number to the input variables of your imported services.

In these steps, you will create Invoke activities for the variables, insert an Assign activity before the Invokes, and then use that Assign to pass values to the service input variables.

You are doing these tasks in this order because the Invoke activities must be in the process before you can access their input variables. You need to copy the account number to the input variables of your imported services.

1. Insert an Assign activity between ReceiveInput and Invoke\_PurchasesDemoService\_getPurchasesByAcctNum. Select **Assign** in the palette and then click between those activities.
2. In the Properties view, open the Description tab and change the name from Assign to **AssignInputs**.
3. In the Properties view, open the Details tab, click the Add copy rule... icon ( **+** ).
4. On the From side, expand `request:InputMessage`, and select `Input:String`.
5. On the To side, expand `CICSAcctsDemo_GetAccountDetail_Input:GetAccountDetail`, then `parameters:GetAccountDetail`, and select `AcctNum:string`.

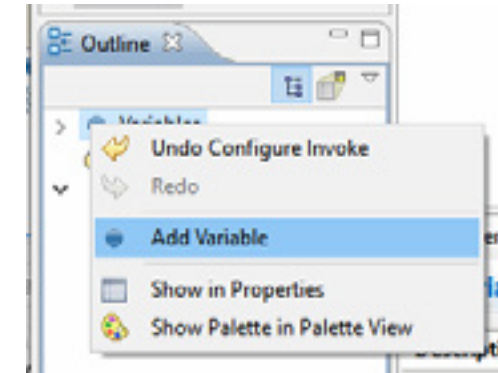


6. Click the Add copy rule... icon ( **+** ) again.
7. On the From side, expand `request:InputMessage`, and select `Input:string`.
8. On the To side, expand `PurchasesDemo_GetPurchasesByAcctNum_Input:GetPurchasesByAcctNum`, then `parameters:GetPurchasesByAcctNum`, and select `AcctNum:string`. Click **OK**.
9. Save the project.

## Creating a variable to store the sum

The GetPurchasesByAcctNum service returns several pieces of information about potential purchases. For this tutorial, you will only be concerned with the price. The next step is to find the sum of the prices of all potential purchases. You will store the sum in a temporary variable named varSum.


1. In the Outline view, right-click **Variables** and select **Add Variable**.



2. Name the variable **varSum**, then click **OK**.
3. In Name field of the Type Selector dialog, type **int** to filter available types, then double-click **integer**.
4. Save the project.

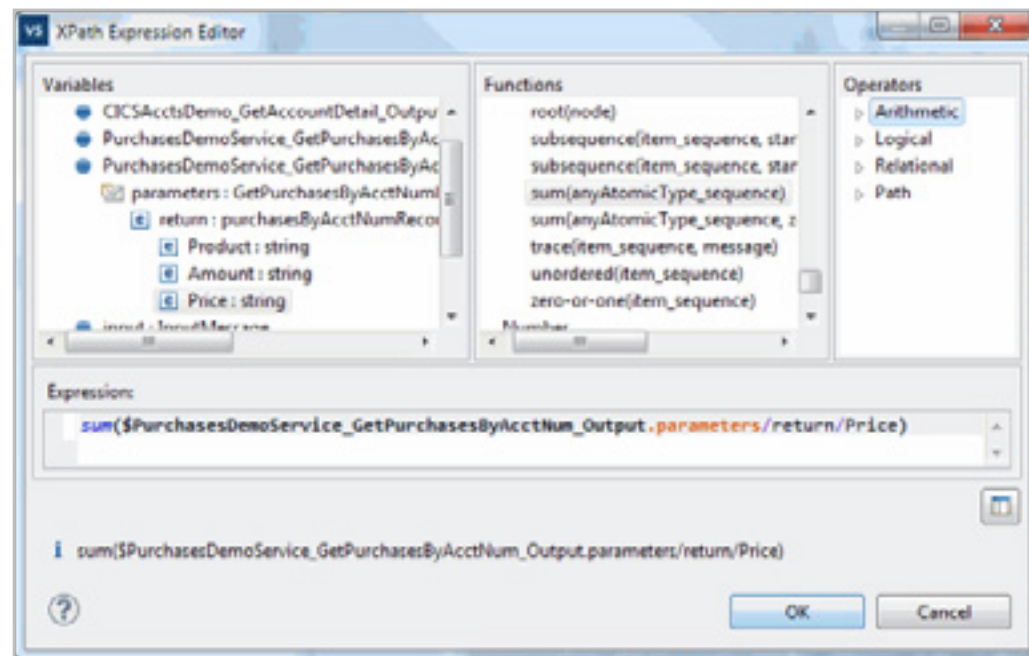
## Storing the sum of purchases

The sum function returns the sum of all of the nodes in a node set. You will use it now to find the sum of all of the Price nodes in the list of purchases.

1. Insert an Assign activity between Invoke\_CICSAcctsDemo\_GetAccountDetail and ReplyWithOutput. Name it **AssignSum**.
2. Select **AssignSum**. In the Properties view, open the Details tab and click the Add copy rule... icon (  ).
3. In the From menu, select **Expression**, then click **XPath Expression Editor...**
4. In the Functions tree, expand **Node** and double-click **sum**. `sum(anyAtomic_sequence)` is added to the Expression field.
5. With `anyAtomic_sequence` selected, in the Variables tree, expand `PurchasesDemoService_GetPurchasesByAcctNum_Output`, then `Parameters:GetPurchasesByAcctNumResponse`, then `return:PurchasesByAcctNumRecord`, then double-click `Price:string`.
6. In the expression that replaces `anyAtomic_sequence`, delete the `[1]`, so the whole expression looks like this:  

```
sum($PurchasesDemoService_GetPurchasesByAcctNum_Output.parameters/return/Price)
```


Click **OK**.

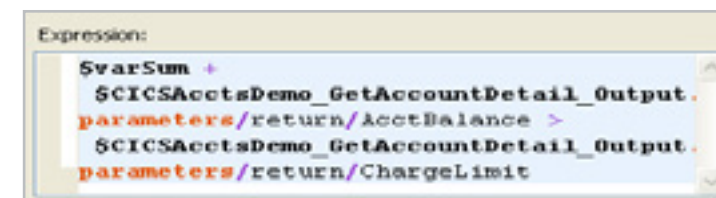


7. On the To side, select `varSum:integer`, and click **OK** and save the project.

## Using If and Else activities to determine the Output message

You have calculated and stored the sum of the cost of all the potential purchases. You can now create an If activity that tests whether the sum plus the current account balance will exceed the account charge limit.

1. Insert an If activity between AssignSum and ReplyWithOutput.
2. In the Properties view, open the **Details** tab and click the pencil (  ) to open the XPath Expression Editor.
3. Delete the default expression, `true()`.
4. In the Variables tree, double-click `varSum:integer`.
5. In the Operators tree, expand **Arithmetic** and double-click **+** (**addition**).
6. In the Variables tree, expand `CICSAcctsDemo_GetAcctDetail_Output:GetAccountDetailResponse` then `parameters:GetAccountDetailResponse` then `return:AccountDetailRecord` then double-click `AcctBalance:string`.
7. In the Operators tree, expand **Relational** and double-click **>** (**greater than**).
8. In the Variables tree, expand `CICSAcctsDemo_GetAcctDetail_Output:GetAccountDetailResponse` then `parameters:GetAccountDetailResponse` then `return:AccountDetailRecord` then double-click `ChargeLimit:string`. Click **OK**.

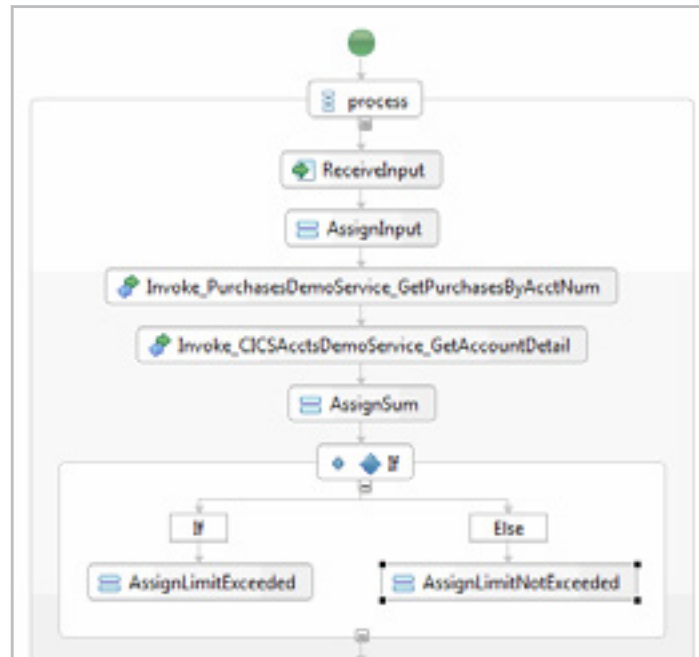


9. Save the project.

## Changing the Output message

The last step before deploying and testing is to add Assigns that will change the Output message depending on the credit status of the account.

1. Place an Assign activity inside the If activity. Name it **AssignLimitExceeded**.



2. Click **AssignLimitExceeded**. In the Properties view, open the **Details** tab and click the Add copy rule... icon ( + ).
3. In the From menu, select **Fixed Value**. In the text field that appears, type (including the quotes):  
**"These purchases would exceed the account credit limit."**
4. On the To side, expand `response:OutputMessage`, then expand `payload:Response` and select `Output:string`. Click **OK**.
5. Add an Else activity to the If activity. Right-click on the If and select the Else.
6. Place an Assign activity inside the Else activity. Name it **AssignLimitNotExceeded**.
7. Click **AssignLimitNotExceeded** and add a copy rule. +
8. In the From menu, select **Fixed Value**. In the text field that appears, type (including the quotes):  
**"Below the credit limit. The purchase is OK!"**
9. On the To side, expand `response:OutputMessage` and `payload:Response`, and select `payload:Response`. Click **OK**. Save the project.

## Deploying and testing

It's time to deploy and test your new process. This process requires account numbers. For this tutorial, valid account numbers are 20000, 20001, 20002, 20003, 20004, 20006, 20007 and 20008. Try to input 20000. The result should be a message warning you that the purchases would exceed the account's credit limit. Try 20003 -- the result should be a message saying the purchase is ok.

*20005 is not a valid account number, a fact that will be helpful in Tutorial 7.*

1. From the Actions menu, select **Deploy to Process Server**.
2. Enter the name, username and password for the server. The defaults are:  
name: **localhost** username: **admin** password: **secret**
3. In the Deployment Succeeded dialog box, click **Test Service...** to launch the Web Services Explorer.
4. Select SOAP11BINDING from the left panel of the Web Services Explorer.
5. You will see a single Input field. Enter an account number, for example, 20000, and click **Go**.
6. If you entered 20000, you should see the message:  
**"Below the credit limit. The purchase is OK!"**  
Try other values, such as 20004.

*See what happens if you try invalid account numbers, such as 1 or BBB. In the next tutorial, you will create fault handlers that respond to invalid input.*

## Using debug mode to check values

You have tested the process and found that both the 'purchase ok' and the 'credit limit exceeded' message may appear, depending on the account number input. You do not know, however, if they are appearing when they should. Perhaps the process is making an error and showing the wrong message. To determine whether that is happening, you must know the values of the sum of purchases, the account balance and the credit limit. One way to see those values is to use the Process Design Studio's debug mode.

The debug mode allows you to follow the progress of your process and inspect the values of variables as activities execute. This can be very helpful for finding the source of problems in your processes.

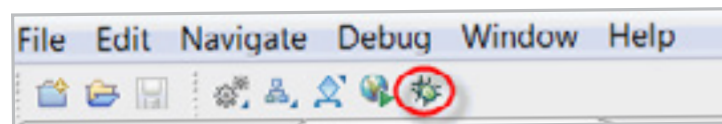
In debug mode, the process executes until it hits a *breakpoint*. A breakpoint is a marker you can add to most activities in your process. When debug mode encounters a breakpoint, execution of the process stops and you have an opportunity to examine the current state of the elements that compose your process. You can also *step* through your process after a breakpoint -- executing only one item at a time so you can see how things change.

A breakpoint on an activity is indicated in the BPEL Editor by a blue circle, like this:




To remove a breakpoint, right-click the activity and select *Remove Breakpoint*.

1. In the BPEL Editor, right-click the If activity and select **Add Breakpoint**.
2. Deploy the process again, but this time, deploy in debug mode by clicking the debug button near the top of the process designer.



3. If prompted, enter the name, username and password for the server. The defaults are:  
name: **localhost** username: **admin** password: **secret**
4. Input **20000** and click **Go**. You will see the Debug view of the Process Design Studio. (If you do not, then switch to the Process Design Studio, and from the Window menu, open the Debug view.)  
  
The breakpoint is on the If activity, so the debug view can currently show the value of variables before the If activity.

Name	Value
input	<message xmlns:ns="h
CICSAcctsDemo_GetAccountDetail_Input	<message xmlns:ns="h
PurchasesDemoService_GetPurchasesByAcctNum_Input	<message xmlns:ns="h
output	<message xmlns:ns="h
PurchasesDemoService_GetPurchasesByAcctNum_Output	<message xmlns:ns="h
varSum	<temporary-simple-ty
CICSAcctsDemo_GetAccountDetail_Output	<message xmlns:ns="h

5. Double-click on **varSum**. Its value should be 957.5.
6. Double-click on **CICSAcctsDemo\_GetAccountDetail\_Output**. Find ChargeLimit and AccountBalance (you may have to scroll down). ChargeLimit should have a value of 1000, and AccountBalance should have a value of 458.25.
7. The value of varSum plus the value of AccountBalance is greater than the value of ChargeLimit, so the 'credit limit exceeded' message should show if you input account number 20000. Fortunately, it does.
8. When you are ready, click **OK** to dismiss the box that contains variable values.
9. In the Debug view, click the Resume button (  ) to allow the process to continue to the end (or the next breakpoint, whichever comes first).
10. Try a few other account numbers in debug mode -- does the correct message appear?

*You can double-click any variable in debug view to inspect its contents.*

*You can redeploy without debug mode--just choose File > Deploy to Process Server...*

*It is very unlikely that you will ever want to publish a completed process in Debug mode. One reason is that it adds code to the process that is usually not needed for the process to run as expected.*

Reference image of the completed process

