



## Tutorial 7: Fault Handling

---

### CONTENTS

Setting up and adding a fault message

Updating copy rules

Enabling a Catch All fault handler

Adding a Catch All fault handler

Returning a fault message

Checking your work

Validating input

Throwing a user-defined fault

Catching a user-defined fault

Returning an incorrect length fault message

Checking your work

Hiding fault handling: the orange X

Finding a service-defined fault

Catching a service-defined fault

Deploying and testing

Reference image of the completed project

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

Fault handling is an important part of a robust production process. Processes do not always perform as expected. Sometimes an issue can be anticipated. For example, users will sometimes input invalid data. Sometimes an issue is completely unexpected. Fault handling helps address these concerns. Depending on the issue, the process may be able to recover or it may need to display (and possibly log) information and exit.

This tutorial introduces fault handling in Process Design Studio.

You will start this tutorial by opening the project you completed in Tutorial 6.

You will add a Catch All to detect and respond to any issue. You will then add a Catch for a user-defined fault, which will detect invalid account formats. Finally, you will add a Catch for a service-defined fault, which detects invalid account numbers.

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
- A completed project file from Tutorial 6.

Let's get started.

## Setting up and adding a fault message

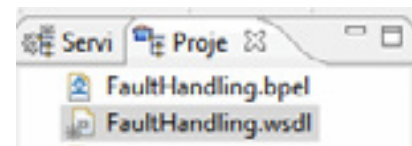
By default, project files are stored in:

My Documents\Micro Focus\Verastream\ ProcessDesigner\ workspace

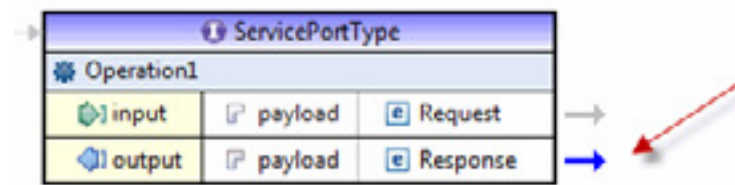
To open the Project Explorer, from the Windows menu choose Project Explorer.

For this tutorial, you'll use the project file from the previous tutorial (with a different name) and add a string to the Output message for a fault message. To add the string, you make the Output variable a complex type, so it can hold both credit check and fault messages.

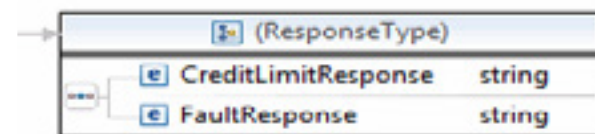
1. Open the completed project from Tutorial 6. (Choose **File > Open Project...**)
2. Create a copy of the project. Choose **File > Save As**. Name the new project **FaultHandling**. Click OK.
3. To open the WSDL Editor, from the Project Explorer, double-click **FaultHandling.wsdl**.
4. To open the Schema Editor, double-click the arrow to the right of Response.



5. Select Output and in the Properties view, on the General



- tab, name the output CreditLimitResponse.
6. Right-click ResponseType and choose Add Element, to create another new element.
  7. Select the other new element, and in the General tab and change its name to FaultResponse. You should now have two elements named CreditLimitResponse and FaultResponse.



8. Close the Schema Editor and the WSDL Editor, and save the project.

## Updating copy rules

Now you must update the copy rules.

1. Select **AssignLimitNotExceeded** and in the Details tab, double-click to open the copy rule that includes the literal **"Below the credit limit..."**
2. On the To side, expand `response:OutputMessage`, and `payload:Response`. Select `CreditLimitResponse:string`, then click OK.
3. In the BPEL Editor, select **AssignLimitExceeded**. In the Properties view, open the Details tab and double-click the single copy rule to open it.
4. On the To side, expand `response:OutputMessage`, and `payload:Response`. Select `CreditLimitResponse:string`, then click **OK**.
5. Save the project.

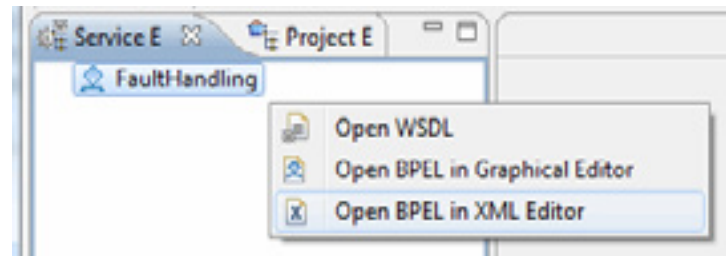
## Enabling a Catch All fault handler

Some web services include service faults and some do not. Service faults are declared in the WSDL file (the file that contains the XML code that defines a web service). For services without them, or that do not have one declared for general problems, you can use a Catch All fault handler. You must first enable the Catch All by adding a snippet of XML code to the WSDL. Before using any XML in a production environment, you should thoroughly test it to ensure it meets your expectations and does not introduce unexpected consequences.

The XML Editor has a design view and a source view. Use the tabs at the bottom of the editor's window to switch between them. You should add this code in the source view.

**IMPORTANT:** If you copy and paste the XML, you may have to copy and paste it into a plain text editor (such as Notepad) first, then copy and paste from the text editor into the XML Editor. Doing this 'cleans' the text of any unwanted (and potentially troublesome) formatting information that can be passed through a clipboard.

1. From the Service Explorer, right-click the top item, **FaultHandling**, and choose **Open BPEL in XML Editor**.



2. At the bottom of the XML Editor, click the **Source** tab.
3. In the XML Editor, locate the first line that starts with `<bpel:import ...`. Click at the beginning of that line and press the Enter key.
4. Type or copy and paste this XML into the space you created (it must be entered exactly as it appears below).
5. Close the XML Editor and save the project.

```
<ext:failureHandling xmlns:ext="http://ode.apache.org/activityRecovery">
  <ext:faultOnFailure>true</ext:faultOnFailure>
</ext:failureHandling>
```



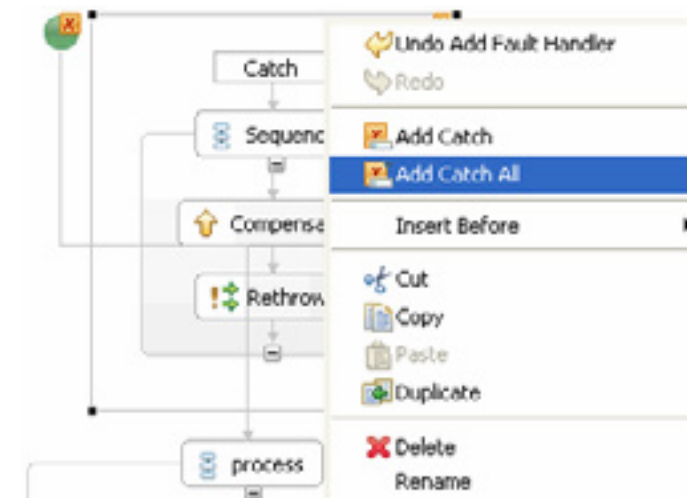
## Adding a Catch All fault handler

It is time to add the Catch All fault handler. You will first add a Catch handler (which you will return to a bit further along in the tutorial), then add a Catch All, then remove the Catch.

1. In the BPEL Editor, right-click the green circle that starts your process diagram and select Add Fault Handler.



2. Right-click the box around the fault handler and select Add Catch All.
3. Delete the original Catch activity and the Sequence beneath it.



4. Save the project.

*A Catch All catches any fault that isn't intercepted by a prior Catch statement.*

*It is best to use a Catch for specific errors whenever possible, so that you can return information about the kind of fault that occurred. This tutorial will provide a few examples of doing that. However, you cannot anticipate every possible fault. For example, you might check the input in various ways, but not consider the possibility that an invoked service is not available. A Catch All is the backup plan--it can tell you **that** something happened, but it cannot tell you **what** happened.*

## Returning a fault message

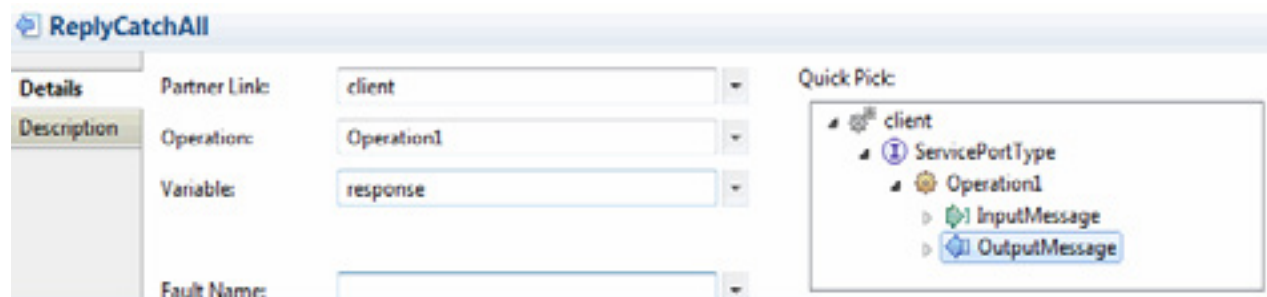
Scope is an important BPEL activity that provides context for other activities, such as variables, faults and events. A scope creates an opportunity to handle these kinds of activities without impacting activities outside it. An analogy in some programming languages would be curly braces, {}. For more information on scope, see the Process Design Studio Help.

The Reply activity allows the Catch All handler to respond with the Output message you copied text into.

If you are creating a fault handler that is NOT at the process level, you must add an Exit activity after each Reply activity within the fault handler.

You need to add a fault message to the Catch All handler to inform the user when a fault occurs. The fault message will be passed to the FaultMessage element you created in the Output variable.

1. In Catch All, delete the Compensate and Rethrow activities.
2. Add an Assign activity to the sequence of the Catch All. Name it **AssignCatchAllFault**.
3. Click **AssignCatchAllFault**. In the Properties view, open the Details tab and click **Add copy rule...** .
4. In the From drop down, select **Fixed Value**. In the text field that appears, type (include the quotes):  
**"An unidentified fault has been caught."**
5. On the To side, expand response:OutputMessage, then payload:Response, and select FaultResponse:string. Click **OK**.
6. Inside the Catch All sequence, after AssignCatchAllFault, insert a Reply activity. Name it **ReplyCatchAll**.
7. Select **ReplyCatchAll**, and in the Properties view select the **Details** tab.
8. In the Quick Pick pane, expand process and select OutputMessage.
9. Unless you are planning on further processing, insert an Exit activity after the ReplyCatchAll and name it **ExitCatchAll**.

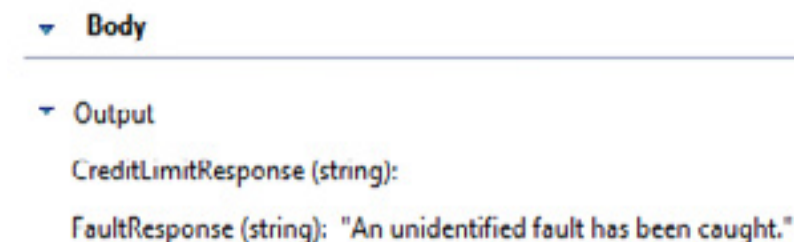


10. Save the project.

## Checking your work

This is a good time to check your work by deploying the service and entering an invalid account number. If the Catch All works as expected, instead of seeing no result, you should see the message you just created, "A fault has been caught."


1. From the File 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. You will see a single Input field. Enter a valid account number, for example, 20004, and click **Go**.
5. If you entered 20004, you should see the message:  
**"These purchases would exceed the account credit limit."**
6. Now try an invalid account number, such as BBB. You should see the message:  
**"An unidentified fault has been caught."**

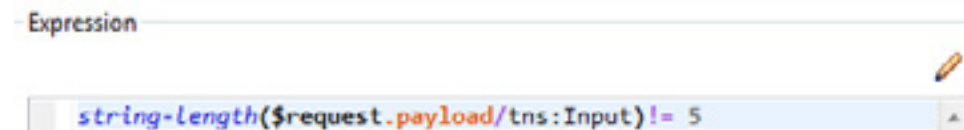


## Validating input

*Some types of input errors, such as format errors, can be identified immediately. It is often more efficient to identify them right away than to rely on an invoked process to do so.*

The most common fault in a process that accepts user input is often the result of invalid input. Since you know invalid input is a potential issue, you will create a Catch statement to provide a level of input validation. If the input is invalid, you can inform the user not only that there has been a fault, but what the fault was (invalid input). For this tutorial, you know that valid account numbers are five characters long. You will add an If activity that tests the input to determine whether it is the right length.

1. In the BPEL Editor, add an If activity between ReceiveInput and AssignInput.
2. In the Properties view of the If activity, open the Details tab and click the pencil  to open the XPath Expression Editor. Delete the default expression, `true()`.
3. In the Functions tree, expand **String**, then double-click **string-length**. This inserts `string-length(string)` in the Expression field with String highlighted.
4. In the Variables tree, expand `request:InputMessage` and `payload:Request`, and then double-click `Input:string` to replace `String` with `$request.payload/tns:Input`. Click at the end of the expression.
5. In the Operators tree, expand **Relational** and double-click **!= (not equal)**.

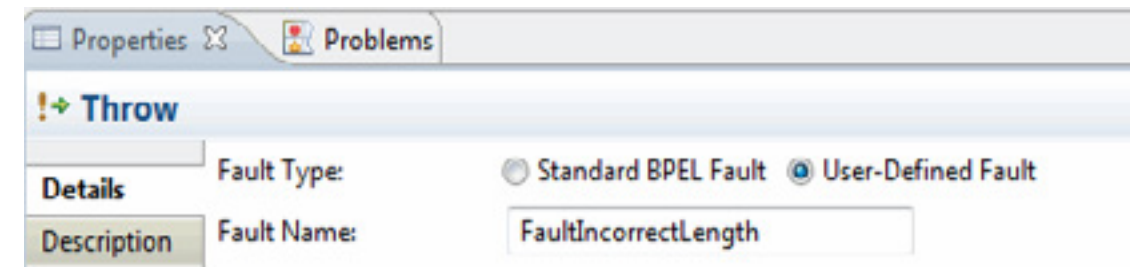


6. Type a space, and then the number **5**. Click **OK**.

## Throwing a user-defined fault

If the input is the wrong number of characters, activities inside the If activity will be called. You will place a Throw activity inside the If. A Throw activity throws a fault, which you will name. You will then create a Catch activity to receive the fault.

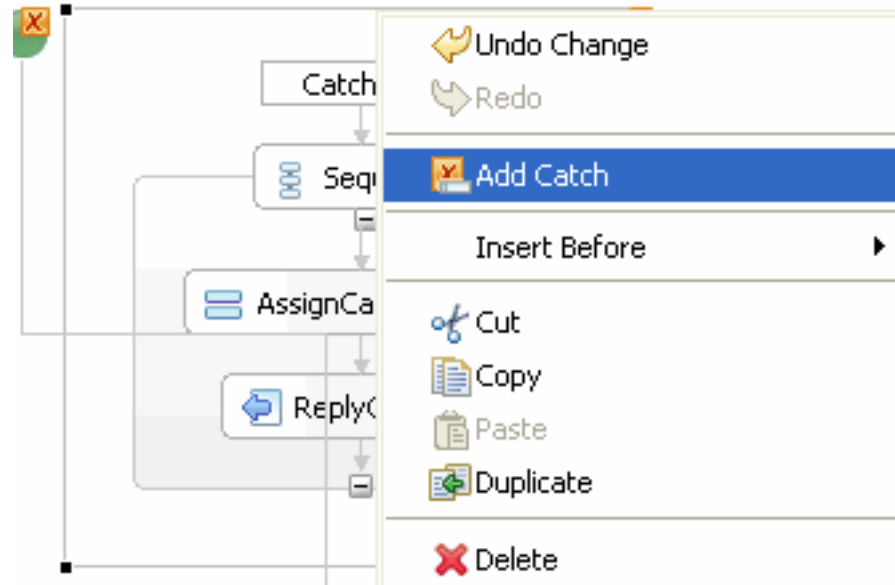
1. From the Palette, under Faults, place a Throw activity inside the If activity you just added.
2. Select **Throw**, and in the Properties view, open the Details tab and select **User-Defined Fault**.



3. Name the fault **FaultIncorrectLength**.
4. Save the project.

## Catching a user-defined fault

1. Right-click the box around Catch All; select **Add Catch**.



*Notice that when you select a Fault Name, the name of the Catch activity changes in the BPEL Editor from Catch to the name you have selected, in this case, FaultIncorrectLength.*

2. Delete the new Compensate and Rethrow activities.
3. Select **Catch**, and in the Properties view, open the Details tab and select **User-Defined Fault**.
4. For Fault Name, select **FaultIncorrectLength**.

Fault Type:  Standard BPEL Fault  Service Fault  User-Defined Fault

Fault Name:

5. Save the project.

## Returning an incorrect length fault message

The steps to return an output message describing the user-defined fault are similar to those for a Catch All message.

1. Add an Assign under FaultIncorrectLength. Name it **AssignIncorrectLengthFault**.
2. Click **AssignIncorrectLengthFault**. In Properties view, open the Details tab and click **Add copy rule...** (+).
3. In the From dropdown, select **Fixed Value**. In the text field that appears, type (include the quotes):  
**"The account number entered is not five characters long."**
4. On the To side, expand `response:OutputMessage`, then `payload:Response`, and select `FaultResponse:string`. Click **OK**.
5. Inside the FaultIncorrectLength sequence, after AssignIncorrectLengthFault, insert a Reply activity. Name it **ReplyIncorrectLength**.
6. Select **ReplyIncorrectLength**, and in the Properties view select the Details tab.
7. In the Quick Pick pane, expand process and select **OutputMessage**.
8. Insert an Exit activity following the Reply. Name it **ExitIncorrectLength**.
9. Save the project.

## Checking your work


It's time to check your work again. This process now has four potential messages, two for a valid account number ('credit limit exceeded' and 'purchase ok'), and two for faults ('...not five characters...' and 'a fault has been caught').

1. From the File 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. Try the following account numbers to see all four potential messages:

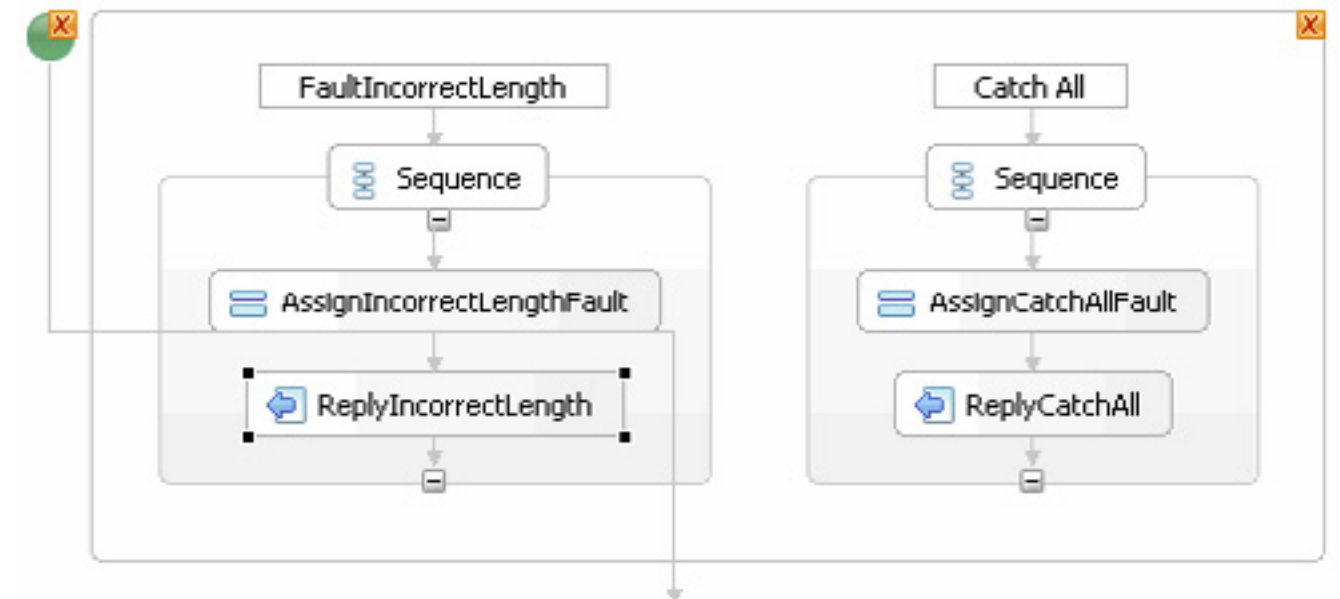
*Feel free to try more account numbers before moving on.*

20000  
20004  
BBB  
20005

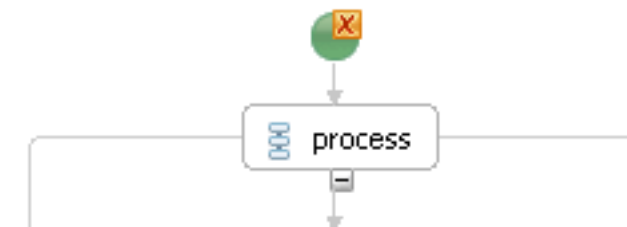
## Hiding fault handling: the orange X

When you create a Catch or Catch All, the green circle at the top of the process flow diagram changes to show an orange X  .

A similar X also appears on the right side of the flow diagrams of faults. Clicking the Xs in the green circle toggles the display to either show or hide the Catch and Catch All flows. In this graphic, the Catch and Catch All that you created are visible:



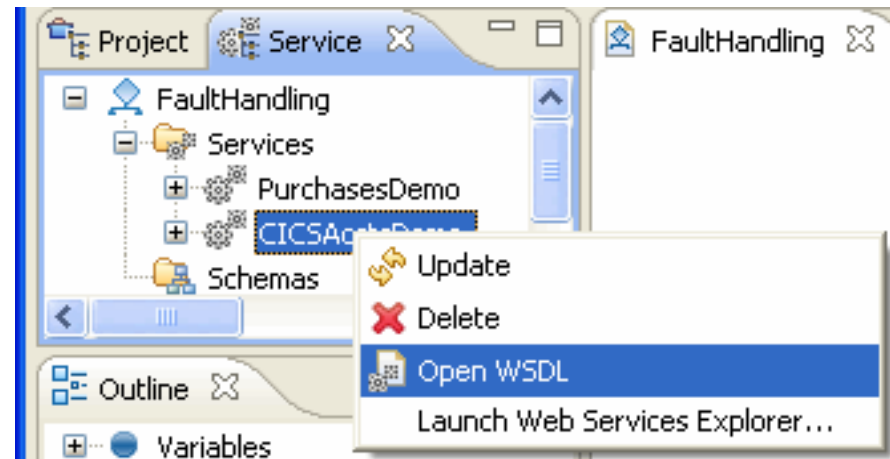
In this graphic, they are hidden. Toggle between the two views by clicking the orange X.



## Finding a service-defined fault

Services you import may also output faults. The WSDL Editor shows the interface to a service. If a service outputs a fault, it is shown in the WSDL Editor.

In the Service Explorer, right-click **CICSAcctsDemo** and select **Open WSDL**.



You can see the fault in the WSDL Editor:

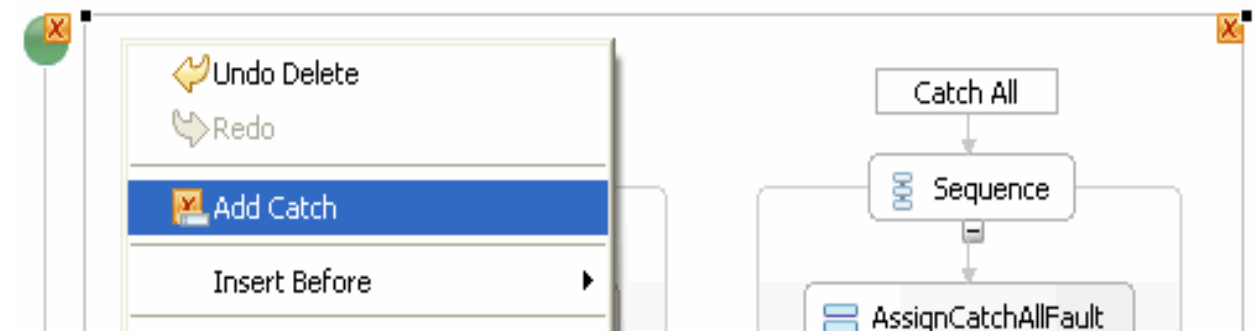
CICSAcctsDemo		
GetAccountDetail		
input	parameters	GetAccountDetail
output	parameters	GetAccountDetailResponse
CICSAcctsDemoException	fault	CICSAcctsDemoException

Of course, now that you know the fault is there, you will want to use it. Hopefully, you can get information about the kinds of faults that are thrown by the services you import. In this case, the CICSAcctsDemo service throws a fault when an invalid account number is input. You can catch any fault in the WSDL and decide how to handle it. Here, you will just pass it to the Output message.

## Catching a service-defined fault

You will start by adding another Catch activity and set it up to catch the fault you are interested in. You will then copy the message you receive to the Output message and finish with a Reply activity.

1. Right-click the box around your processes' fault handlers and select **Add Catch**.



2. Select the **Catch** statement, and in the Properties view, in the Details tab, select **Service Fault**. Faults that are available in the current scope are listed under Fault name. Select **CICSAcctsDemoException**.
3. In the Catch activity **CICSAcctsDemoException**, delete the **Compensate** and **Rethrow** activities.
4. Add an **Assign** activity to **CICSAcctsDemoException** after **Sequence**. Name it **AssignCICSException**.
5. Click **AssignCICSException**. In the Properties view, open the **Details** tab and click **Add copy rule...** ( + ).
6. On the **From** side, expand `CICSAcctsDemoException:CICSAcctsDemoException` then `fault:CICSAcctsDemoException`, and select `message:string`.
7. On the **To** side, expand `response:OutputMessage`, then `payload:Response`, and select `FaultResponse:string`. Click **OK**.
8. After **AssignCICSException**, insert a **Reply** activity. Name it **ReplyCICSException**.
9. Select **ReplyCICSException**, and in the Properties view open the **Details** tab. In the Quick Pick pane, expand **process** and select **OutputMessage**.
10. Insert an **Exit** activity after the **Reply**. Name it **ExitCICSException** and save the project

*The order of Catch activities is often important. Drag Catch activities horizontally to reorder them. Catch All should always be last.*

*In step 6, the error message you are assigning to the output variable is provided by the WSDL. It is part of the fault definition in the WSDL.*



## Deploying and testing

You have completed the fault handling project. The process now has five potential messages, two for a valid account number ('credit limit exceeded' and 'purchase ok'), two for faults you defined ('...not five characters...' and 'a fault has been caught'), and one that should show if the account number is five characters long but invalid.

1. From the File 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 in the left panel of the Web Services Explorer.
5. Try the following account numbers to see four of the potential messages:

*Valid account numbers are:*

*20000 through 20004*

*and*

*20006 through 20008*

20000  
20004  
BBB  
20005

Can you think of a way to make the process return the fifth message -- the Catch All's 'undefined fault' message?

## Reference image of the completed process

