



Tutorial 8: Variable-Length Recordsets

CONTENTS

Setting up and importing a service

Accessing the Schema Editor's index view

Creating a record element

Setting the output type to RecordSetType

Changing the input type

Copying inputs

Adding a temporary variable and a For Each activity

Extracting records from the service data

Adding a new record to a recordset

Deploying and testing

Removing the blank record from the output

Reference image of the completed process

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

In Tutorial 5, you created a process that sent a name to an external service and counted the number of records returned by that service. What if you had wanted to build a recordset to store each name returned as a different record? How do you create a recordset with data returned by a service if you do not know how many records the set will contain? For example, in Tutorial 5, you did not know how many names would be returned. How can you create a recordset with the right number of records? The answer is, create a variable-length recordset that can grow to the size required when the process executes.

The CCSDemo service returns several details about each account found. In this tutorial, you will use a variable-length recordset to create a process similar to the one created in Tutorial 5. Instead of counting the number of records, the process will trim the details received from CCSDemo and return a list of names, account numbers, and finance types.

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.

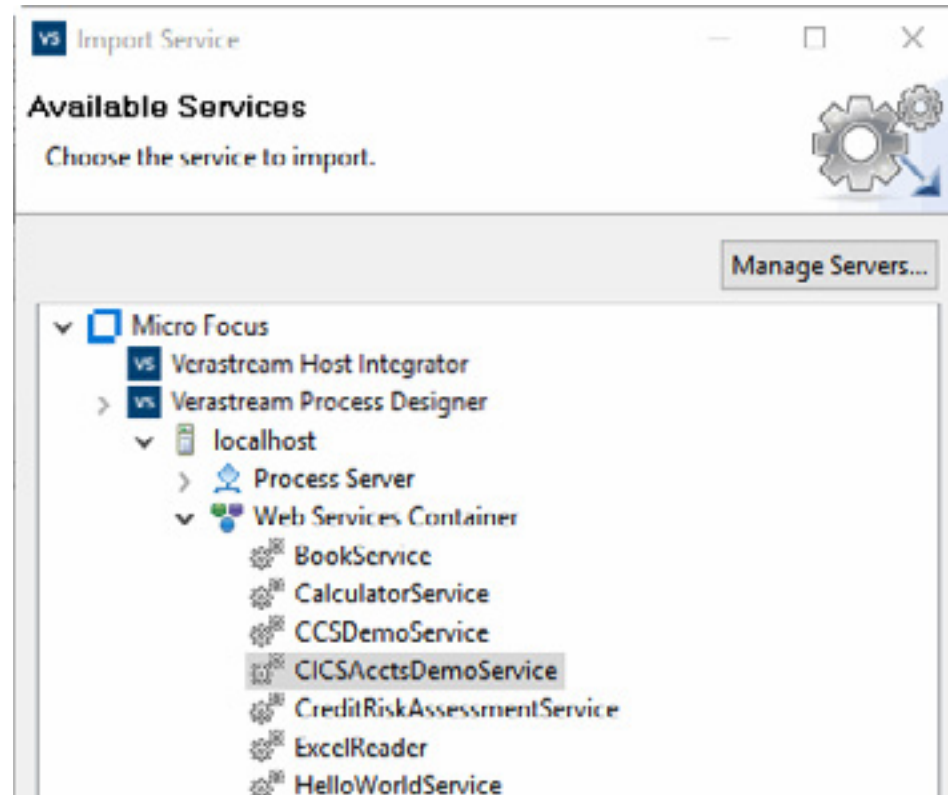
Setting up and importing a service

You can import a service at any time. Select **File > Import Service** to see the **Import Service** dialog, which is dealt with in step 6 on this page.

By default, the process server's name is 'localhost'. If your server has a different name, replace 'localhost' with the name of your server.

You start this tutorial by following the same steps as the setup in Tutorial 5. Create a new project, and then import the CCSDemo service.

1. Start the Process Design Studio (**Start > Micro Focus Verastream > Process Designer > Process Design Studio**).
2. From the File menu, select **New Project**.
3. Name the project **VariableLengthRecordsets** and click **Next**.
4. On the Import Services dialog, click **Add**.
5. Select **Import a registered service** and click **Next**.



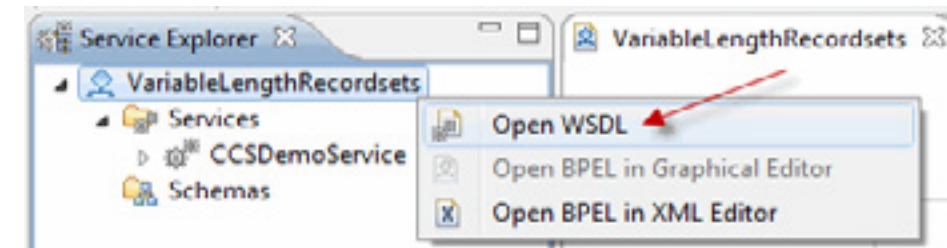
If you do not see the server where the Process Server resides under Micro Focus/Verastream Process Designer, click **Manage Servers** and follow the prompts to add the server to the list.

6. Expand Verastream Process Designer, <server_name> and Web Services Container, and select **CCSDemoService**.
7. On the Name and Confirm dialog, the Name field is pre-populated with the name **CCSDemoService**, click **Finish**, and then **OK**.
8. In the BPEL Editor, delete the DoSomethingHere activity (and the AssignValue it contains) from your project. Save the project.

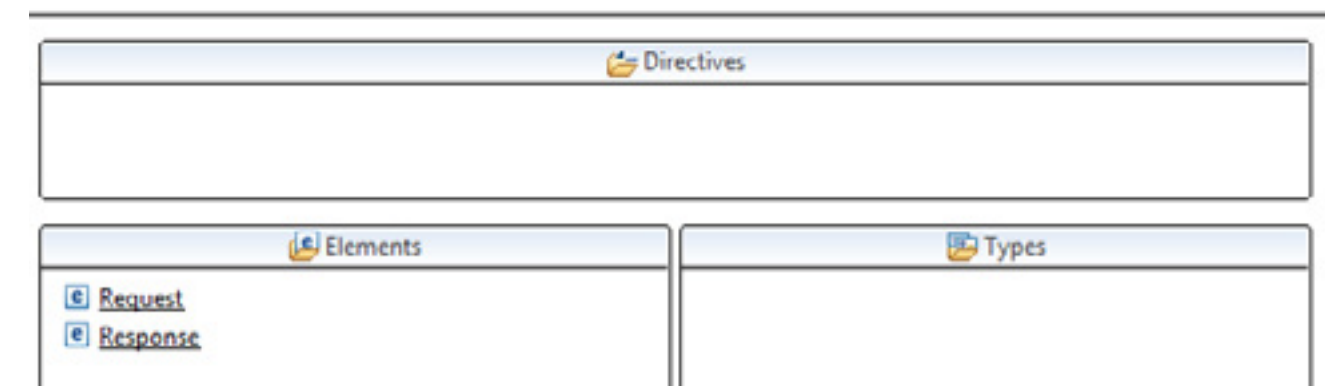
Accessing the Schema Editor's index view

For this tutorial, you will create two new types and one new kind of element. The most convenient place to create new types and elements is the index view of the WSDL Editor.

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



2. The WSDL Editor opens to its default view, which shows Input and Output variables. To only see the Output variable, click on the arrow to the right of Output.
3. To see the index view of the Schema Editor, click the index view icon (the small box in the upper-left corner of the editor).

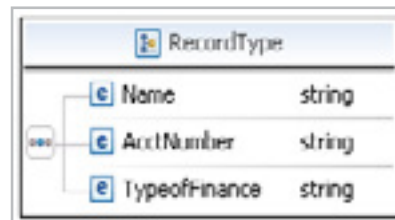



Creating a record element

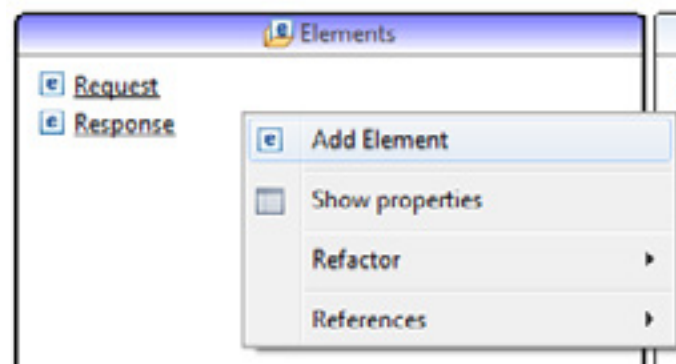
The names used in this tutorial are chosen to indicate the purpose of items in the tutorial. You may want to name types and elements in your project something more descriptive than *RecordType* and *Record*.

A *type* defines a data structure. Complex types can include other types. To do so, you first create an *element* that defines an association with a type. In this tutorial, your output will be a recordset type. To create it, you first define a record type, which describes the structure of one record in your recordset. Then you create a record element, then a recordset type with a reference to the record element.

1. Right-click the Types pane of the index view, and choose **Add Complex Type**. A type named *NewComplexType* is created. Double-click it to isolate it in the Schema Editor.
2. Select **NewComplexType**. In the Properties view, open the **General** tab and change its name to **RecordType**.
3. Right-click **RecordType** and select **Add Element**. Do this three times, so the *RecordType* contains three elements.
4. Change the name of the first element to **Name**, the second to **AcctNumber**, and the third to **TypeofFinance**. (To change an element name, select it, and then on the General tab of the Properties view, change the Name field.)



5. Click the index view icon () to return to index view.
6. Right-click in the Elements pane and select **Add Element**. Name the new element **Record**.

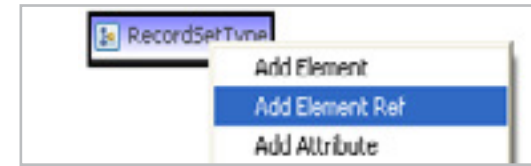


7. Select the **Record** element. In the Properties view, open the General tab, and from the Type menu choose **Browse...**
8. Type **Re** to filter the list, then double-click **RecordType** and save the project.

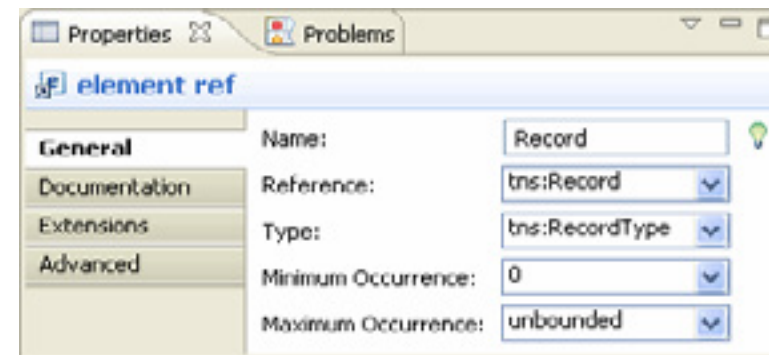
Setting the Output type to RecordSetType


You will now create a *RecordSetType* that refers to your new element 'Record', and set its maximum number of records to 'unbounded.' You can then set the Output to *RecordSetType*.

1. Right-click the Types pane of the index view, and choose **Add Complex Type**. A type named *NewComplexType* is created. Double-click it to isolate it in the Schema Editor.
2. Select **NewComplexType**. In the Properties view, open the **General** tab and change its name to **RecordSetType**.
3. Right-click *RecordSetType* and select **Add Element Ref**.



4. Select the new element reference. In the Properties view, open the **General** tab. In the Reference dropdown, select **Browse...**, and then double-click **Record**.
5. In the General tab, set Minimum Occurrence to **0**, and Maximum Occurrence to **unbounded**.

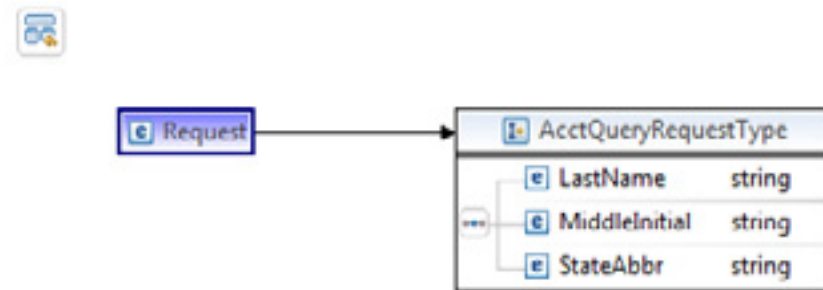


6. Click the index view icon () to return to index view.
7. In the Elements pane, select **Response**, and on the General tab, from the Type menu, select **Browse...**
8. Type **Re** to filter, then double-click on **RecordSetType**.
9. Save the project.

Changing the input type

The process you are using requires a Name, Middle Initial and State in order to return a list of accounts. You must change your input type to accept those three parameters.

1. Return to the Schema Editor's index view. In the Elements pane, select **Request**, and on the General tab, from the Type menu, select **New...**
2. Name the new type **AcctQueryRequestType**, then click **OK**.
3. In the Types pane, double-click **AcctQueryRequestType** to isolate it in the Schema editor.
4. Right-click **AcctQueryRequestType** and select **Add Element**. Do this three times, so that the RecordType contains three elements.
5. Change the name of the first element to **LastName**, the second to **MiddleInitial**, and the third to **StateAbbr**. (To change an element name, select it; open the General tab of the Properties view and change the Name field.)



6. Close the Schema Editor and the WSDL Editor (if it is open).
7. Save the project.

Copying inputs

You have done the hard part. Now you just need to copy inputs to inputs and outputs to outputs, as you have in earlier tutorials.

1. In the BPEL Editor, place an Assign activity between ReceiveInput and ReplyWithOutput. Name it **AssignInputs**.
2. On the Service Explorer, expand **CCSDemoService**, and drag **AccountSearch** into the BPEL Editor to invoke the service. Place it between AssignInputs and ReplyWithOutput.
3. Select **AssignInputs** and create three rules (**+**):

```
From: request:InputMessage > payload:Request > LastName:string
```

```
To: CCSDemo_AccountSearch_Input:AccountSearch > parameters:AccountSearch > LastName:string  
**
```

```
From: request:InputMessage > payload:Request > MiddleInitial:string
```

```
To: CCSDemo_AccountSearch_Input:AccountSearch > parameters:AccountSearch > MiddleInitial:string  
**
```

```
From: request:InputMessage > payload:Request > StateAbbr:string
```

```
To: CCSDemo_AccountSearch_Input:AccountSearch > parameters:AccountSearch > State:string
```

4. Save the project.

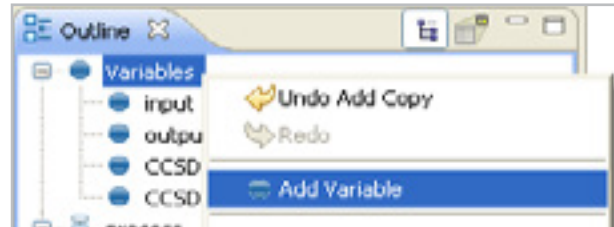
Creating similar copy rules is made easier by the copy button in the Copy Rules dialog box. To 'clone' a copy rule, select it in the Properties tab, click the pencil icon, and in the Copy Rule dialog click the copy button. The 'cloned' copy rule opens with the nodes expanded to the same location as the original copy rule. Change the To and From selection, then click OK to save the new copy rule.




Adding a temporary variable and a For Each activity

The next step is to create a temporary variable to hold individual records as you extract them from the service data. You will copy them, one-by-one, out of the service data and into your output recordset using a For Each activity.


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



2. Name the new variable **tempRecord**.
3. In the Type Selector dialog, type **Re** to filter the list, then double-click **Record**.
4. Place a For Each activity between Invoke_CCSDemo_AccountSearch and ReplyWithOutput.
5. In the For Each Properties view, open the Details tab. Scroll down to Final counter value, and click the pencil icon () to open the XPath Expression Editor.
6. In the Expression field, delete the 1.
7. In the Functions tree, expand Node, double-click **count**.
8. With `item_sequence` highlighted, expand the `CCSDemo_AccountSearch_Output:AccountSearchResponse`, then `parameters:AccountSearchResponse`, then double-click `return:AccountRecord`. **item_sequence** is replaced by `$CCSDemo_AccountSearch_Output.parameters/return[1]`.
9. Delete `[1]`, so the whole expression looks like this:

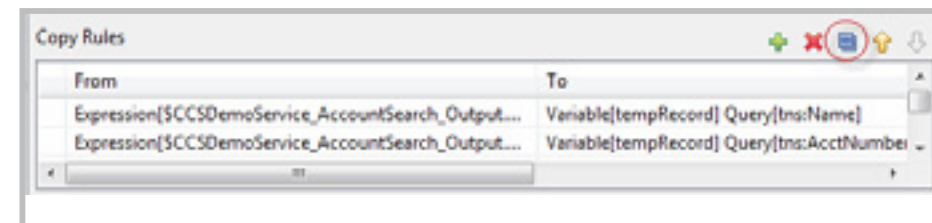
```
count($CCSDemo_AccountSearch_Output.parameters/return)
```
10. Click **OK** and save the project.

Extracting records from the service data

1. Place an Assign activity inside the For Each and name it **AssignAddRecords**.
2. Select **AssignAddRecords** and in the Properties view, open the Details tab and click the copy rule icon ().
3. In the From menu, select Expression then click **XPath Expression Editor...**
4. Expand `CCSDemo_AccountSearch_Output:AccountSearchResponse` then `parameters:AccountSearchResponse` then `return:accountRecord` then double-click `Name:string` to create the expression:

```
$CCSDemo_AccountSearch_Output.parameters/return[1]/Name
```
5. Replace the 1 inside the brackets with `$Counter`. Click **OK** when the expression looks like this:

```
$CCSDemo_AccountSearch_Output.parameters/return[$Counter]/Name
```
6. On the To side of the copy rule, expand `tempRecord:Record` and select `Name:string`. Click **Apply**.
7. To clone the copy rule, select it, and then click the copy button.




8. Replace `Name` with `AcctNumber`. Click **OK** when the expression looks like this:

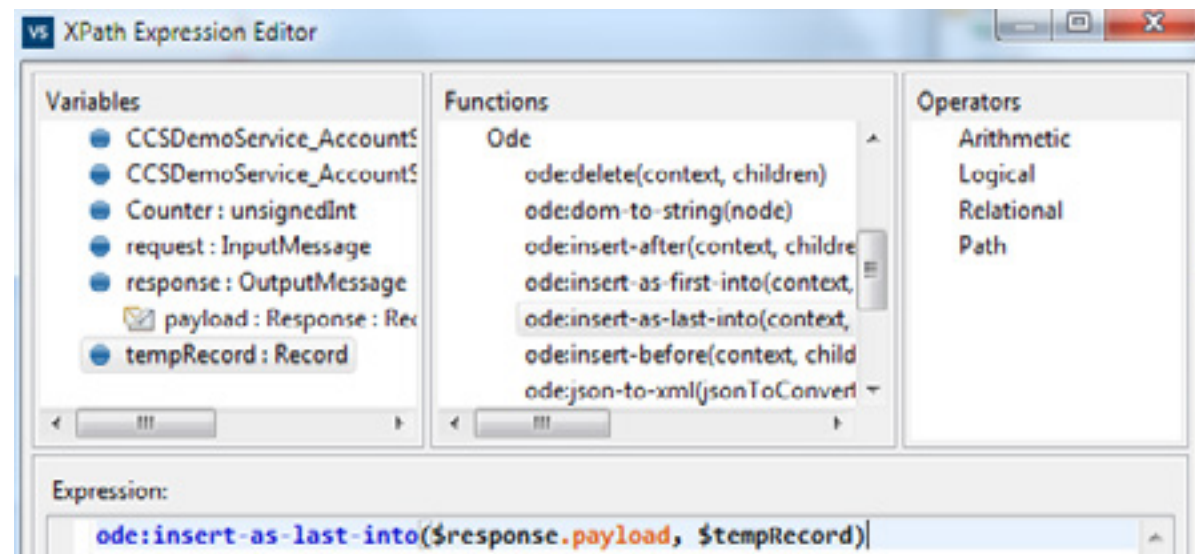
```
$CCSDemo_AccountSearch_Output.parameters/return[$Counter]/AcctNumber
```
9. On the To side of the copy rule, expand `tempRecord:Record` and select `AcctNumber:string`. Click **Apply**.
10. Clone the first rule again. Replace `Name` with `FinanceType`.
11. On the To side of the copy rule, expand `tempRecord:Record` and select `TypeofFinance`. Click **OK**.
12. Save the project.

Adding a new record to a recordset

The Ode extensions were implemented by the Apache Ode team. They help users accomplish tasks that would be very difficult in BPEL without them, including several tasks relating to recordsets. All of the Ode extensions available in the XPath Extension Editor are supported in the Verastream Process Server.

You are ready to create a copy rule that will copy the record you just stored in tempRecord into the output recordset. To accomplish this, you will use the Ode XPath extension, ode-insert-as-last-into. This extension adds a record as the last item in a recordset. Its *context* parameter takes a recordset, and its *children* parameter takes a record to be added.

1. Select **AssignAddRecords** and in the Properties view, open the Details tab and click the copy rule icon ().
2. In the From menu, select **Expression** then click **XPath Expression Editor...**
3. In the Functions tree, expand **Ode**, and double-click `ode:insert-as-last-into`.
4. Select the **context** parameter. In the Variables tree, expand `response:OutputMessage`, and double-click `payload:Response:RecordsetType`.
5. Select the **children** parameter. In the Variables tree, double-click `tempRecord:Record`. Click **OK**.



6. On the To side of the copy rule, expand `response:OutputMessage` and select `payload:Response:RecordsetType`.
7. Click **OK**.
8. Save the project.

Deploying and testing

It's time to deploy and test your new process. The last page of this tutorial is a reference graphic showing the completed process in the BPEL Editor.

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. You will see three inputs: LastName, MiddleInitial, and State.
6. Enter the following values:
For LastName input **Smith**
for MiddleInitial input **C**
for StateAbbr input **RI**
then press **Go**


The output will be a list of records comprised of a name, an account number and a type of finance. There should be fourteen names -- the same number that the process in Tutorial 5 counted.

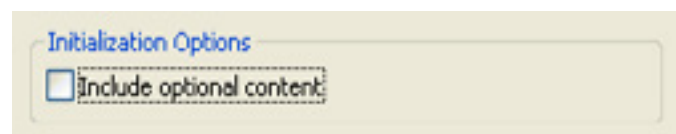
Removing the blank record from the output

If you followed the steps in this tutorial closely, the first record returned in the test result should be blank. This is because when the Output variable was initialized, a record containing blank data was created and added to the dataset. When the `ode:insert-as-last-into` function added records, they were added to the end of the recordset, after the record containing blank data.

It is considered a best practice to leave the 'Include optional content' checkbox marked. Only uncheck it when you have a reason to, such as in this case.

To remove the blank first record from the results, the Output variable's initialization rule needs to be revised. The revised initialization rule will initialize the variable without putting blank data in its elements. You will accomplish this by turning off the 'Include optional content' checkbox in the second Generate XML for Variable dialog box.

1. In the Outline view, expand Variables, and select Output.
2. In the Properties view, Initialization tab, click to create a new initialization ().
3. Clear the **Include optional content** checkbox. Notice that the `<tns:Record>` structure is removed from the initialization XML. Click **OK**.



4. Save the project.
5. Repeat the steps for deploying and testing. The blank record at the beginning of the results should now be gone.

Congratulations! You have completed the last tutorial. See the Process Design Studio help for more information about the topics covered.

Reference image of the completed process

