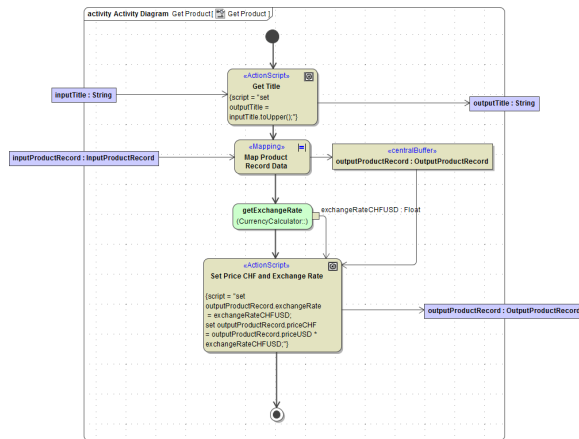


# Activities Lesson 2 MD18



The next step in the development process is to model the implementation of the operation **getProduct**. At the end of this development step, your activity diagram will look like the following:



Components

## On this Page:

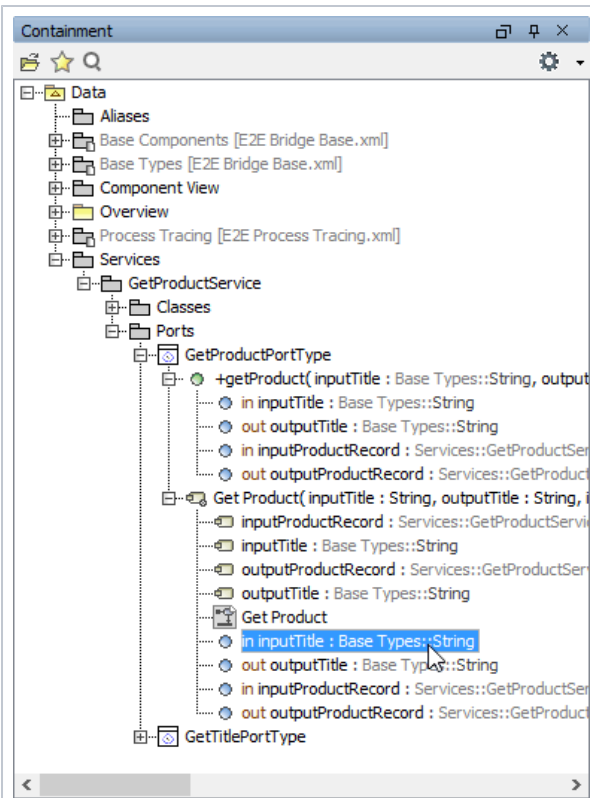
- Adding Parameters to the Activity Diagram
- Copying UML Elements
- Mapping Input to Output
- Using Call Operation Actions
- Using Action Script
  - Completing the Object and the Control Flow
  - Processing the Data Using Action Script
- Implementing the Class Operation

## Adding Parameters to the Activity Diagram

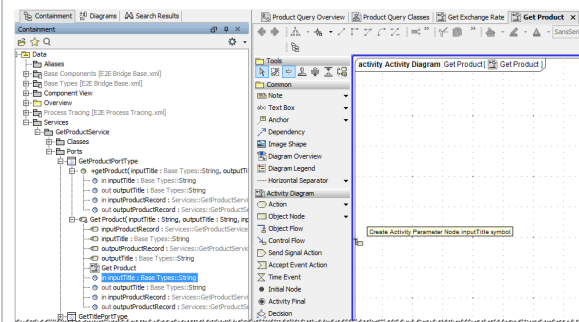
The activity diagram **Get Product** will contain actions to define an exchange rate and to receive and process input data.

Objects represent object flow through activities and can change their state. Objects may originate outside of the context of an activity diagram – in other words, they are input parameters to the activity diagram. On the other side, activity diagrams may produce objects that are used as output. They can be passed as output parameters to an outside context, which can be a calling activity diagram or a port type operation.

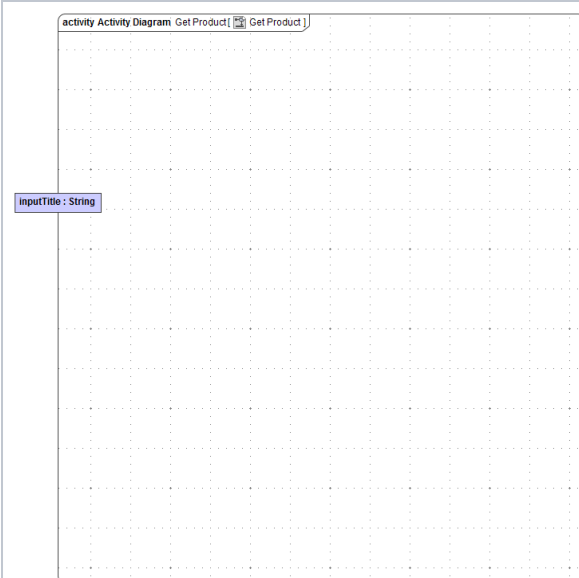
Previously, you defined the operation **getProduct**, its parameters, and the implementing activity diagram. Now, you will add the two parameters to the activity diagram in the diagram pane to model the object flow. Remember that each activity parameter node works as a container of a parameter. They have a different color than normal object nodes that are drawn in the activity diagram and indicate that these nodes contain activity parameters. In order to visualize input and output parameters, you can place them on the left or right side of the diagram frame. The left side corresponds to input, the right side to output.



Select the parameter **inputTitle** in the containment tree.



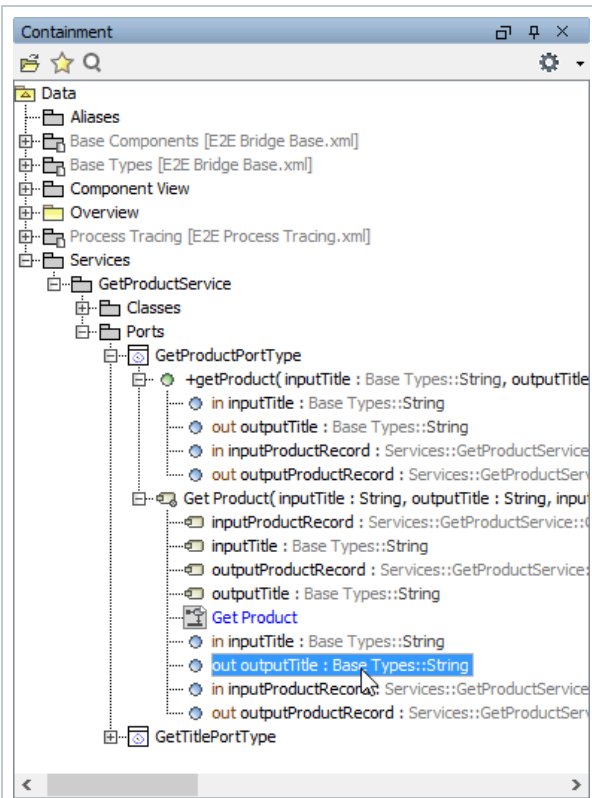
As it is an input parameter, drag the selected parameter to the left border of the diagram frame until the diagram frame gets surrounded by a blue rectangle.



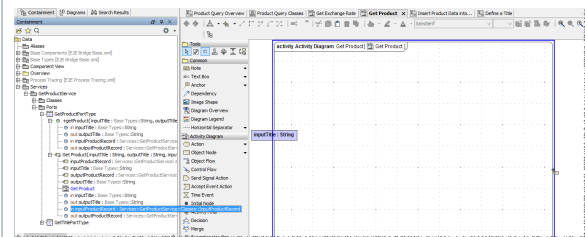
Release the mouse button.

The input activity parameter node has been drawn in the activity diagram and is displaying the name of the **parameter** it is containing. On the right side of the colon, the type of parameter (**String**) is displayed.

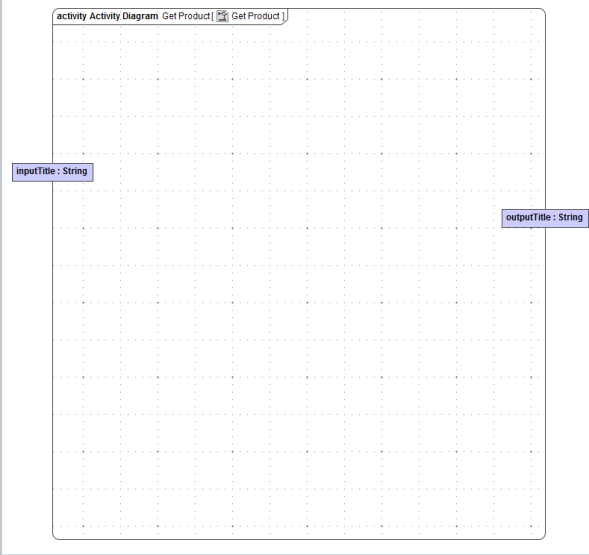
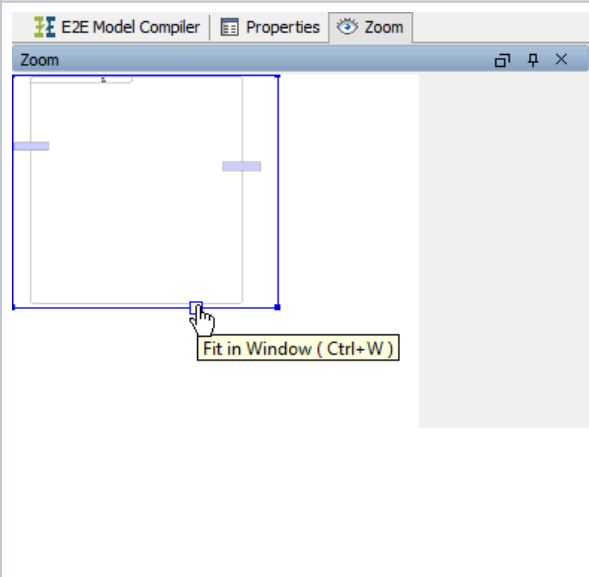
Although you were dragging the **parameter** and not the **activity parameter node** from the containment tree, MagicDraw placed the activity parameter node on the activity diagram.



Now, add the output parameter to the activity diagram. Select the parameter **out putTitle** in the containment tree.



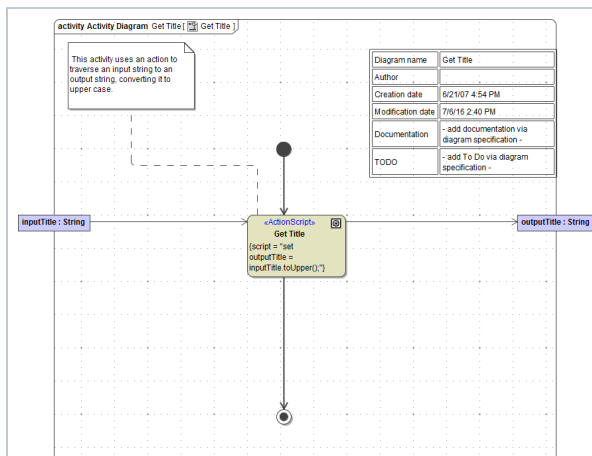
As it is an output parameter, drag the selected parameter to the right border of the diagram frame until the diagram frame gets surrounded by a blue rectangle.

	<p>Release the mouse button. The output activity parameter node has been drawn in the activity diagram and is displaying the name of the <b>parameter</b> it is containing.</p>
	<p>In the lower left corner of MagicDraw, next to the E2E Model Compiler, you can find the <b>Z</b>oom tab. The blue frame marks the section of the diagram you are currently viewing in the diagram pane. You can shrink or enlarge the section by dragging the little blue handles in the corners of the blue frame.</p>

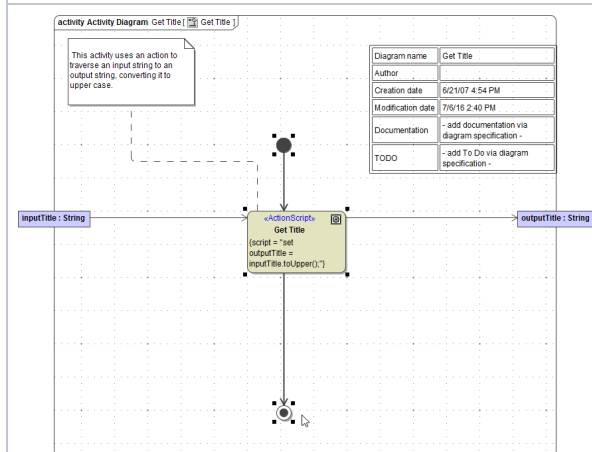
In order to get a better overview, click the white rectangle to automatically resize the diagram section. Now, the whole diagram fits into the diagram pane.

## Copying UML Elements

Before implementing the new functionality, you will copy some UML symbols from the port type operation diagram **Get Title** to the new port type operation diagram **Get Product**.

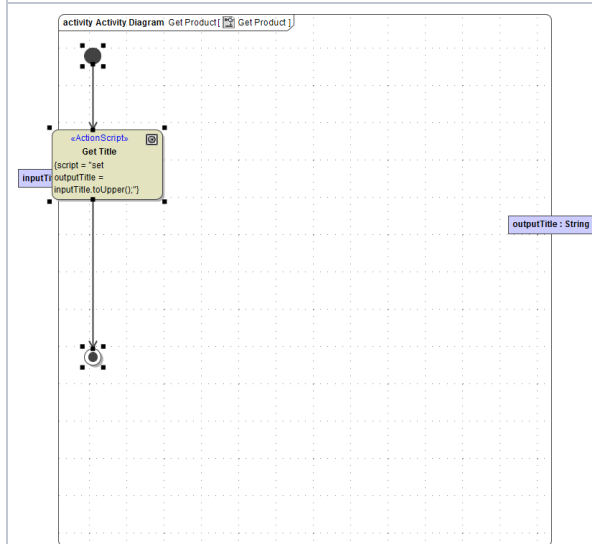


In the containment tree, navigate to the activity diagram **Get Title**, which has been created in lesson 1. Double-click it to open it in the diagram pane.



Select the items as shown in the picture on the left while holding down the **Shift** key: the initial node, the action node **Get Title**, and the final node.

Press **Ctrl - C** on your keyboard to copy them to the clipboard.



Switch back to the activity diagram **Get Product**.

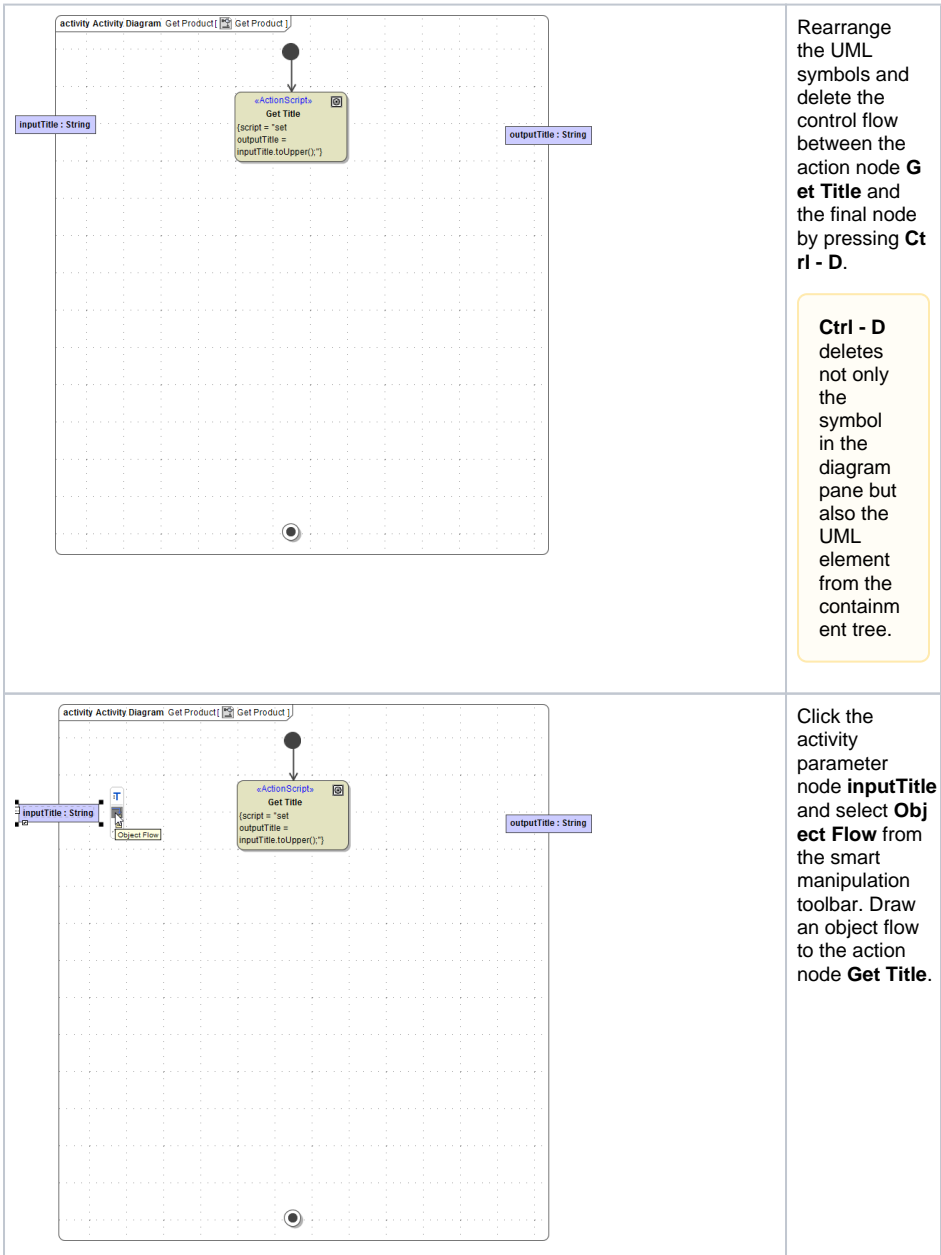
Paste the copied items with **Ctrl - V**. Note that MagicDraw copied the control flows, too, although they were not selected.

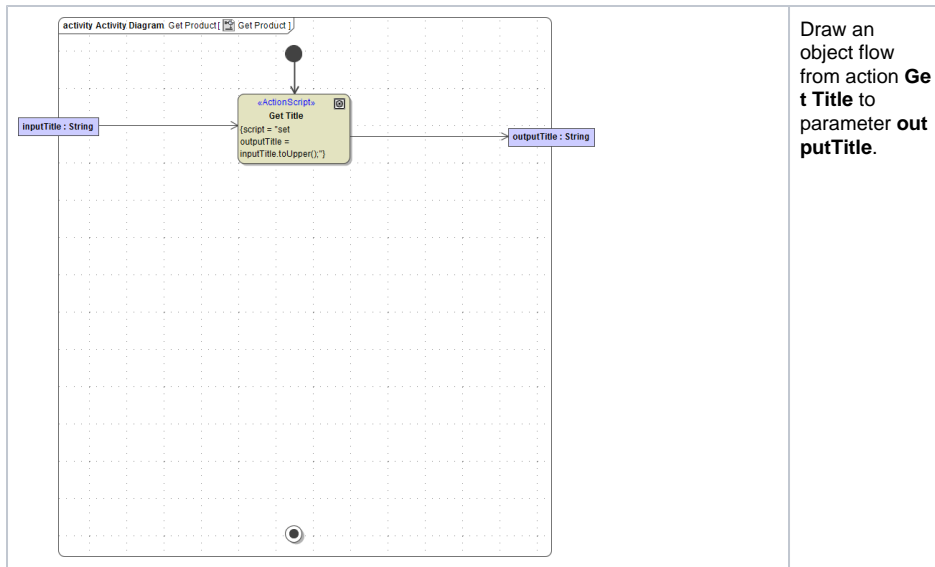
Copying UML symbols **from one diagram to another** results in duplicate elements in the containment tree with the same name. The duplicated elements exist twice, in the activity tree of the source diagram as well as in the activity tree of the target diagram.

Copying UML symbols **within the same diagram** only results in copying the graphical element - the symbols. The same symbol appears twice in the diagram pane, but the UML element in the containment tree only once. Symbols are only representations of the UML elements in the containment tree.


Deleting the UML element in the containment tree would remove all symbol representations in the diagram pane.

In order to create a duplicate UML element **within the same diagram**, paste the copied symbol with **Edit - Paste with New Data** or **Ctrl - E**. In this case, MagicDraw will not only create a new symbol with a new name but also a new element in the containment tree.

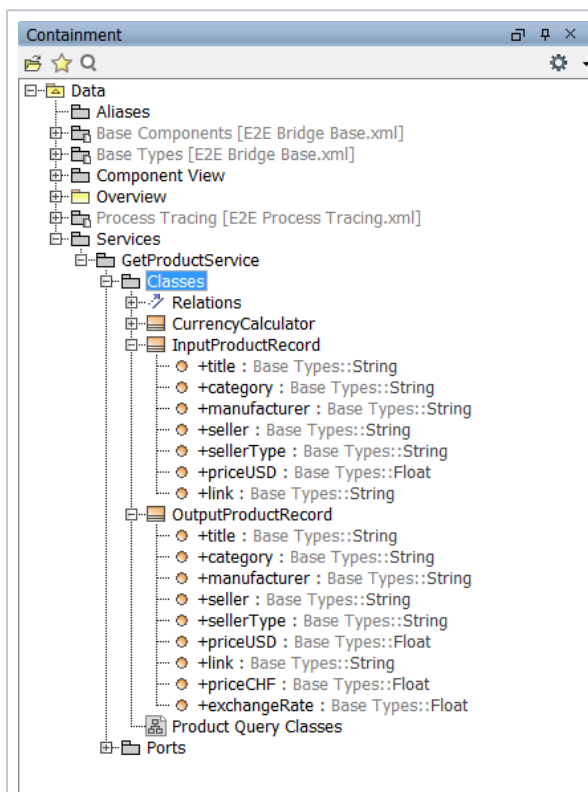




Draw an object flow from action **Get Title** to parameter **outputTitle**.

Save  the UML model.

## Mapping Input to Output

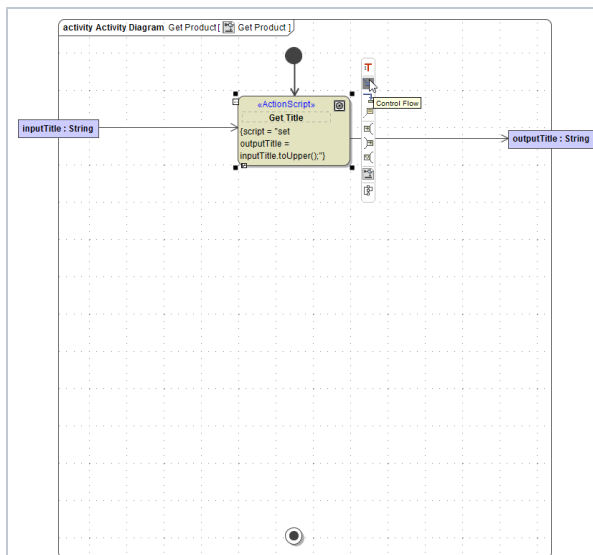


At the beginning of the modeling phase of this Web service, you defined the classes for the input and output data structures.

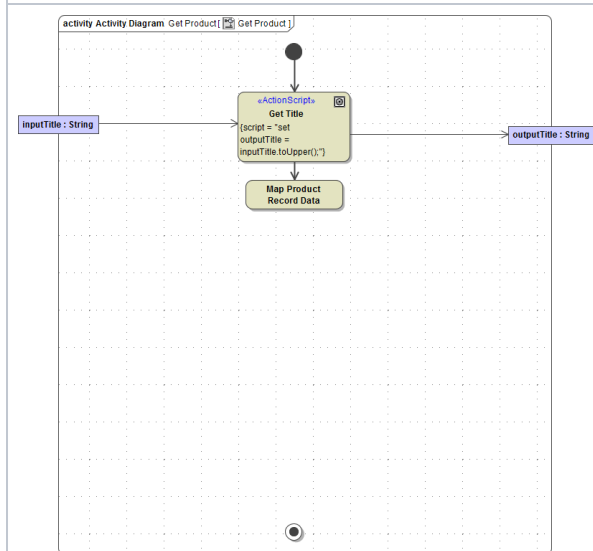
Remember that by defining dependencies between the attributes of the input and output class, the input data can easily be mapped to the output data.

Now, you will initiate this mapping by using the **<<Mapping>>** stereotype in an action. This action then will internally translate the declarations defined in class diagrams into a set of action script statements that actually execute the mapping. Later on in this lesson when tracing the compiled service with the E2E Analyzer, you can see this in more detail.

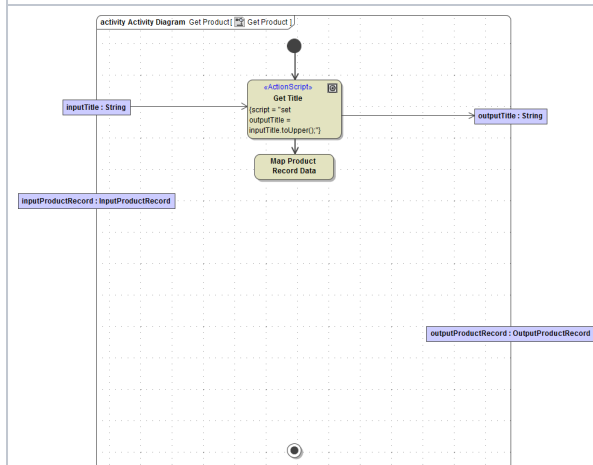
The input and output objects will be added to the diagram as parameters.



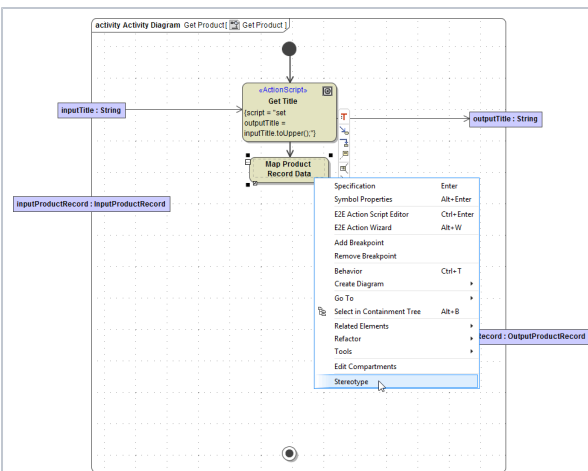
Select the **Control Flow** icon from the smart manipulation toolbar and draw an action node below **Get Title**. MagicDraw draws both the control flow and the action node.



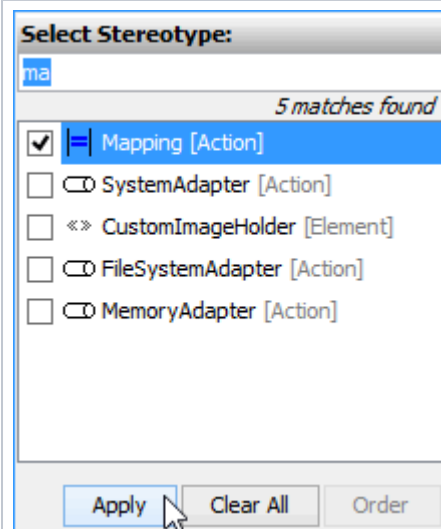
Start directly typing the action node name **Map Product Record Data**. Finish the entry with **Enter**.



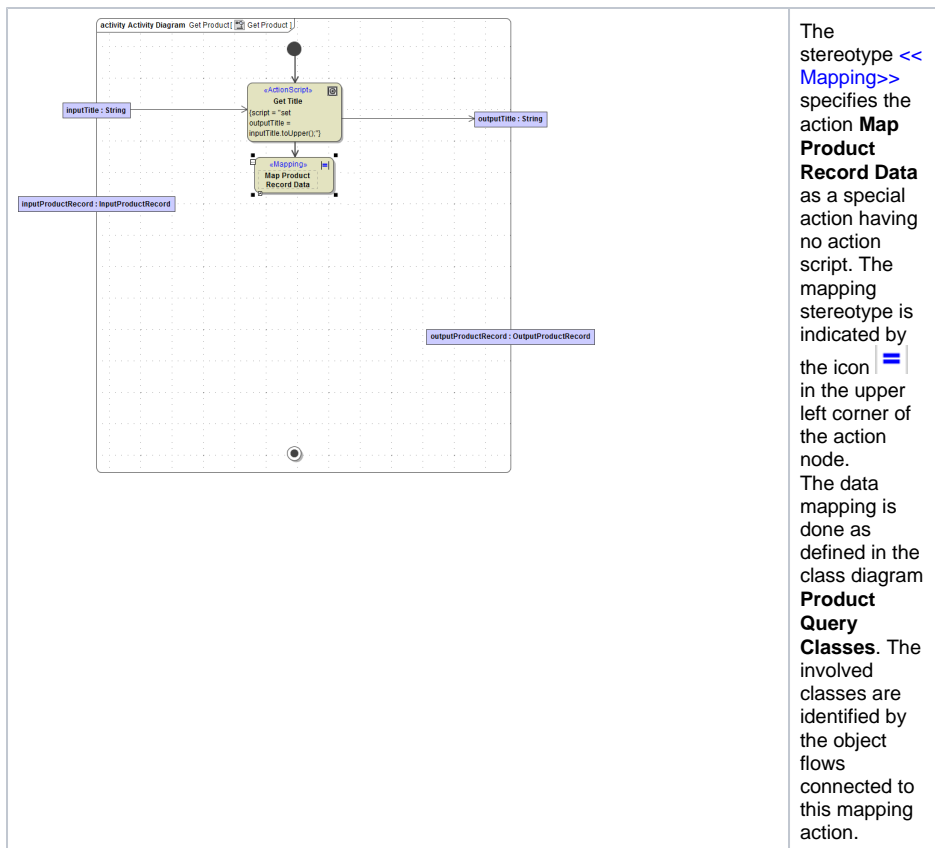
Add the parameters **inputProductRecord** and **outputProductRecord** to the diagram. Select them in the containment tree (below the activity **Get Product**) and drag them to the frame borders of the diagram pane.




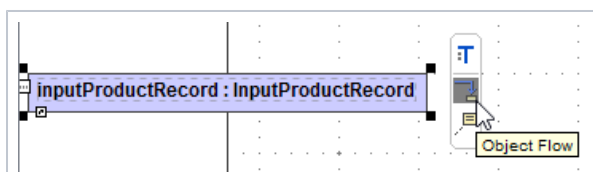
Select the action node **Map Product Record Data** with the right mouse button and choose **Stereotype**.



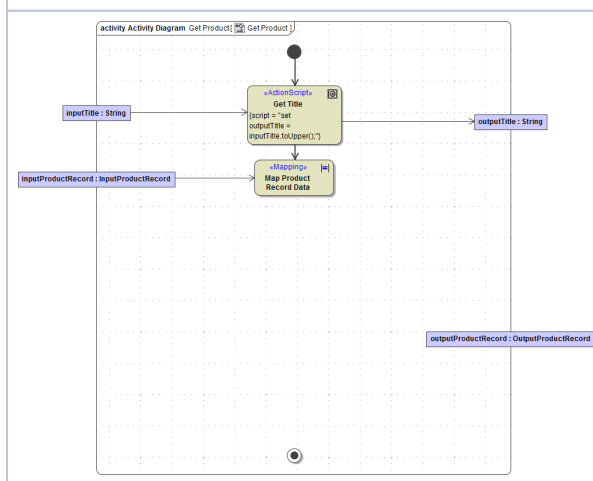
Start typing **ma** to filter the list. Select **Mapping** and click **Apply**.



The stereotype **<<Mapping>>** specifies the action **Map Product Record Data** as a special action having no action script. The mapping stereotype is indicated by the icon  in the upper left corner of the action node. The data mapping is done as defined in the class diagram **Product Query Classes**. The involved classes are identified by the object flows connected to this mapping action.

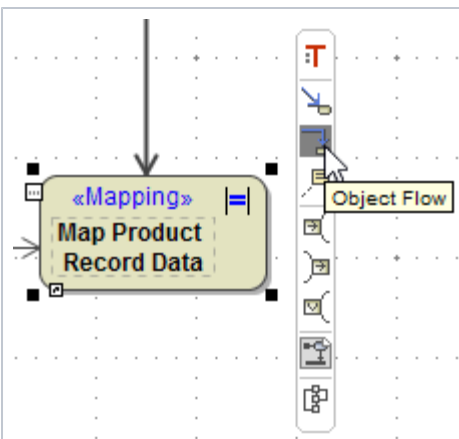


Click the parameter **inputProductRecord** and select **Object Flow** from the smart manipulation toolbar.

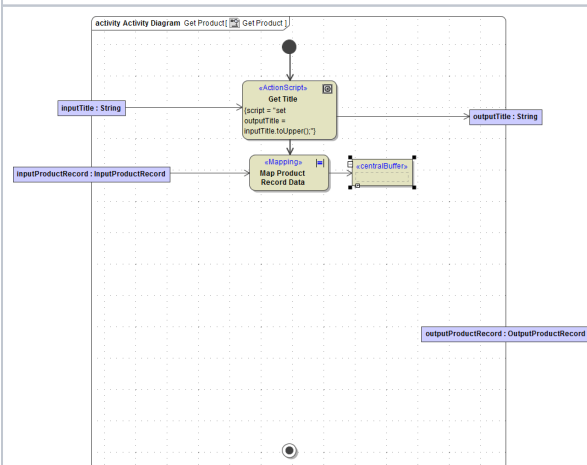


Draw an object flow from parameter **inputProductRecord** to action **Map Product Record Data**.

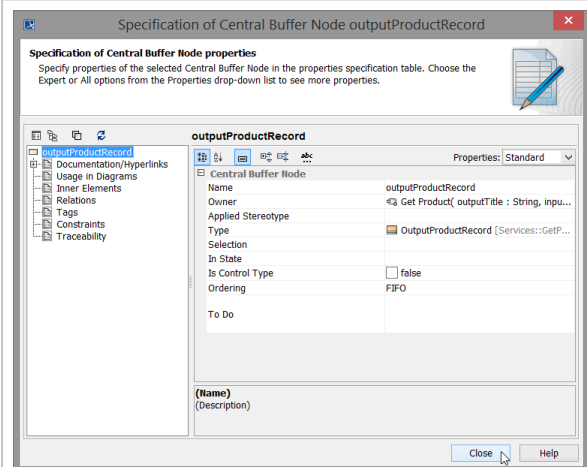
As the currency conversion is not done yet, the output data of the mapping action will not flow directly to the output parameter **outputProductRecord**, but will be temporarily stored in a **Central Buffer Node**. This UML symbol is used to store information only within the activity diagram. The information is not passed outside the context of the activity diagram.



Click the action **Map Product Record Data** and select **Object Flow** from the smart manipulation toolbar.



Click the diagram pane next to the action node **Map Product Record Data** to create a central buffer node. Action node and central buffer node will be connected by an object flow automatically.



Double-click the object to open the specification dialog. Assign the name **outputProductRecord** and select the type **OutputProductRecord** from the type list. Remember that the list can be filtered by typing the initial characters in the field **Type**.

Note, that the object has the same name and type as the parameter.

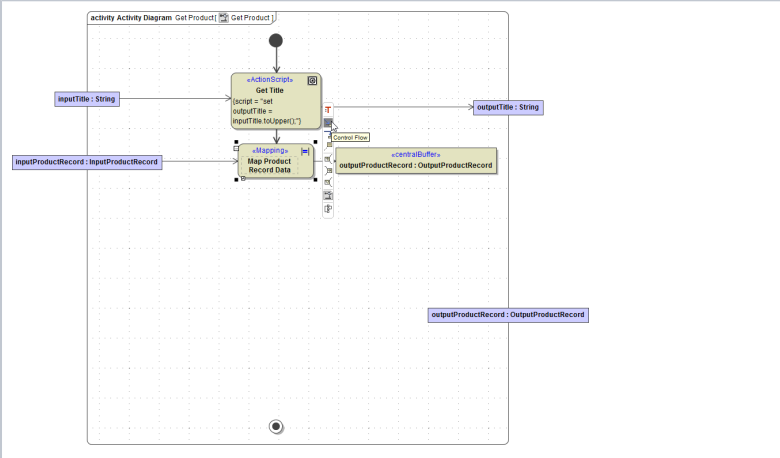
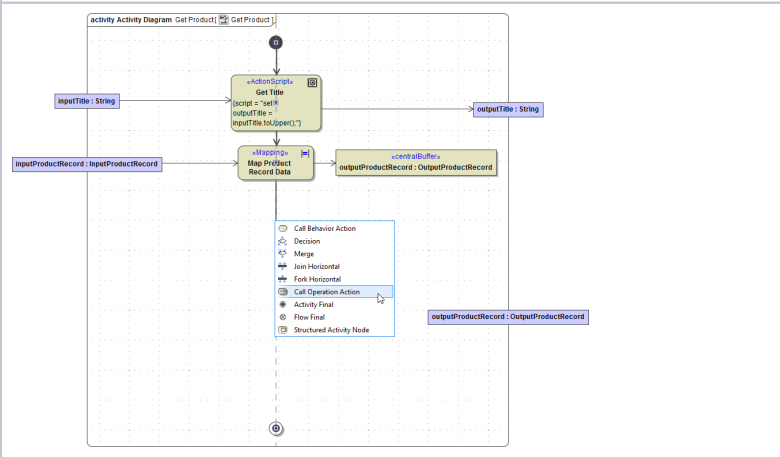
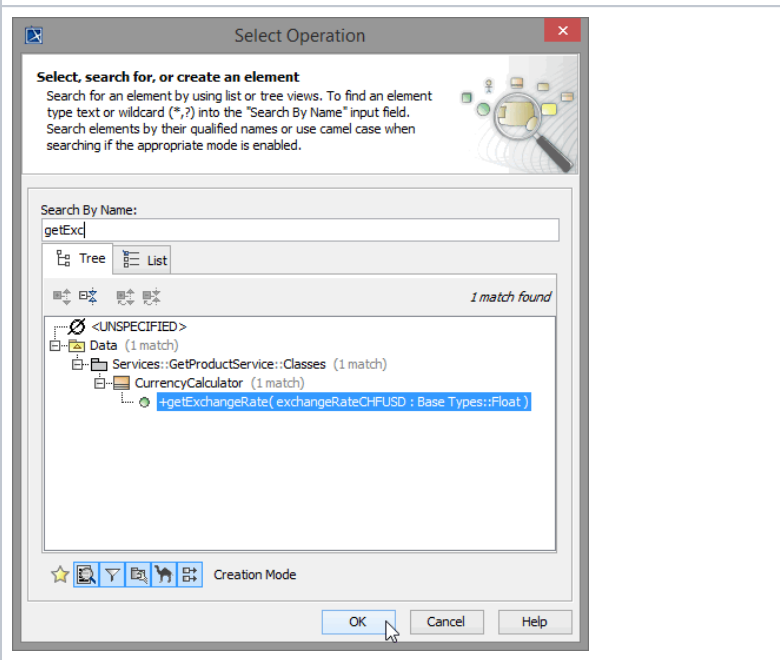
Click **Close**.

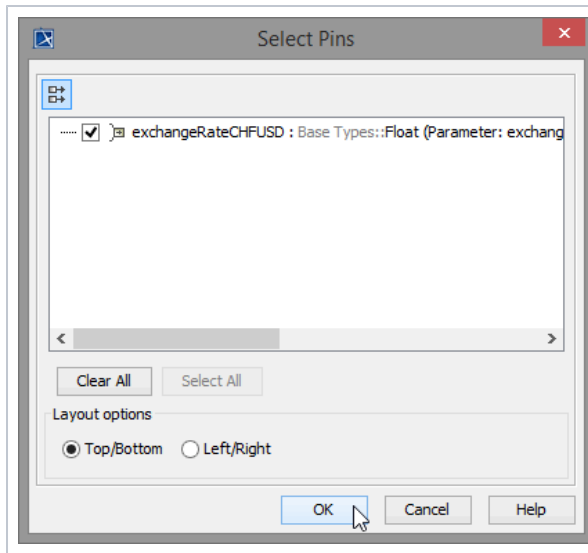
While the central buffer node is used for the object flow within the context of the activity diagram, the activity parameter node is used to pass the data to the outside context. The object flow only leads to the parameter node when the object **outputProductRecord** does not change its state anymore. Here, this will be right after the currency calculation that is implemented further below.

Save  the UML model.

# Using Call Operation Actions

In the next step, you will call a class operation in order to get the exchange rate for CHF / USD. The operation **getExchangeRate** is a member of class **CurrencyCalculator**. The operation returns a parameter called **exchangeRateCHFUSD** being of type **Float**. In UML, operations are called using **Call Operation Actions**.

	<p>Click the action node <b>Map Product Record Data</b> and select <b>Control Flow</b> from the smart manipulation toolbar.</p>
	<p>Move the mouse cursor down, but this time click the right mouse button. A list of possible UML symbols appears.</p> <p>Choose <b>Call Operation Action</b>.</p>
	<p>In the <b>Select Operation</b> dialog, filter the list by typing <b>getExc</b> and select the operation <b>getExchangeRate</b>.</p> <p>Click <b>OK</b>.</p>



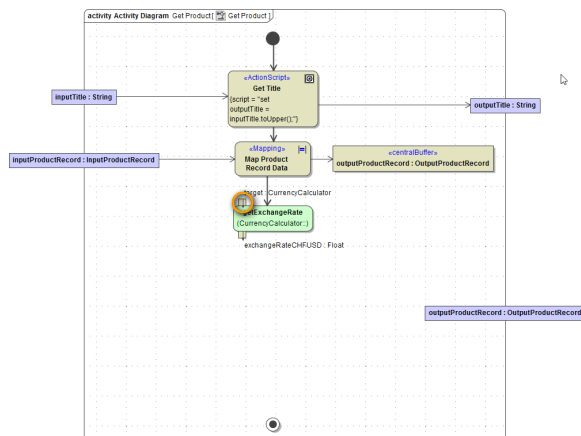
Call operation actions and call behavior actions may have input and output pins. An input pin is a pin that holds input values to be consumed by an action. An output pin is a pin that holds output values produced by an action.

In the **Select Pins** dialog, click **OK**.

The call operation action node is displayed in a different color than normal actions. It visualizes an action calling an operation. The name of the called operation is displayed in the call operation action node. As the operation **getExchangeRate** has the output parameter **exchangeRateCHFUSD**, an output pin is docked to the action node. Name and type of the object is displayed next to the pin. By default, a target pin is created automatically, too (see orange mark in the picture below).

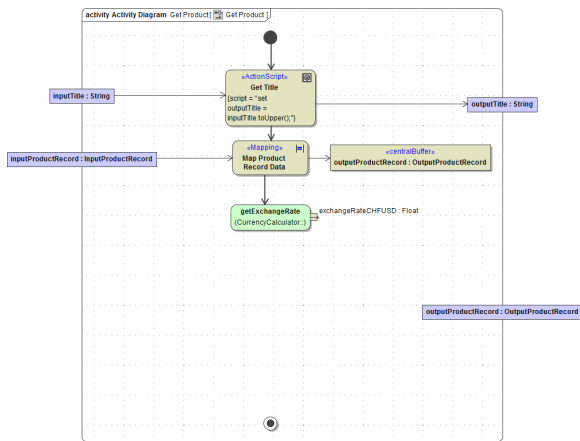
For static operations, target pins are not necessary and must be deleted. The target pins are only needed for non-static operations.


Delete the target pin. Remember to use **Ctrl - D** to delete the element and not only its symbol.



An output pin holds output values produced by an action. The value can be delivered to other actions via object flows. At the same time, objects can be transformed to other objects having a different name using pins. This increases modeling speed and helps avoiding modeling errors. It would be equivalent to use a central buffer node instead of a pin, however, it is best practice to use pins in this case.

As it is an output pin and in order to visualize the information flow, we recommend dragging the output pin to the right border of the call operation action node.



Save  the UML model.

## Using Action Script

In the next (and last) action, you will use the float **exchangeRateCHFUSD**, which was received from operation **getExchangeRate**, in order to convert the price in USD to the price in CHF. The price in USD was taken from the input object (**inputProductRecord**) and has already been mapped to the object **outputProductRecord**. In the next step, you will update this object with the calculated price in CHF and the exchange rate, before the object is returned to the calling client.

## Completing the Object and the Control Flow

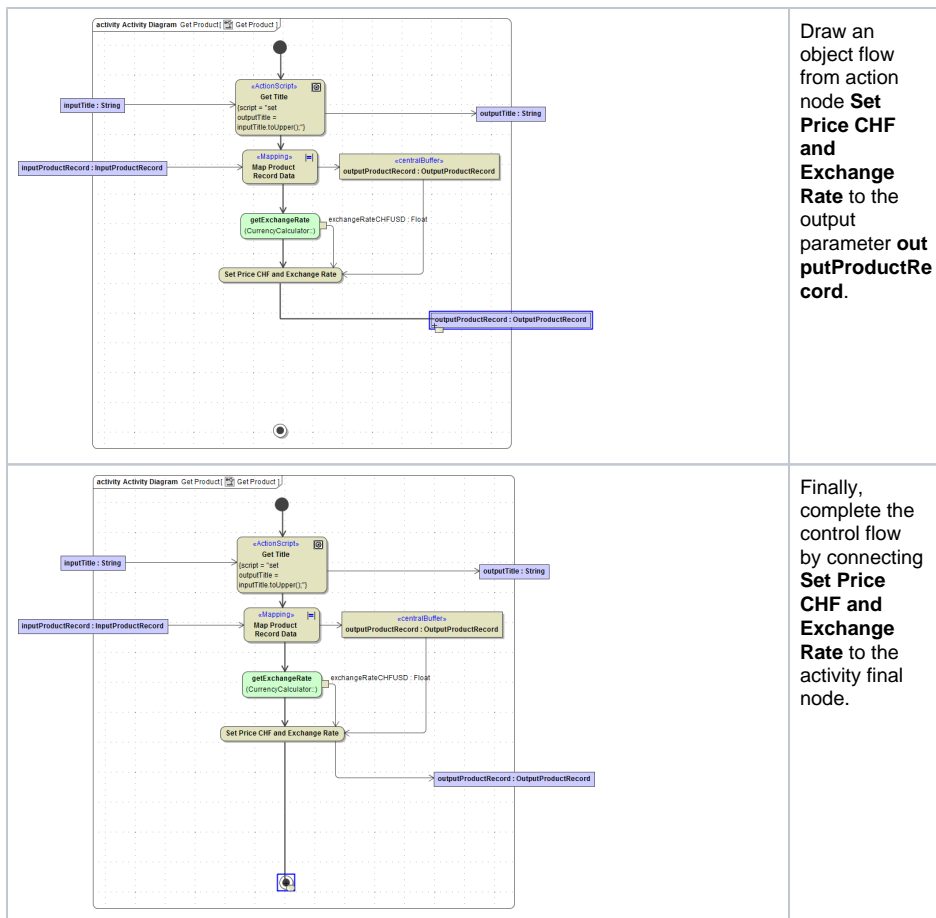
Before editing the action script of the next action, you will first complete the control and the object flow. Having completed the object flow you get full editing support of the Action Script Editor.

	<p>Select the call operation <b>getExchangeRate</b> and choose <b>Control Flow</b> from the smart manipulation toolbar.</p>
	<p>Draw a new action node and immediately start typing its name <b>Set Price CHF and Exchange Rate</b>.</p>

The new action **Set Price CHF and Exchange Rate** needs some input data: the exchange rate and the output product record data, which has to be updated.

	<p>Select the output pin <b>exchangeRateCHFUSD</b> and choose <b>Object Flow</b> from the smart manipulation toolbar.</p>
	<p>Draw a object flow from the output pin to the new action node <b>Set Price CHF and Exchange Rate</b>.</p>
	<p>Now, draw an object flow from the central buffer node <b>outputProductRecord</b> (to which the input product record data had been mapped) to the new action node as well. This is required, as the action needs the reference to this object in order access and update its attributes.</p>

Now, the processed data must be passed to the output parameter.

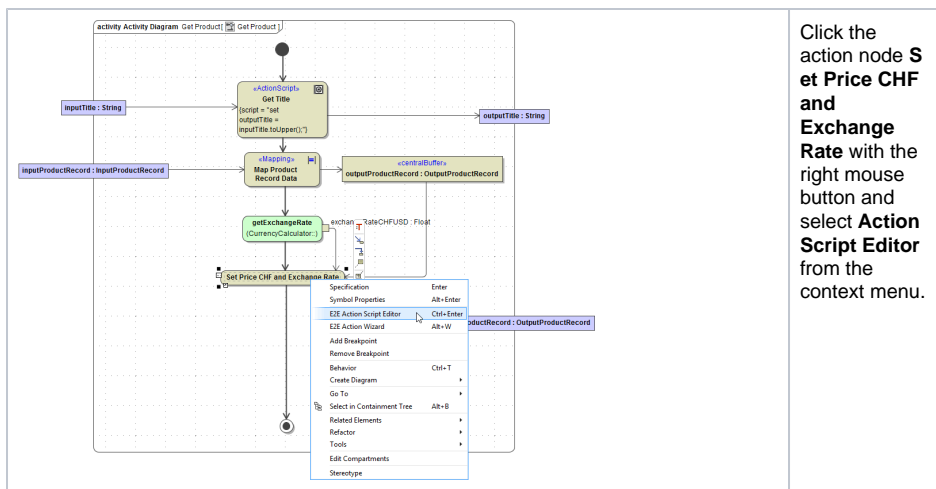


Draw an object flow from action node **Set Price CHF and Exchange Rate** to the output parameter **outputProductRecord**.

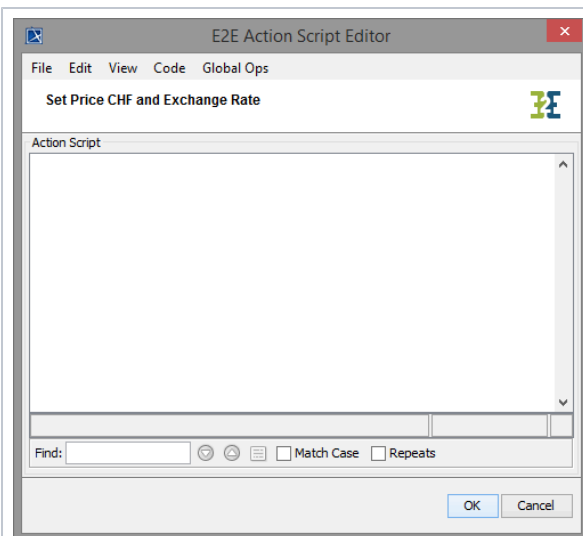
Finally, complete the control flow by connecting **Set Price CHF and Exchange Rate** to the activity final node.

## Processing the Data Using Action Script

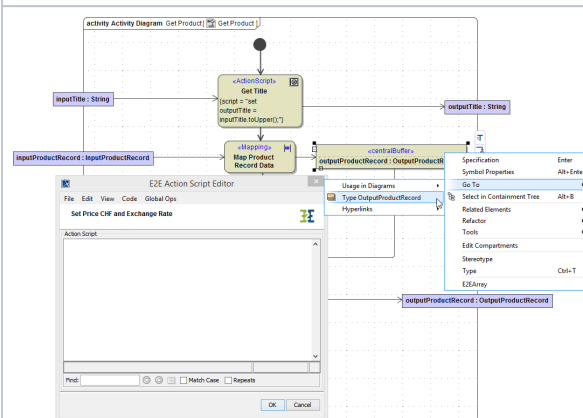
In the next modeling step, you will define the action script of the action **Set Price CHF and Exchange Rate**. Within this action script, you will set the exchange rate to the output parameter and perform the currency calculation.



Click the action node **Set Price CHF and Exchange Rate** with the right mouse button and select **Action Script Editor** from the context menu.



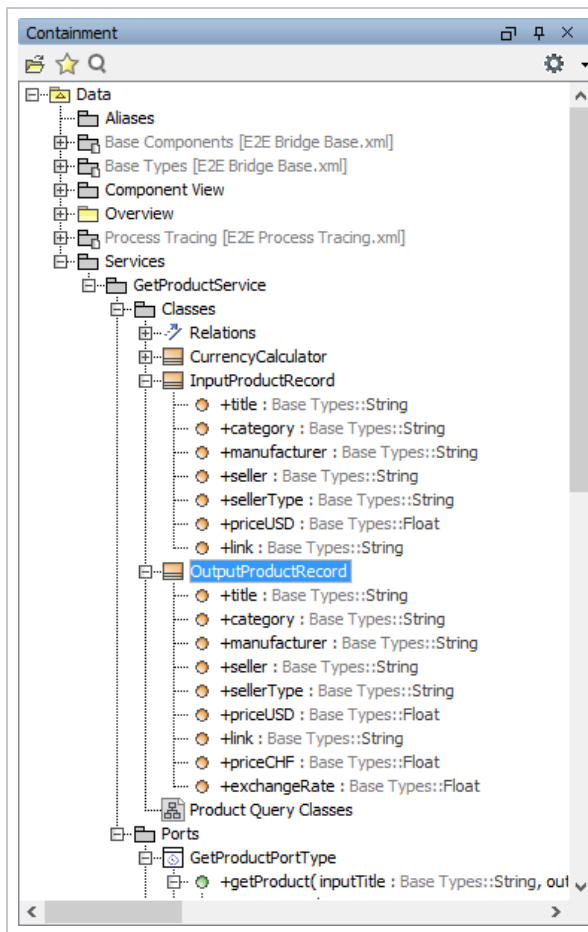
The **E2E Action Script Editor** opens.



Remember that you defined the attribute **exchangeRate** in class **outputProductRecord**.

Although the Action Script Editor is open, you can check the class attributes in between.

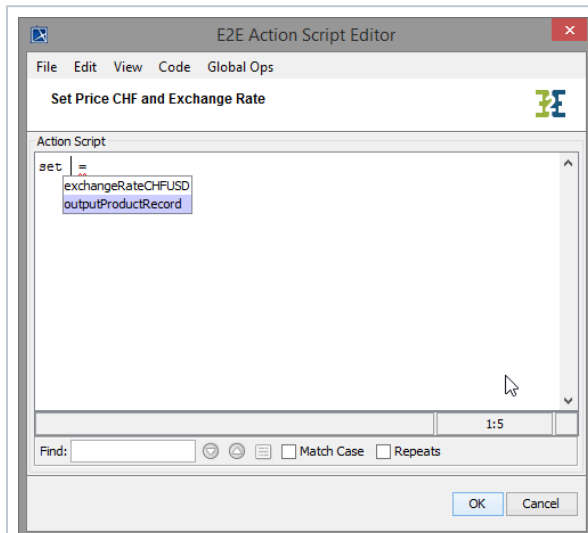
Select the central buffer node **outputProductRecord** and select **Go To > Type OutputProductRecord** from the context menu.



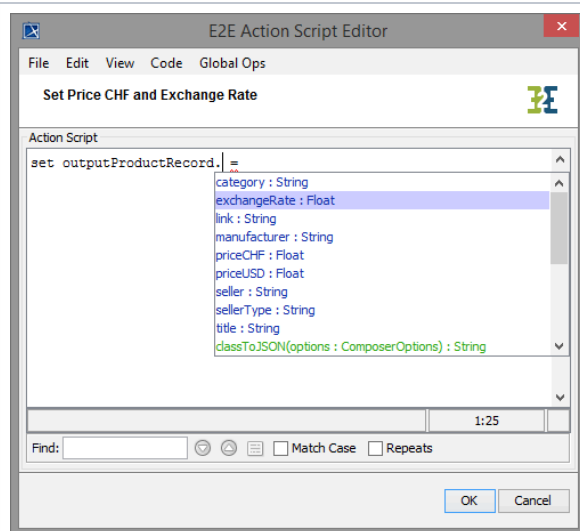
The class definition of **OutputProductRecord** gets selected in containment tree.

Expand the tree to view all attributes of **OutputProductRecord**. In the meantime, the Action Script Editor stays open. Continue working in the Action Script Editor now.

You can reference the attribute **exchangeRate** of class **outputProduct-Record** by writing the object name followed by a dot and the attribute name (`outputProductRecord.exchangeRate`). Use the set assignment statement in order to assign a value to an attribute being of a base type (here **Float**).



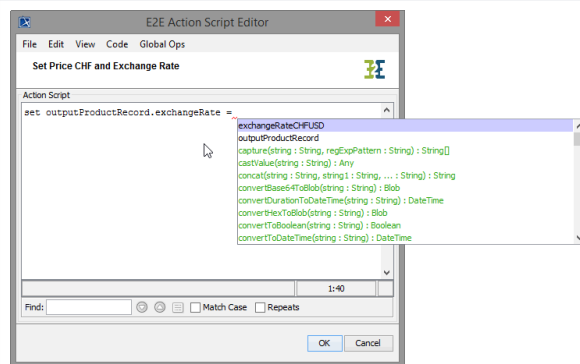
Adding a space after set lists all available objects of the current action. Select the object **outputProductRecord** and press **Return**.



Now, you need to reference the attribute **exchangeRate** of the object **outputProductRecord**. After adding a dot behind the object name, all attributes of the object (blue) and all available operations (green) will be listed. The list of operations contains the operations of a corresponding base class (for instance a String) as well as **Any Type Operations** (operations that are applicable to any possible type).

Select the attribute **exchangeRate**.

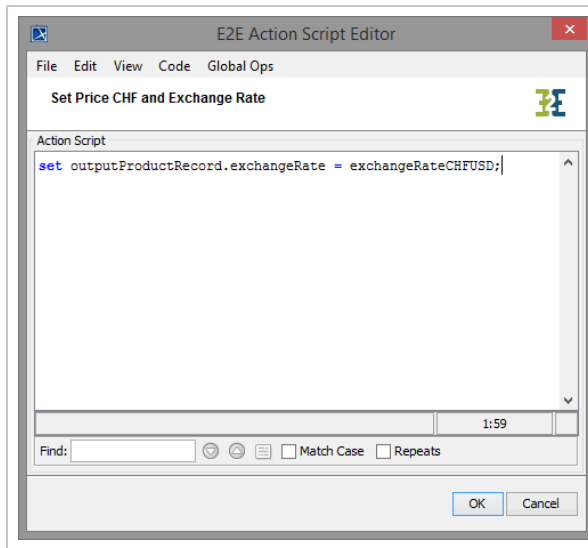
The notion **exchangeRate:Float** in the suggestion list indicates that the attribute is of base type **Float**.



Note that an equal sign (=) has already been added after entering set.

Go to the end of the line and press **Ctrl - Space** to force the list of suggestions for possible input objects. Select **exchangeRateCHFUSD**.

If Action Script Editor does not offer you the accessible objects automatically, you can force the list of suggestions by pressing **Ctrl - Space**.



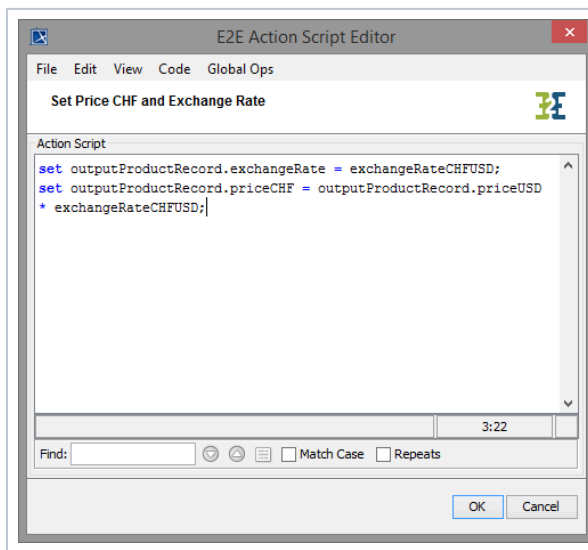
Every action script statement has to be finished with a semicolon (;), which you have to add to complete the statement.

In the second action script statement, you will perform the currency calculation. The price in USD will be converted to the price in CHF. You will use arithmetic operators in the set assignment statement and assign the value of attribute **priceUSD** multiplied with the value of object **exchangeRateCHFUSD** to the value of attribute **priceCHF**.

Enter the following action script using the script editing features.

```
set outputProductRecord.priceCHF = outputProductRecord.priceUSD *
exchangeRateCHFUSD;
```

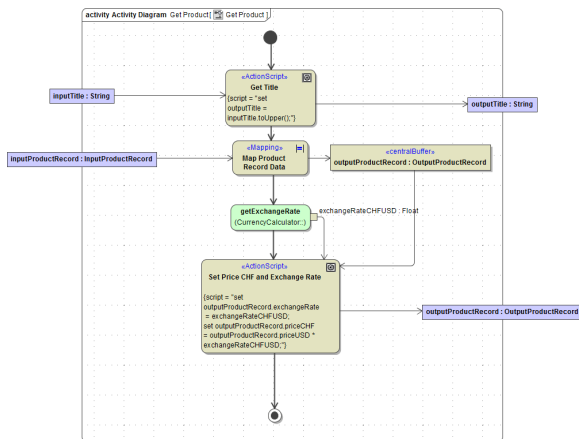
The factor **exchangeRateCHFUSD** is known within the action script as it was passed into the action node via the output pin of call operation action **getExchangeRate**.




The action script is complete.

Press the **Ctrl - Enter** keys to close the Action Script Editor.

The action script is now shown in the tagged value **script** within the action node **Set Price CHF and Exchange Rate**. Note that the stereotype **<<ActionScript>>** was applied to the action node as well.



Save  the UML model.

Next, you will implement the class operation **getExchangeRate()**.

## Implementing the Class Operation




Besides attributes, there are operations that are defined on classes. Basically, these operations can be called within a service instance but not externally by other processes, clients, etc. However, if you like to publish class interfaces, for instance as Web service interfaces, you can give the class the stereotype **<<E2ESOAPPortType>>**. A port type accumulates operations that a client can call on a Web service. If you want to publish other interfaces like SAP RFC modules or HTTP operations, the Bridge provides other appropriate stereotypes.

While defining the **Classes** you defined the class **CurrencyCalculator**, which has no attributes and one operation **getExchangeRate**.

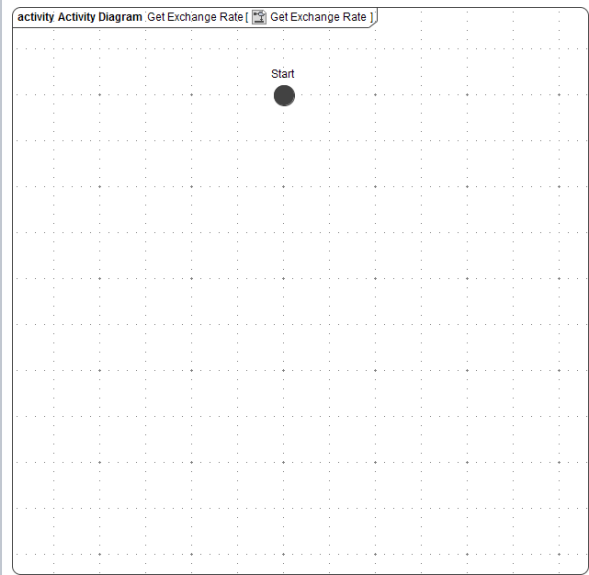
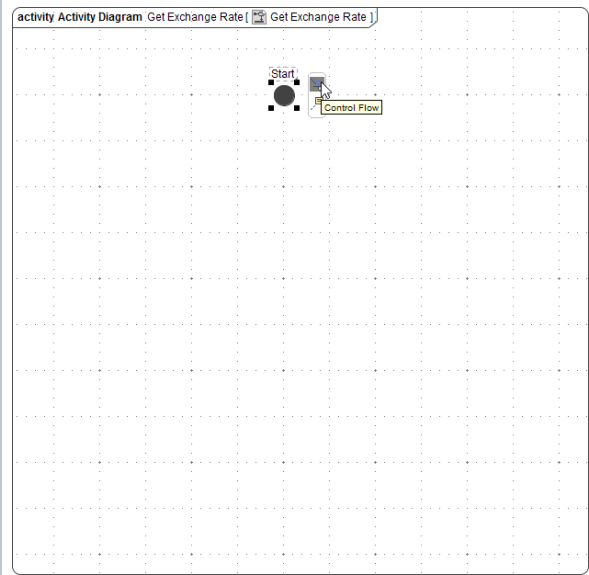
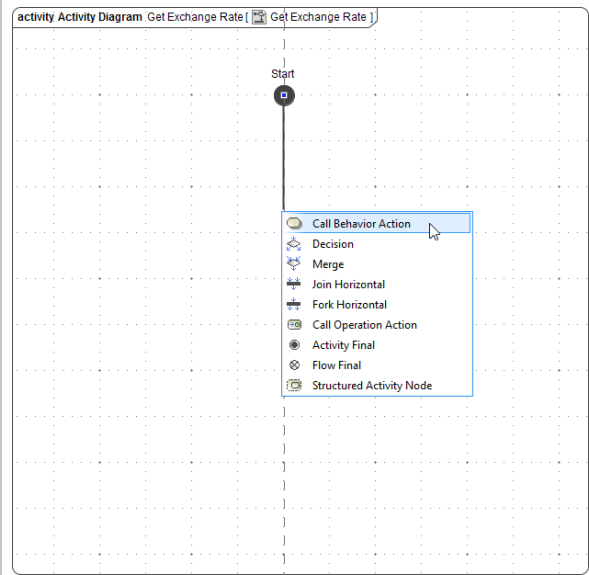
The operation **getExchangeRate** is defined as static. That means, no instance of the class has to be created to use the operation. In chapter **Using Call Operation Action**, you directly called this operation in the activity diagram via the **Call Operation Action**, in order to get the exchange rate for the currency conversion.

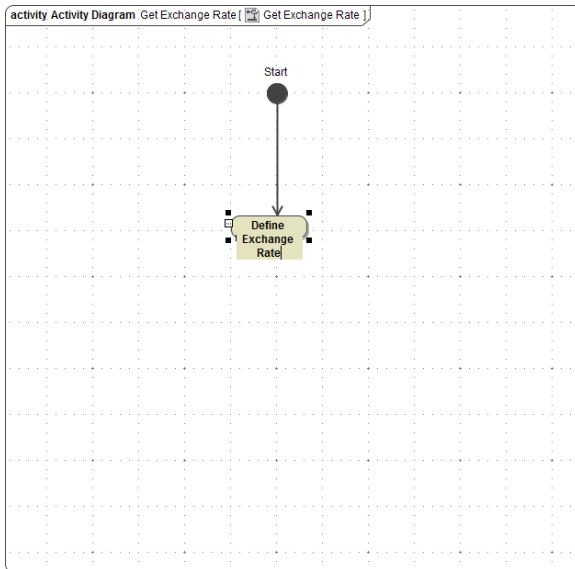
Now, you will implement this operation in the activity diagram, you already defined before. It is already assigned to the class operation **getExchangeRate**. The activity diagram is located in the containment tree directly within the class element. Navigate to the activity diagram **Get Exchange Rate** (in **Data - Services - GetProductService - Classes - Currency Calculator**) and open it.

Now add the required diagram components. Add the following to the diagram pane:

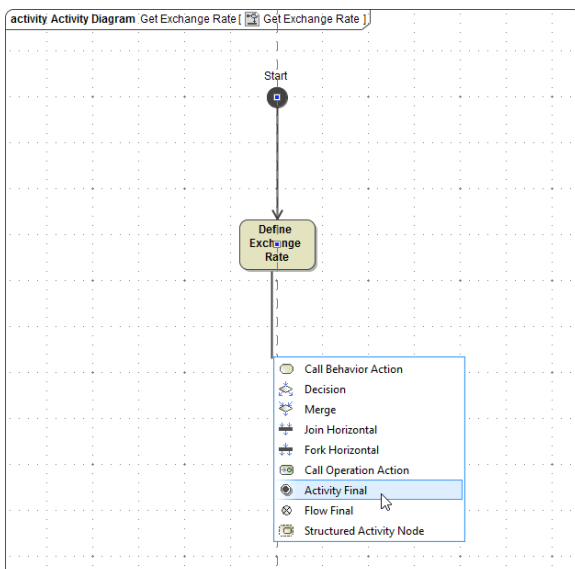
- an initial node 
- an action node 
- and an activity final node 

The most effective way to draw this is described below.

 <p>The diagram shows a single 'Start' node, represented by a solid black circle, on a grid background. The title bar of the diagram pane reads 'activity Activity Diagram Get Exchange Rate'.</p>	<p>First, add an initial node to the activity diagram.</p> <p>Just after drawing the UML element start typing a name. Assign the name <b>Start</b>.</p>
 <p>The diagram shows the 'Start' node. A mouse cursor is hovering over the 'Control Flow' icon in the smart manipulation toolbar, which is located near the top of the diagram pane.</p>	<p>Select the initial node and choose the <b>Control Flow</b> icon from the smart manipulation toolbar.</p>
 <p>The diagram shows a vertical line extending downwards from the 'Start' node. A context menu is open, listing various actions. The 'Call Behavior Action' option is highlighted by the mouse cursor.</p>	<p>Drag the control flow downward and right-click on the diagram pane below the initial node.</p> <p>From the context menu, select <b>Call Behavior Action</b> as node type the control flow is leading to.</p>

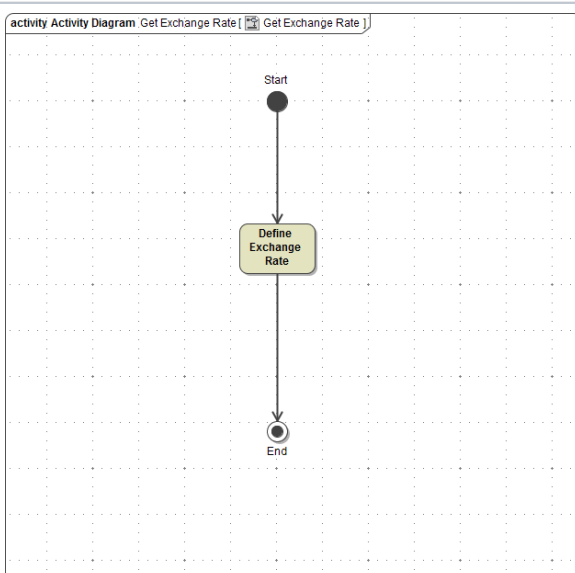


Assign the name **Define Exchange Rate**.



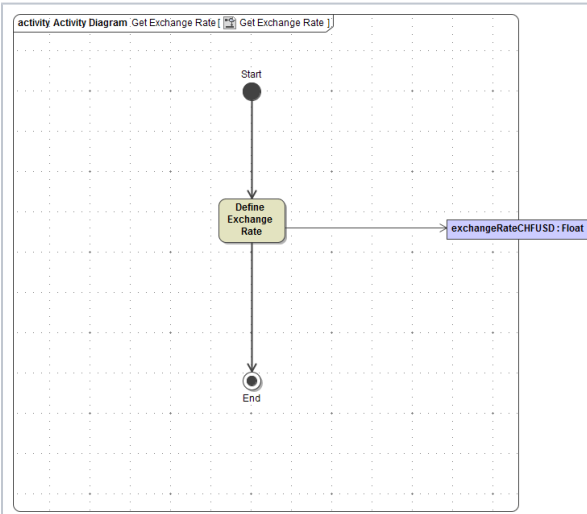
Select the action node and draw another **Control Flow** again. As ending point of the control flow select **Activity Final**.

Assign the name **End**.



The activity diagram should look like this.

Giving initial and final nodes names makes maintaining and debugging the Web service easier, as they can be identified via their name. Especially if there are more ending points in the same diagram we recommend giving them a name. Note that the names are displayed in the containment tree now.



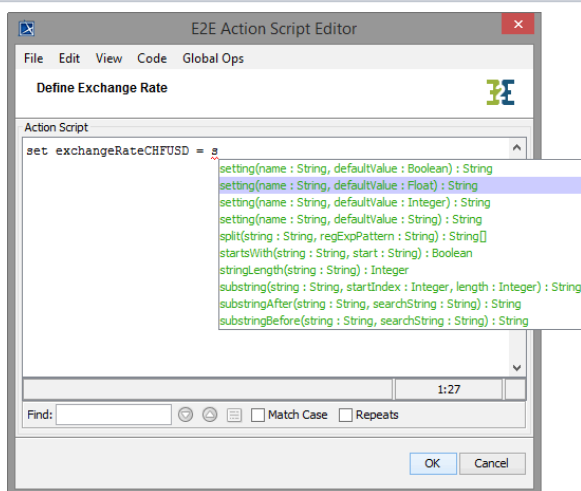
Select the parameter **exchangeRateCHFUSD** in the containment tree.

As it is an output parameter, drag it on the diagram pane on the right border of the diagram.

Draw an object flow from the action node to the output parameter.

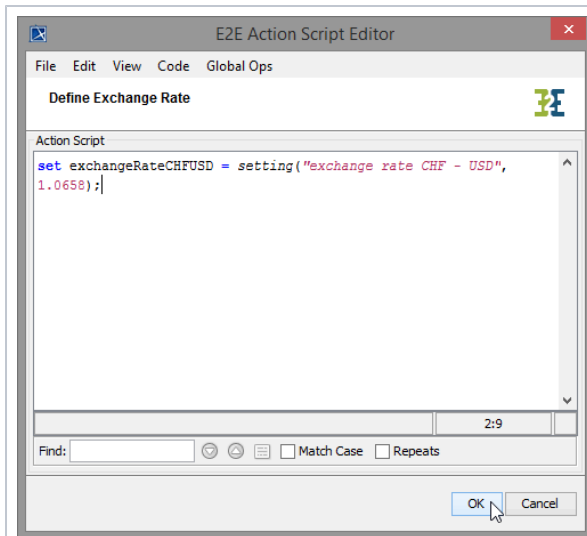
Frequently, it is necessary to store literals global to an xUML service. In the Bridge context, it is possible to define name value pairs that are configurable by the E2E Embedded Runtime respectively the E2E Bridge. These name value pairs are called settings.

For the moment, you will hard code the exchange rate as a fix value that can be edited via the xUML service settings. In lesson 3, you will extend the service by implementing a call to an external Web service in order to get the exchange rate.



Open the **Action Script Editor** and start with the set assignment statement:  
`set exchangeRateCHFUSD =`

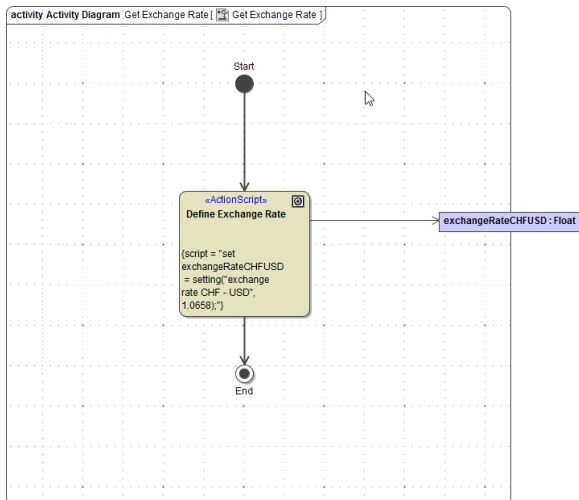
After the equal sign, type **s** and open the suggestion list with **Ctrl - Space**. Select the second setting statement offered as you want to assign a float value.



Replace the **name** by "exchange rate CHF - USD". Replace defaultValue by, for instance, 1.0658. This statement will create a configurable setting of the xUML service, whose value is assigned to the variable **exchangeRateCHFUSD**.

Finish the statement with a semicolon and close the Action Script Editor.

The implementations of the activity diagrams are finished now.



Save  the UML model.

The next step in the development process is to modify the component diagram.