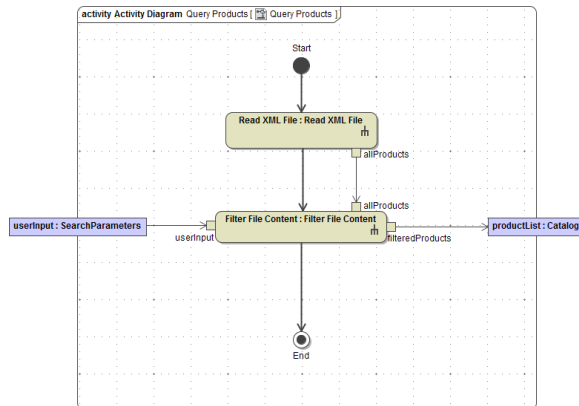


Reading the XML File MD18



The first action calls the activity **Read XML File** and contains actions to read the XML file and to create the output object **allProducts**. To retrieve the data from the XML file you need to use the file system adapter.

The second action calls the activity **Filter File Content** which uses the entered keywords to filter the catalog data. After the outlining of the activities, the activity diagram **Query Products** will look like shown in the picture below.



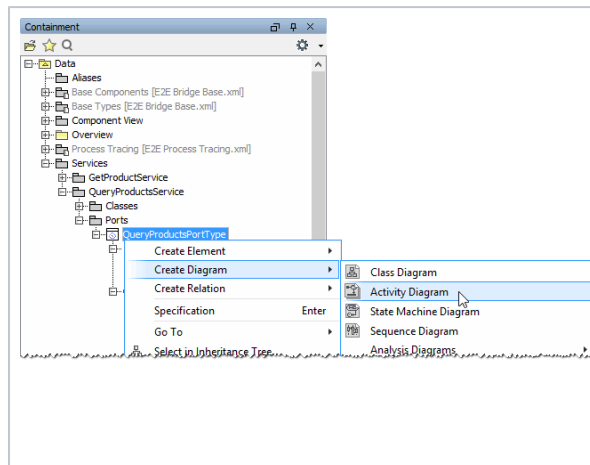
Filtering the File Content

On this Page:

- [Outlining the Activities](#)
- [Implementing the Activity Read XML File](#)
- [Reading the File with the File System Adapter](#)
- [Transforming the XML Data](#)

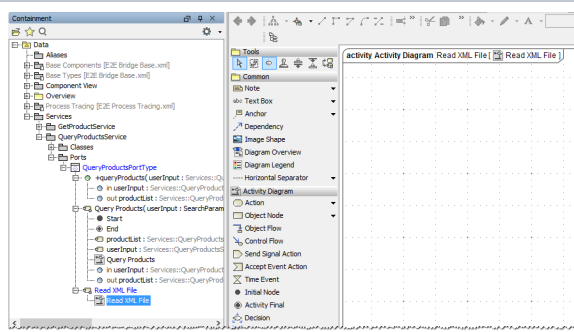
Outlining the Activities

To continue the activity diagram **Query Products**, you need to create the activity **Read XML File** that is called at this point.



As an activity diagram has to be assigned too, you will create the activity and the corresponding diagram at the same time.

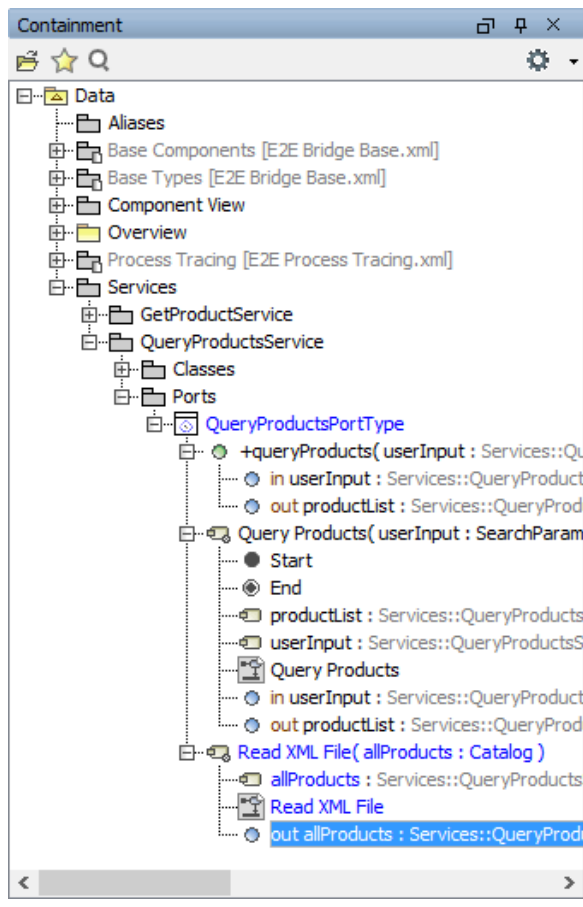
Select **QueryProductsPortType** in the containment tree. Create the activity diagram **Read XML File**.



The activity diagram **Read XML File** has been created as well as the activity having the same name.

The new diagram is opened in the diagram pane. In the containment tree, it is displayed below the activity.

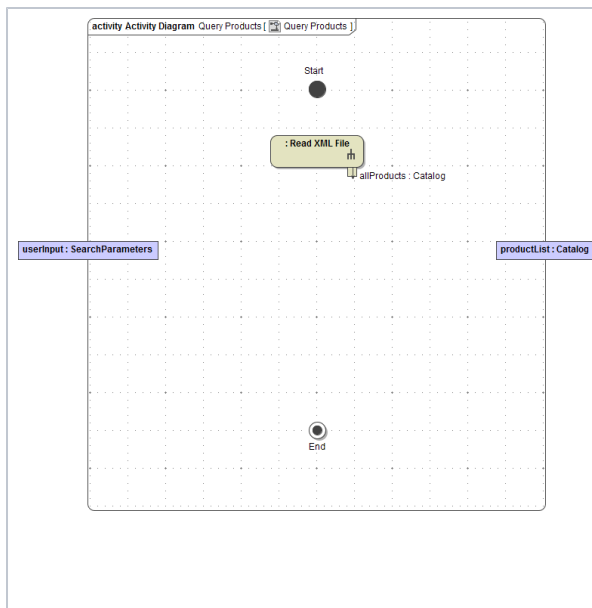
Creating an activity diagram in the containment tree, the diagram owner (the activity) will get the same name.



The action **Read XML File** calls the activity **Read XML File**, which returns the object **allProducts**.

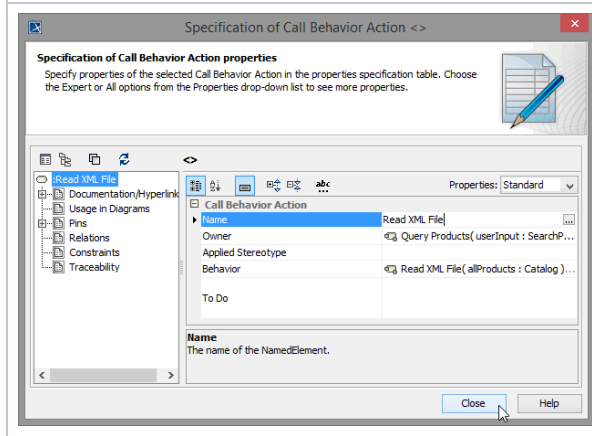
In the containment tree, select the activity **Read XML File** and create a new parameter:

- **allProducts : Catalog** (complex type)
- direction **out**

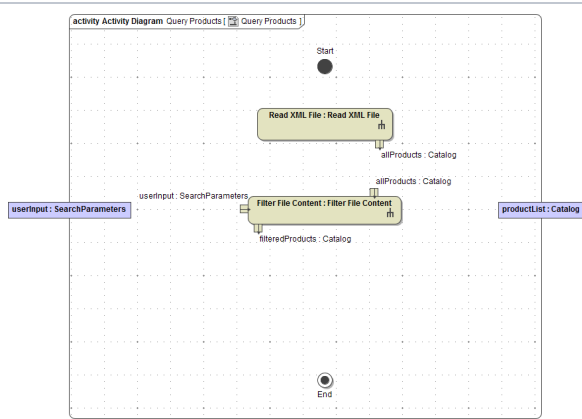


Return to the activity diagram **Query Products**. Drag and drop the activity **Read XML File** to the diagram pane.

A new action node including the output pin **allProducts** is created on the diagram pane. As **allProducts** is an output pin, move it to the right side of the action node.



Open the specification dialog of the action node and assign the name **Read XML File**.

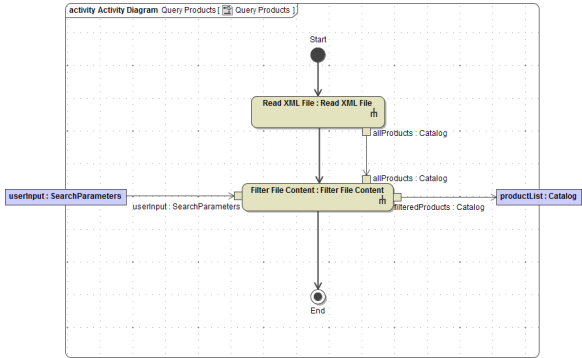
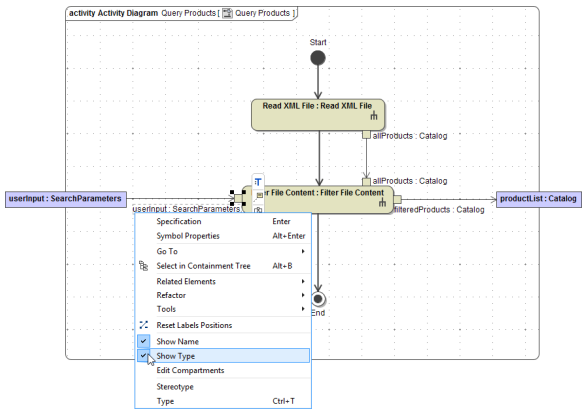


Now, create the second activity **Filter File Content**. Select **QueryProductsPortType** in the containment tree and create the activity along with the activity diagram like you with **Read XML File**.

Create the parameters:

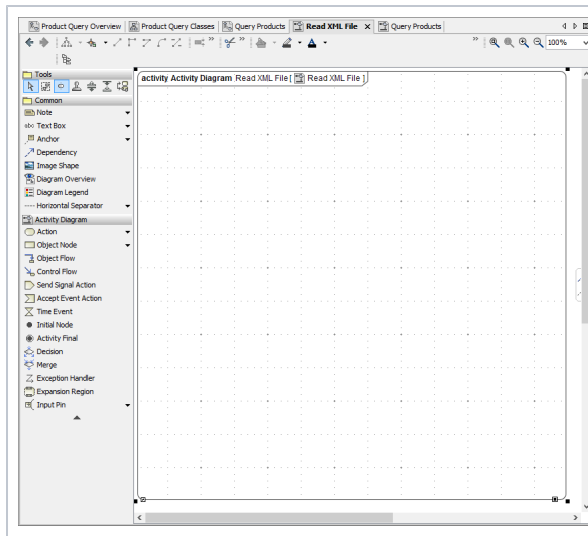
- **userInput : SearchParameters** (complex type), direction in.
- **allProducts : Catalog**, in.
- **filteredProducts : Catalog**, out.

Drag and drop the activity to diagram **Query Products** to create the action node and assign the Name **Filter File Content**.

	<p>The value of the pin filtered Products is used as output of the SOAP operation for the time being. Move it to the right side of the action node and draw the required object flow from the pin to the output parameter productList.</p> <p>Also, draw the object flows from the output pin all Products to the corresponding input pin and from the input parameter userInput to the corresponding pin.</p> <p>Finish the diagram by drawing the necessary control flows.</p>
	<p>The names of the pins are displayed including their type. For a well-arranged diagram pane, you can reduce the display to the pins names.</p> <p>Open the context menu on a pin and disable the checkbox Show Type.</p>

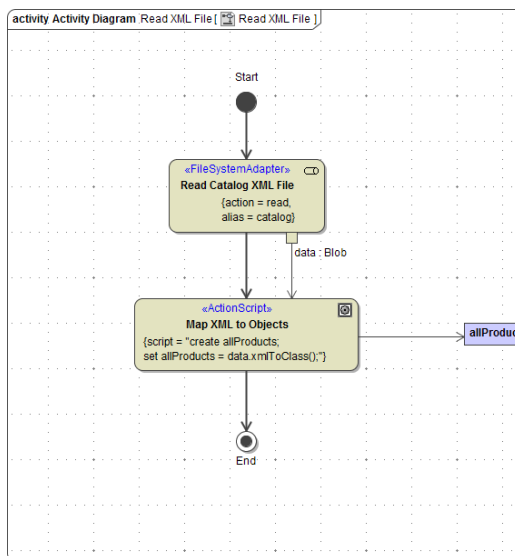
Save  the UML model.

Implementing the Activity Read XML File



Double-click the action **Read XML File** either in the containment tree or in the diagram pane to open the empty activity diagram again.

In the picture below, the completed activity diagram provides an overview of all activities you are going to implement in **Read XML File**.



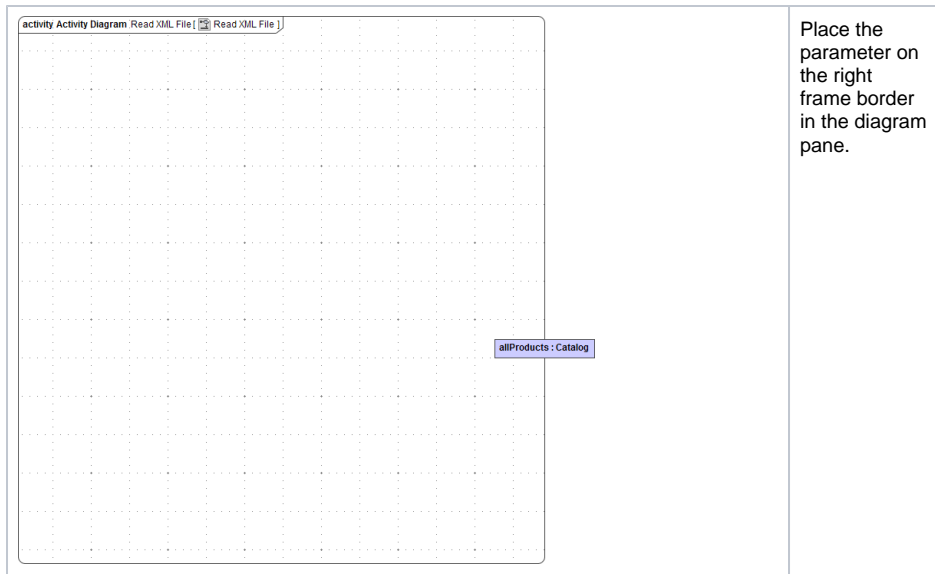
When reading from a file (see action node **Read Catalog XML File**), the data is stored in an object **data**, which is of base type **Blob**. A blob is a **binary large object** that can store a large amount of binary data (e.g. pictures, web pages, and so on). To read the XML file, you can use the file system adapter that facilitates the file access. All you need to do is to define the location where the file is stored.

In the activity diagram, you define the action of the file system adapter (for instance **read**). In the component diagram, you customize the file system interface (file name and path).

It is also possible to define the path and name of a file dynamically in the activity diagram. For more details refer to the [xUML Services Reference Guide](#).

The output object of the file system adapter when reading a file must always be named **data**.

After the XML data has been stored in a blob, the binary data is transformed and assigned to the output object **allProducts**. The output object is passed to the caller, which is the action **Read XML File** in the activity diagram **Query Products**.

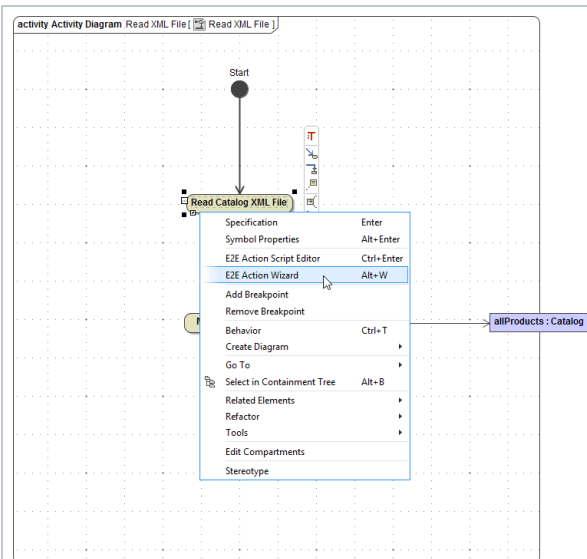


Draw the activity diagram as shown in the picture below:

Element	Name
Initial and final nodes	Start and End
Action nodes	Read Catalog XML File
	Map XML to Objects

Draw all required control and object flows as shown in the picture. In the next step, you will finish the implementation of the activity **Read XML File**.

Reading the File with the File System Adapter



Select the action node **Read Catalog XML File** and choose **E2E Action Wizard** from the context menu.

In the stereotype list select **FileSystemAdapter**.

Click **Next**.

Action Wizard

Enter Properties
Specify the properties of the new action.

Action
read

Alias
<UNDEFINED> **New**

Mode
<UNDEFINED>

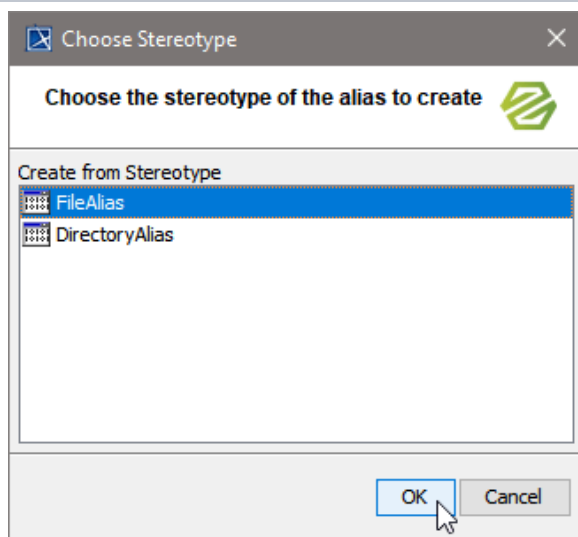
Resource
<UNDEFINED>

Help Next Finish Cancel

The file system adapter can be used for different purposes, for example, to remove and write files, or to handle directories. Therefore, you need to specify the action that the file system adapter shall perform.

Leave the default action **read** as you want to read the file.

Click **New** to define an alias.



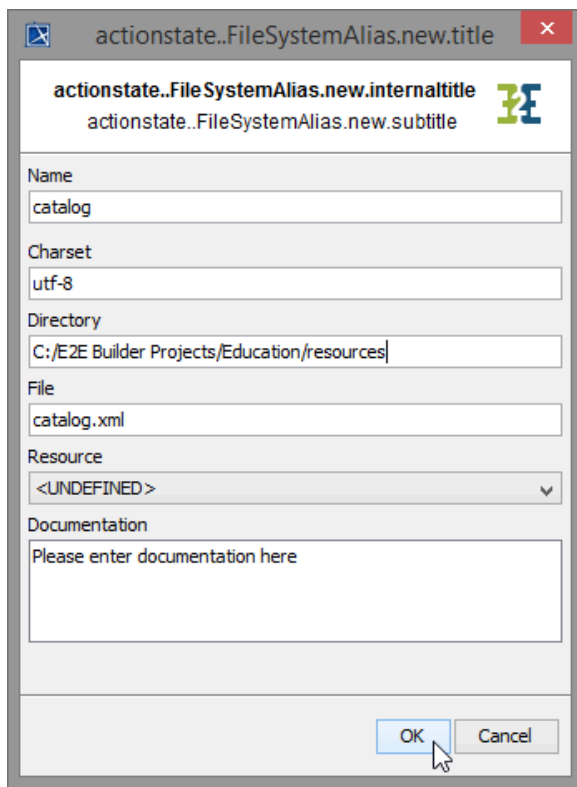
You will now define a pointer that links the physical definition of the backend in the component diagram to the logical definition in the activity diagram, where a file system adapter as a backend is used.

In this way, an action in the activity diagram will be enabled to access a backend defined in the component diagram. Note that the backend has not been defined in the component diagram, yet. This will be done later by the help of the E2E Components Wizard.

Bridge 7
Select stereotype **FileAlias** and click **OK**.

Select stereotype **FileSystemAlias** and click **OK**.

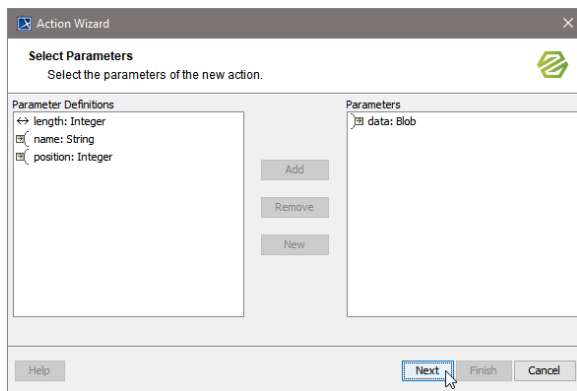
Stereotype **Alias** is deprecated and serves only for backwards compatibility to older versions of the E2E Builder.



Now specify the file details.

Assign the name **catalog** to the new alias. Specify the directory, in which the file catalog.xml can be found and enter **catalog.xml** as the name of the file to be read via this alias.

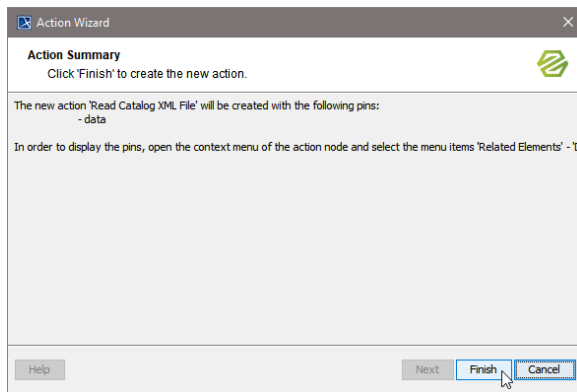
Click **OK** and **Next**.



Remember that the read data is stored in an object **data** of base type **Blob**. This object now is suggested from the wizard as a parameter.

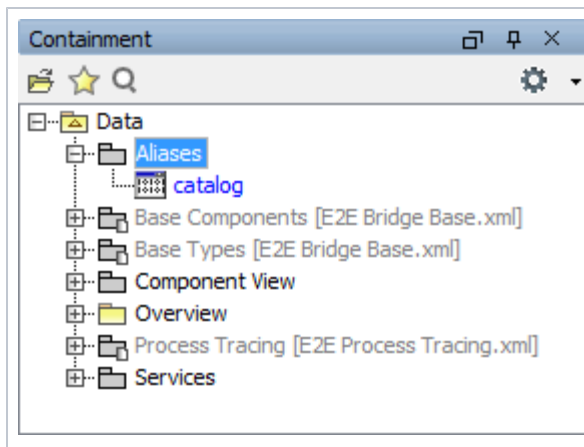
You do not need any further parameters.

Click **Next**.



The wizard presents a summary of all adjustments.

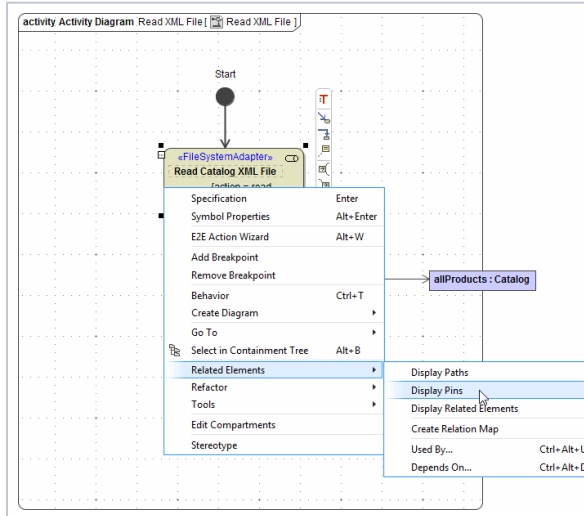
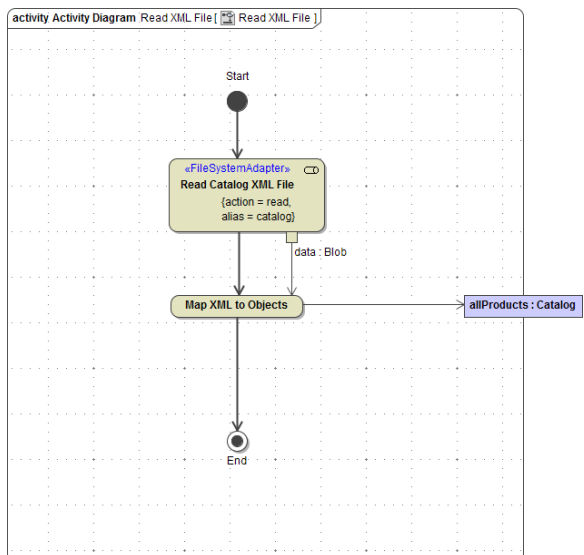
Click **Finish**.



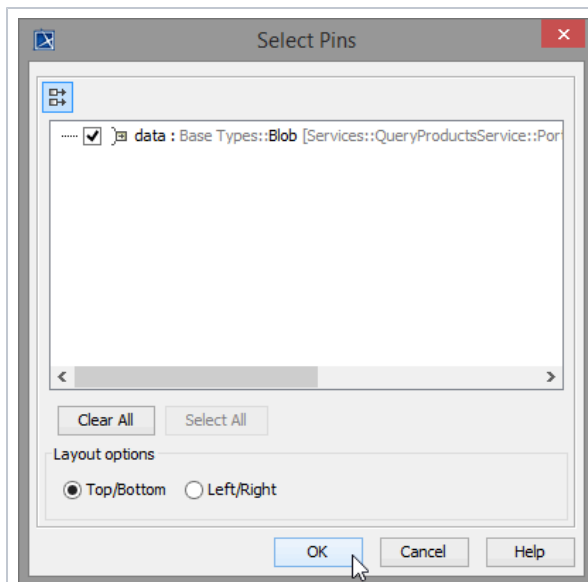
After finishing the wizard, you can find the alias definition in the package **Data / Aliases**.

The stereotype **<<FileSystemAdapter>>** you have defined on the action node is displayed in blue. The tagged values **action=read** and **alias=catalog** are also displayed. The action is ready to read the XML file using the file system adapter (once the backend is defined in the component diagram). The alias is used to look up the path and name of the file in the component diagram.

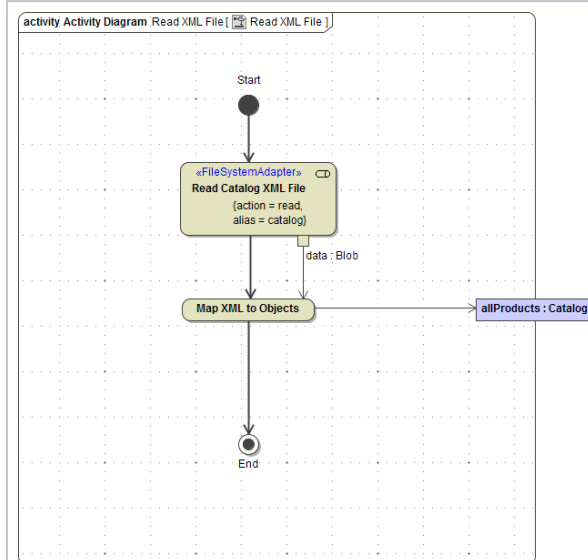
The content of the XML file will be stored in the object **data**; no action script needs to be written. When using the file system adapter, the received data must always be stored in an object named **data**, which needs to be of base type **Blob**. The default character set of a blob object is UTF-8 (see appendix of the [x UML Services Reference Guide](#)).



To display the object **data**, open the context menu of the action node **Read Catalog XML File**. Choose the item **Related Elements > Display Pins**.



In the dialog **Select Pins** check the object **data** and click **OK**.

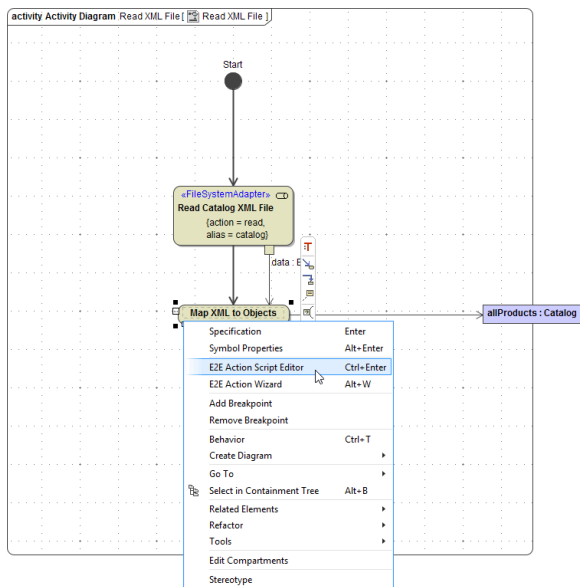


The pin is now displayed on the diagram pane. Move it to the right side of the action node and draw an object flow from the pin to the action node **Map XML to Objects**.

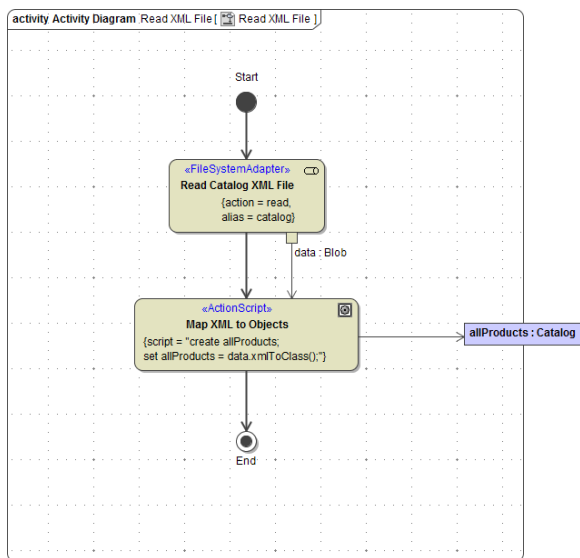
Save  the UML model.

Transforming the XML Data

In the action node **MAP XML to Objects**, you will transform and assign the data of the blob object to the object **allProducts**. Use the Action Script Editor to enter the following action script statements.




Statement	Description
create allProducts ;	This statement creates the object allProducts , which is an instance of class Catalog .
set allProducts = data. xmlToClass ();	The operation <code>xmlToClass</code> converts an XML structure to a class structure. All XML elements are mapped to class associations, respectively <code><<XMLElement>></code> stereotyped attributes, automatically. The operation is called on the blob object data containing the XML data.



To make the automatic data mapping of the `xmlToClass` operation work, the defined class attribute needs to have the same name as the XML element. In this case, you linked the attribute **products** of class **Catalog** to the XML Element **product** by specifying the external name **product**.

For further information about E2E Bridge operations, see [xUML Services Reference Guide](#).

Save  the UML model.