Use Microsoft Excel

Contents

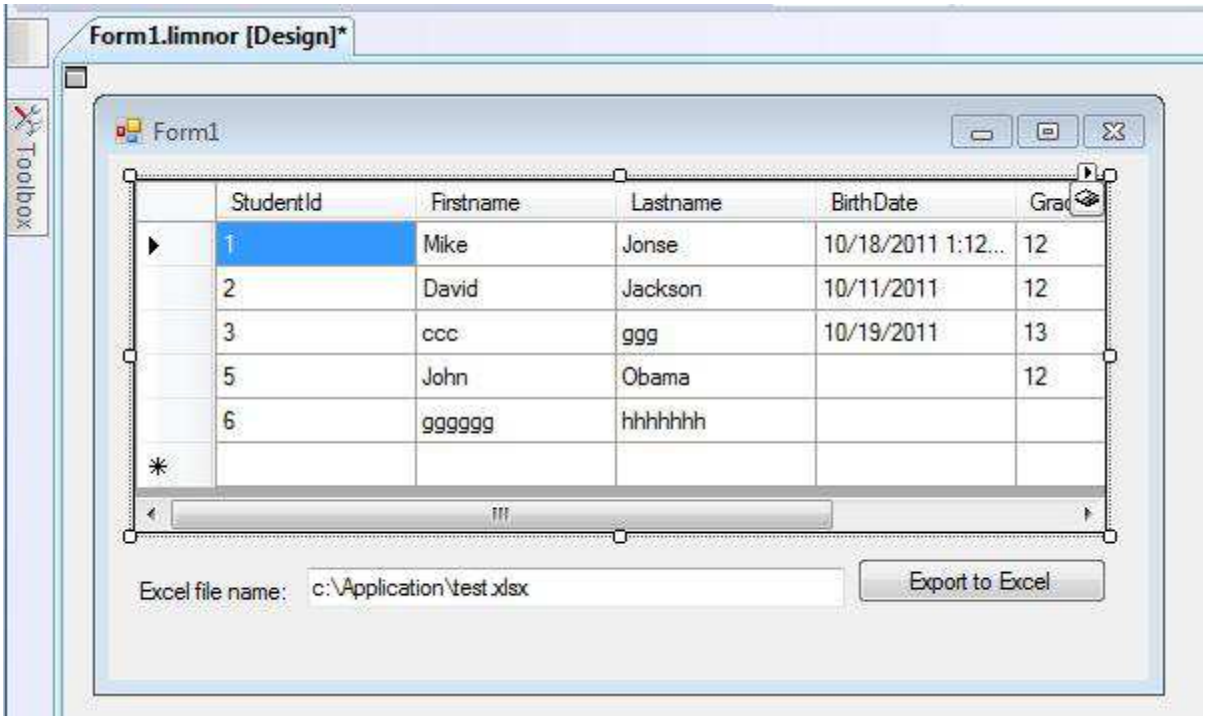
Introduction

Microsoft provides Primary Interop Assembly (PIA) for accessing Microsoft Office. PIA is very powerful, enabling total control of the Office. But the PIA is not intuitive. This sample project is trying to point out pitfalls to watch out. The sample uses PIA for Office 2007.

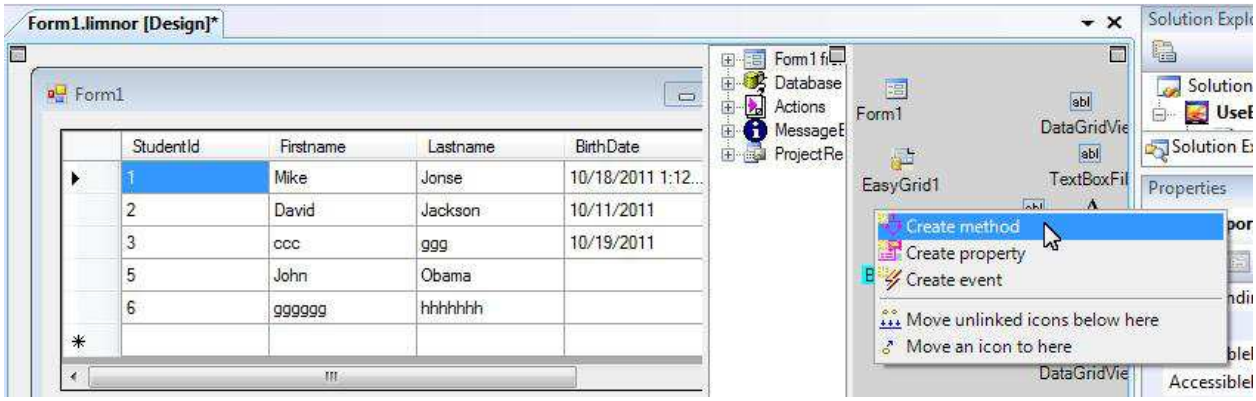
Suppose Microsoft Office is installed and Microsoft Primary Interop Assemblies (PIA) is installed. For downloading PIA for Office 10, see http://www.microsoft.com/download/en/details.aspx?id=3508 or search the internet for “PIA downloads” for PIA matching the version for your office.

Excel Sample: Export Data Grid

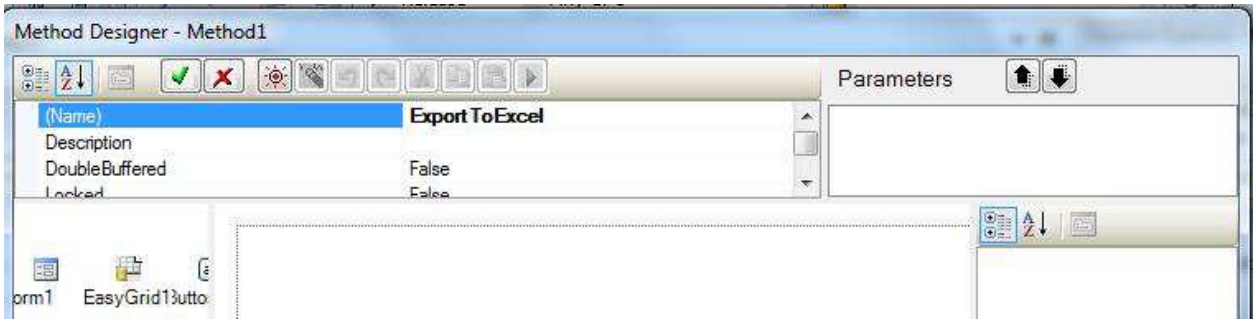
Suppose we use an EasyGrid to show data from a database and we want to send the data in the EasyGrid to an Excel sheet.



We may create a method to do the job of exporting the data in the data grid to an Excel file.



The Method Editor appears. Rename the method to “ExportToExcel”:



In this sample, all the programming with using Excel will be done in developing this method.

Create Excel Application, Workbook, and Worksheet

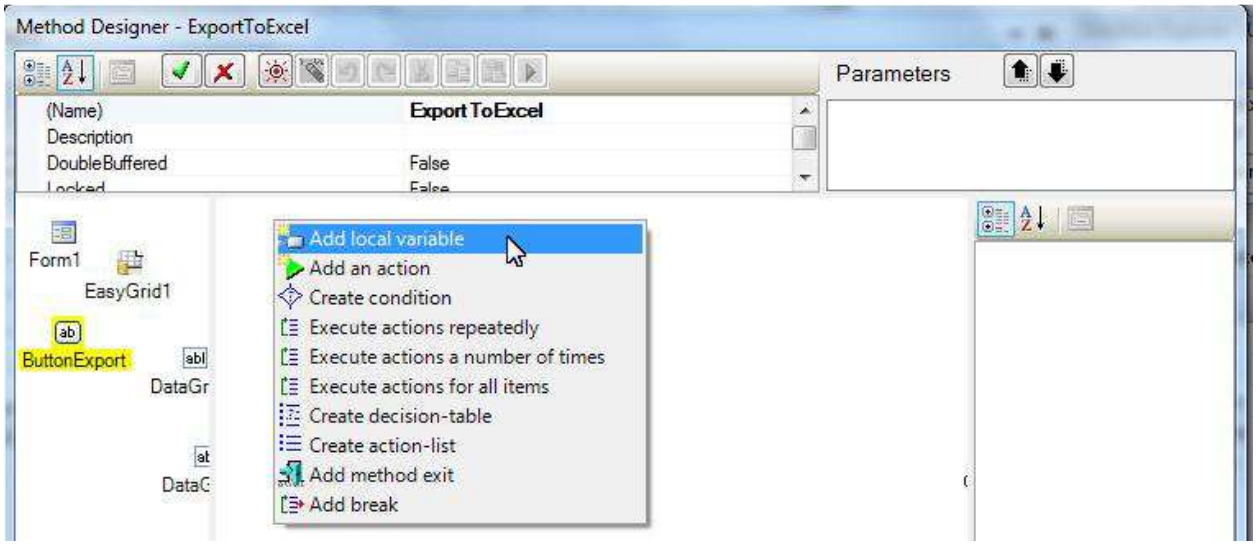
To use Excel, we need to at least get a Worksheet object. To get a Worksheet object we must have a Workbook object. To have a Workbook object we need an Excel Application object.

In the sample, we first create an Excel Application object. The Application object has a Workbooks property which has an “Add” method. We use it to create a new Workbook. The Workbooks property also provides other methods for getting a Workbook. For example, use get_Item to get an existing Workbook; use Open to open a file.

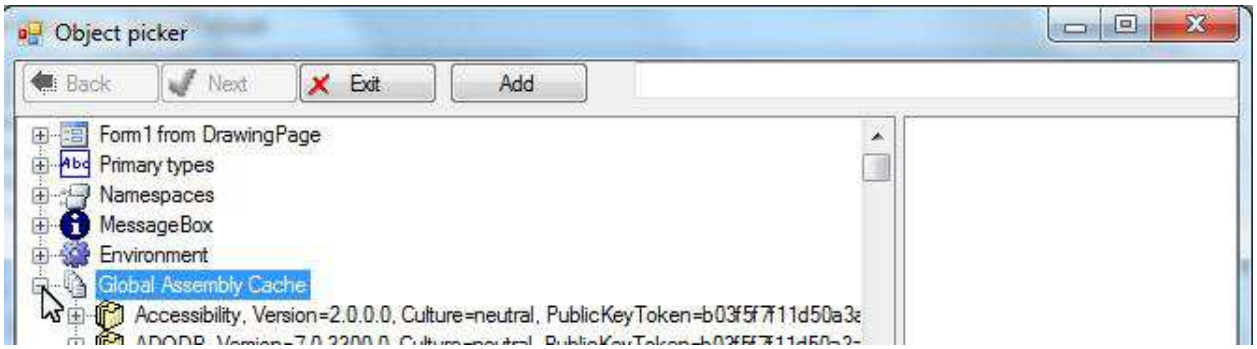
The Workbook has a Sheets property which has an “Add” method. We use it to create a new Worksheet. The Sheets property also provides other methods for getting a Worksheet. For example, use get_Item to get an existing Worksheet.

Create Excel Application

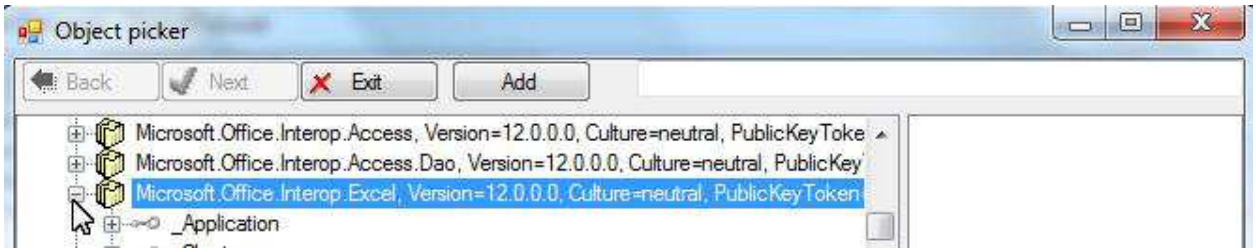
Right-click the Method Editor; choose “Add local variable”:



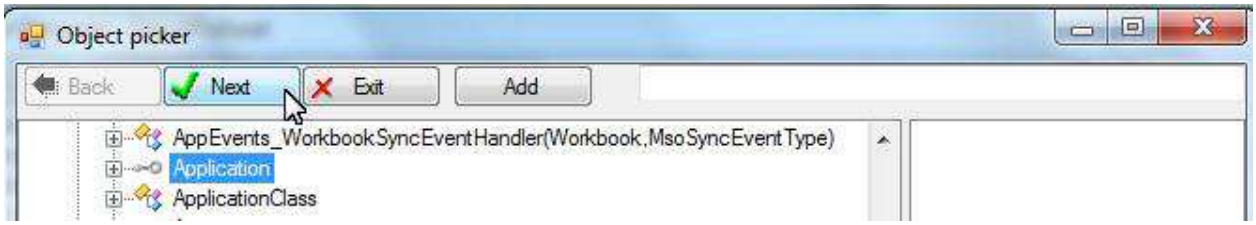
The PIA is under the Global Assembly Cache:



Find Excel in the Global Assembly Cache:

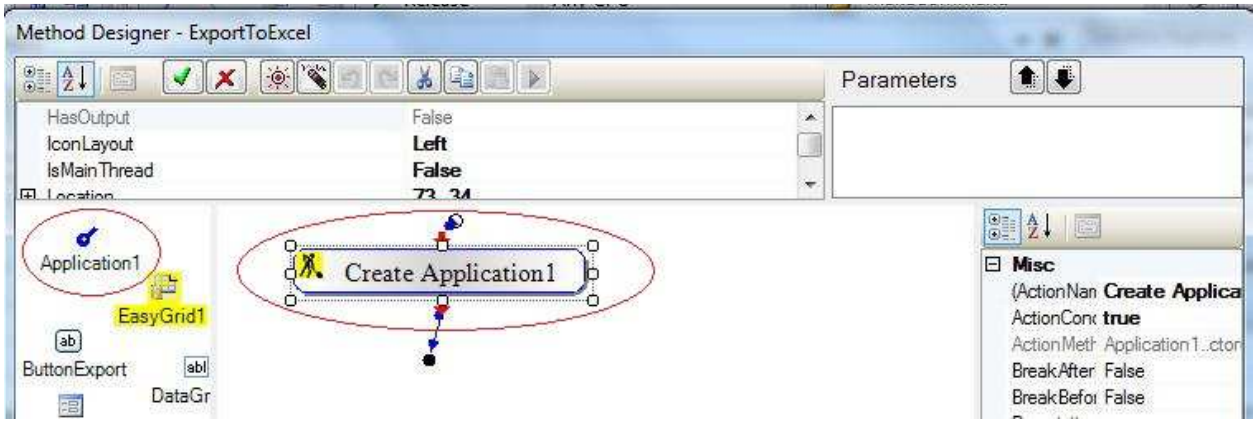


Select Application and click Next:



Select the parameter-less constructor to create an instance of Excel Application:

An Application object appears in the Variable Pane. An action appears in the Action Pane to construct the Application object:

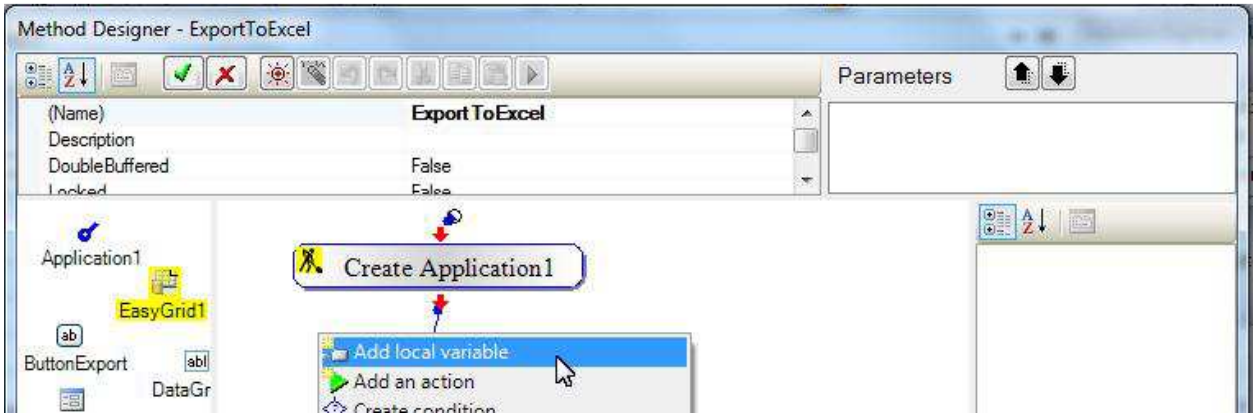


Note the following rules

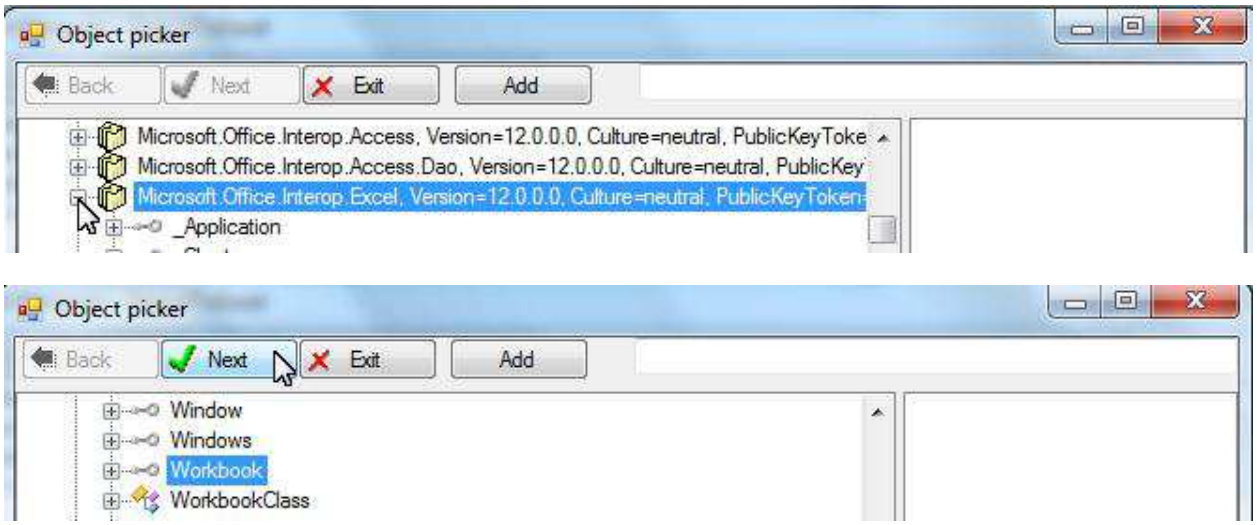
- All PIA objects to be used must be created via “Add local variable” so that their properties, methods and events can be accessed for programming
- Only the Application object should be manually constructed

Create Workbook

We must use “Add local variable” to create a Workbook variable first. Right-click the Method Editor; choose “Add local variable”:

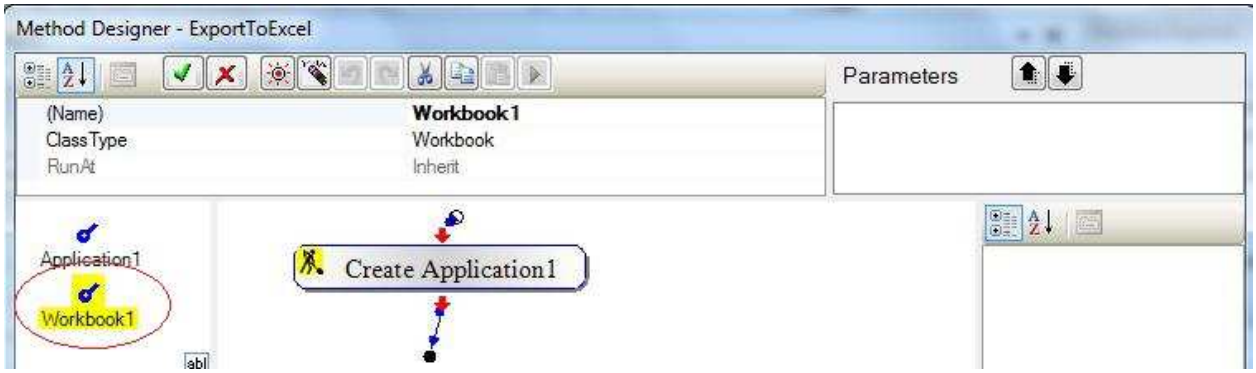


Select Workbook in the Global Assembly Cache:

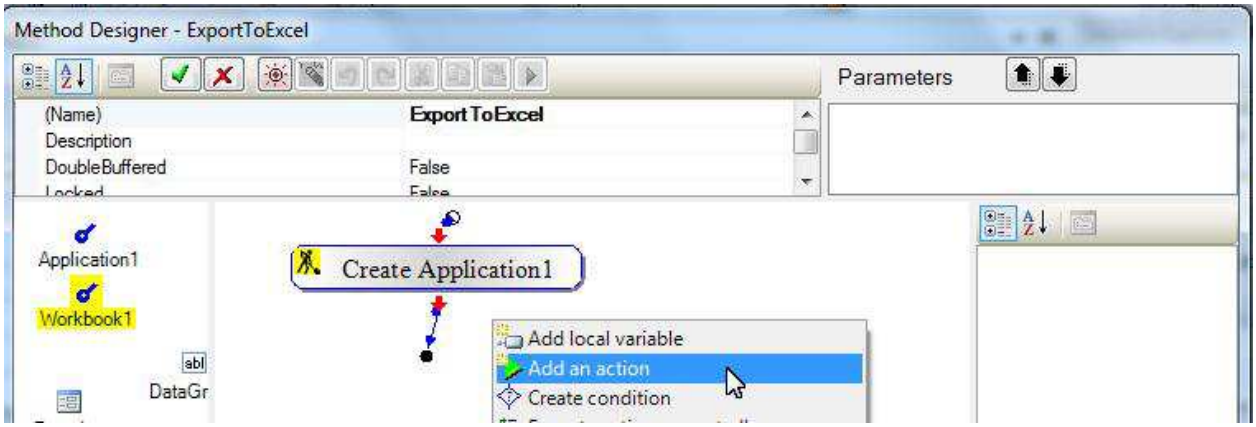


Select “null” because we should not construct an instance of Workbook manually. We need to let the Application object to create Workbook:

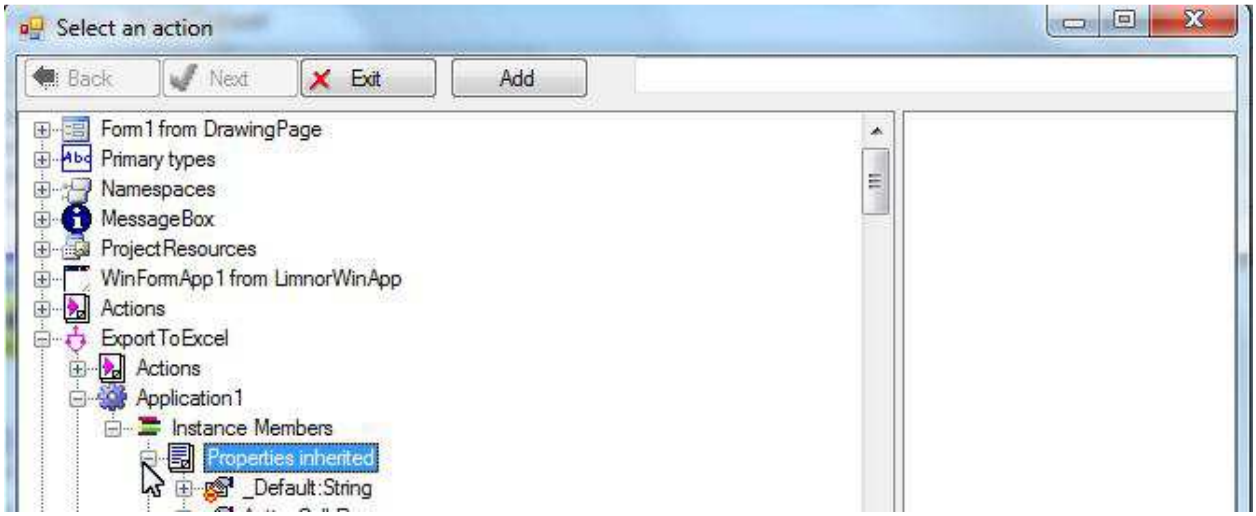
A Workbook variable appears:

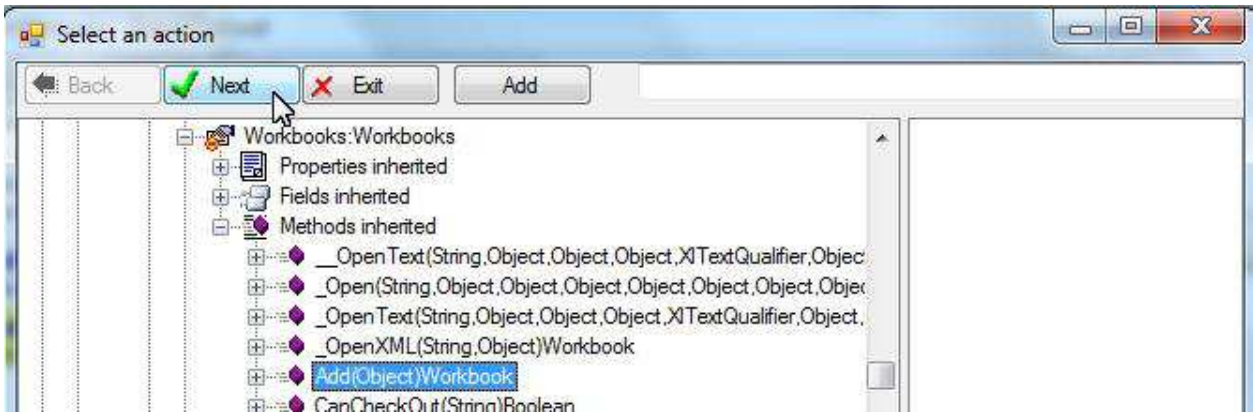


We may use the Add method of the Workbooks property of the Application object to create a new Workbook. Right-click the Action-Pane; choose “Add an action”:

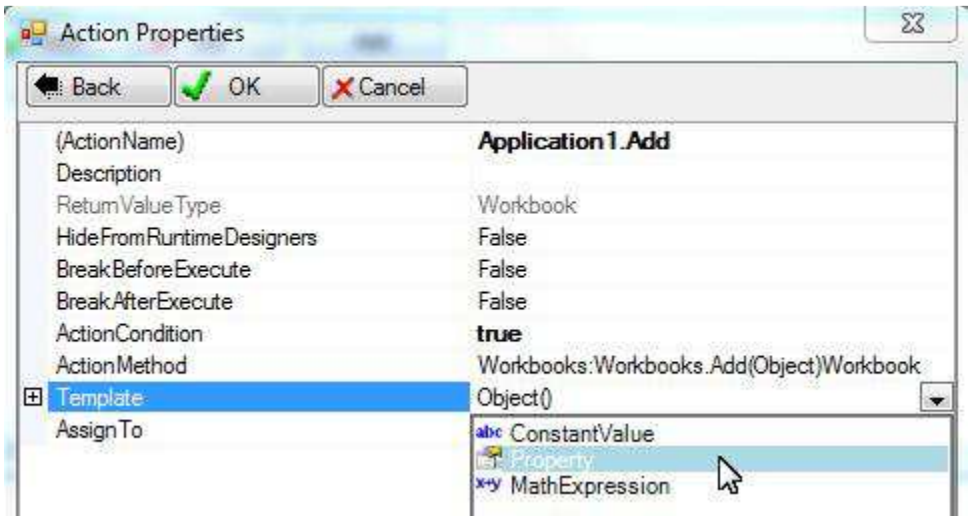


Locate the Workbooks property of the Excel Application and select its Add method:

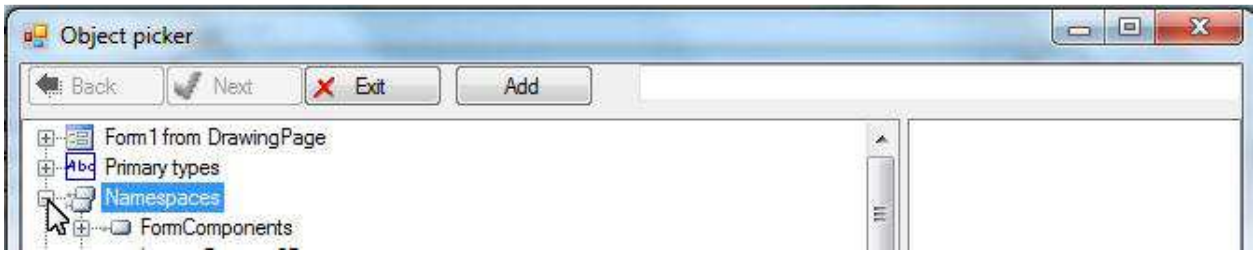




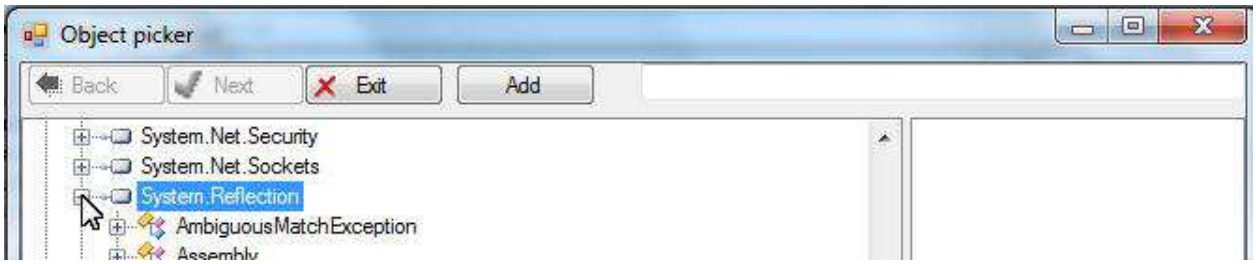
The Add action uses a Template parameter. In this sample, we do not need to use it. For all PIA actions, an unused parameter cannot be left blank, a special value must be provided for it via selecting Property:



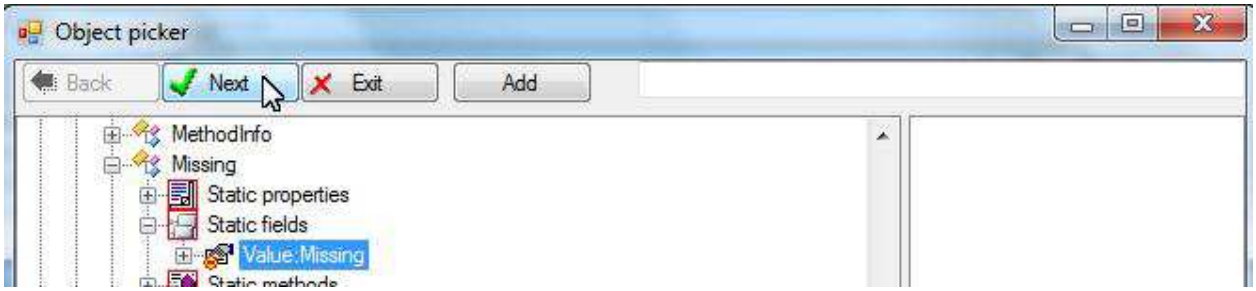
Expand Namespaces:



Expand System.Reflection:

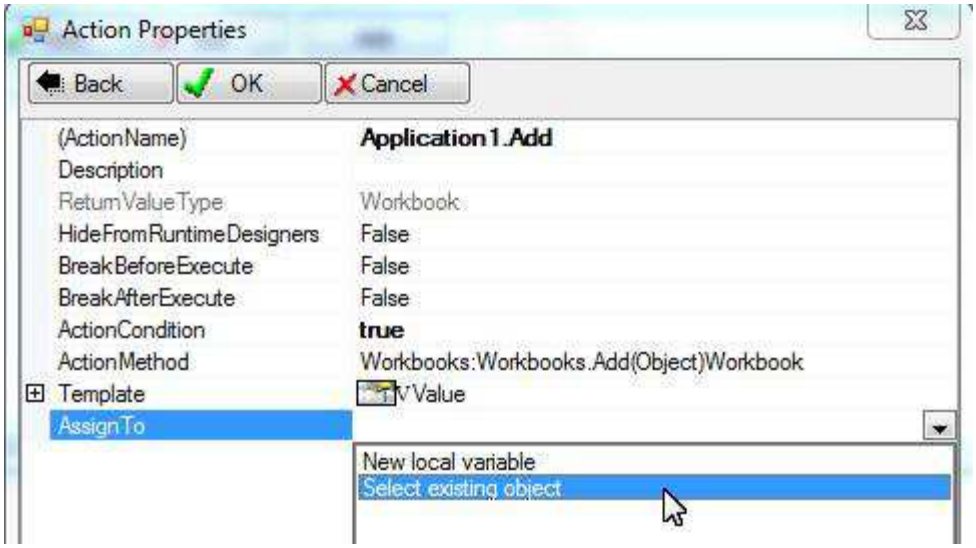


Select “Value” under Missing:

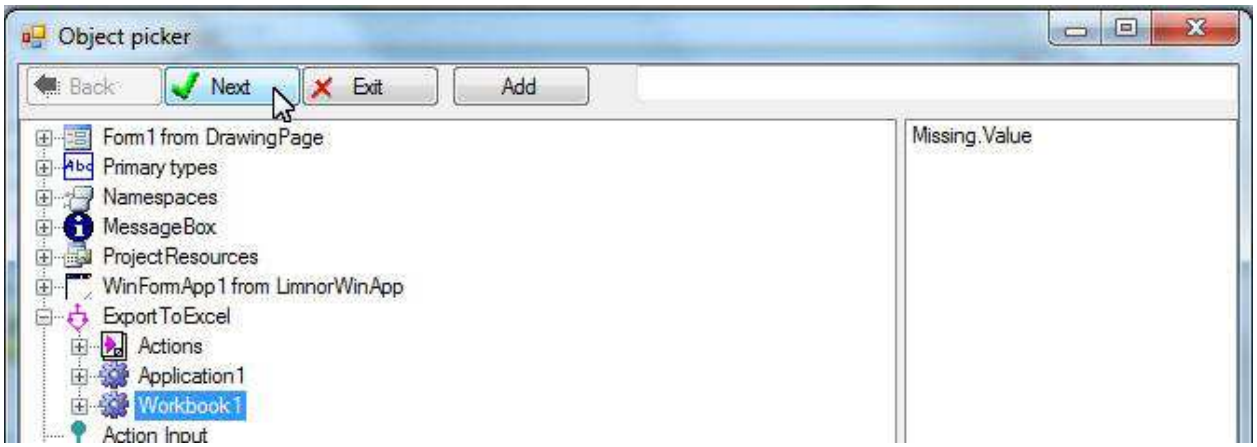


In this way we tell PIA that we do not use this parameter.

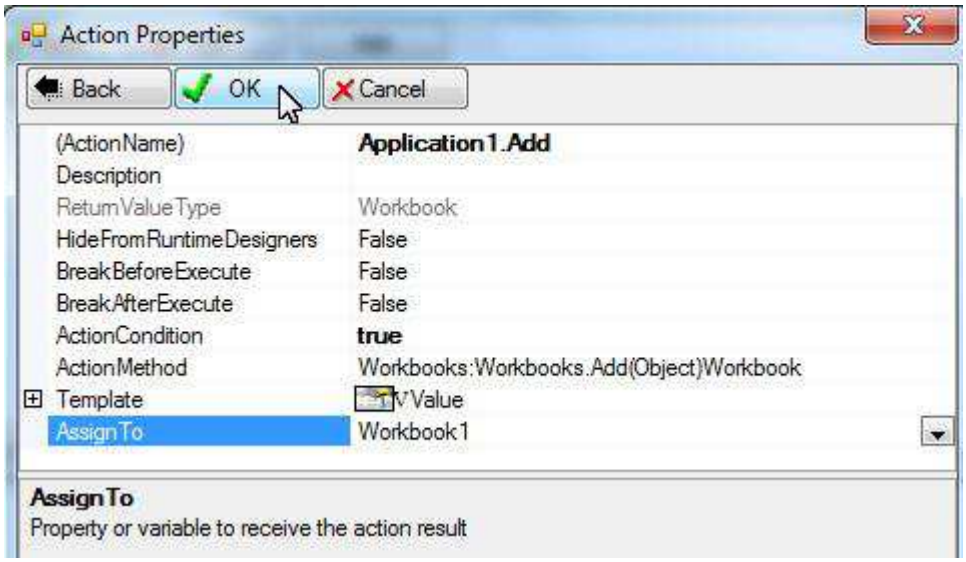
For “AssignTo” property of this action, choose “Select existing object”



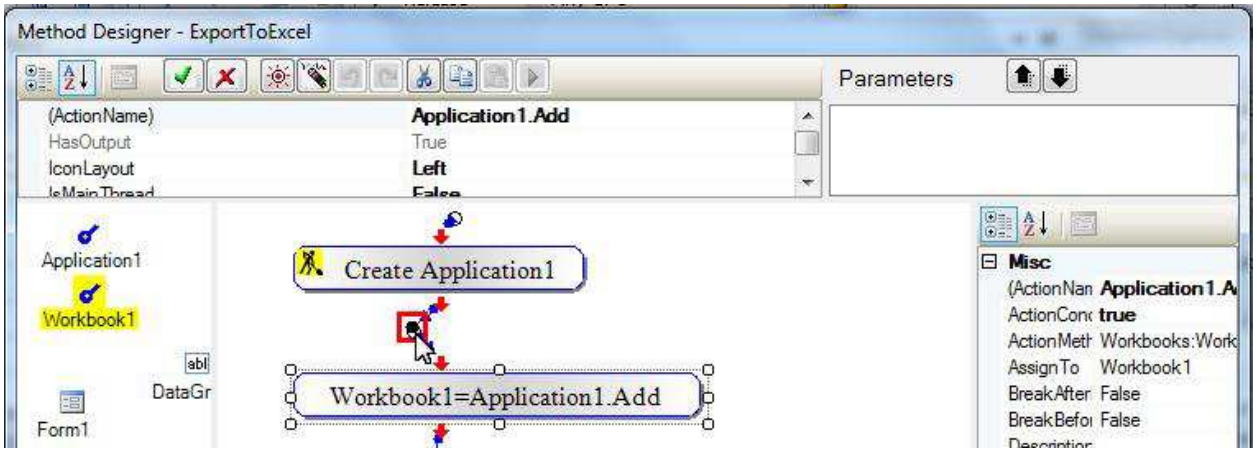
Choose the Workbook variable we already created:



Click OK to finish creating this action:



Link this action to the previous action:



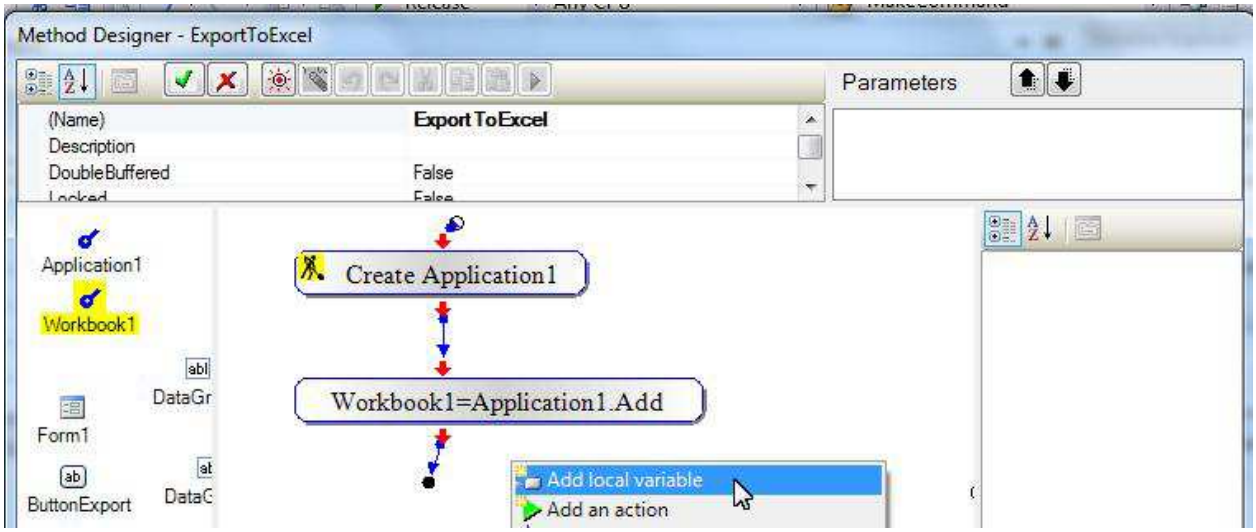
Note the following rules

- Unused parameters in a PIA action cannot be left blank. A special value, Missing.Value, must be used.
- Except the Application object, all other PIA object should not be constructed manually. Some actions can be used to return PIA objects. Specify “AssignTo” property of such an action to a PIA variable to get access to a PIA object.

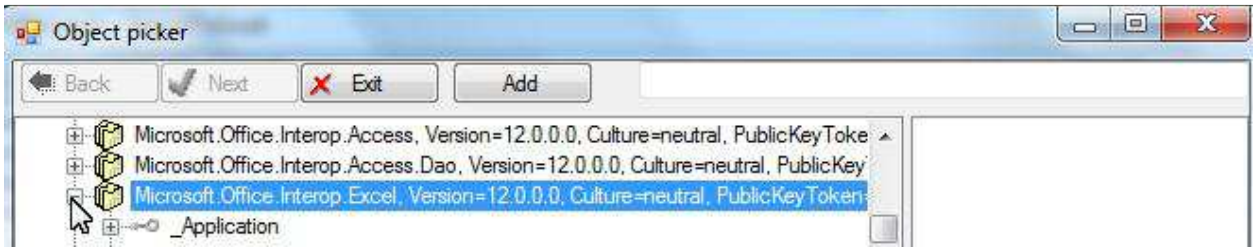
Create Worksheet

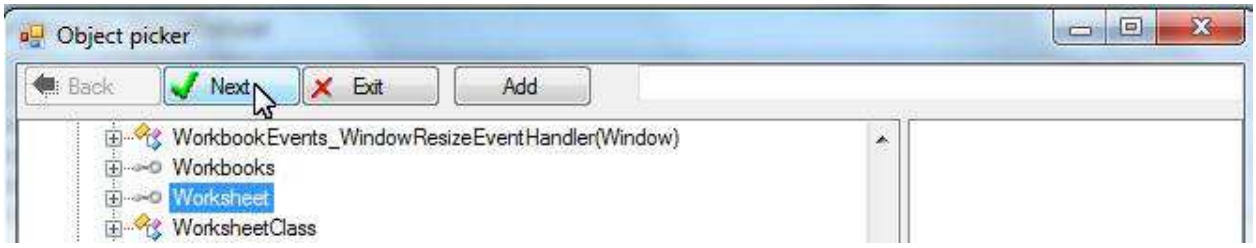
Now we have created a Workbook object. We may use it to created Worksheet.

We must use “Add local variable” to create a Worksheet variable first. Right-click the Method Editor; choose “Add local variable”:

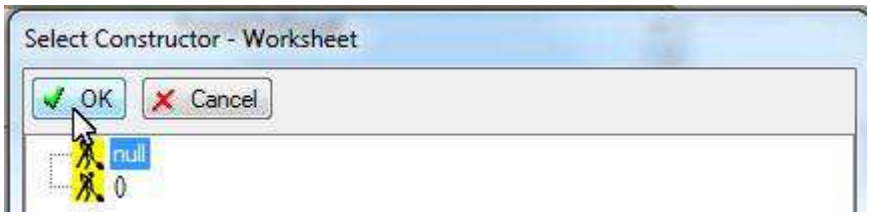


Select Worksheet in the Global Assembly Cache:

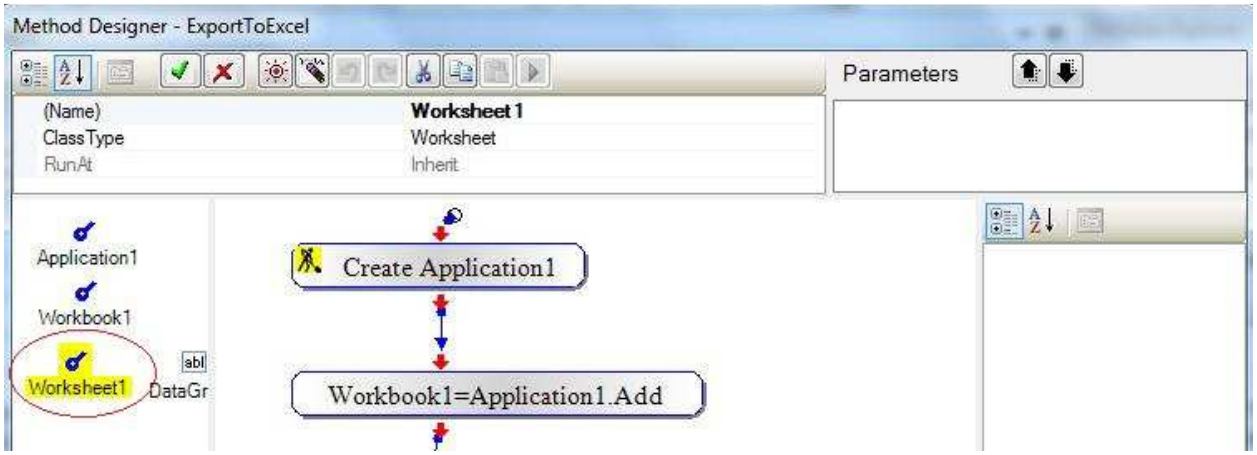




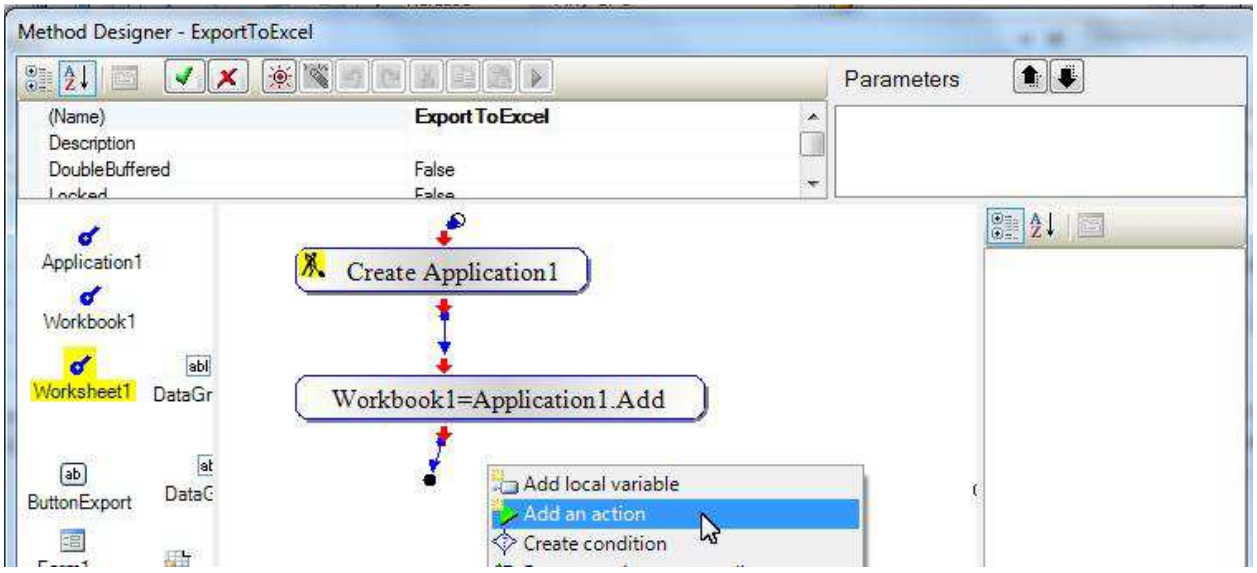
Select “null” because we should not construct an instance of Worksheet manually. We need to let the Workbook object to create Worksheet:



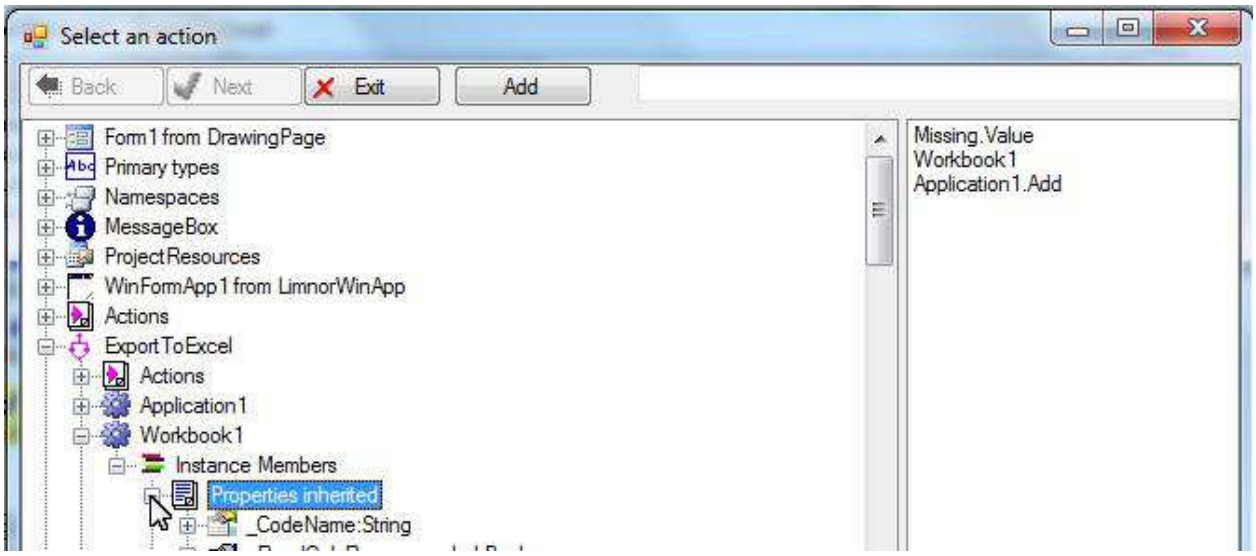
A Worksheet variable appears:

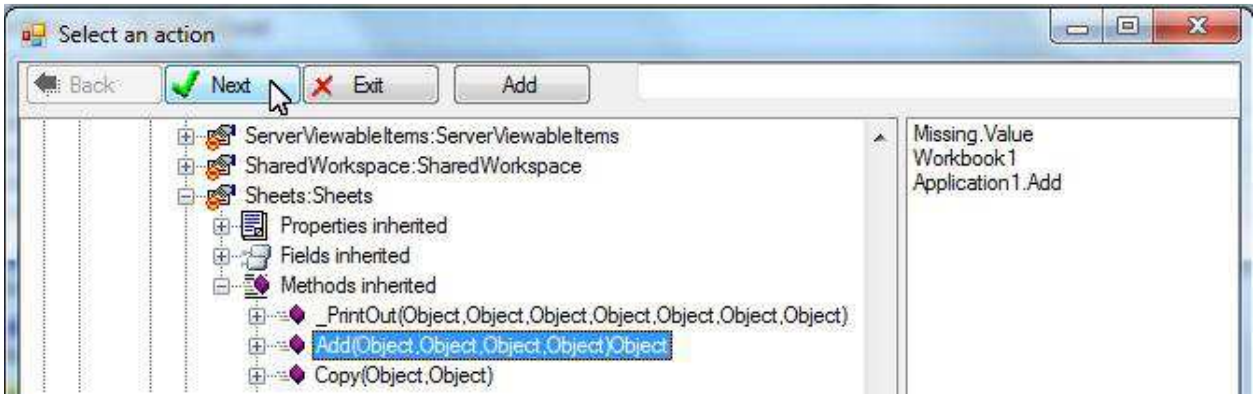


We will use the Add method of the Sheets property of the Workbook object to create a new Worksheet. Right-click the Action-Pane; choose “Add an action”:

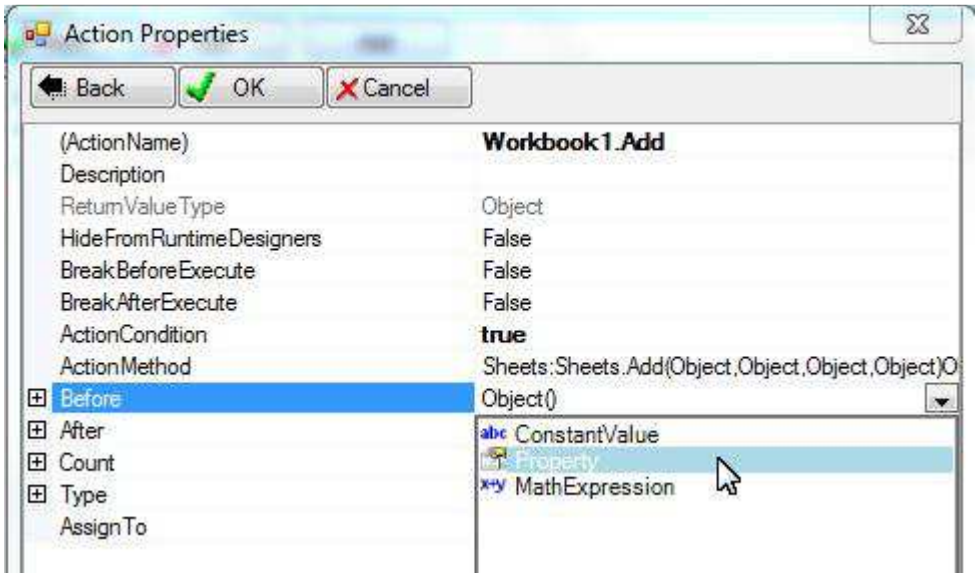


Locate the Sheets property of the Workbook object and select its Add method:

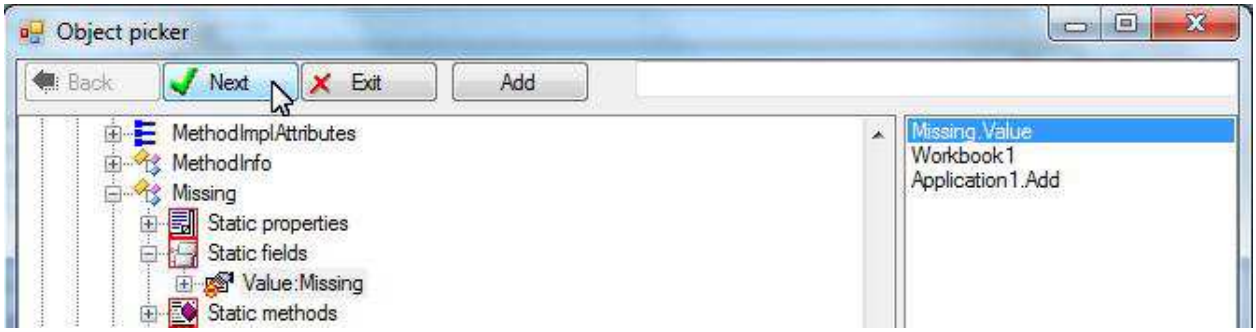




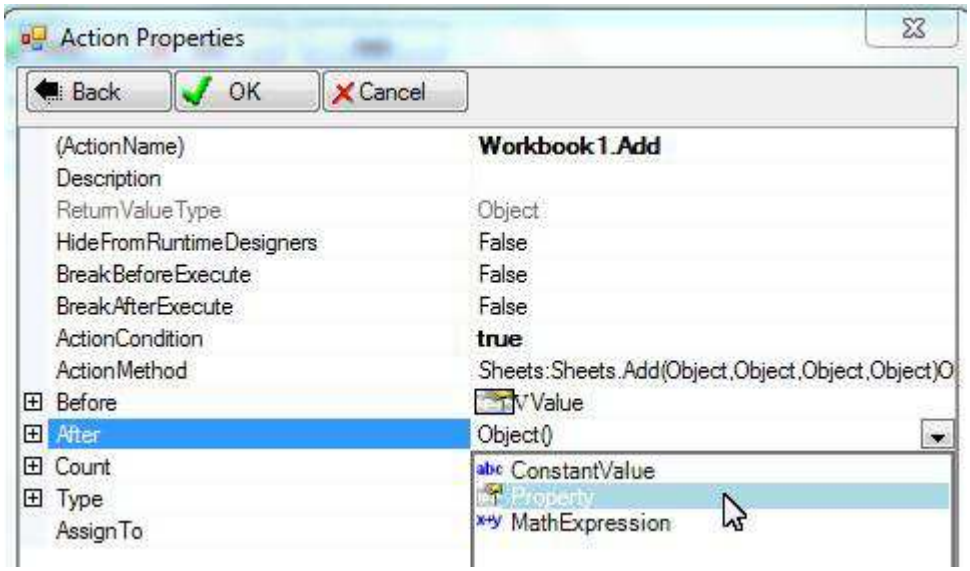
This action has parameters “before”, “after”, “count”, and “type”. For this sample, we do not use these parameters. We need to provide a “Missing.Value” for them. Select “Property” for each parameter:



Since we already used “Missing.Value” before, we may select the “Missing.Value” in the list:

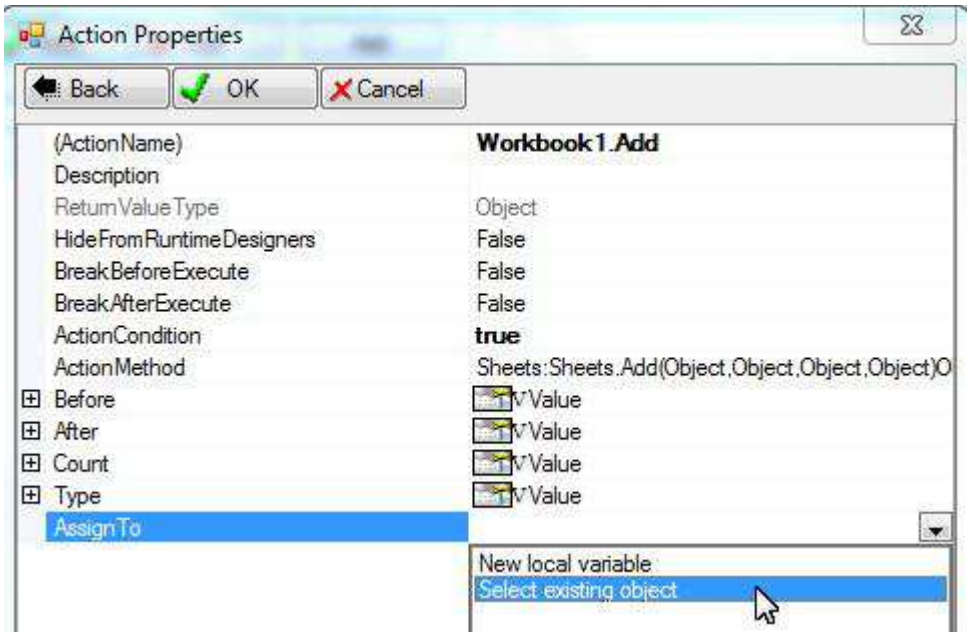


Do it for all the parameters:

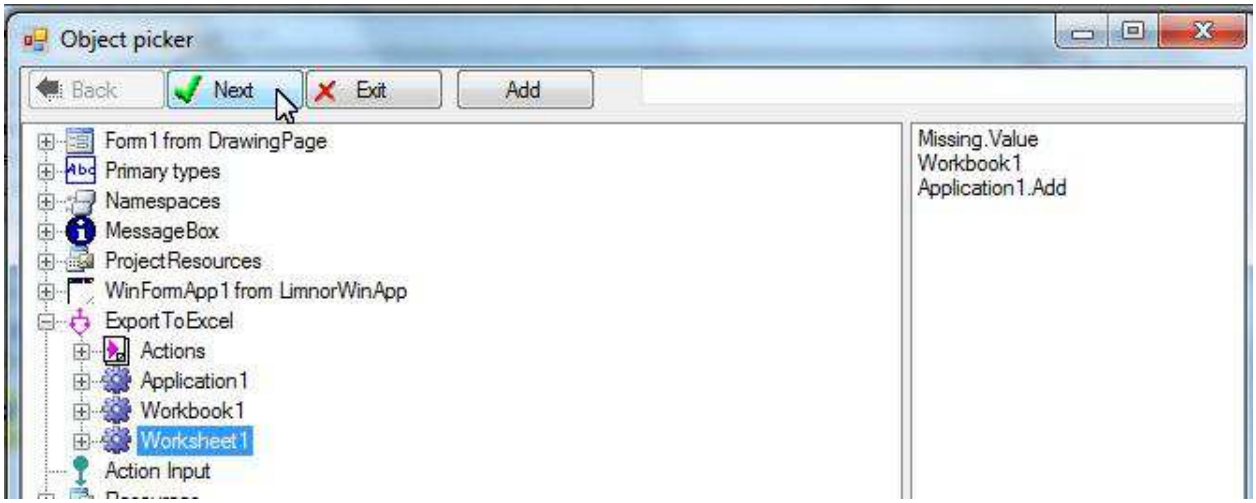


In this way we tell PIA that we do not use these parameters.

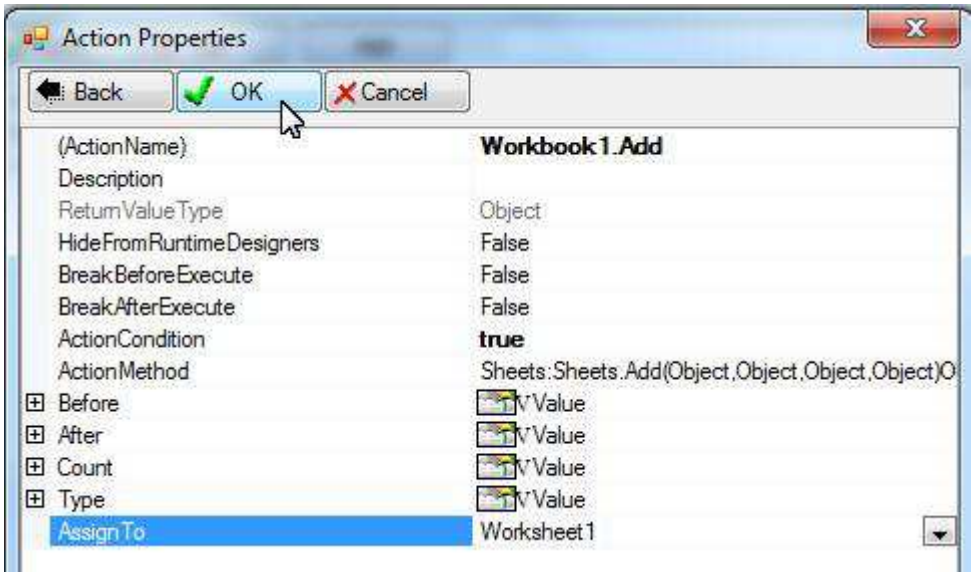
For “AssignTo” property of this action, choose “Select existing object”



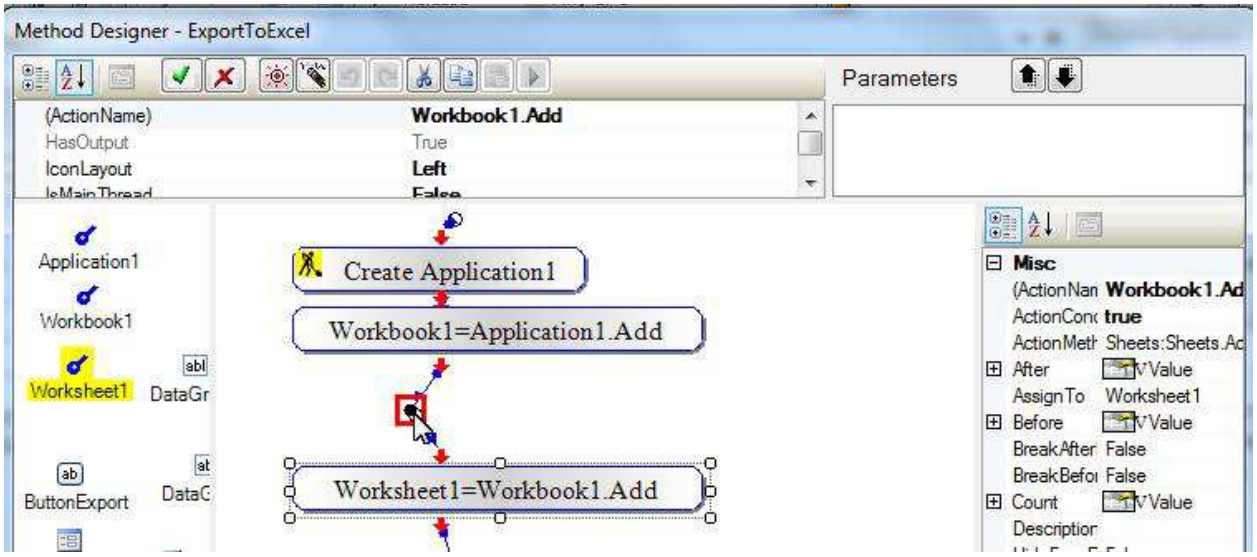
Select the Worksheet variable we already created:



Click OK to finish creating this action



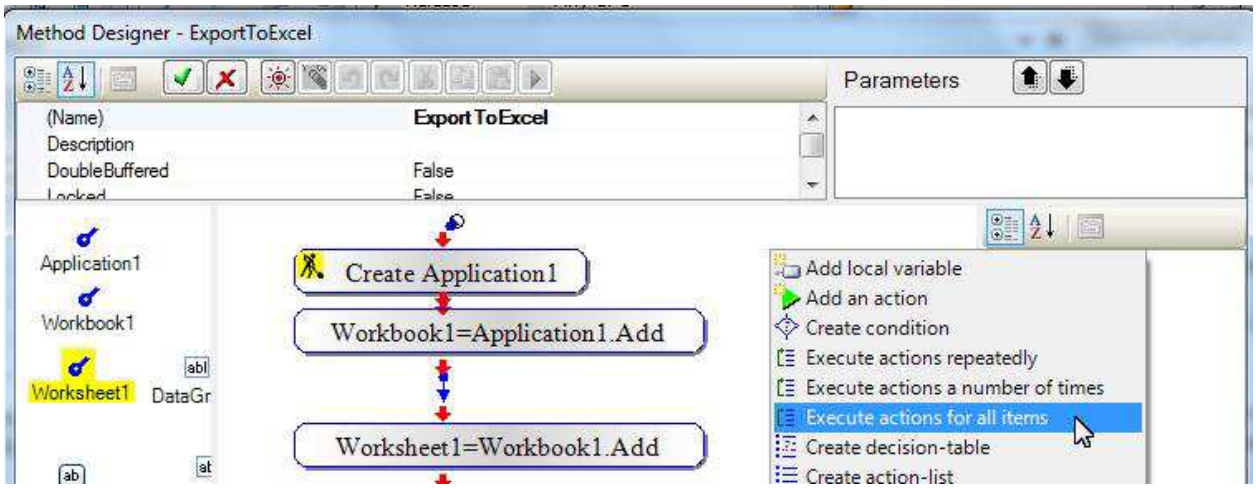
Link this action to the previous action:



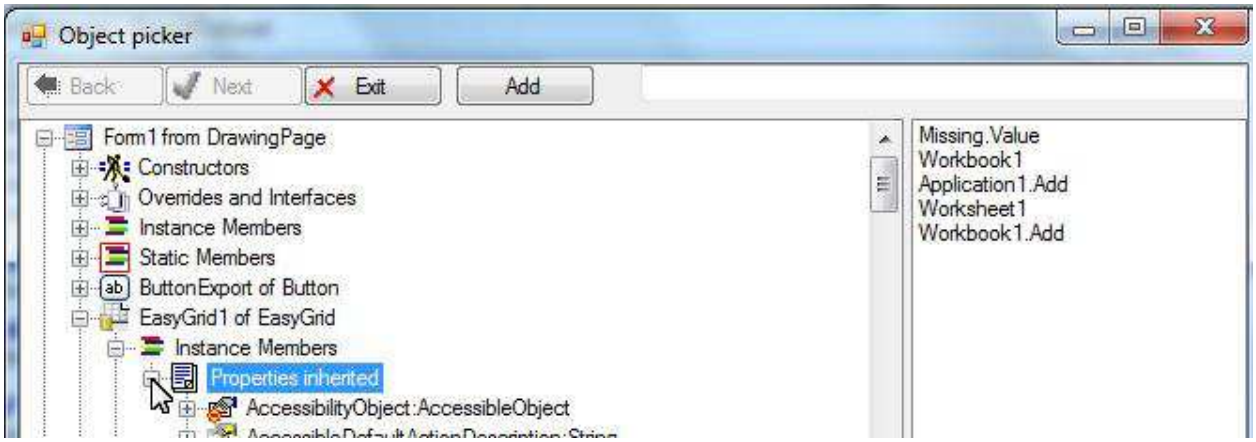
Export Data Grid Headers

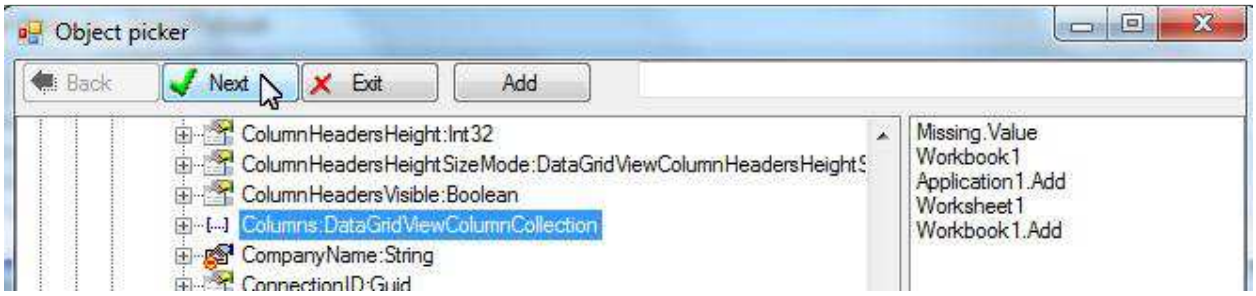
Now we have a work sheet. We may export data to it.

To export data grid headers to the sheet, right-click the Action Pane, choose “Execute actions for all items” to act on all columns:



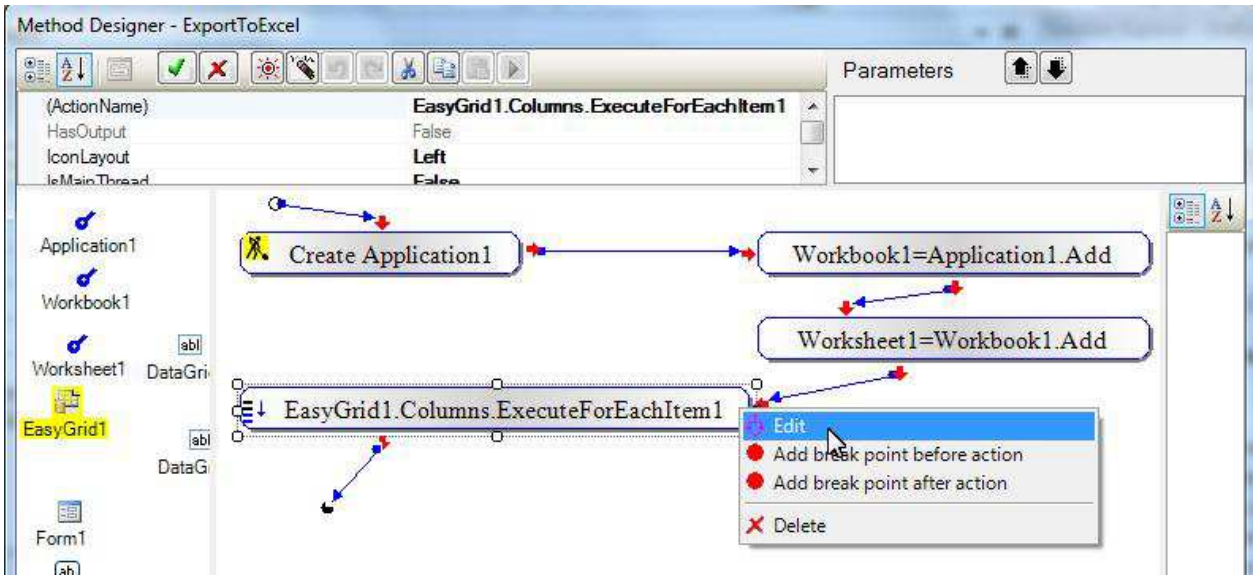
Choose Columns property of the grid:



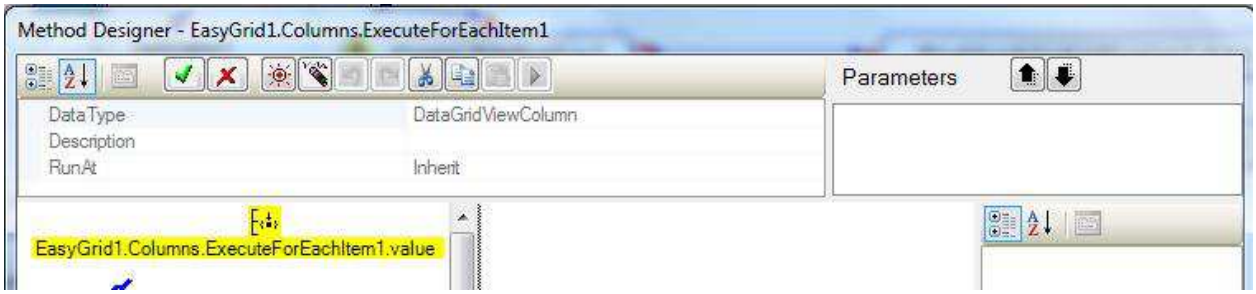


The action appears in the Action Pane. Link it to the last action.

Right-click the action; choose “Edit” to specify actions to work on columns:



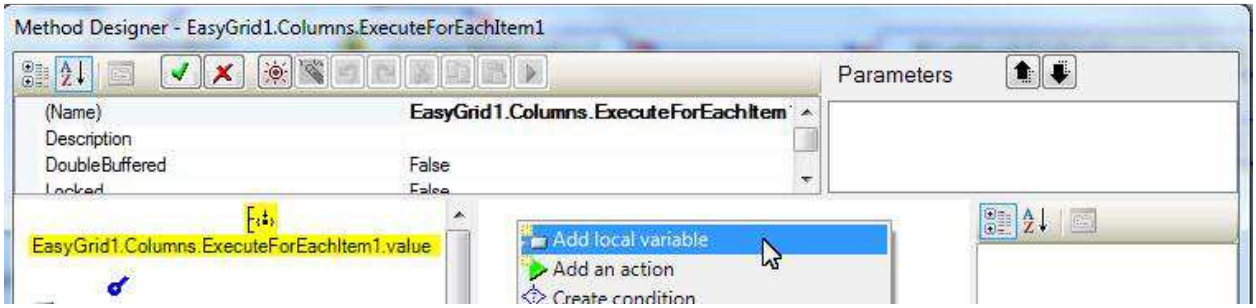
A new Method Editor appears for specifying actions to be executed for every column; the “value” icon represents the column:

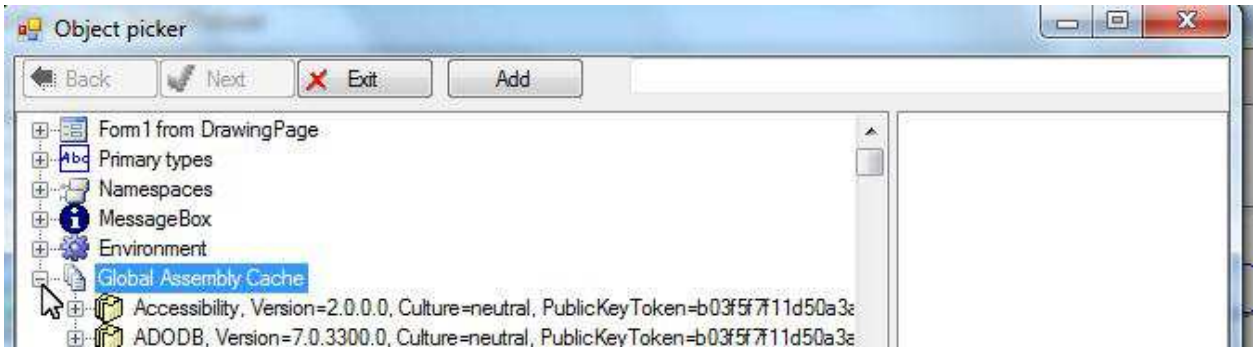


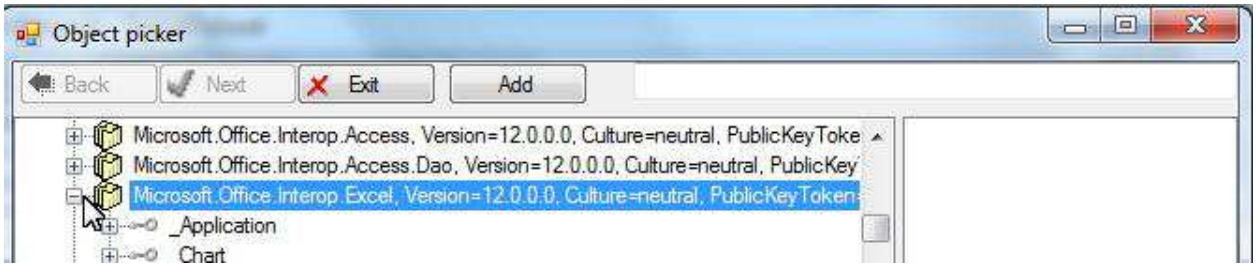
The “value” has a “HeaderText” property. We want to export it to a cell in the sheet. The cell should be at the first row. The row number for the cell is 1, representing the first row. The column number for the cell is the column index plus 1 because the column index starts from 0, not 1.

The cell can be retrieved via a “get_Item” method of the Cells property of the sheet.

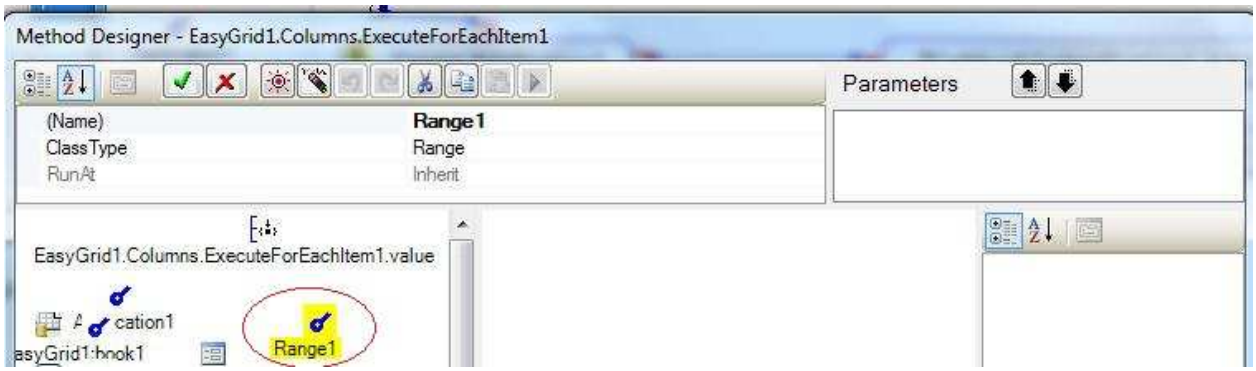
A cell is represented by a Range object. Following the rules, we first create a Range variable; then we use “get_Item” to get the Range object.



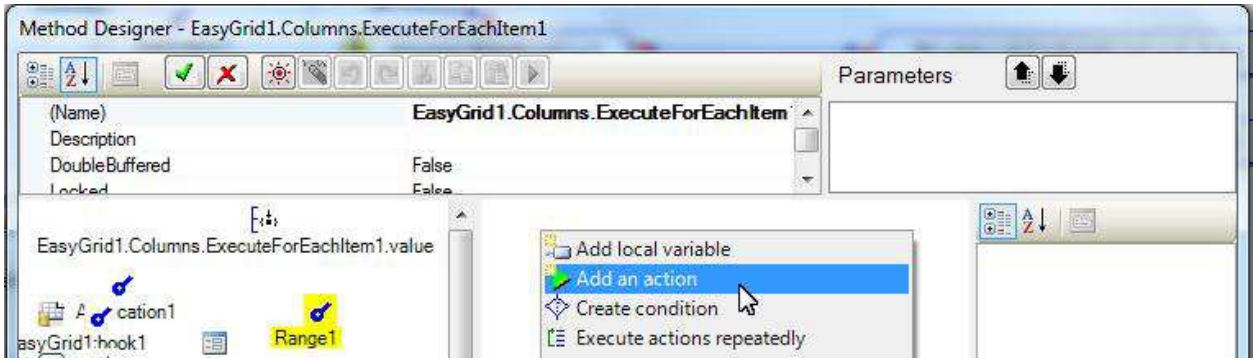




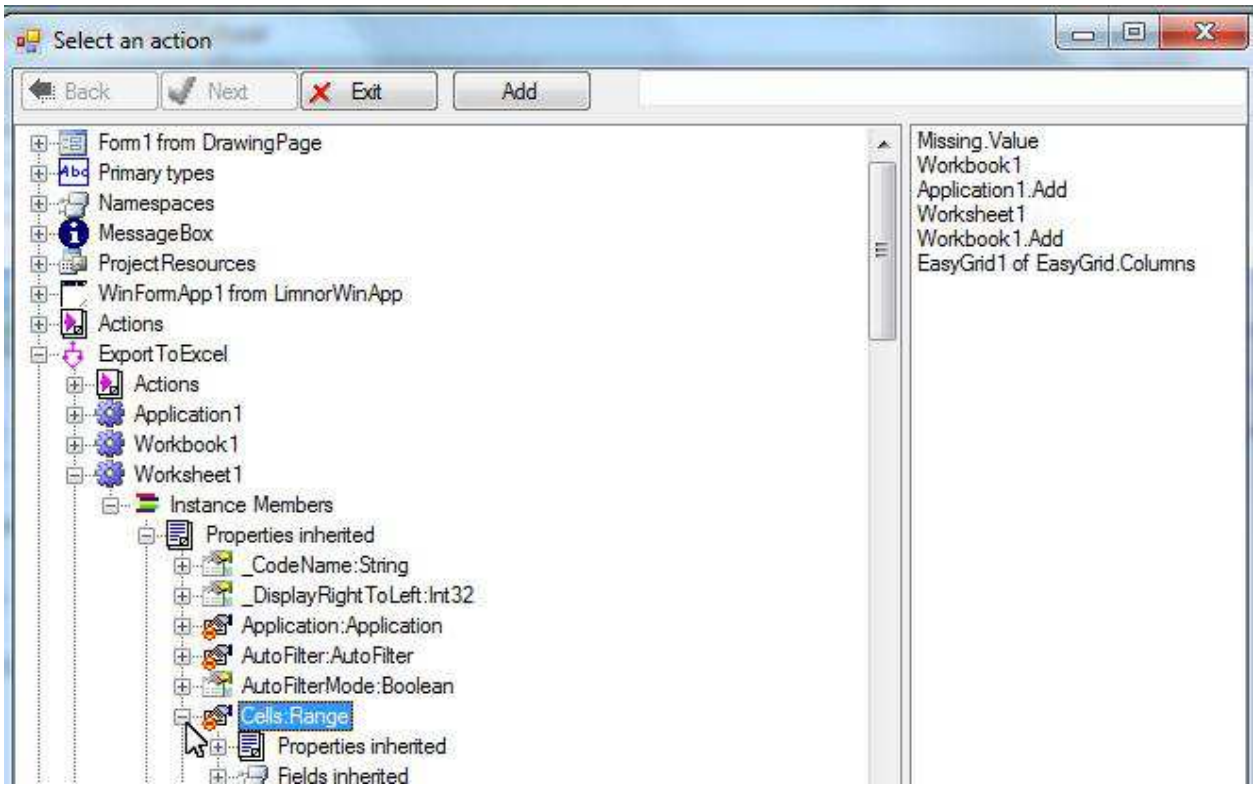
A Range variable appears in the Variable Pane:



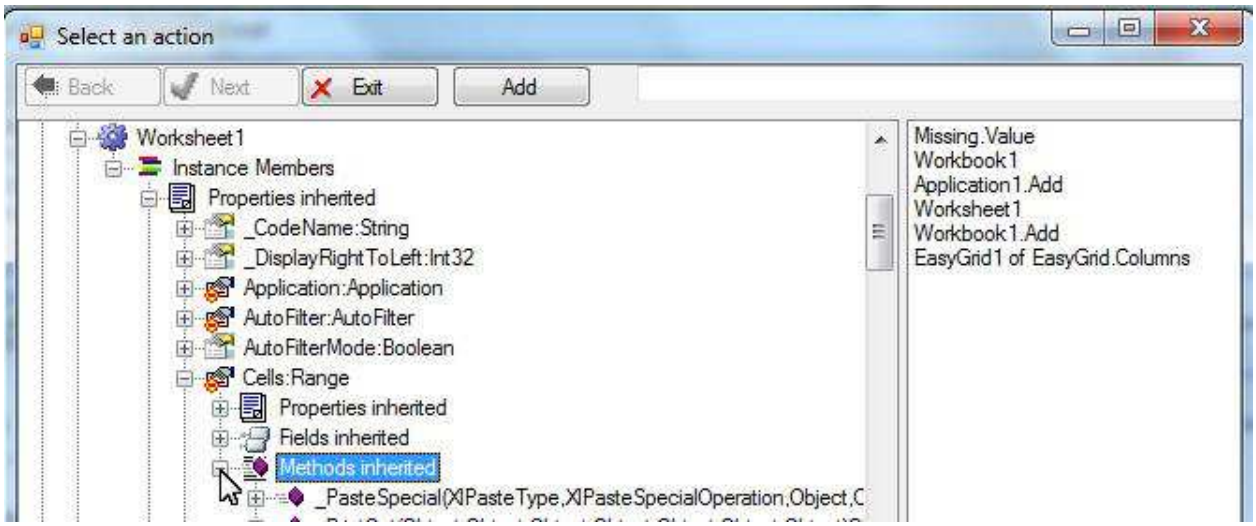
Now we may use the “get_Item” of the cells property of the sheet to get the Range. Right-click the Action Pane; choose “Add an action”:



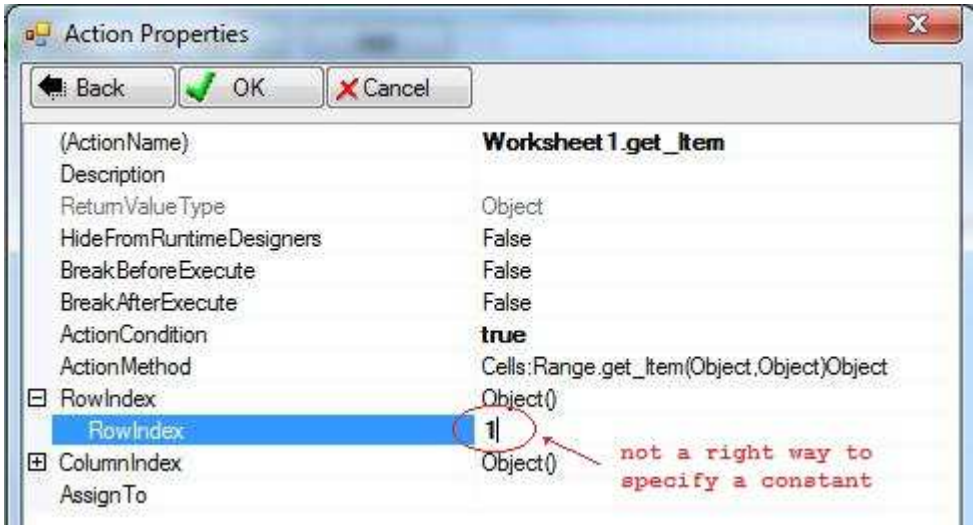
Find the Cells property of the sheet:



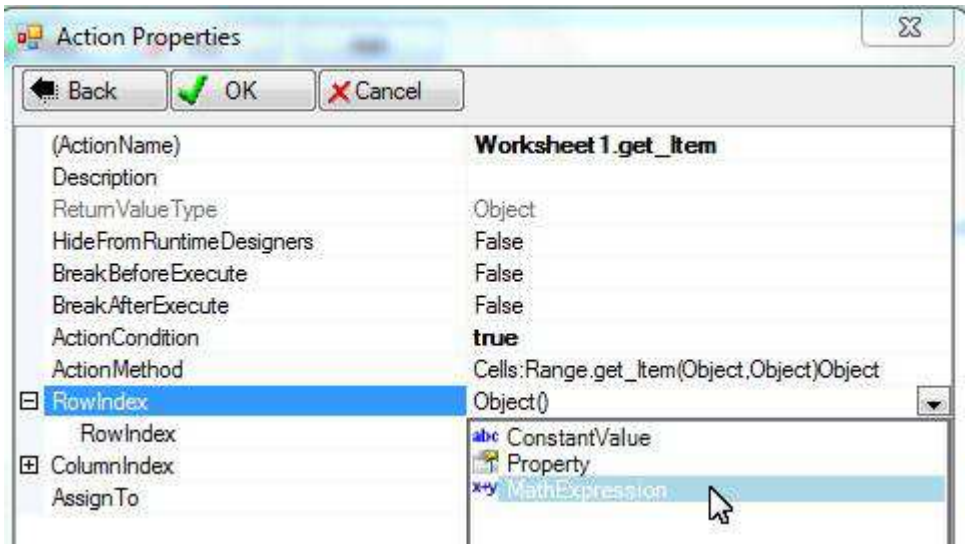
Select get_Item:



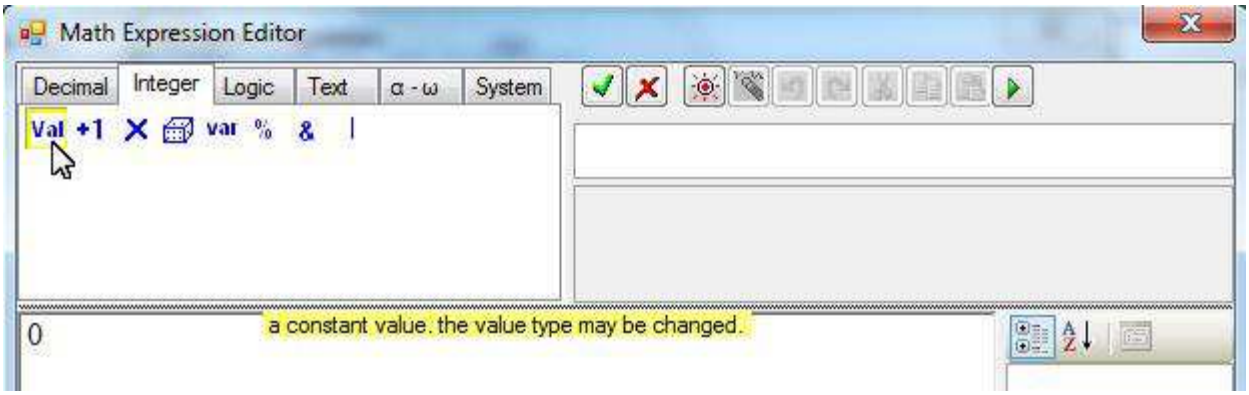
We need to specify RowIndex and ColumnIndex for “get_Item” to get the cell. In our case we want RowIndex to be 1, indicating the first row. But if we specify 1 in the action then PIA will not give us the result we want. The reason is that PIA uses Object as the data type for the parameter, instead of correct data type of Integer.



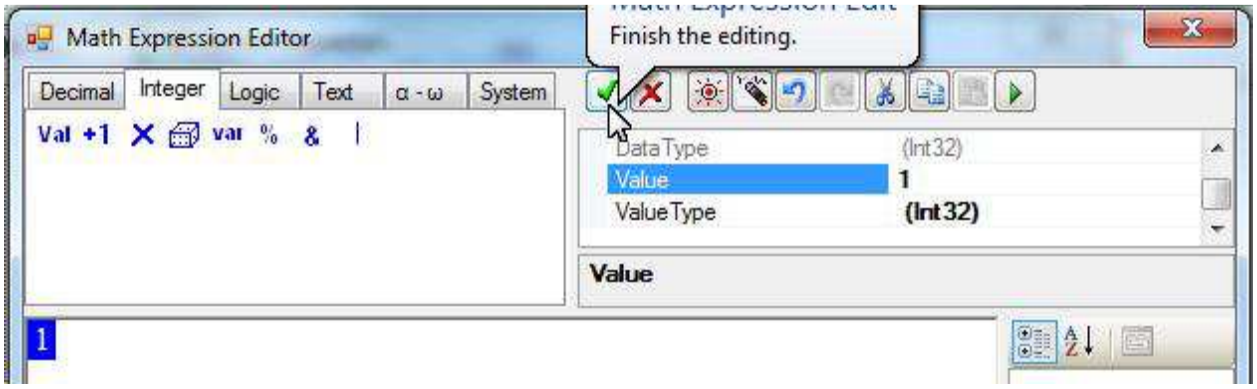
To get around this problem, choose “Math Expression” for the property:



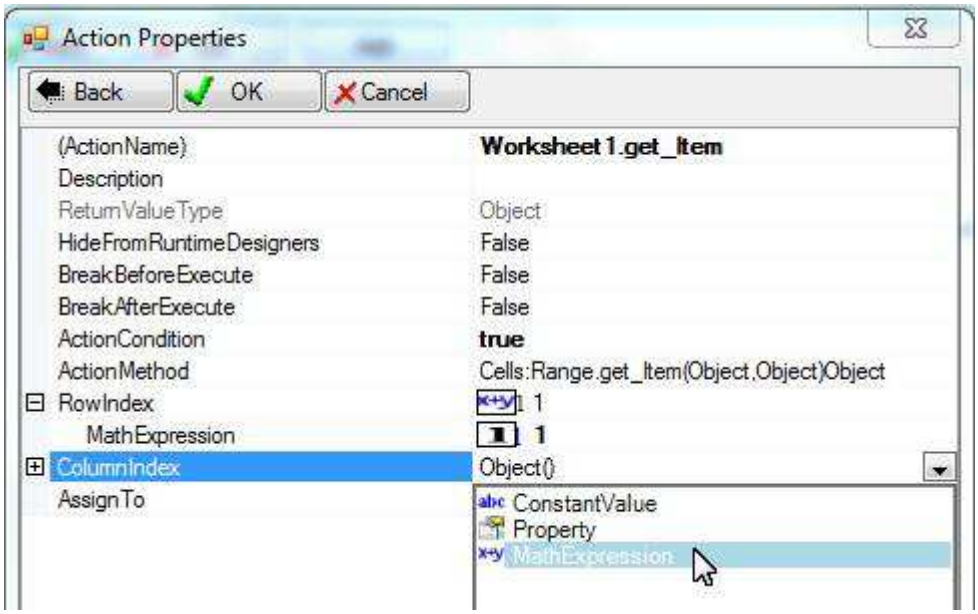
Select an integer constant:



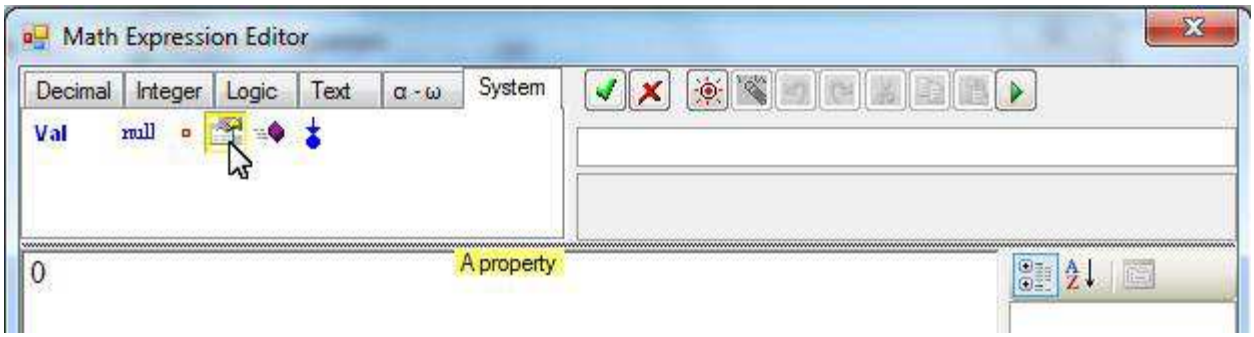
Use 1 as the value:



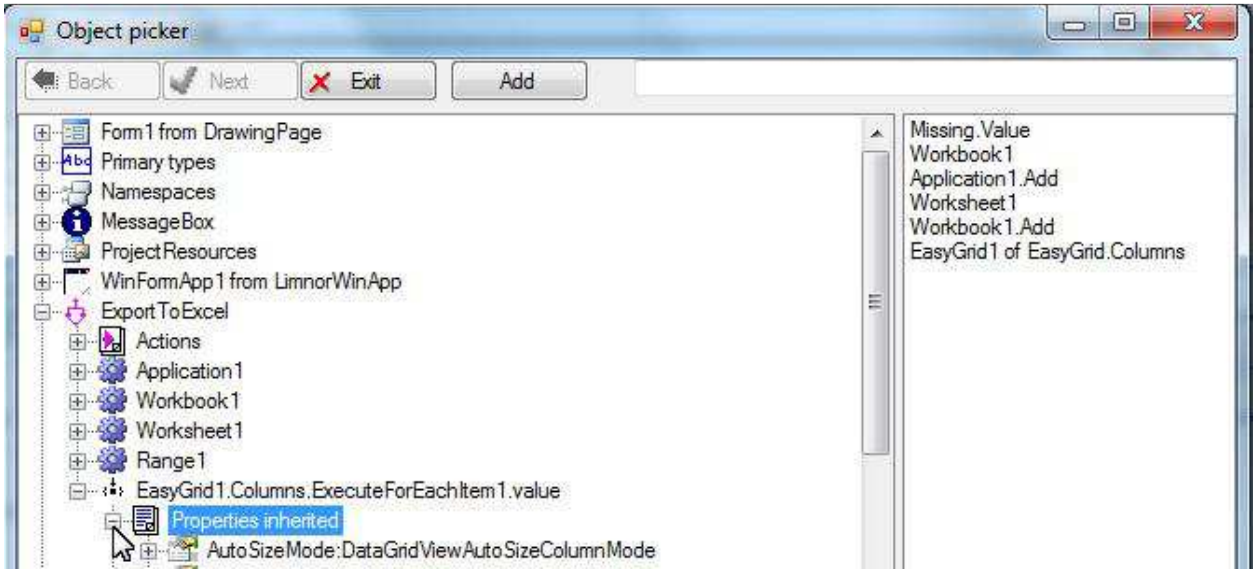
For the ColumnIndex, we want to use the Index property of the column. We might choose “Property”. But because we need a calculation, we have to choose “Math Expression”:

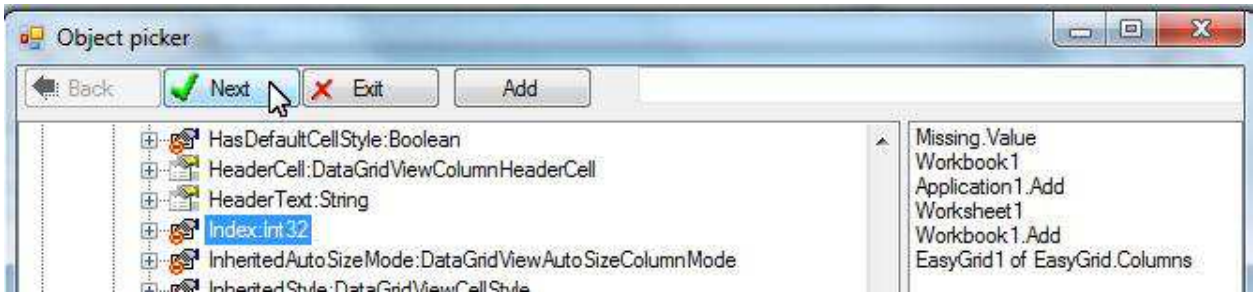


We want to use the Column index of the column being worked on. So, choose Property icon:

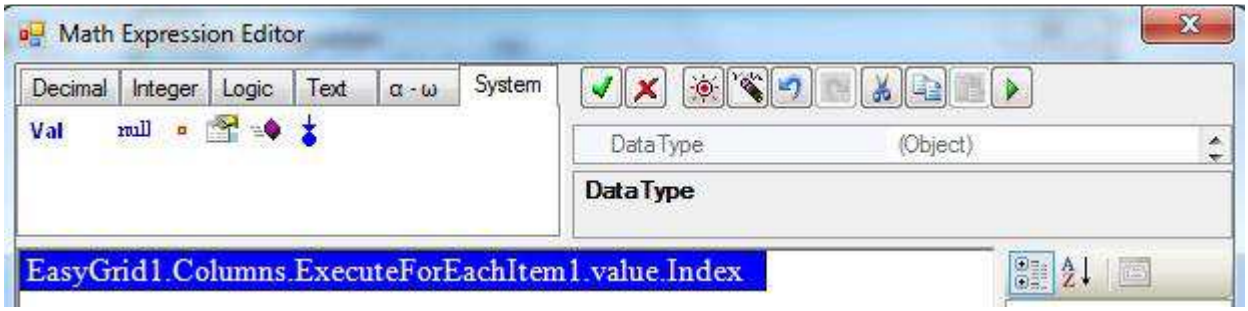


Select the Index property of the “value”:

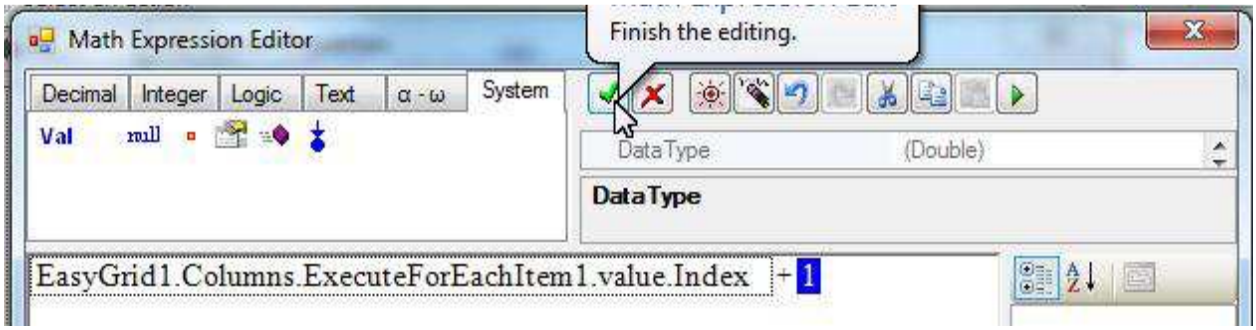




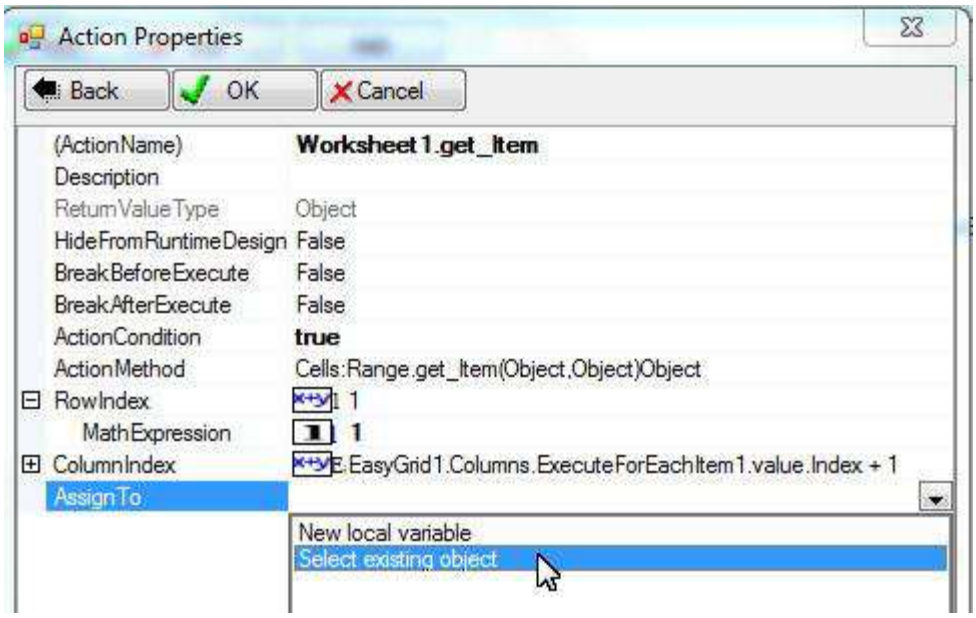
The Index property appears in the expression:



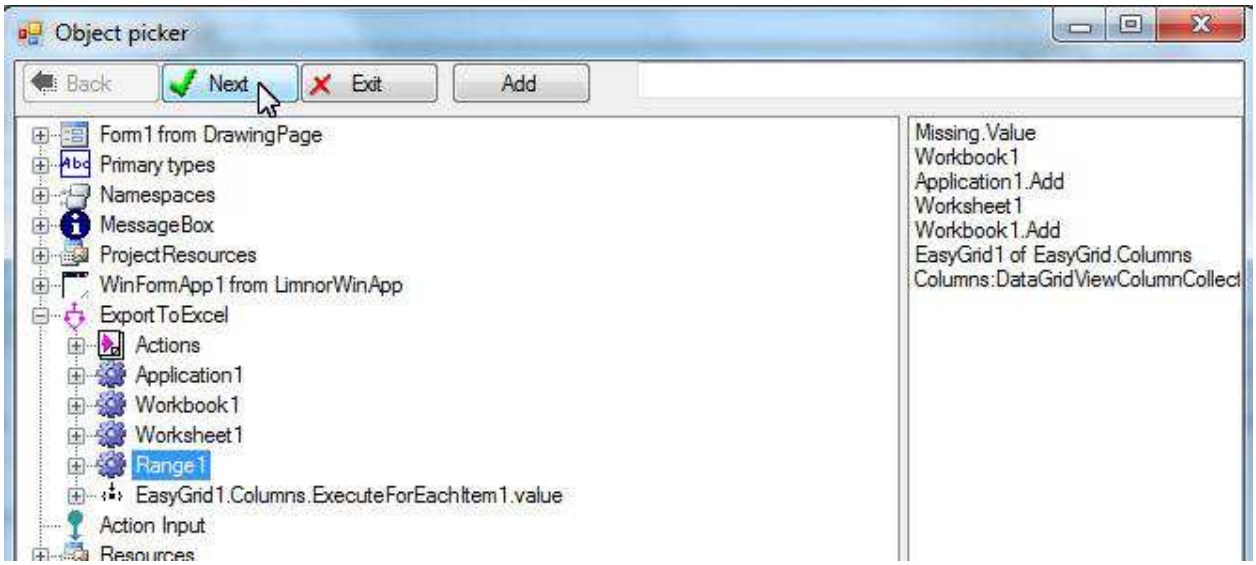
But the column index of the grid starts from 0 and column index of the Excel sheet starts from 1, we must add 1 to the grid column index. So, type "+1" to the expression:



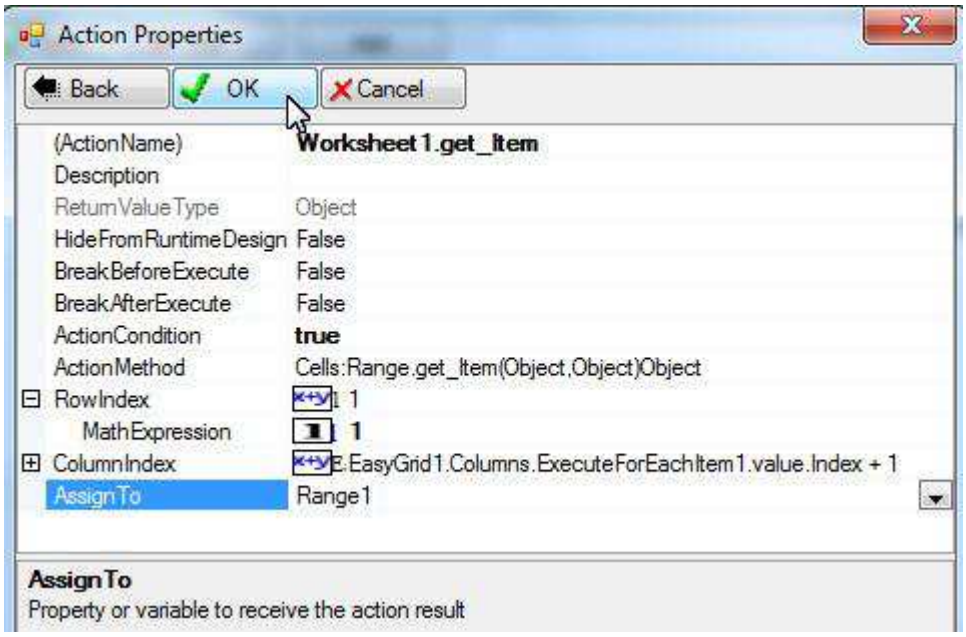
For the "AssignTo" of the action, choose "Select existing object":



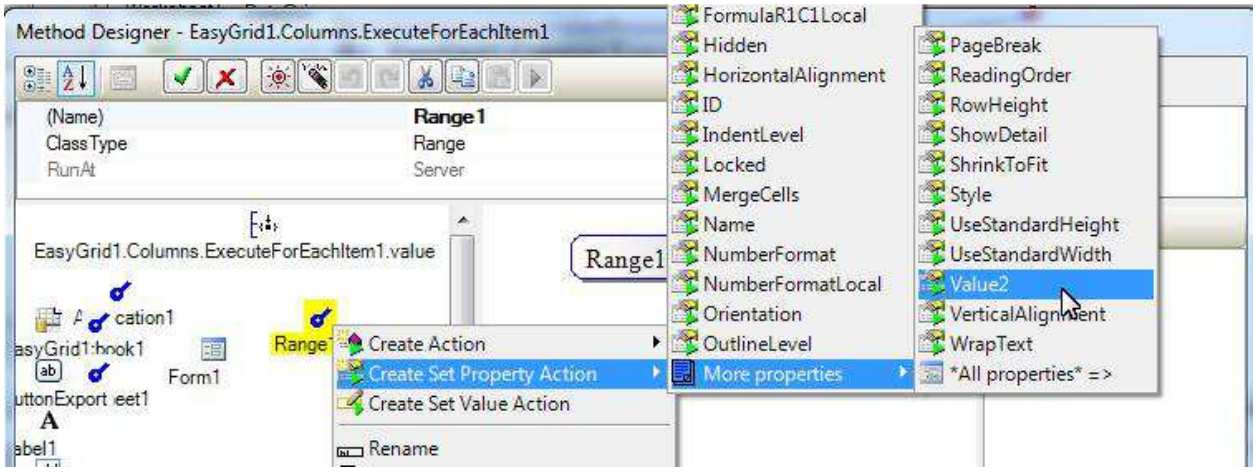
Choose the Range variable:



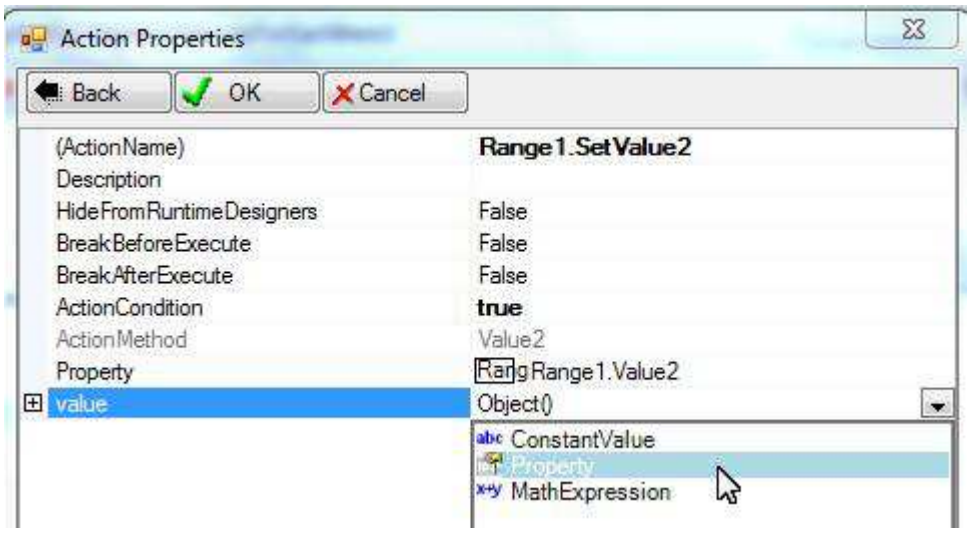
Click OK to finish creating this action:



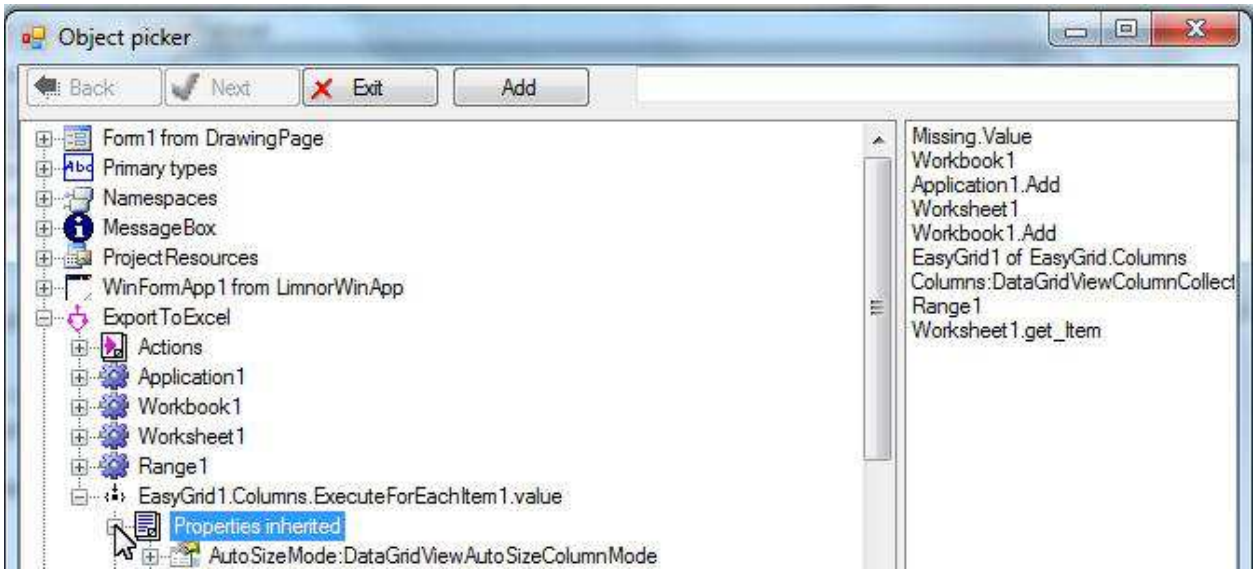
This action gives us the Excel cell corresponding to the grid column. The cell is represented by variable Rang1. We may set its Value2 property to the HeaderText of the column. Right-click Range1; choose "Create Set Property Action"; choose Value2:

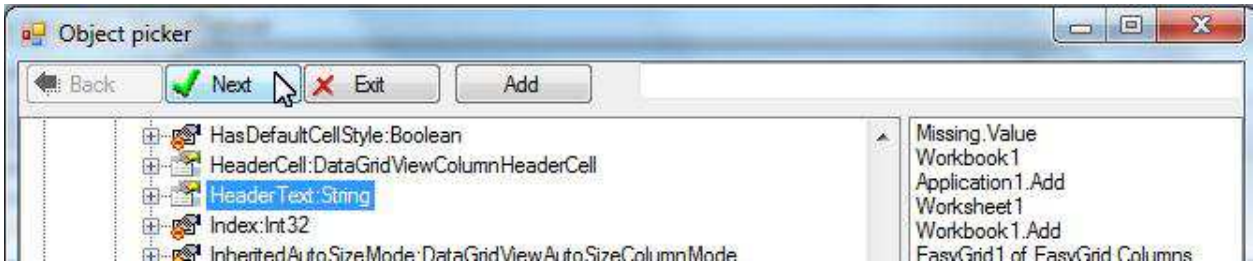


Select Property for the action parameter "value":

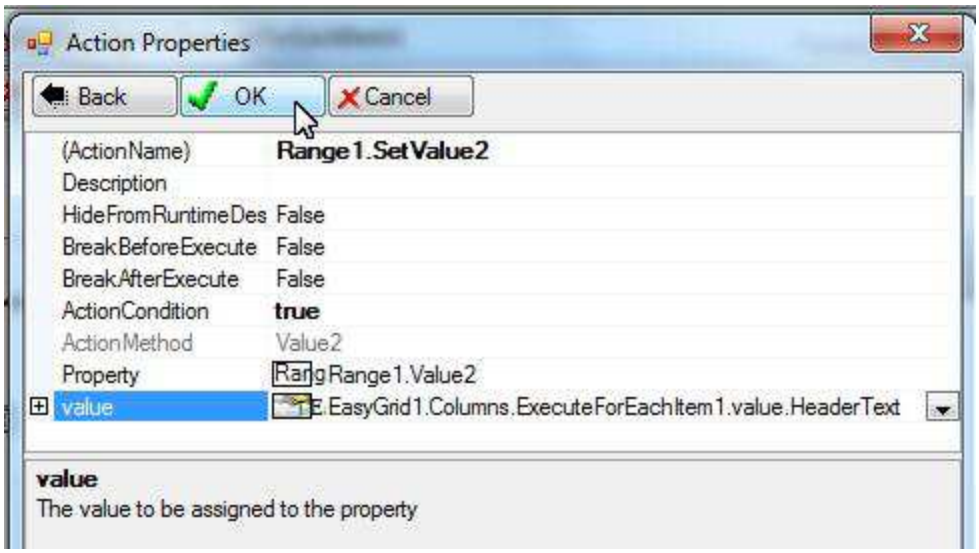


Select HeaderText property of the column:

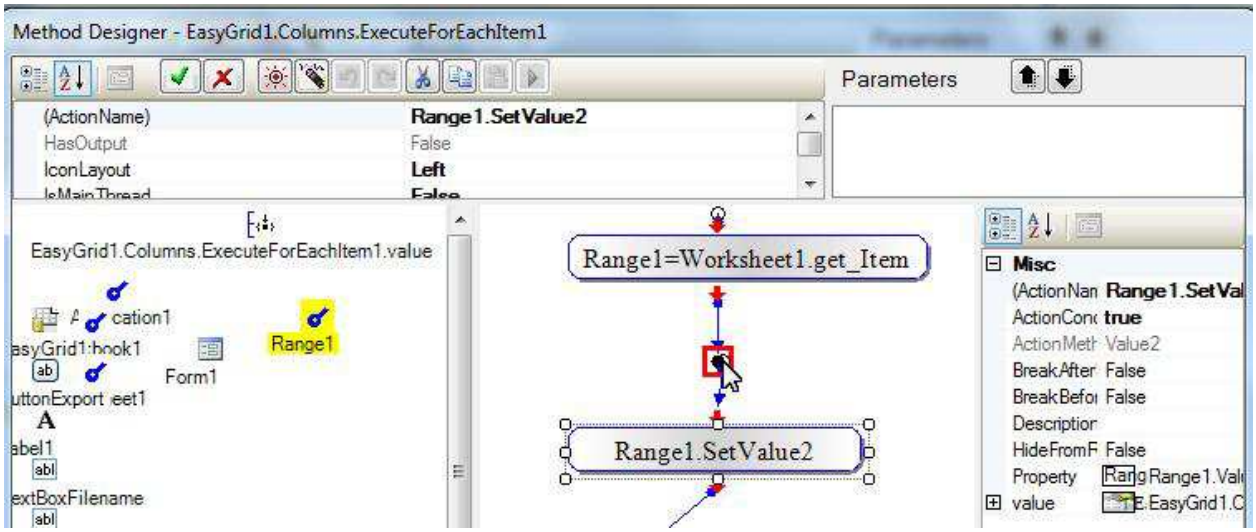




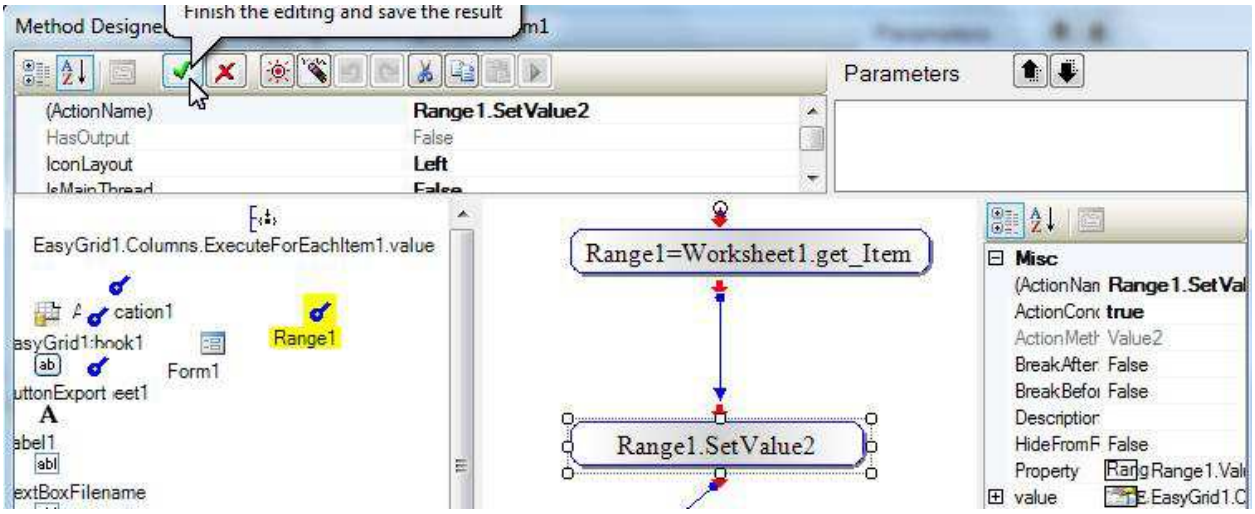
Click OK to finish creating this action:



Link the two actions:



That is all we need to do for each column:



Note the following rules

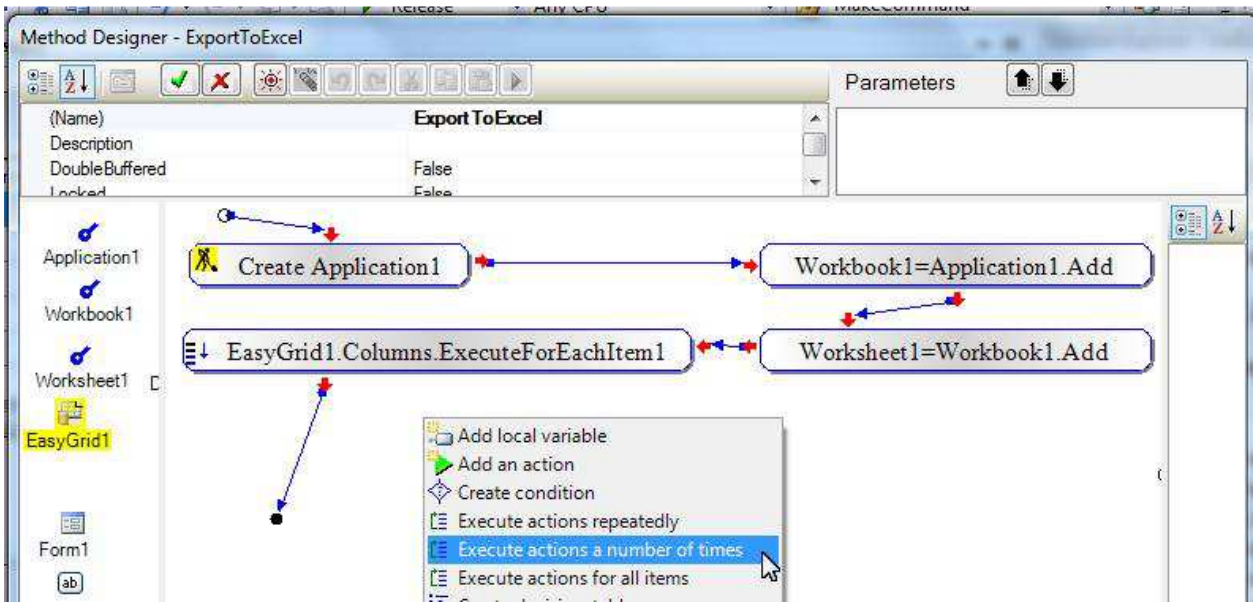
- Row index and column index of Excel cells start from 1, not 0.
- Do not use constants for PIA action parameters which are not strings. Use Math Expression to specify constants of correct types.

Export Data Grid Row to Excel Row

An EasyGrid has a DataRows property. It holds the rows of data for the grid. We may pass each row to a row in Excel sheet.

Create Loop Action

To work on every row, create an action of “Execute actions a number of times”:

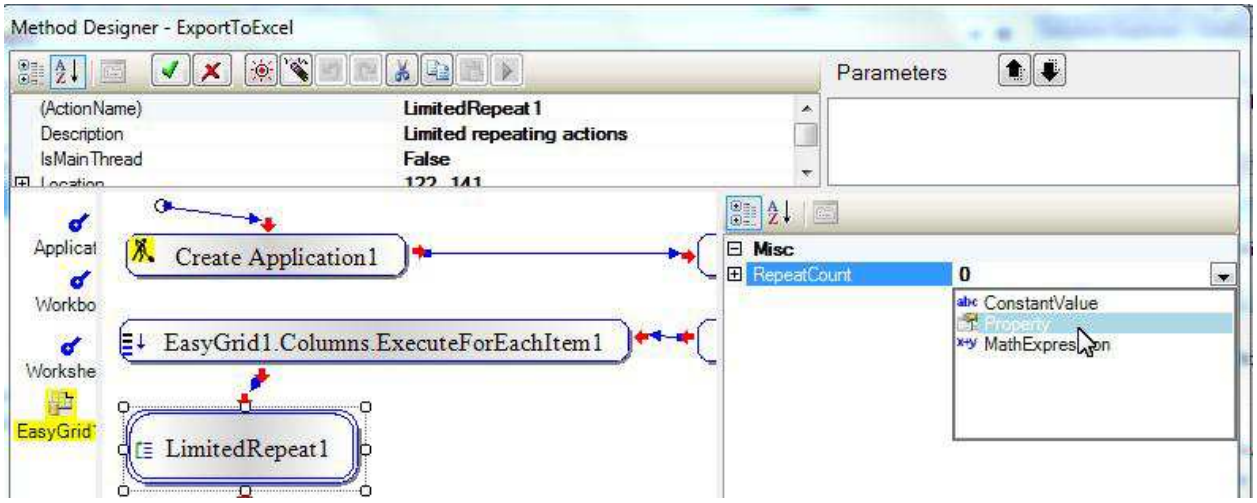


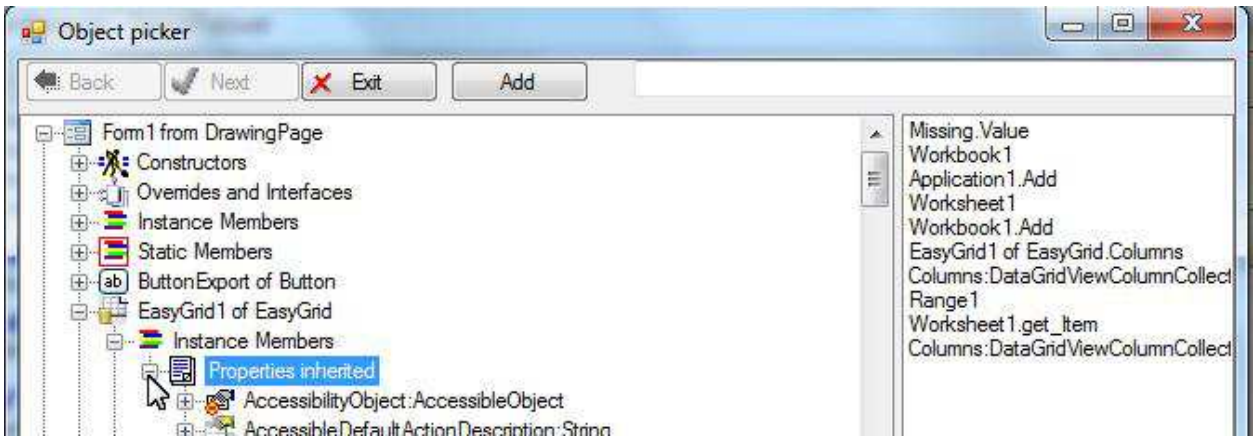
You may notice that when we work on every column, we used “Execute actions for all items”, not “Execute actions a number of times”. The differences are listed below

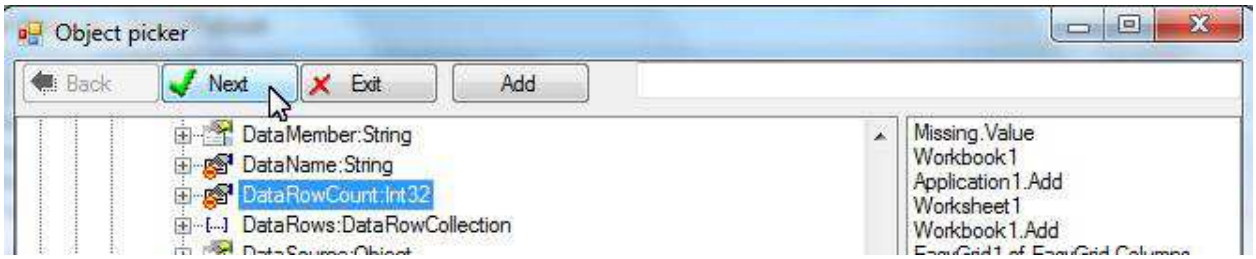
- “Execute actions a number of times” gives us the index of iteration. “Execute actions for all items” does not use iteration index.
- “Execute actions for all items” provides the item object for each iteration. “Execute actions a number of times” does not provide the item object for iteration; we must use the iteration index to retrieve the item object.

We need row index to specify the cells. Because a DataRow object does not have an Index property, we have to use “Execute actions a number of times” so that we may use the iteration index as the row index. A Column object of the grid has an Index property and thus we do not need to use “Execute actions a number of times”.

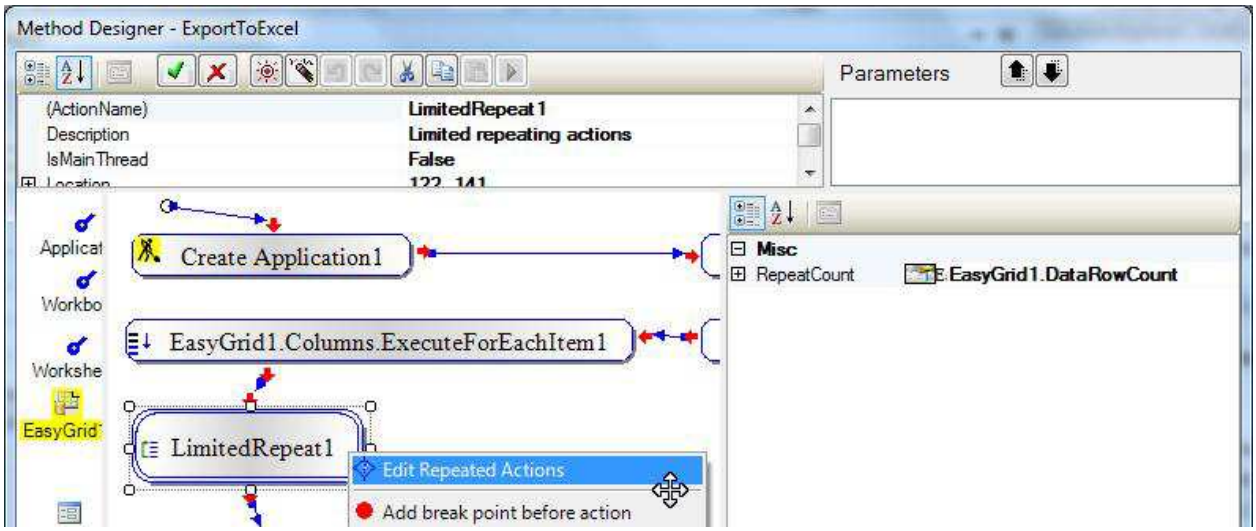
We need to set the RepeatCount of this action to the DataRowCount of the grid:



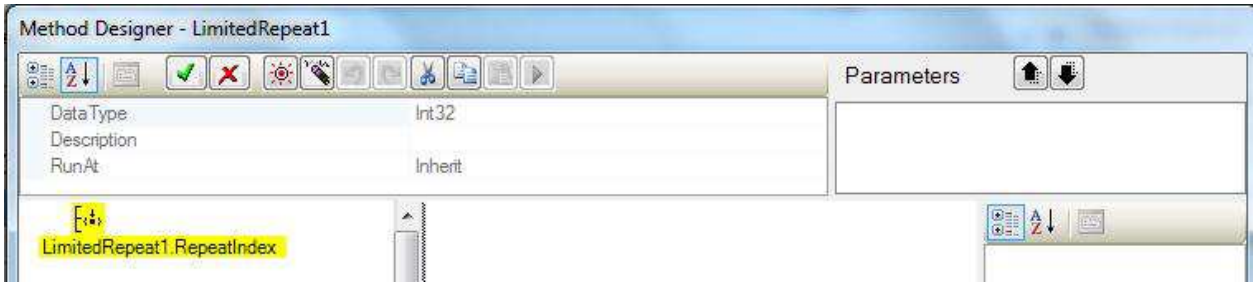




Right-click the action; choose “Edit Repeated Actions” to specify the actions for every row:

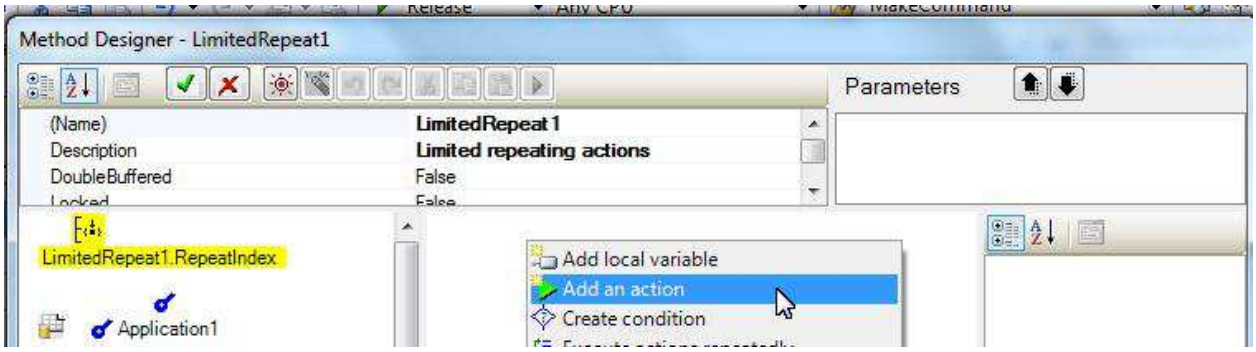


A new Method Editor appears for specifying actions for iteration

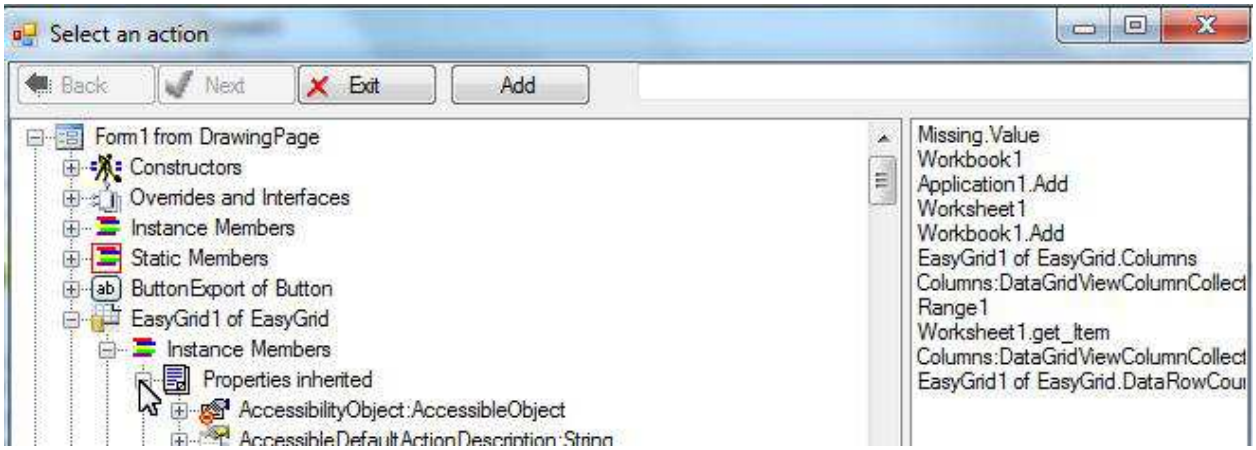


Get Data Row

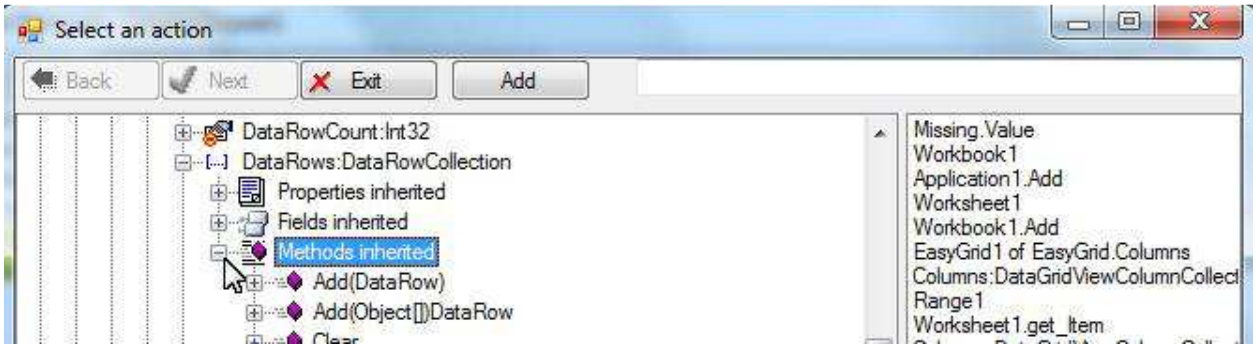
The “RepeatIndex” is the iteration index. We need to use it to retrieve the DataRow object. Right-click the Action Pane; choose “Add an action”:

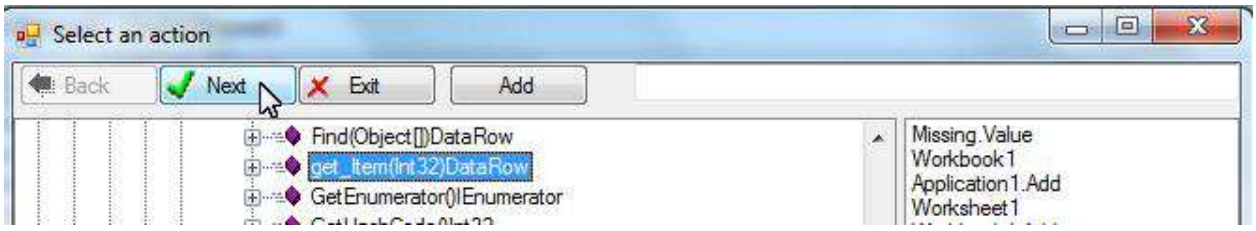


Find DataRows property of the grid:

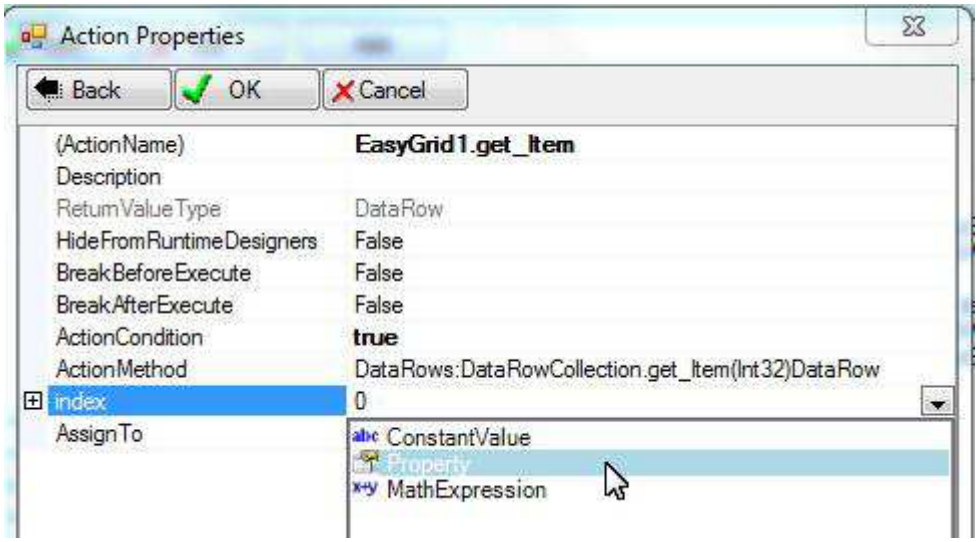


Find and select DataRows' get_Item method:

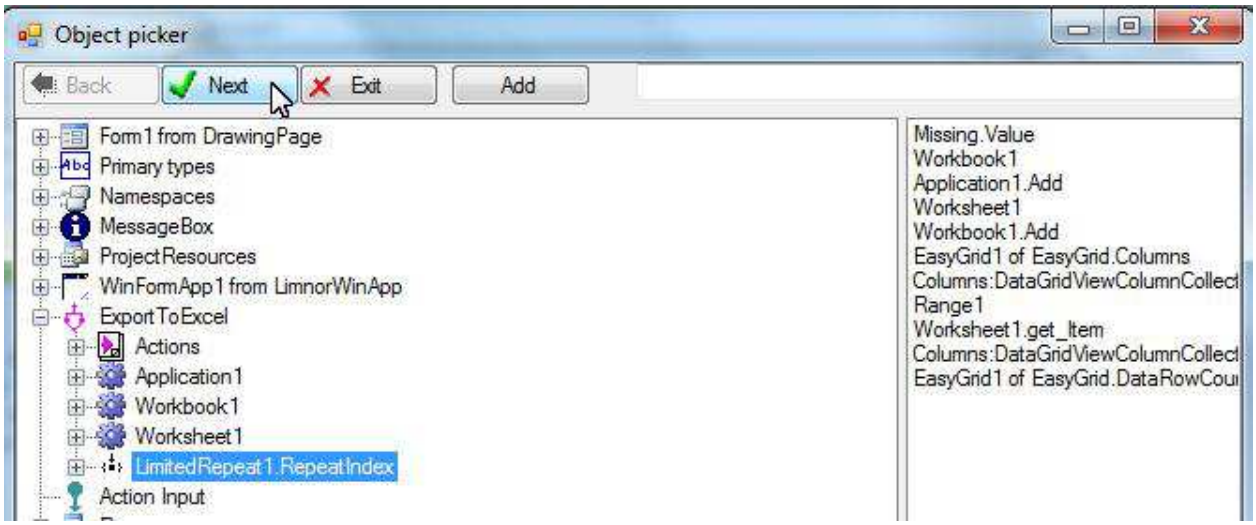




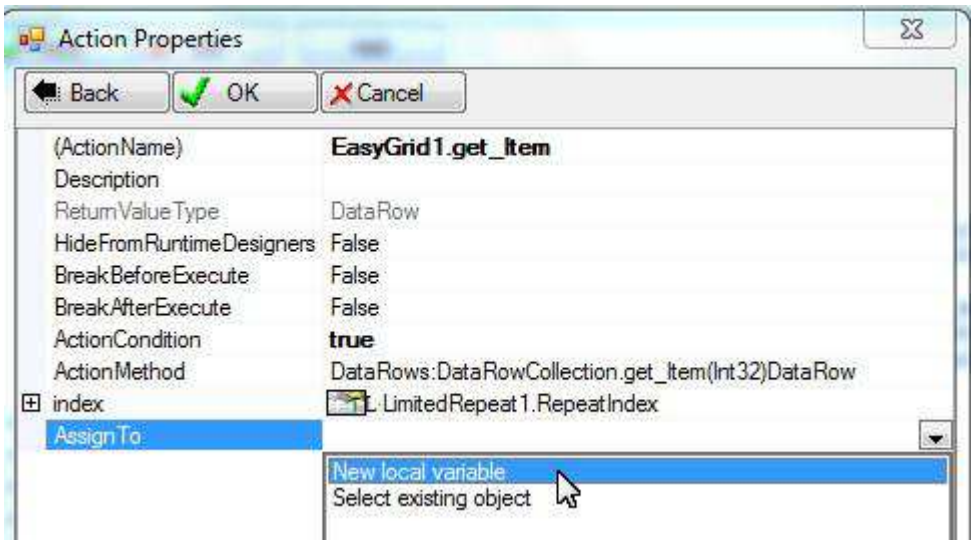
For "index" of this action, choose "Property" because we want to use "RepeatIndex":



Select “RepeatIndex”

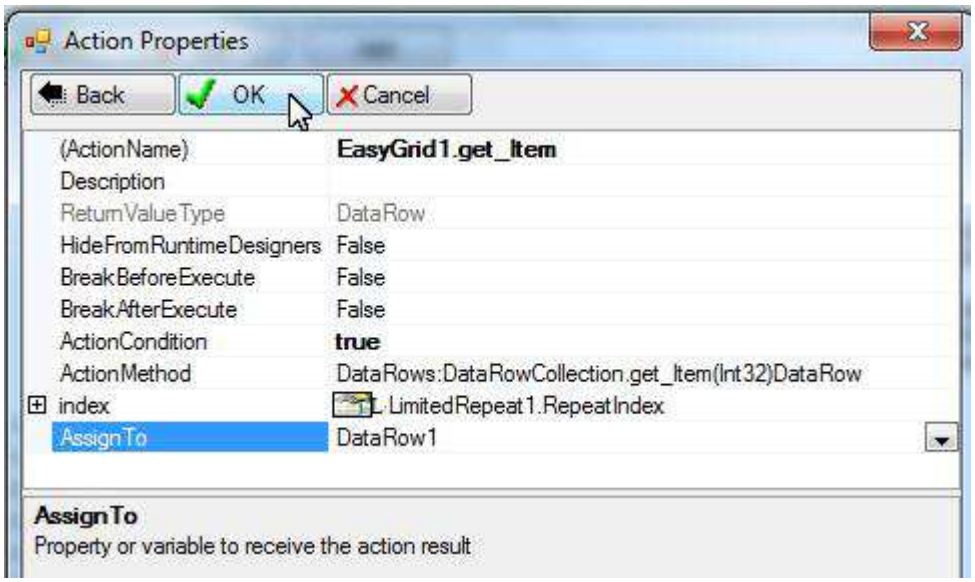


For “AssignTo”, choose “New local variable”:

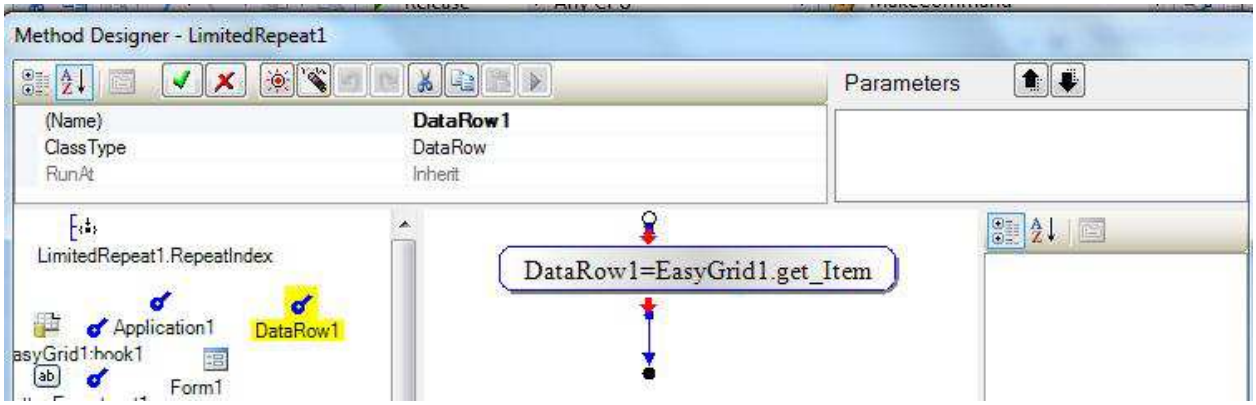


We may choose “New local variable” because this is not a PIA action. We do not need the trouble of creating a variable of the right type first and then use “select existing object”. Choosing “New local variable” will automatically create a variable of the right type for us.

Click OK to finish creating this action:



The action appears in the Action Pane. A new variable named DataRow1 appears in the Variable Pane:



Get Excel Row

Now we have a row of data. We will need to get a row of Excel cells so that we may pass the row of data to the row of cells.

The Cells property of a sheet has a get_Range method. We may use it to get a row of cells by specifying the starting cell and ending cell.

We have retrieved a cell using its row index and column index when exporting column headers. We will do it again here.

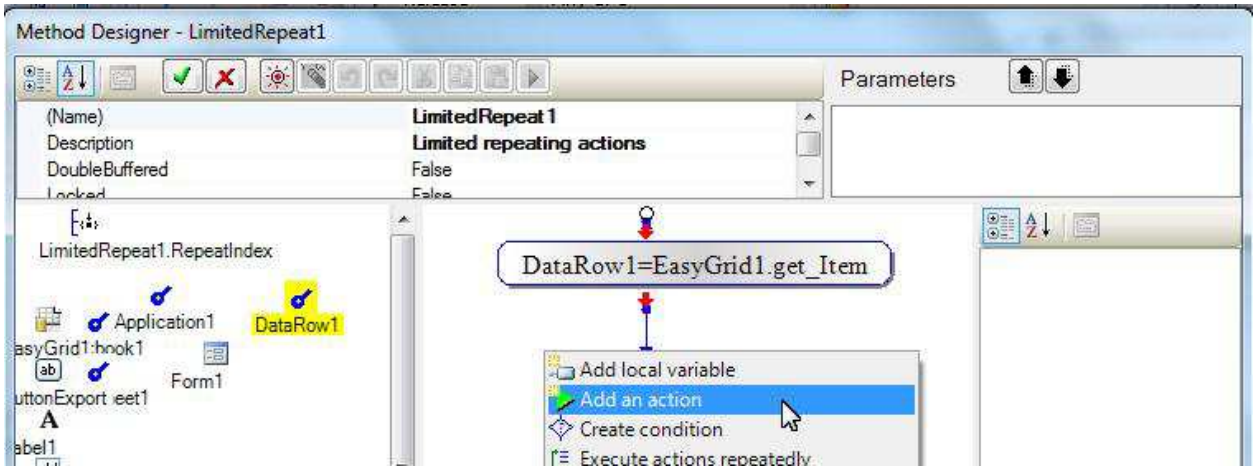
For each row of cells, the row index is RepeatIndex plus 2 because RepeatIndex starts from 0.

Row 1 is the header row; row 2 is the starting row for the data. Thus RepeatIndex + 2 is the row index for the cells.

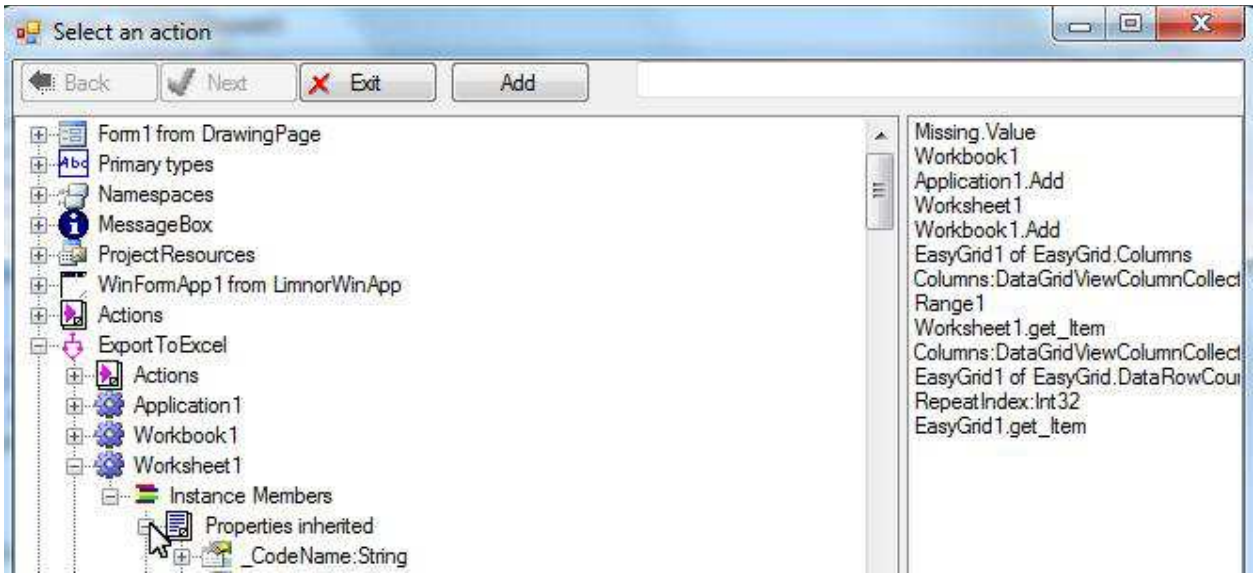
For the starting cell, the column index is 1.

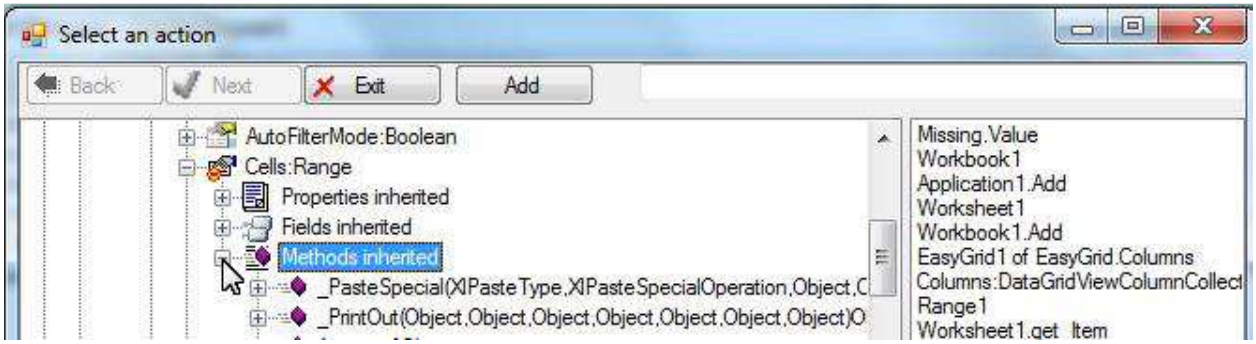
For the ending cell, the column index is the number of columns in the grid.

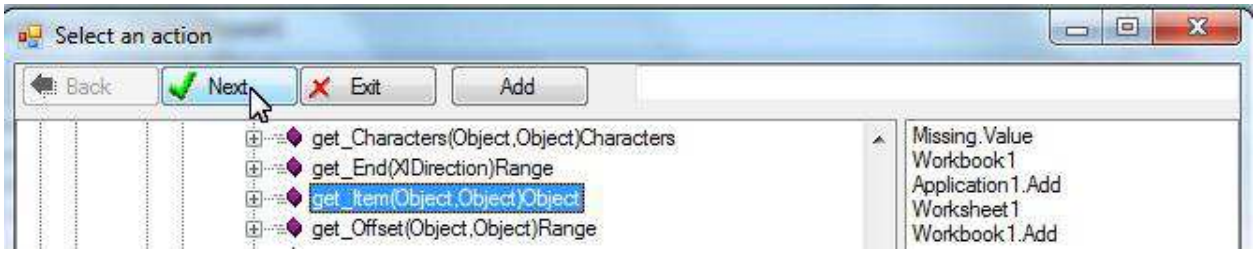
We have determined the row and column indexes for the starting/ending cells. We may retrieve them now. Right-click the Action Pane; choose “Add an action”:



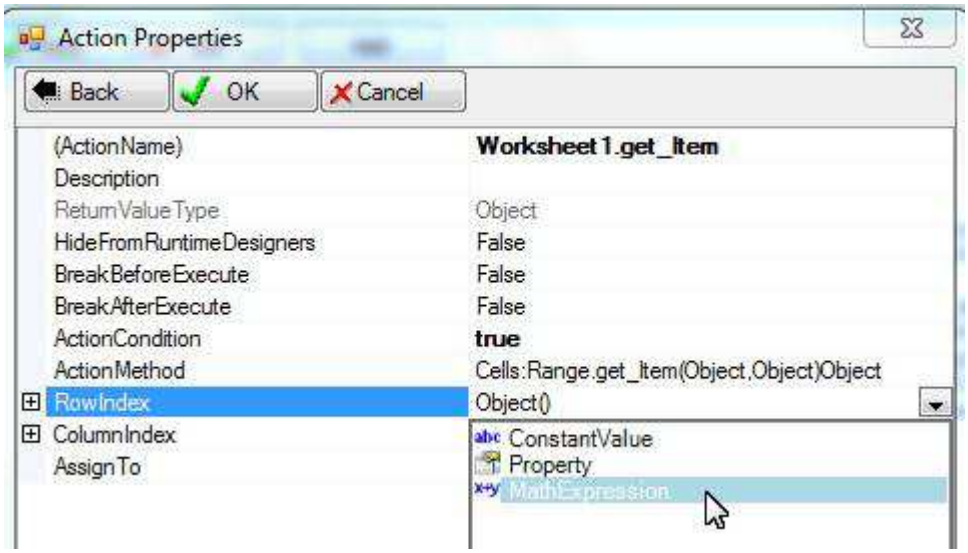
Choose get_Item of the Cells property of the sheet:

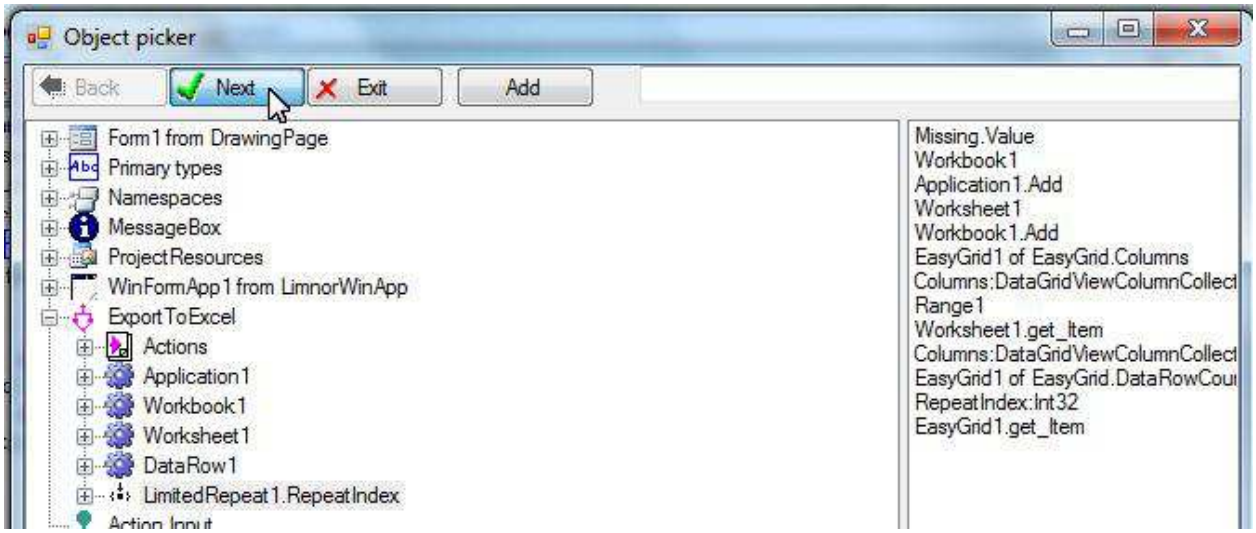


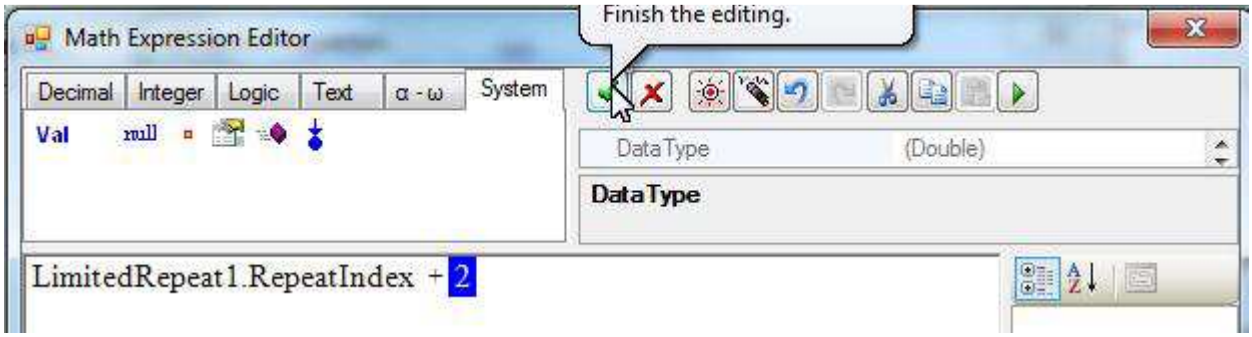




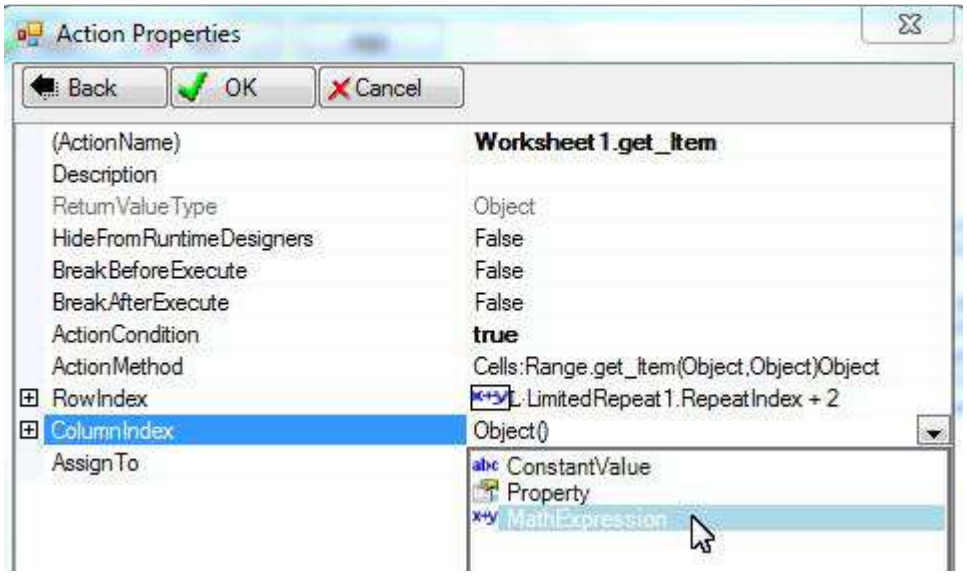
For the RowIndex, use RepeatIndex +2:

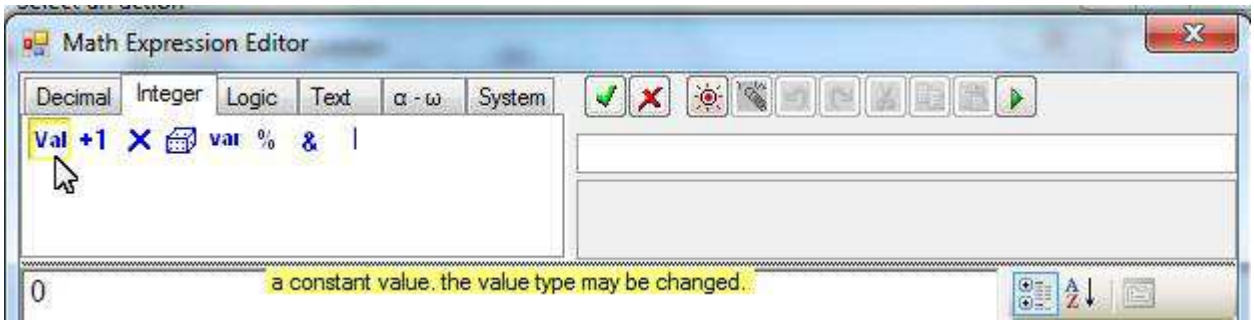


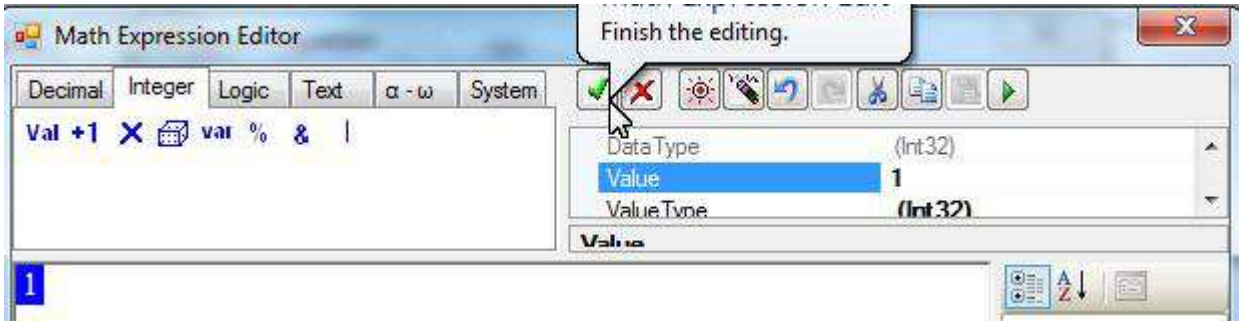




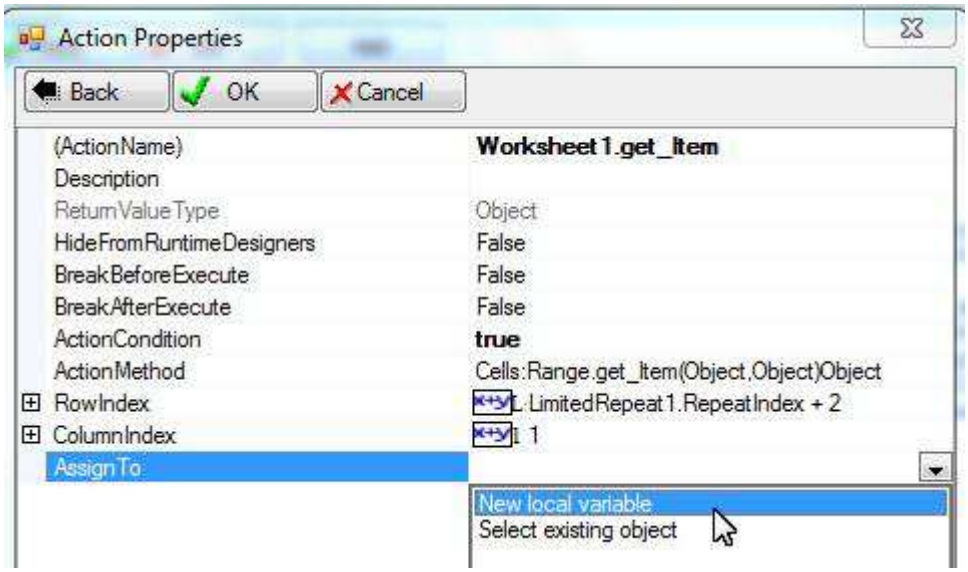
For the ColumnIndex, use 1:



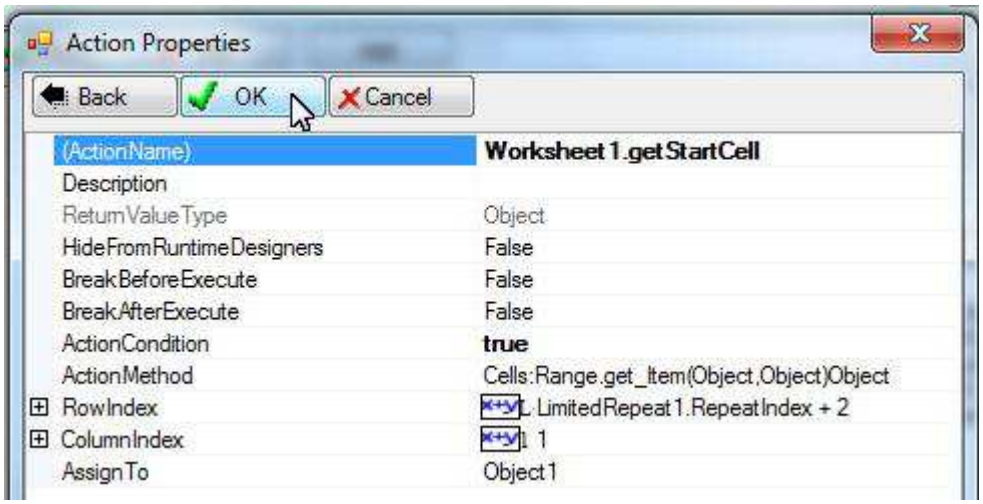




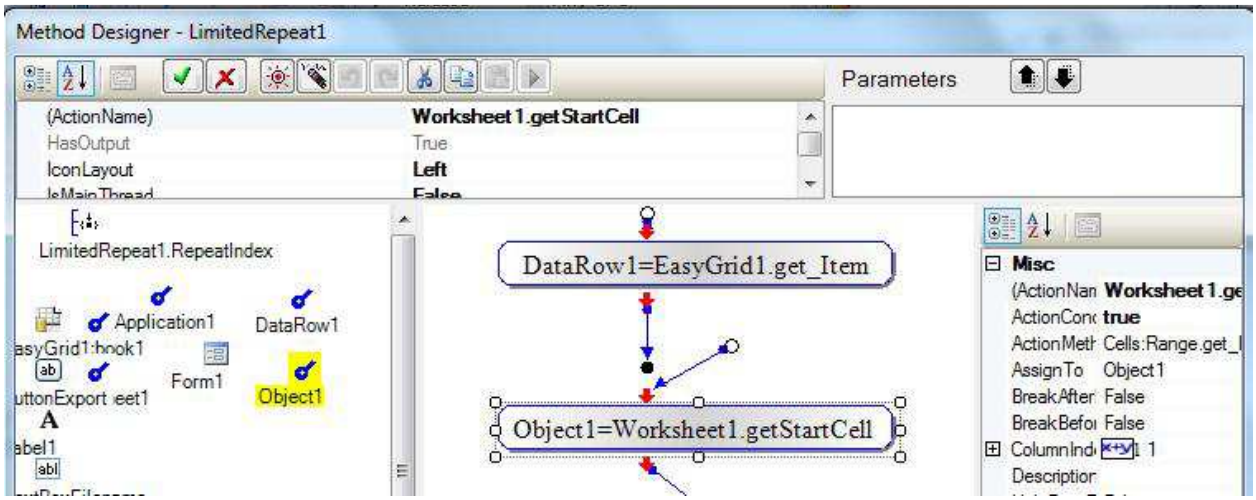
For “AssignTo”, choose “New local variable”:



Rename the action and click OK:



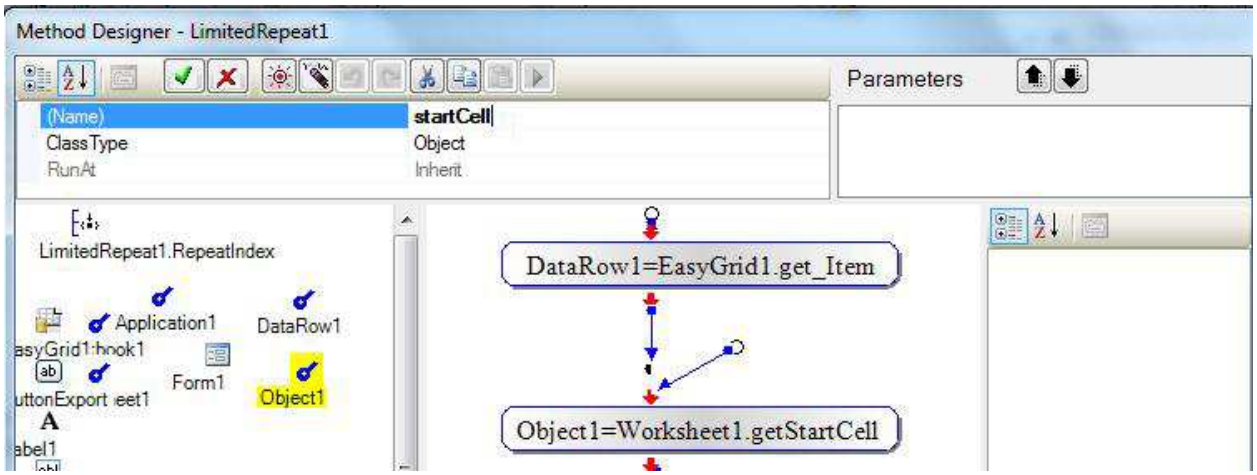
The action appears in the Action Pane. A new variable named Object1 appears in the Variable Pane:



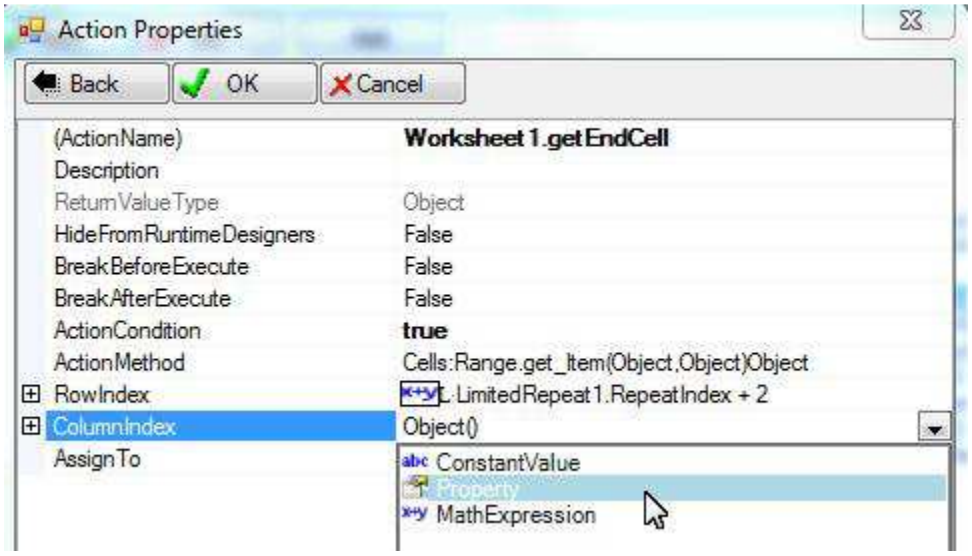
You may remember that we have a rule that says we must create a variable first for a PIA object and choose “Select existing object”, not “New local variable”, to get the PIA object. What we did above is against that rule.

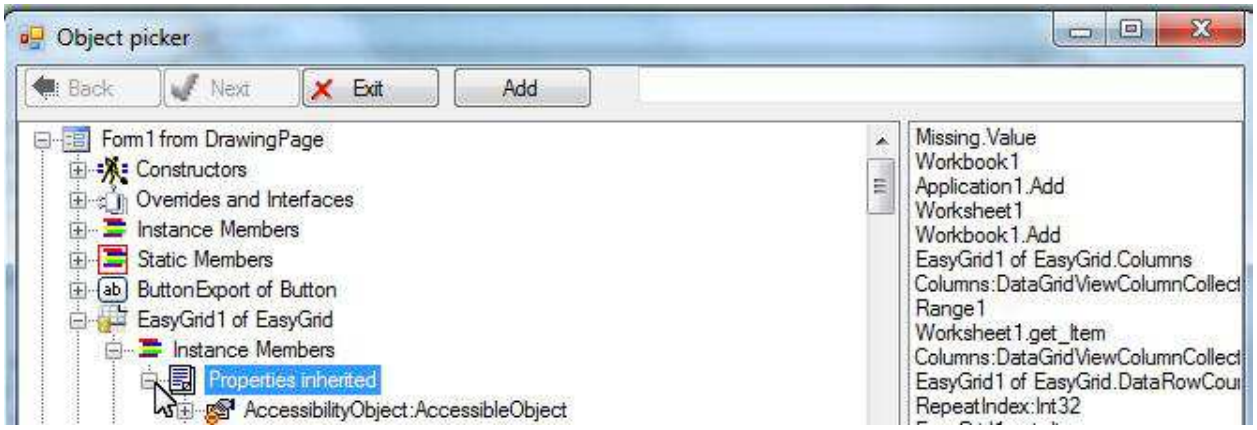
Actually that rule is set so that we may use the properties/methods/events of the resulting object. Since we will not use the properties/methods/events of the resulting object in this case, we do not have to follow that rule.

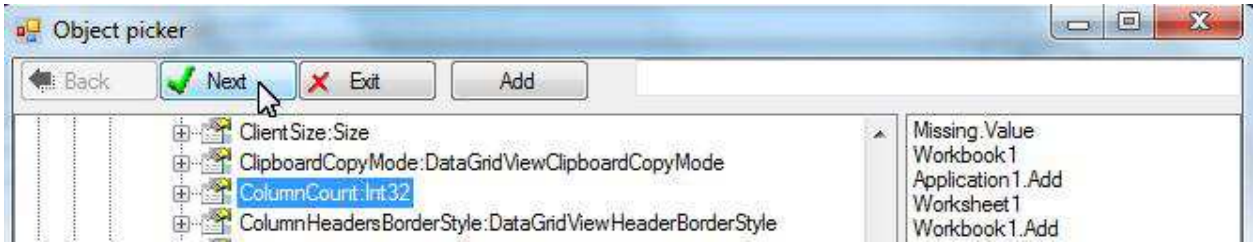
We may rename Object1 to startCell:



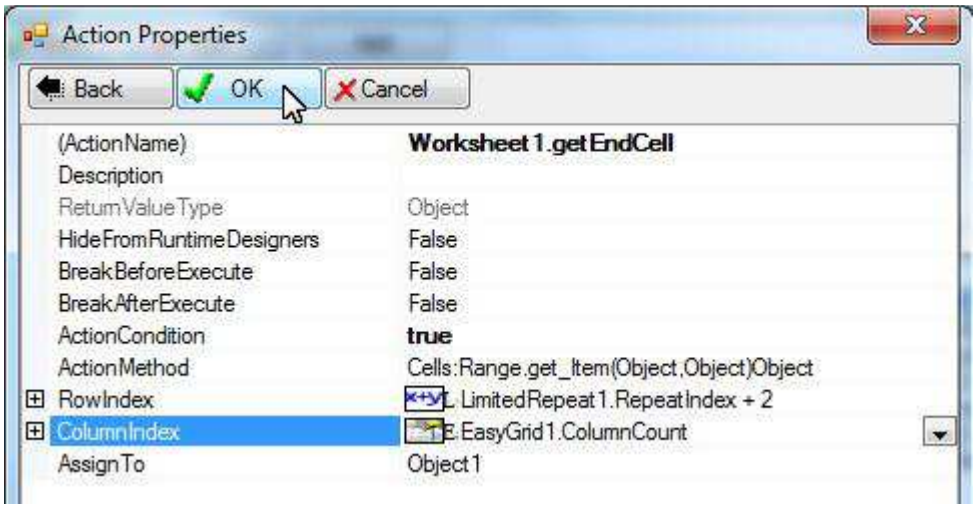
In the same way, we may create an action to get the ending cell. The RowIndex of the action is still RepeatIndex + 2; but the ColumnIndex is the number of columns in the grid:



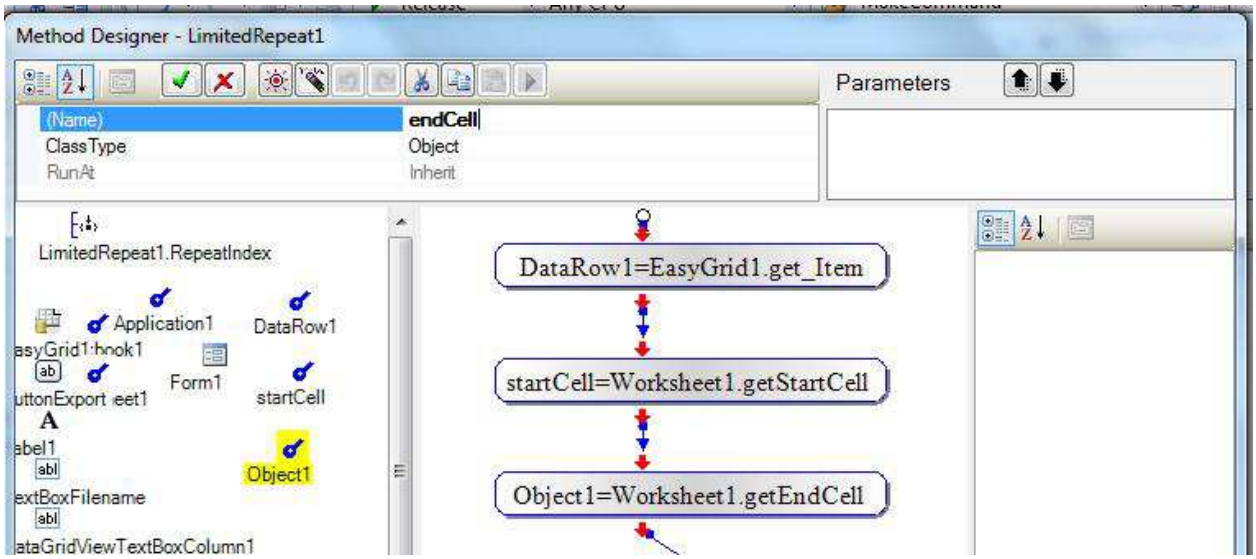




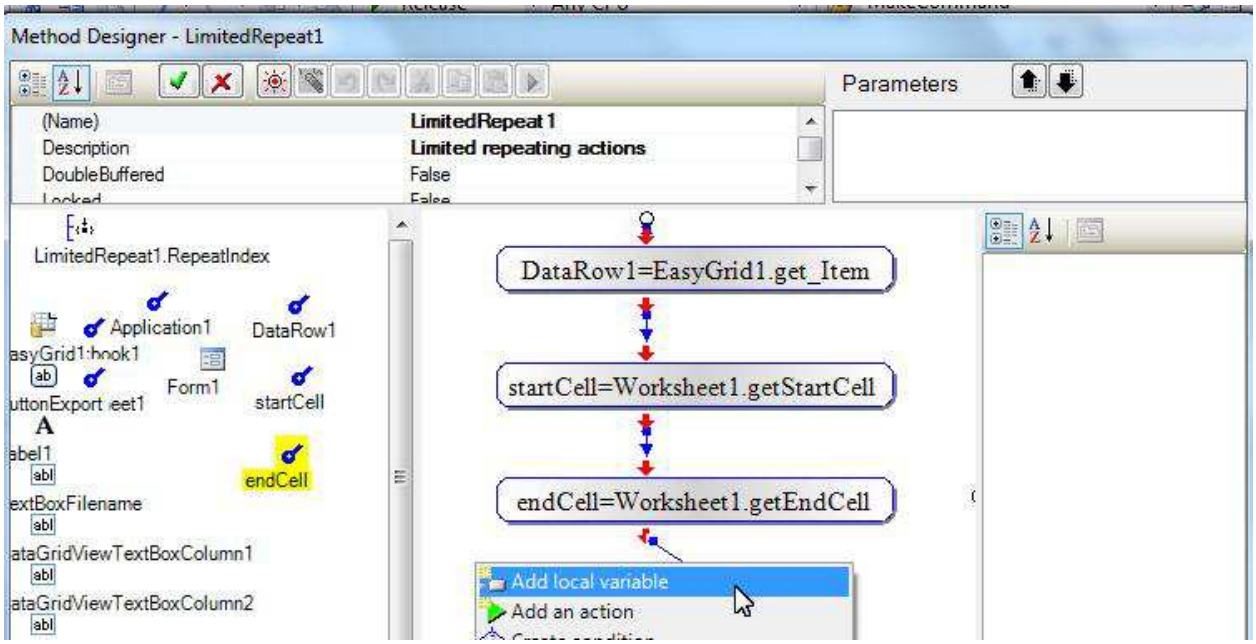
Click OK to finish creating this action.

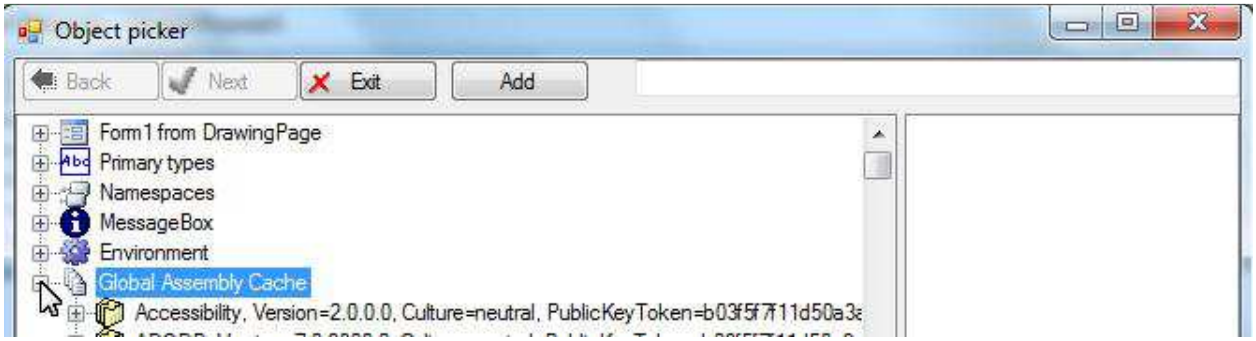


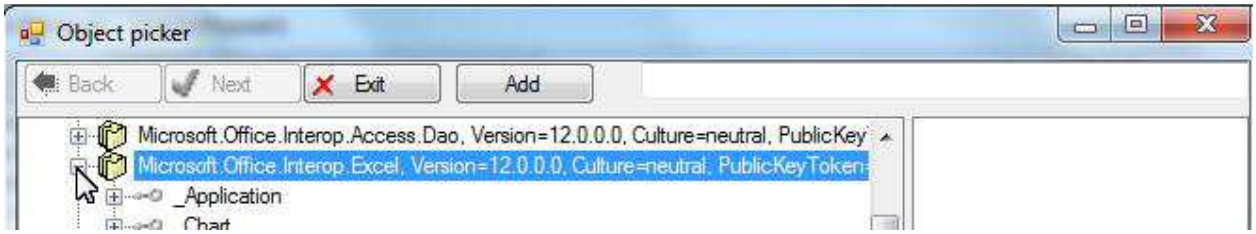
Rename the new variable to endCell:

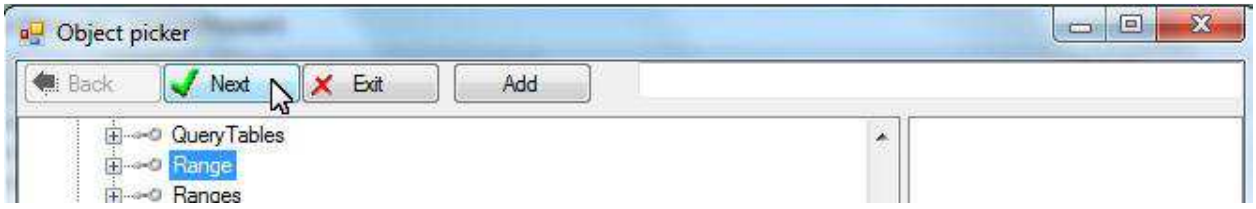


With start and end cells ready we may get Excel Row. First, we need to create a Range variable for the row:

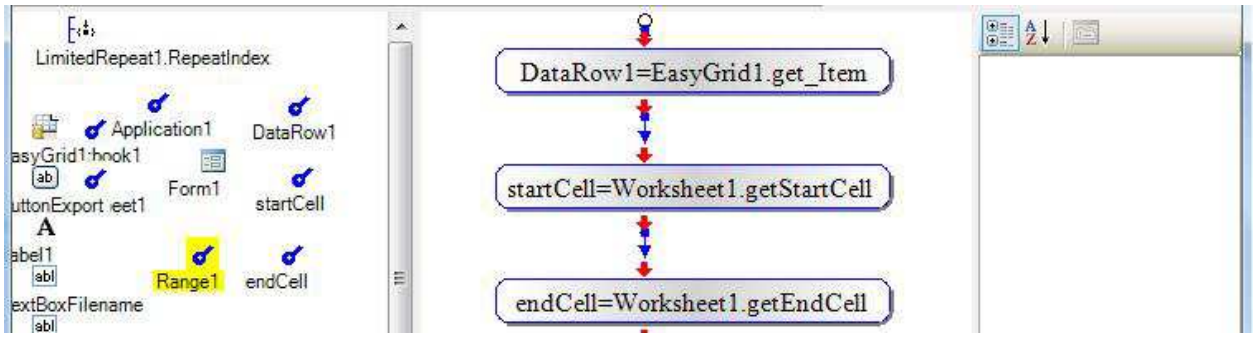




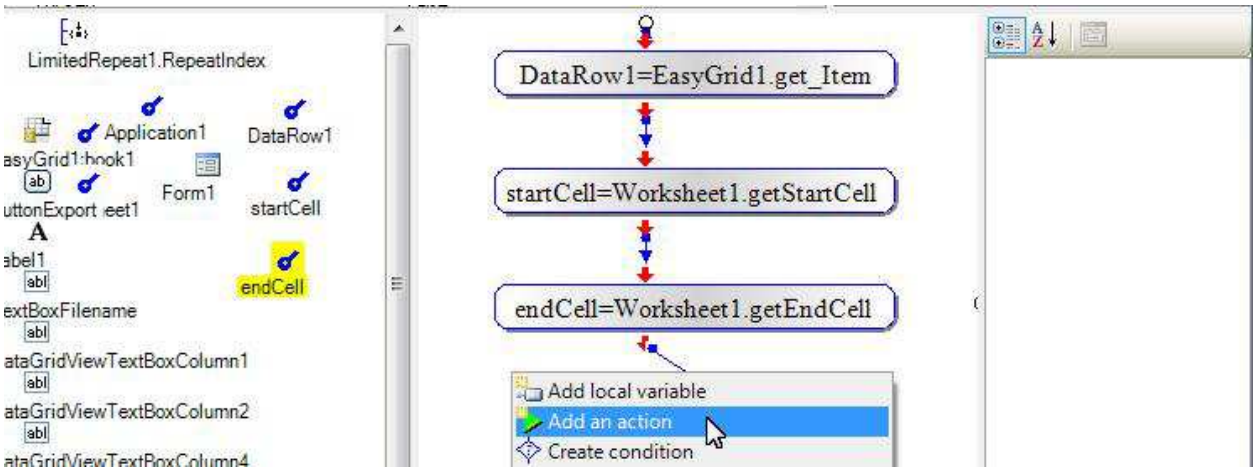


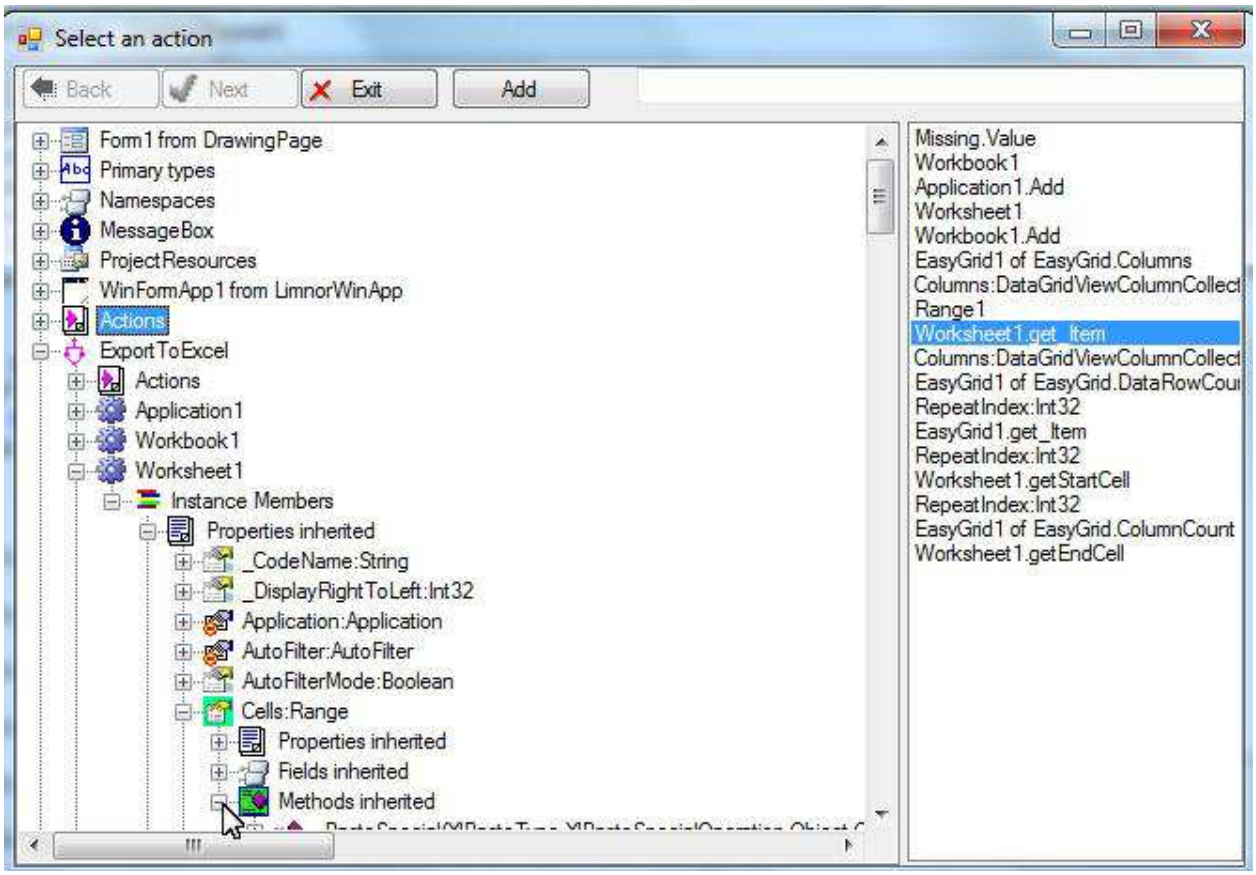


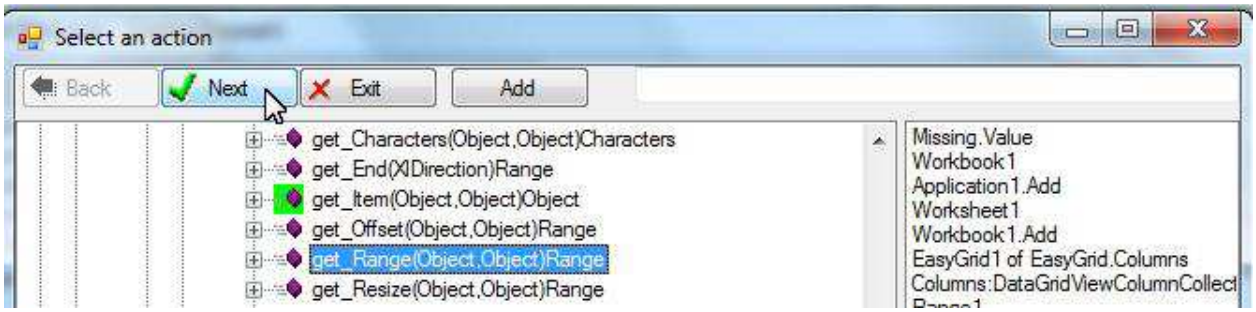
A new variable, Range1, appears:



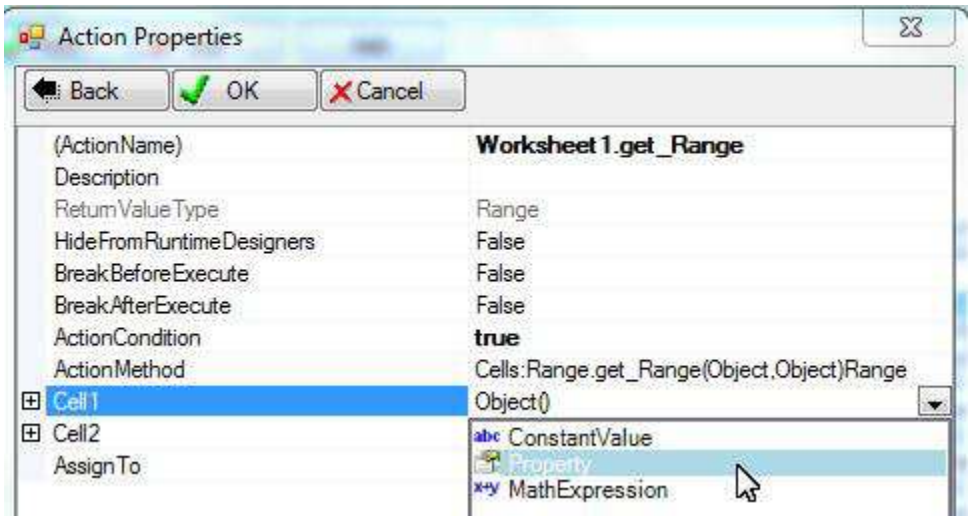
Create a new action by selecting get_Range of the Cells property of the sheet:



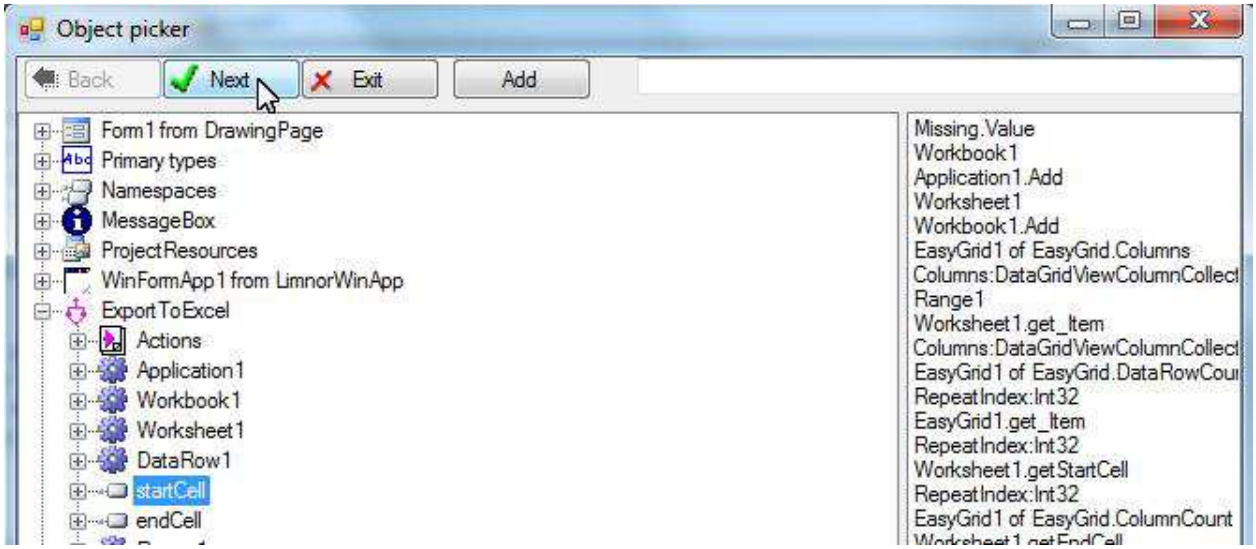




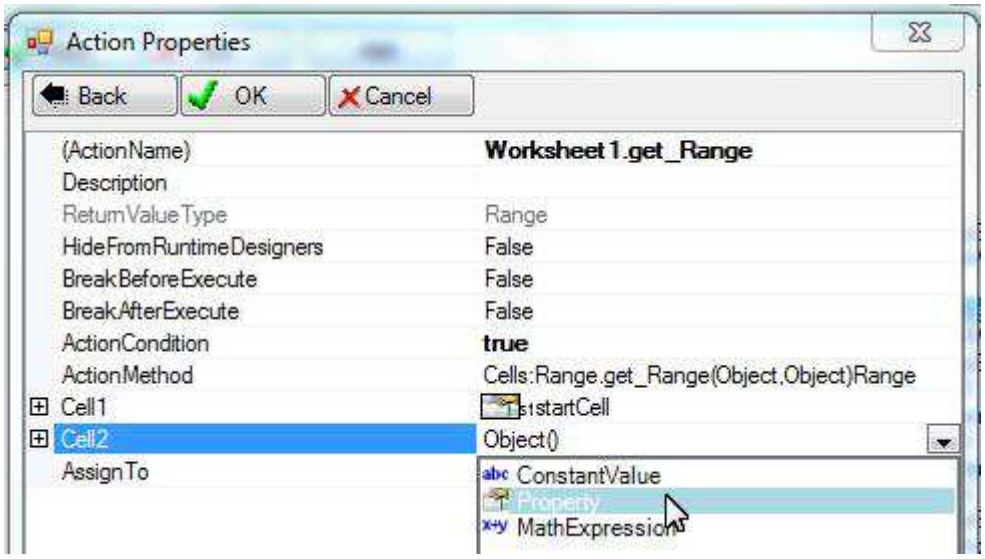
For “Cell1”, choose “Property”:

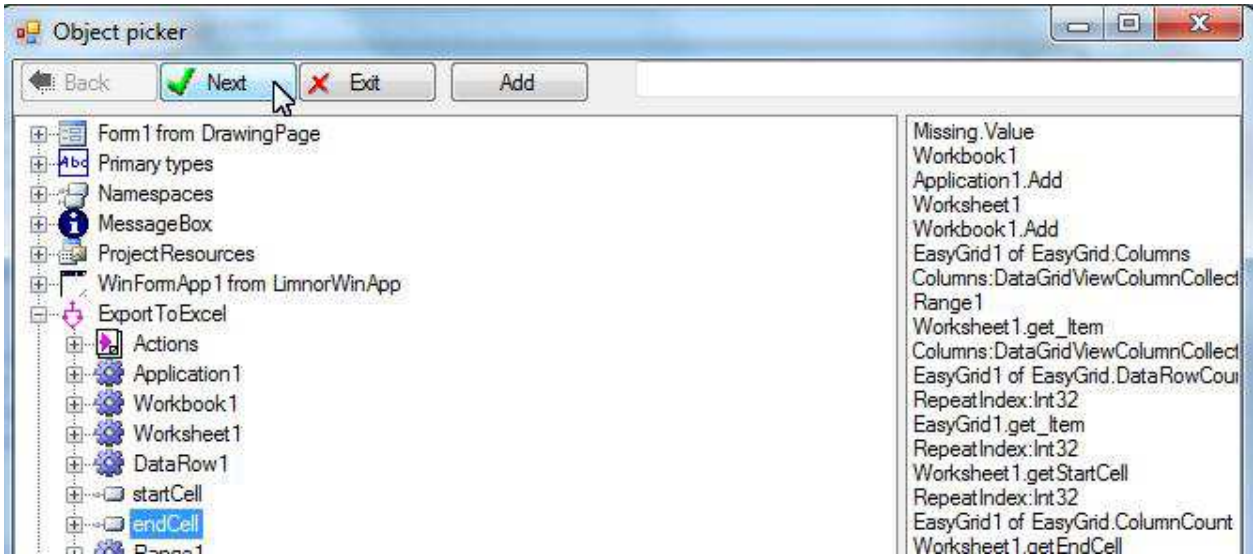


Choose the startCell:

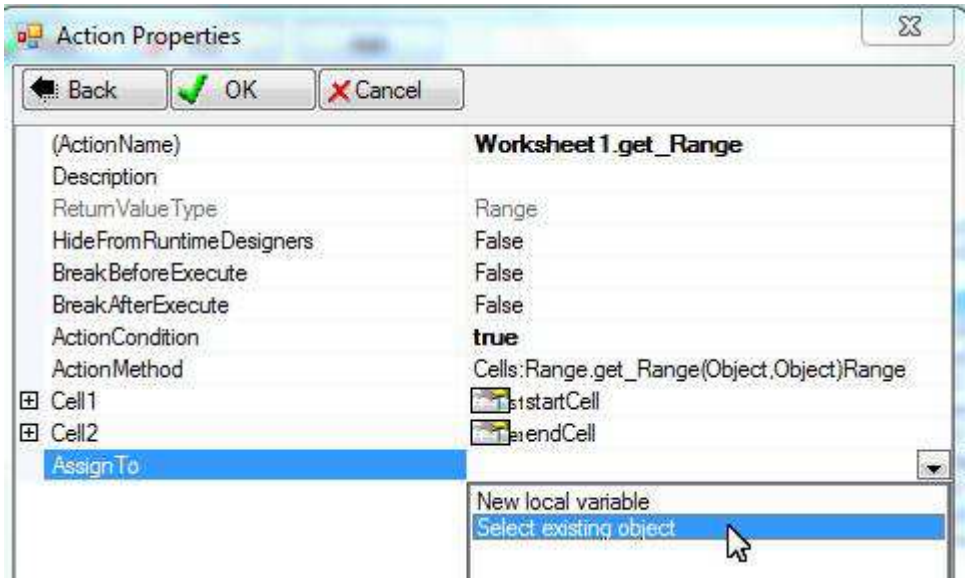


For “Cell2”, choose endCell:

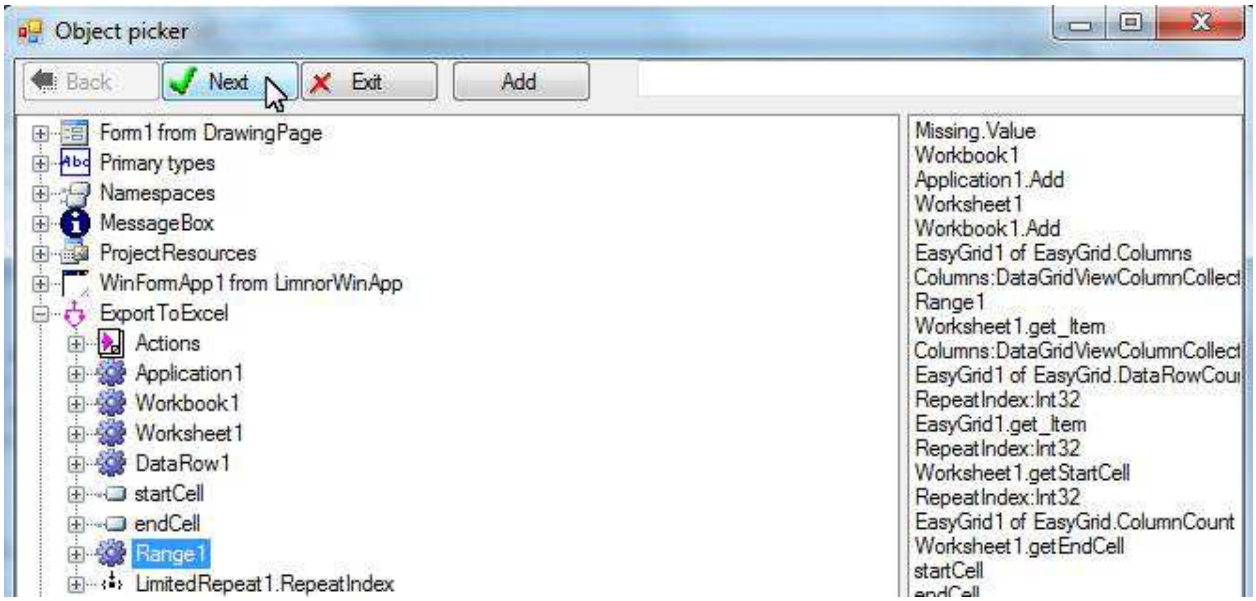




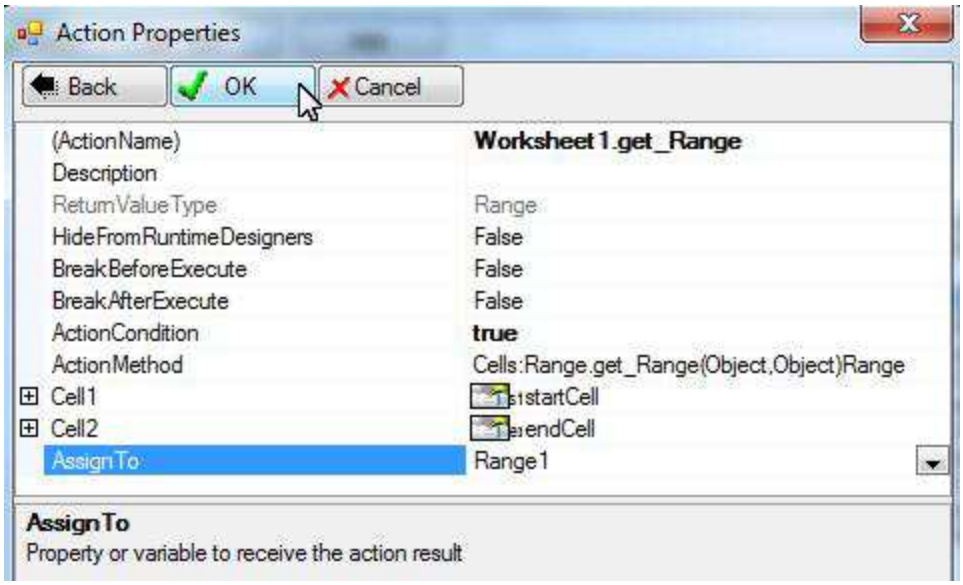
For “AssignTo”, choose “Select existing object”:



Select Range1:

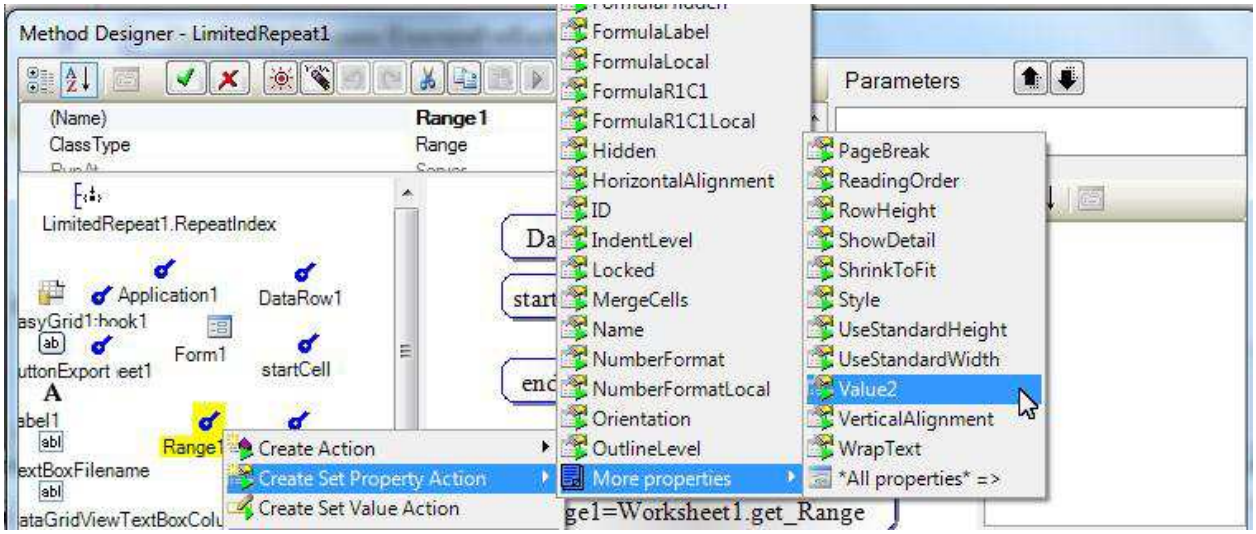


Click OK to finish creating this action:

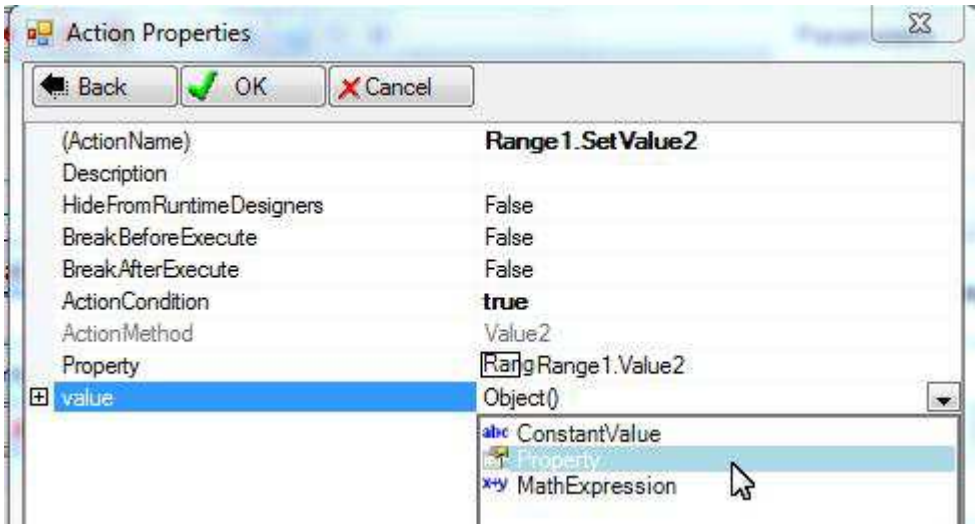


Export Row

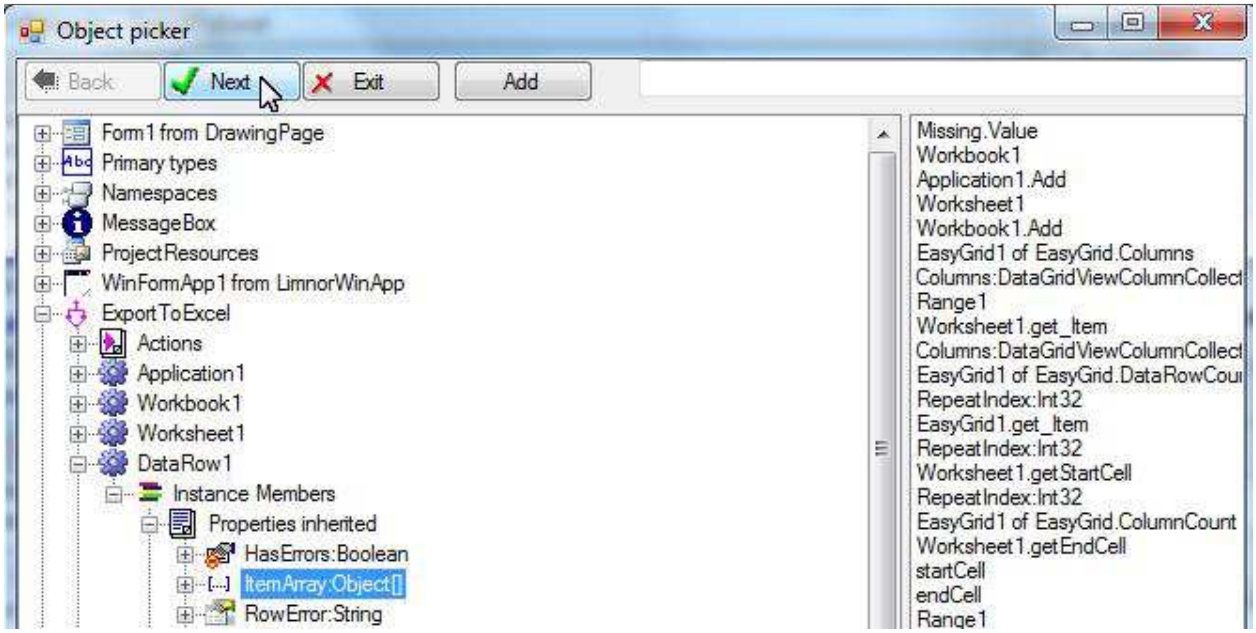
Now we may pass the data row to the Excel row by creating a “Set Property Action” on Range1:



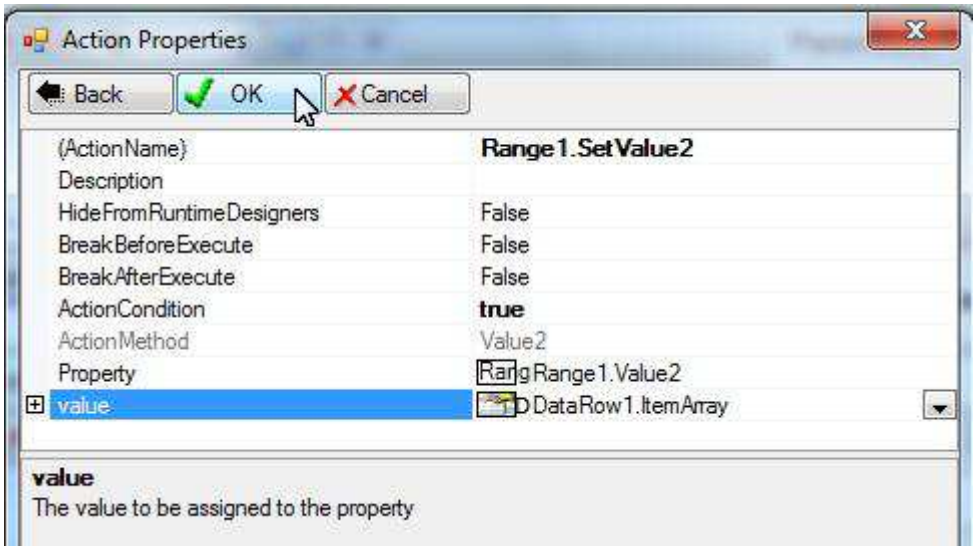
For the value of the action, choose Property:



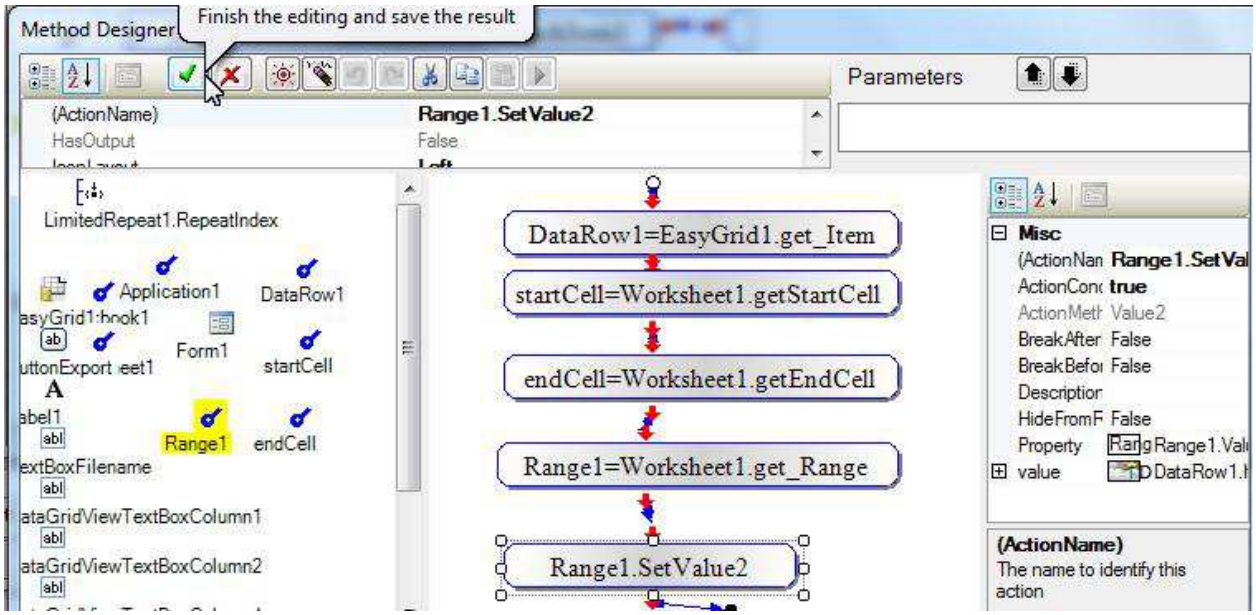
Choose ItemArray property of the data row:



Click OK:



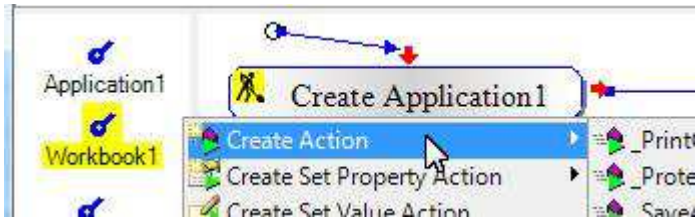
Link all actions together. That is all what we need for exporting a row:



Close Excel and Save

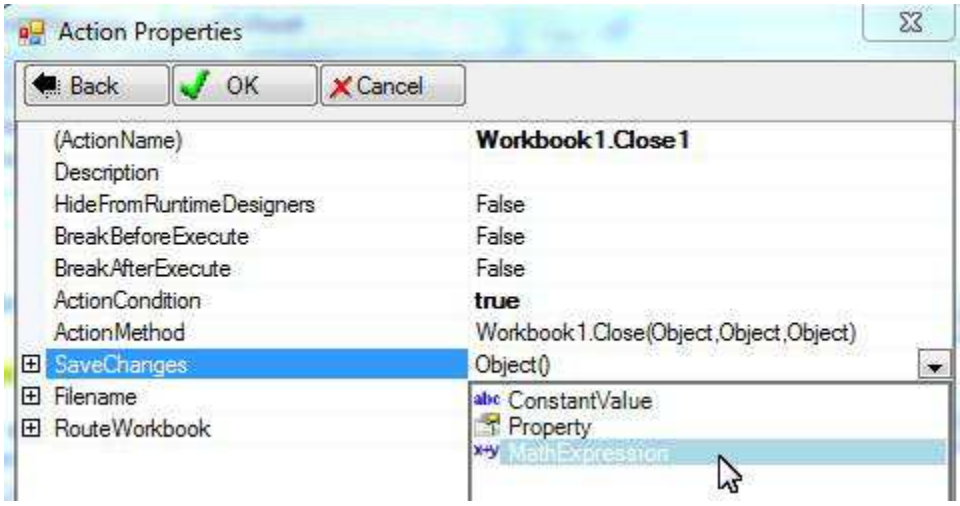
We may save the workbook and close the Excel by a Close action on the workbook.

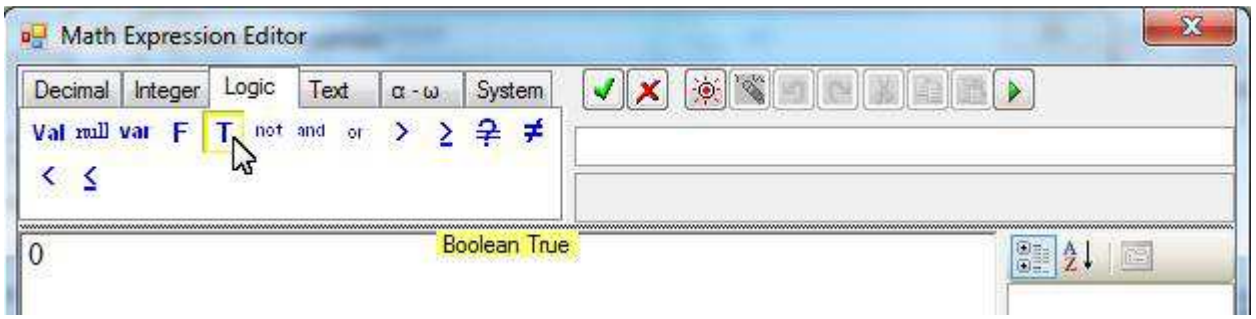
Right-click Workbook1; choose “Create Action”;

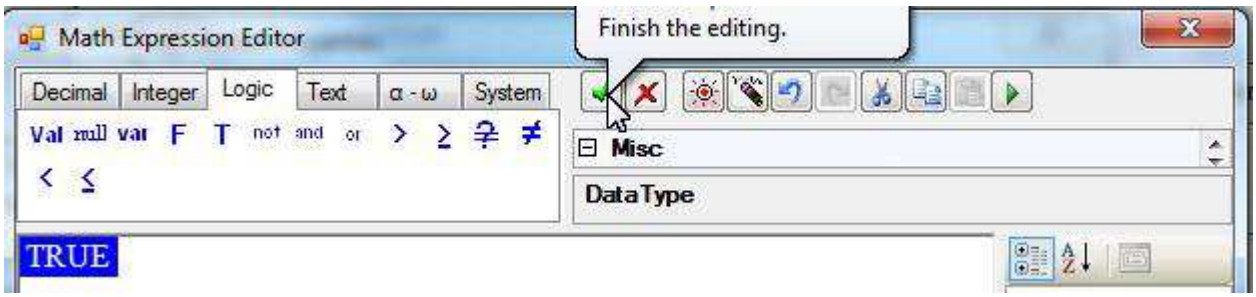


Choose “More Methods”; choose “All methods”; Choose “Close”:

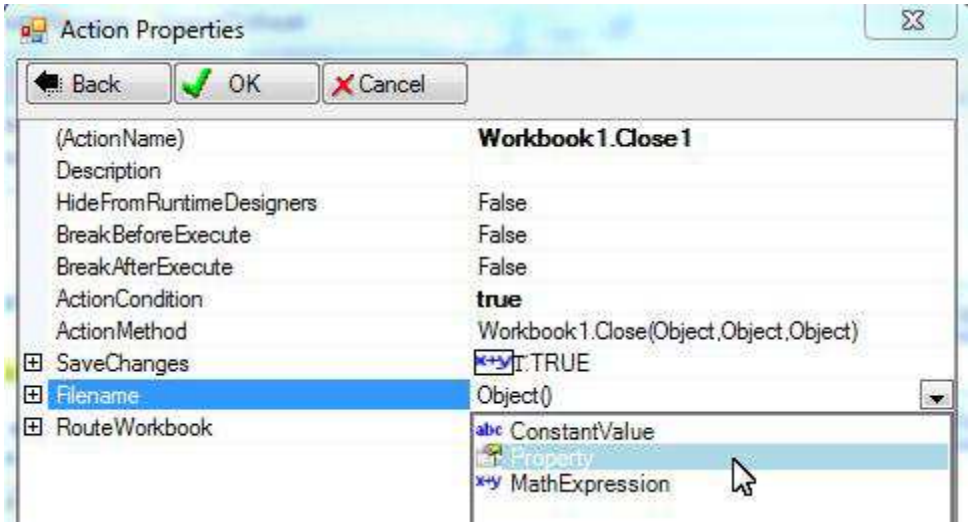
For “SaveChanges”, choose a math expression True constant:

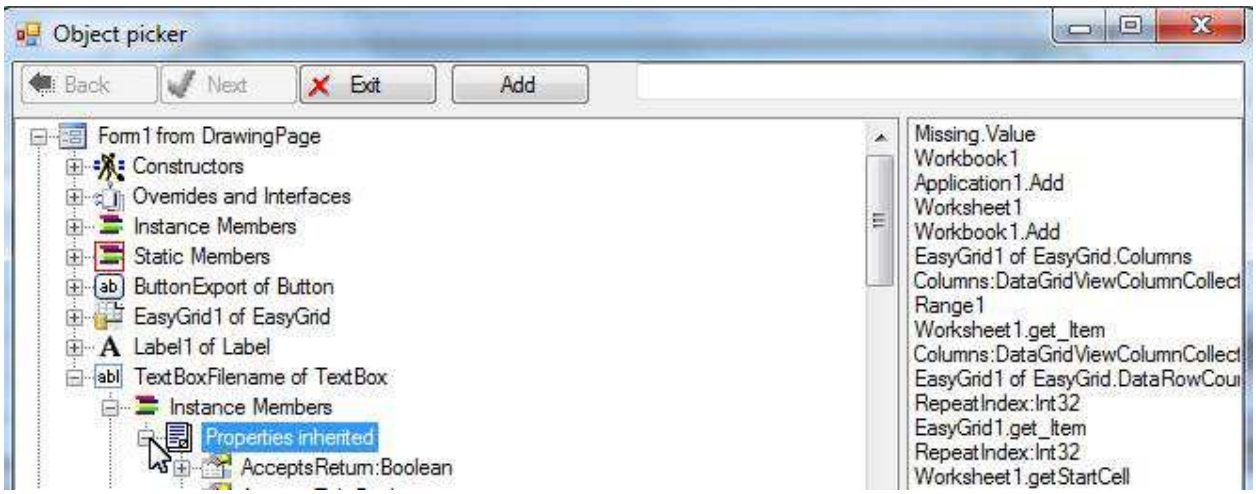




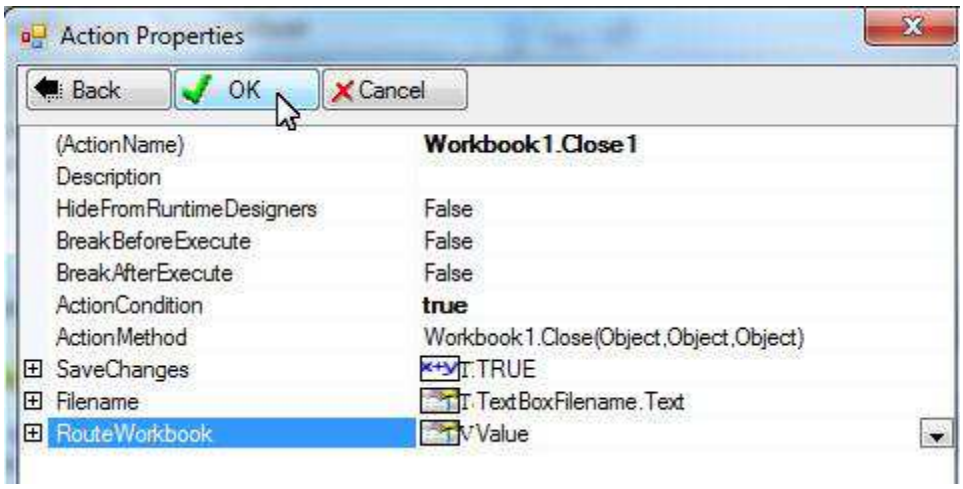


For “Filename”, choose Text of a text box:





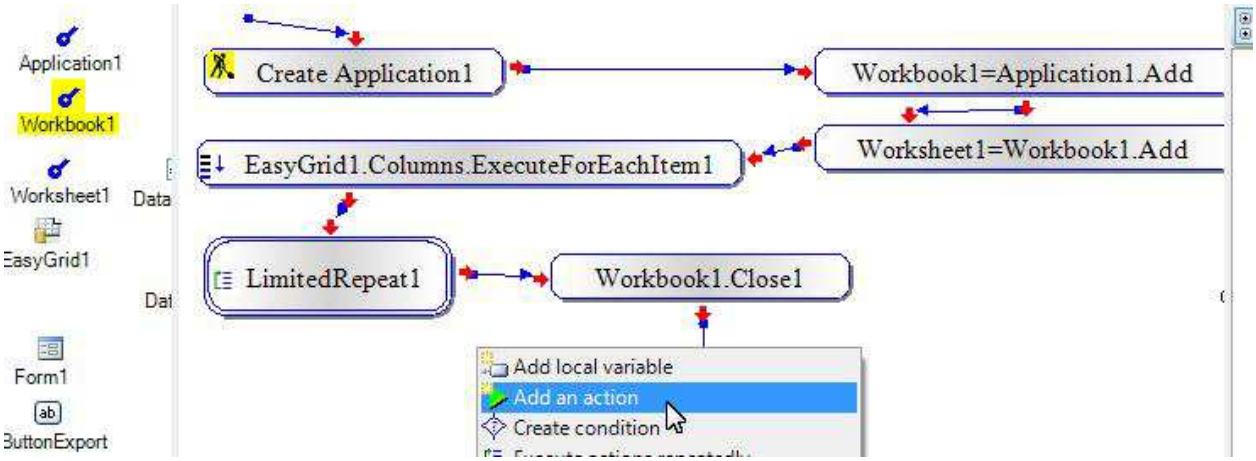
For the other parameter, use Missing.Value as we did before.



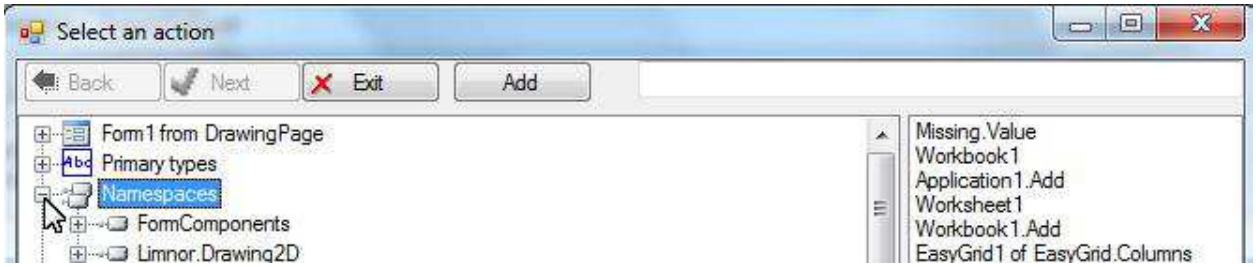
Clean up

PIA needs some clean up. We need to release the Application object and the Workbook object. It can be done by the ReleaseComObject of the Marshal class.

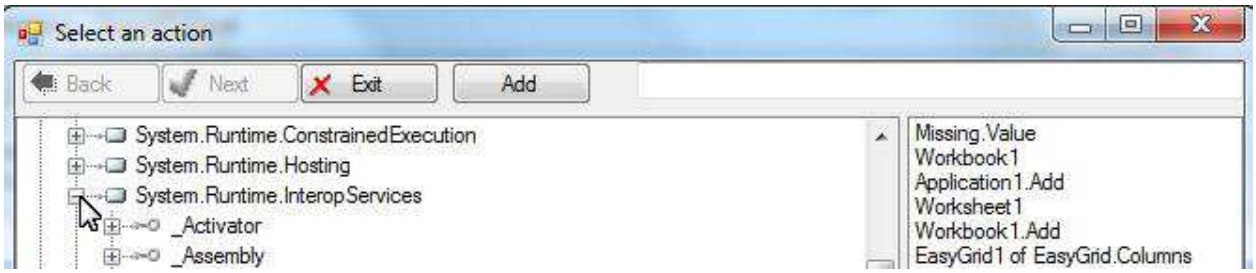
Right-click the Action Pane; choose “Add an action”:



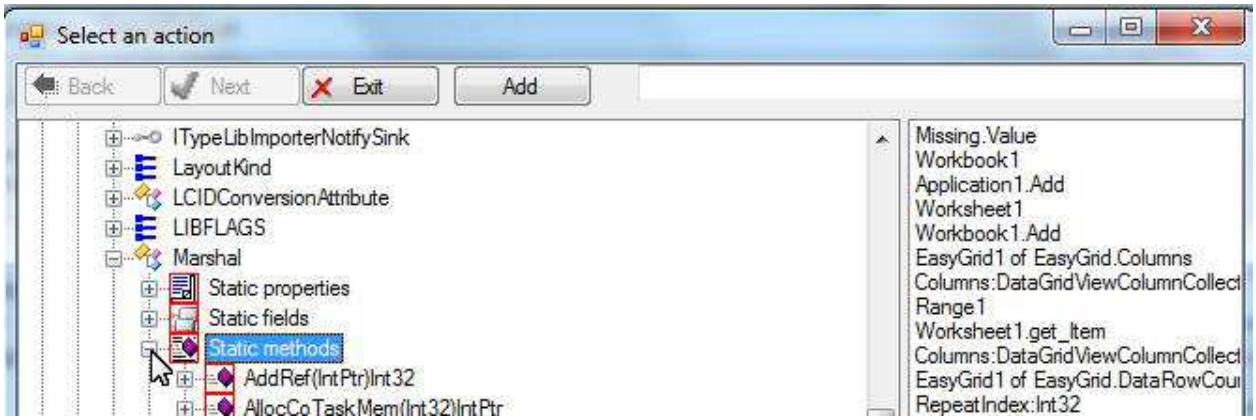
Expand Namespaces:



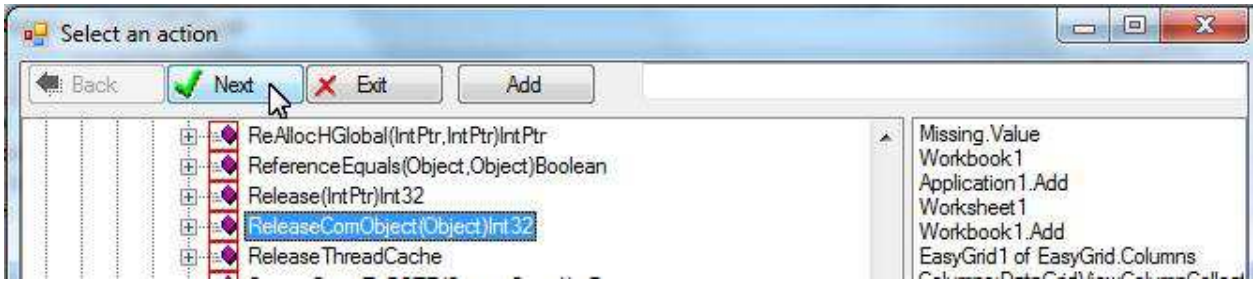
Expand System.Runtime.InteropServices:



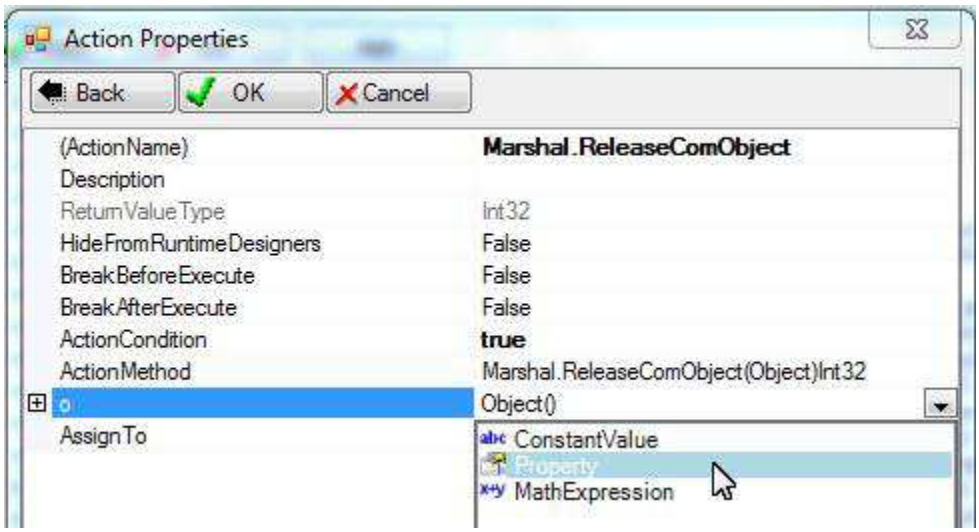
Expand Static methods of the Marshal class:



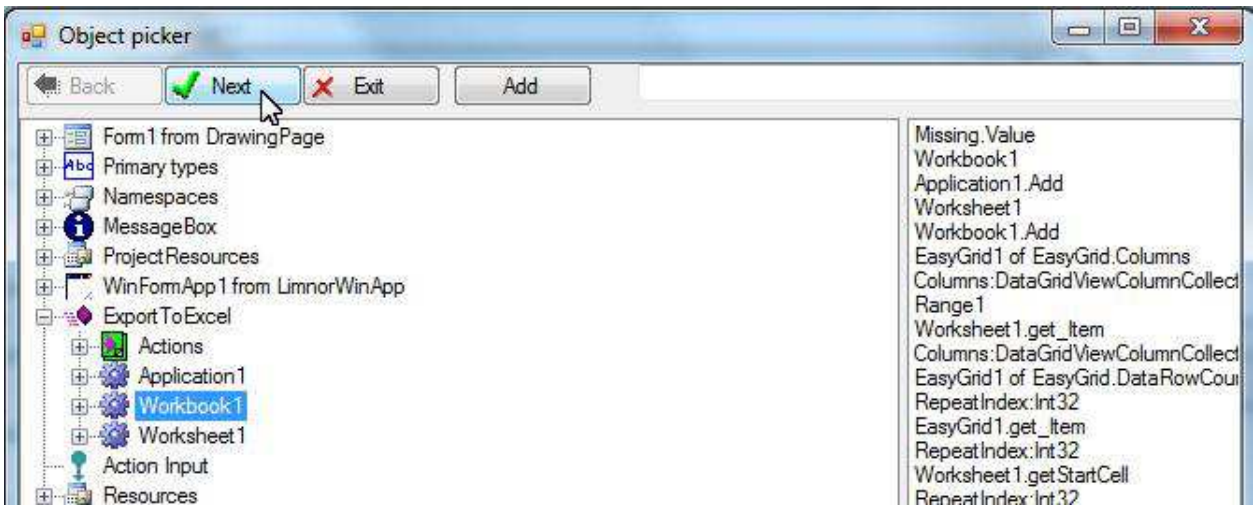
Select ReleaseComObject method:



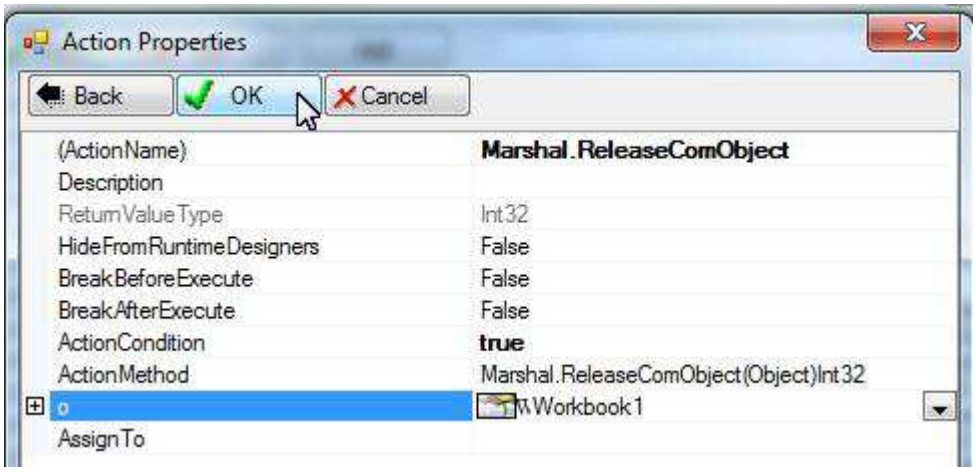
Select Property for parameter “o”:



Select the Workbook object:

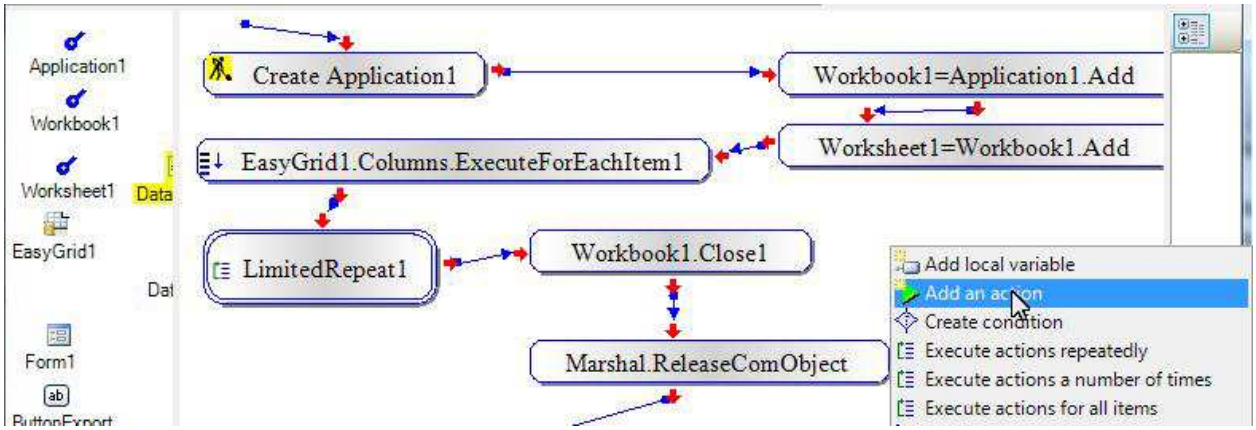


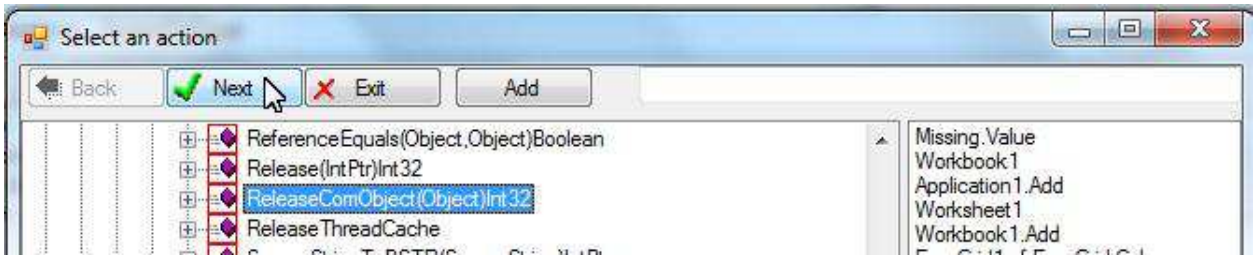
Click OK to finish creating this action:

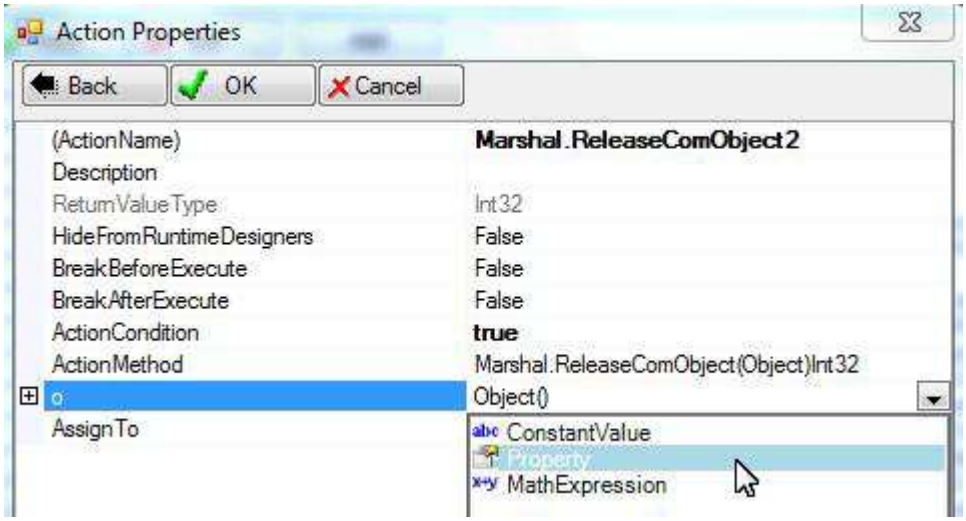


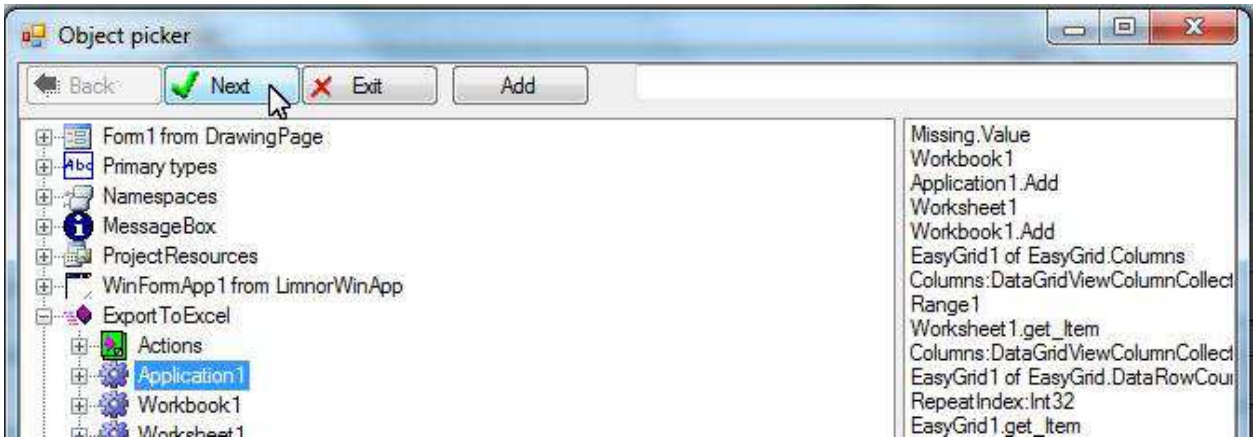
Link the action to the last action.

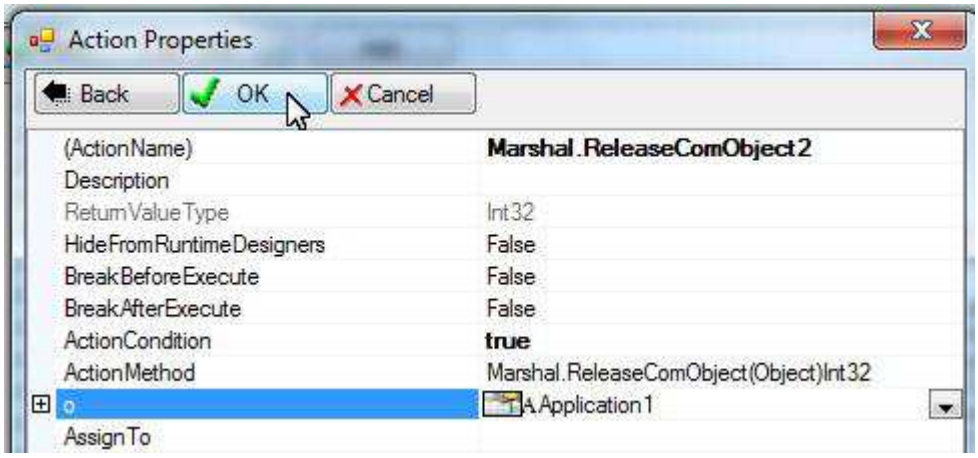
Create another ReleaseComObject action to release the Application object:



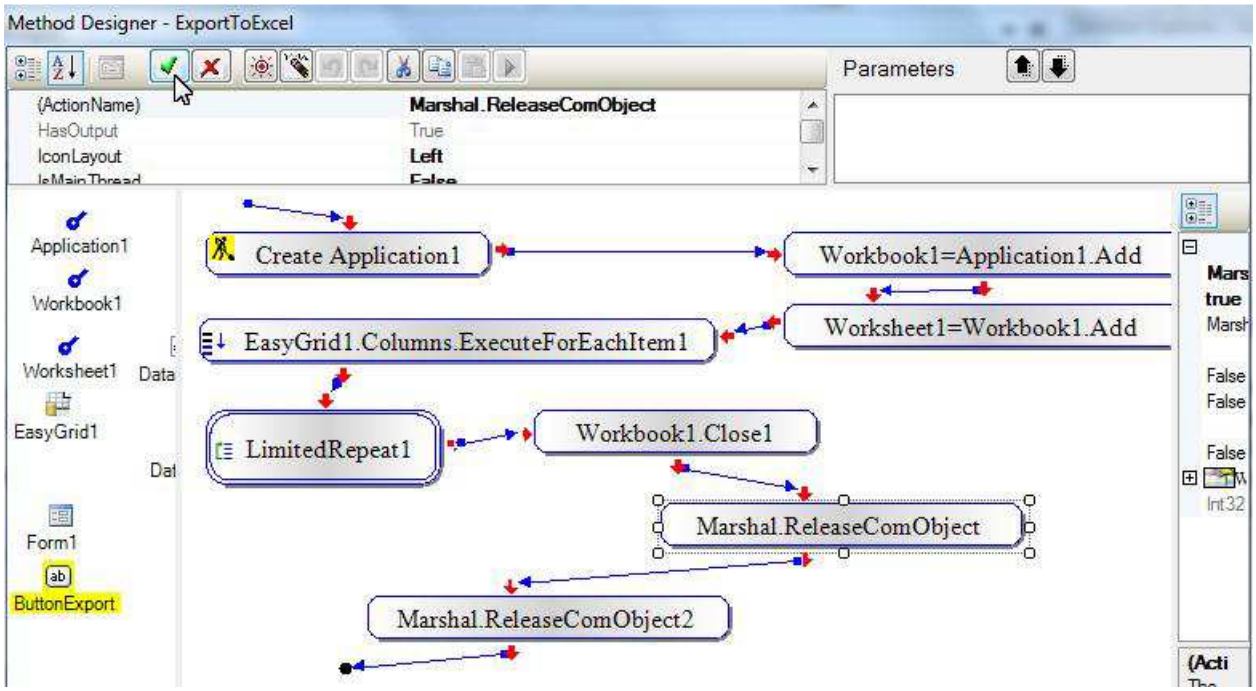








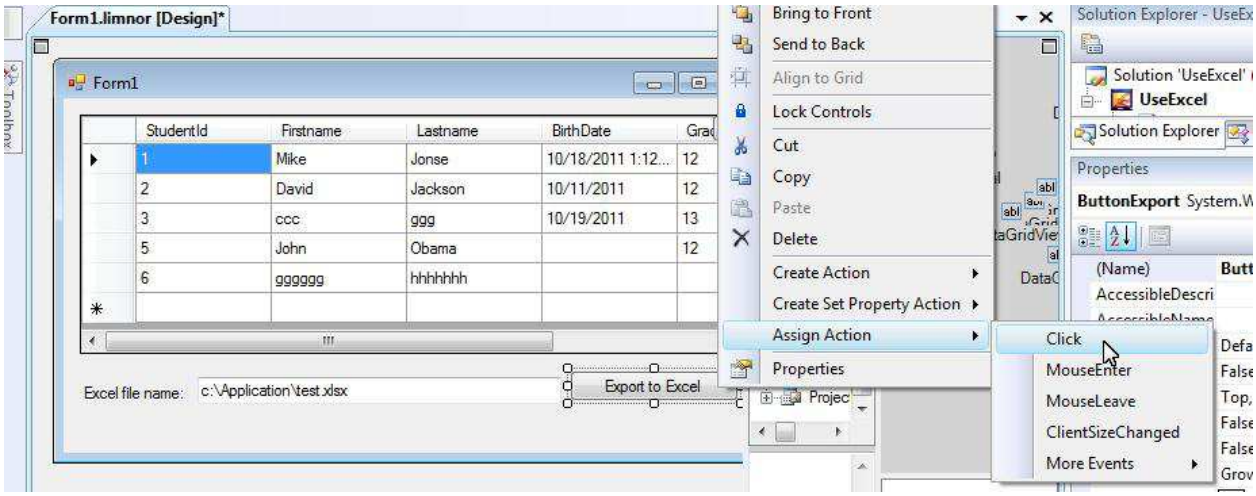
We are done creating this method:



Test

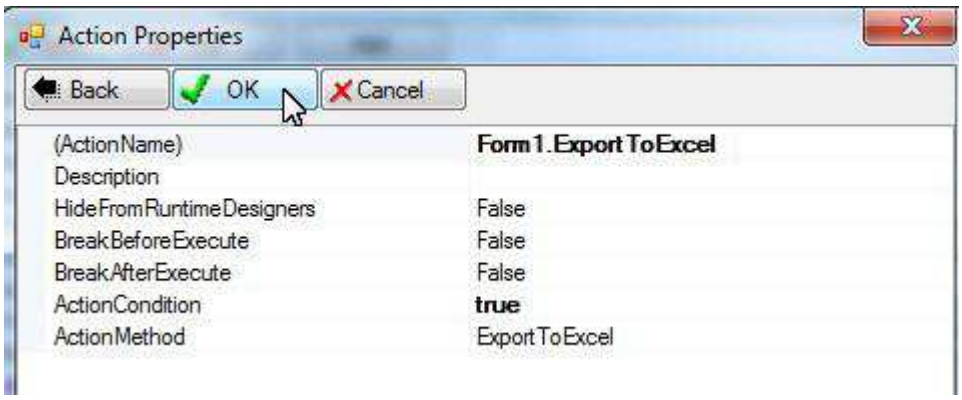
We may execute the new method by clicking a button.

Right-click the button; choose “Assign action”; choose “Click”:

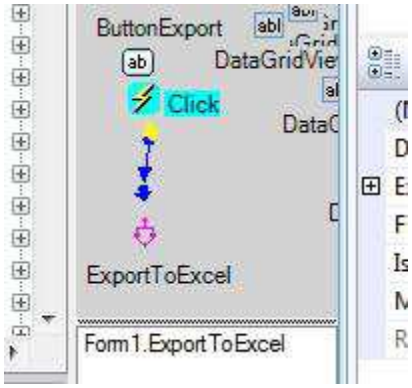


Select the method we just created:

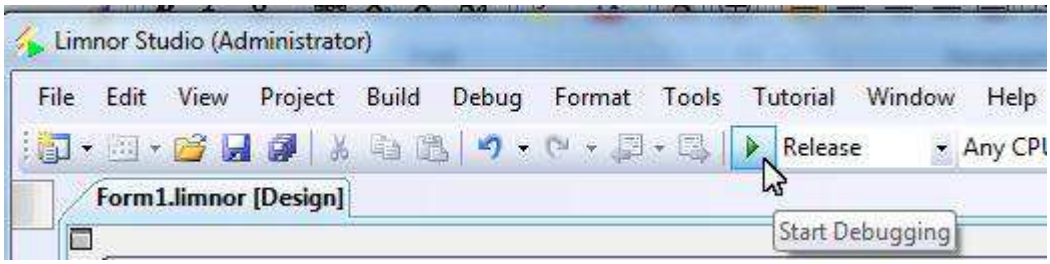
Click OK:



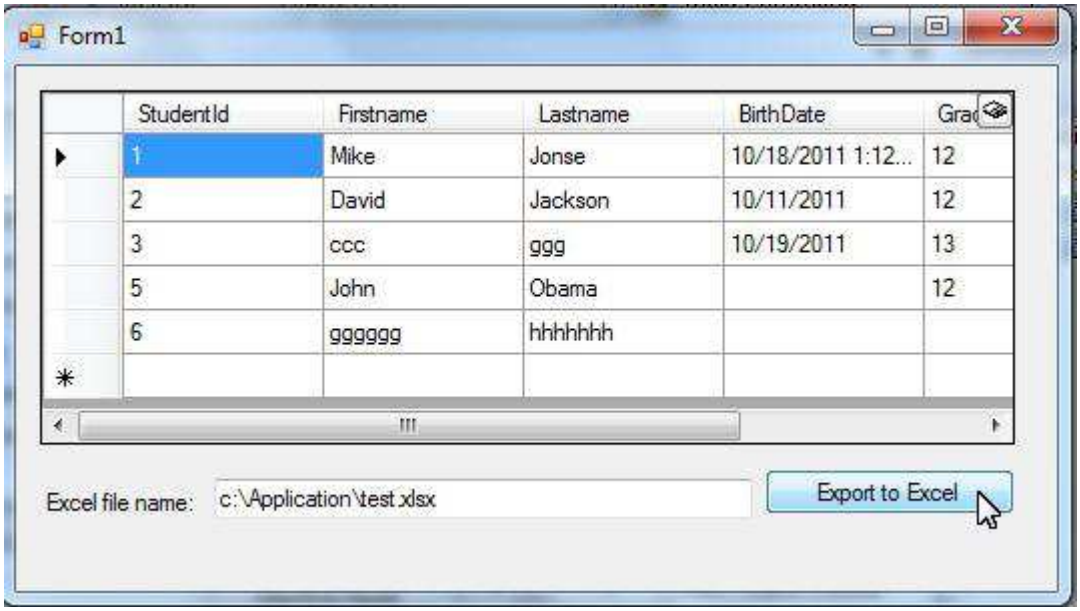
An action is created and linked to the button:



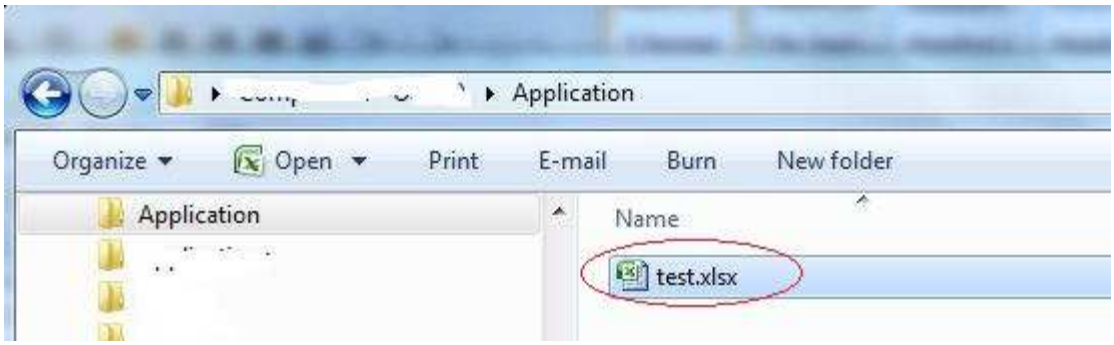
Click Run button to test the application



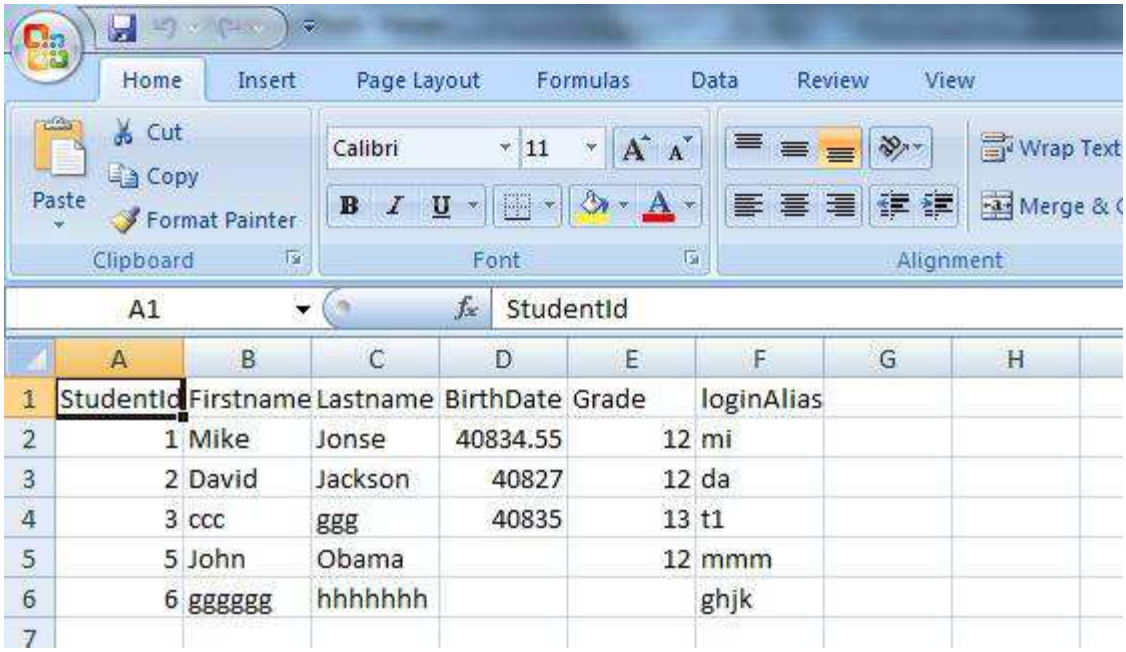
The form appears. Click button “Export to Excel”:



A new file is generated:



The Excel file contains grid data we just exported to it:



	A	B	C	D	E	F	G	H
1	StudentId	Firstname	Lastname	BirthDate	Grade	loginAlias		
2	1	Mike	Jonse	40834.55	12	mi		
3	2	David	Jackson	40827	12	da		
4	3	ccc	ggg	40835	13	t1		
5	5	John	Obama		12	mmm		
6	6	gggggg	hhhhhhh			ghjk		
7								

Feedback

Please send your feedback to support@limnor.com