

Use Arrays and Collections

Contents

Introduction	1
Create “Act on each item” action.....	1
Use Array and Collection in a Method.....	9
Create a method.....	9
Create Action “Execute actions for all items”	10
Specify actions to be executed for each item	12
Access specific items.....	13
Test	20

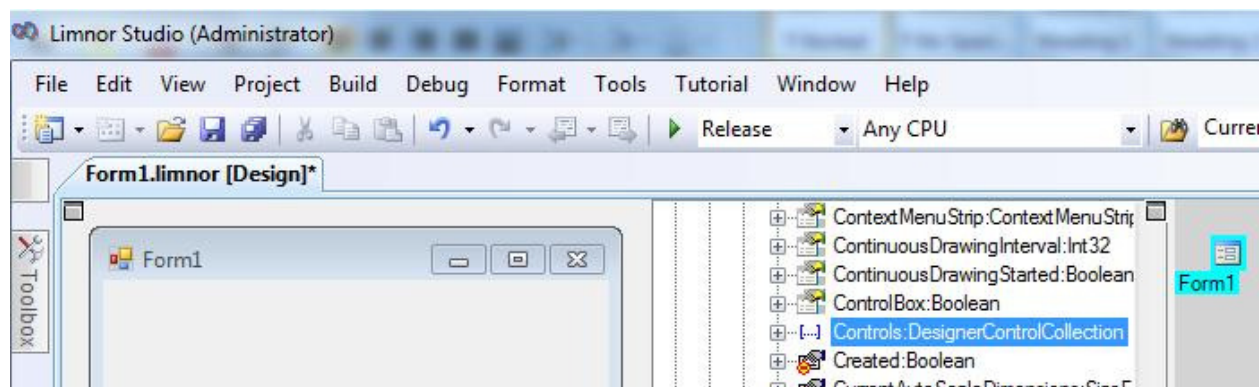
Introduction


Some components and properties are arrays, lists, or collections. Each of them may hold many items. Using of them usually include **getting specific items** or **going through all items**. We'll use an example of processing database records to demonstrate both usages.

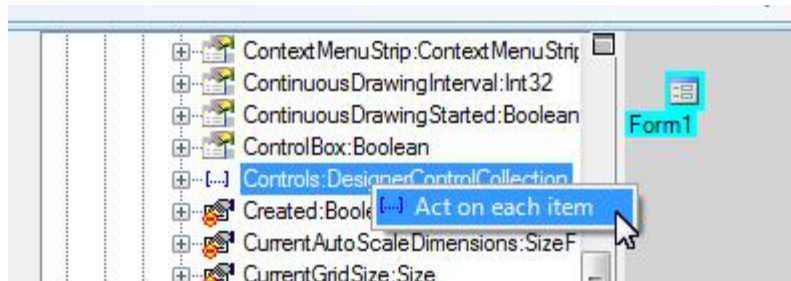
“Going through all items” means that we create some actions to act on each item one by one. We first use a simple example to show how this is done.

Create “Act on each item” action

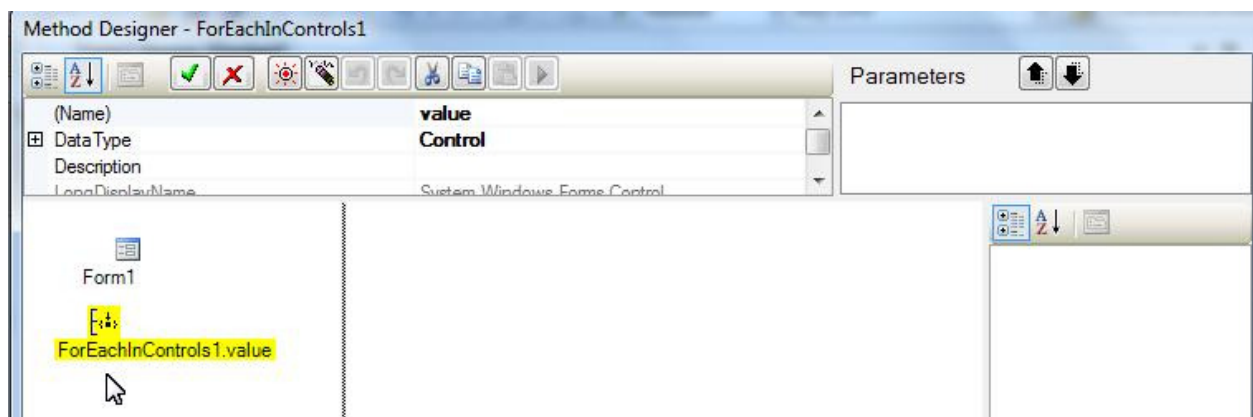
Create a Windows Form Application. A Form is created for us. The form has a Controls property. It is a collection of all controls on the form with their parent being the form.



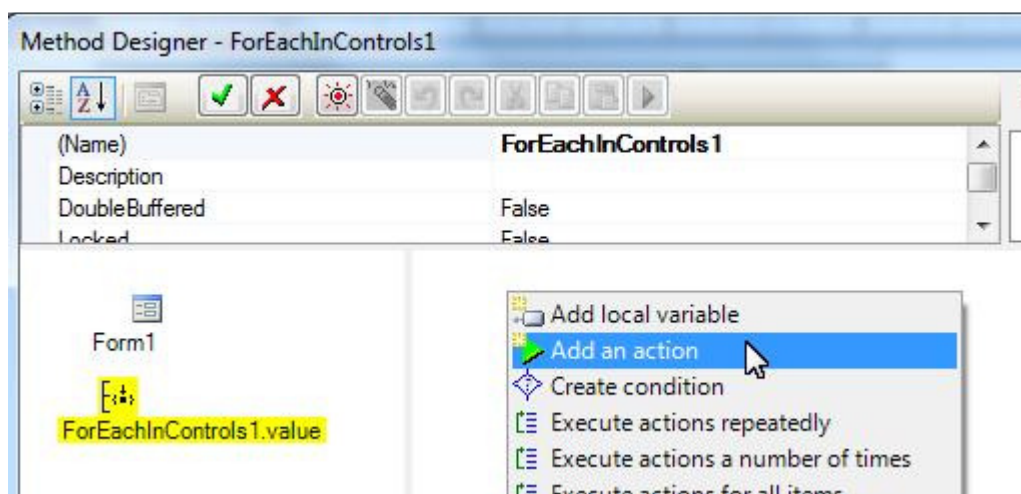
The icon, , beside the property name indicates that the property is an Array or a Collection. We may right-click the property and choose “Act on each item” to make actions to act on each item contained in the array or collection:



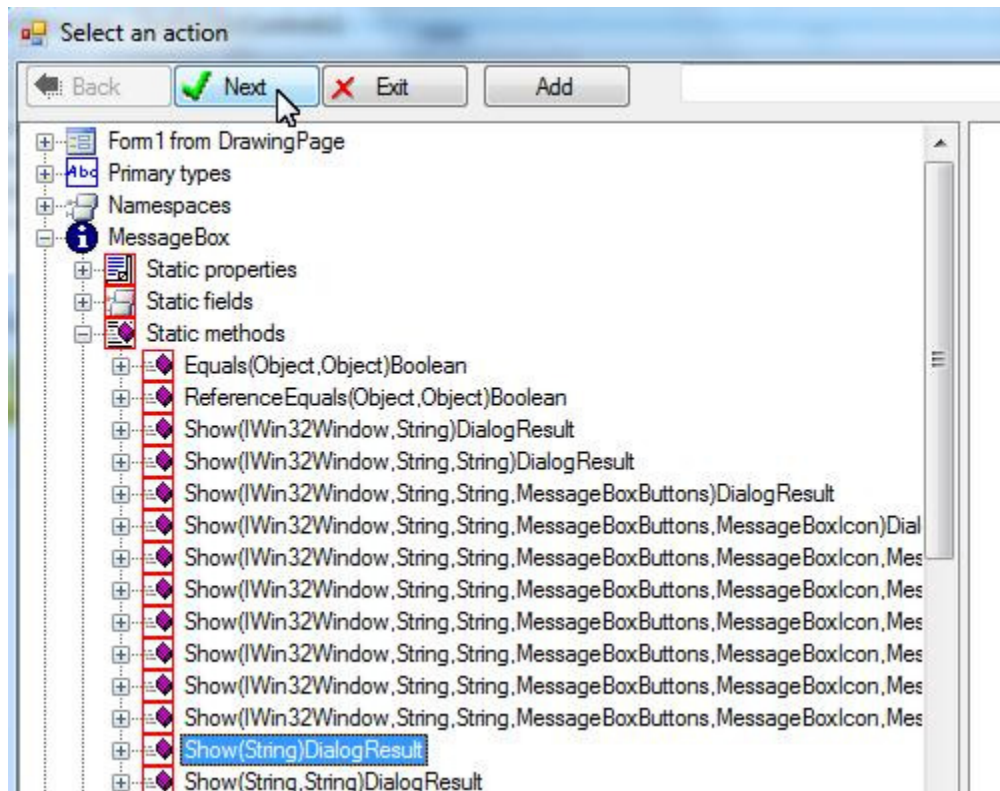
A dialogue box appears to let us create actions to act on each item. The item being acted on is represented by a “value” icon:



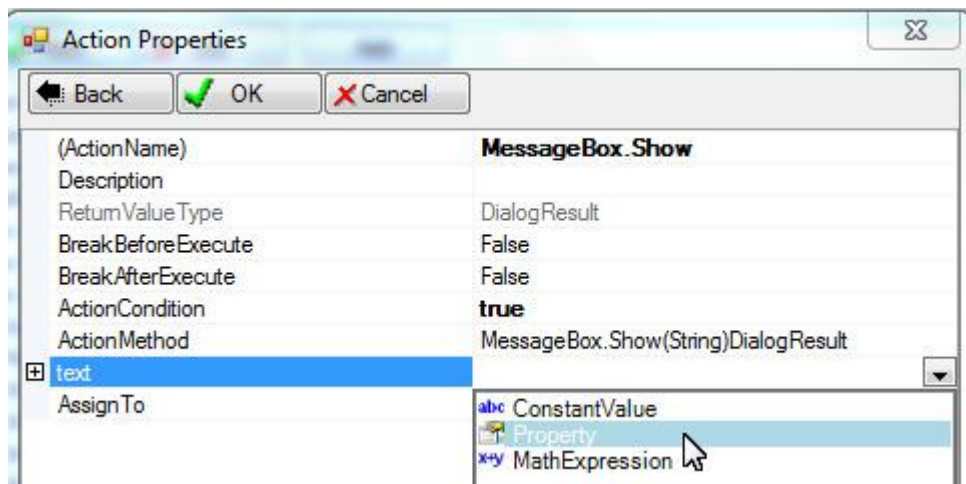
Note that the “value” is a Control on the form. As an example, we create an action to show the name of the control. Right-click the middle pane; choose “Add an action”:



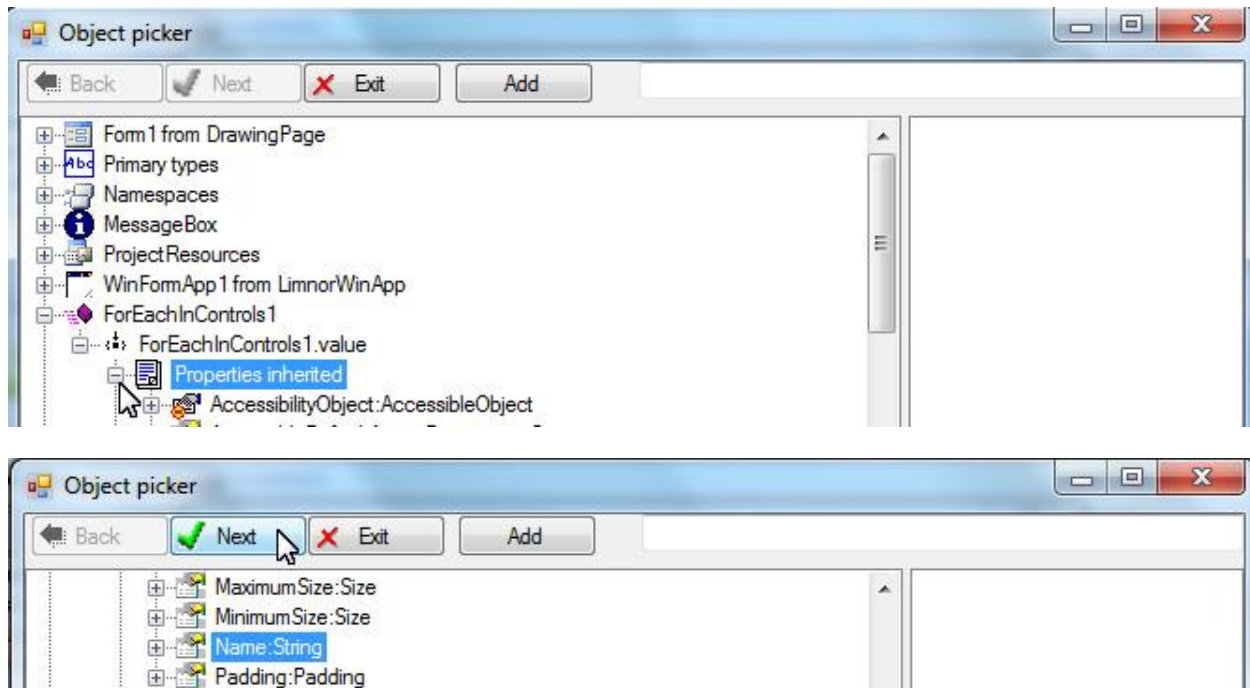
Choose “Show” method of the MessageBox:



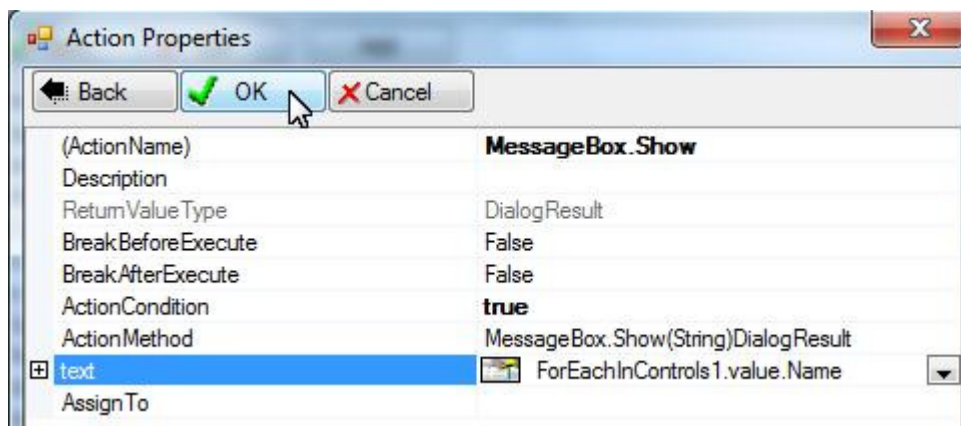
For the “text” of the action, choose “Property”:



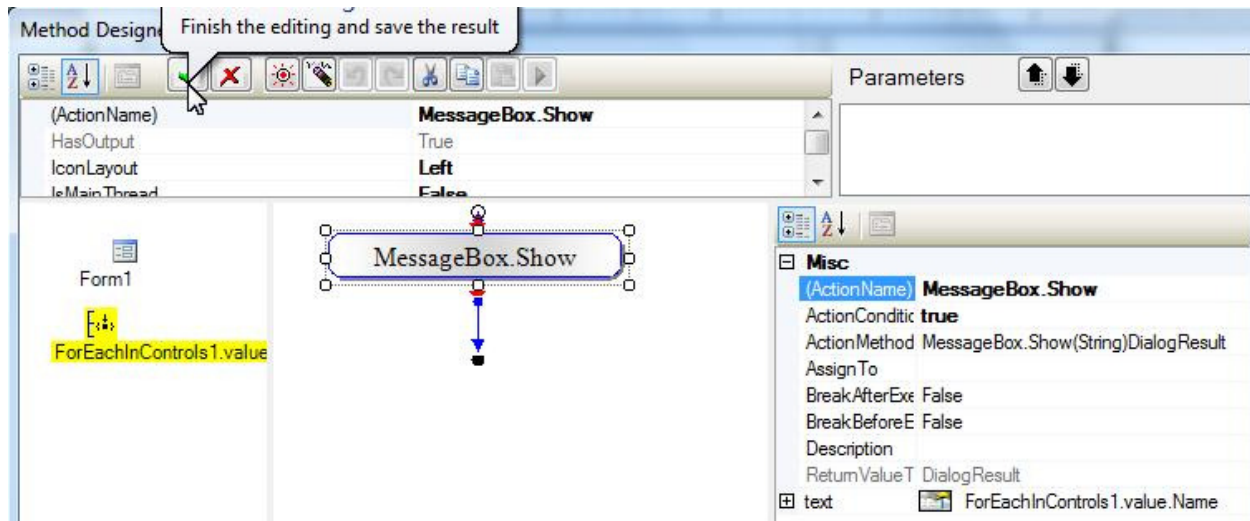
Select the Name property of the “value”:



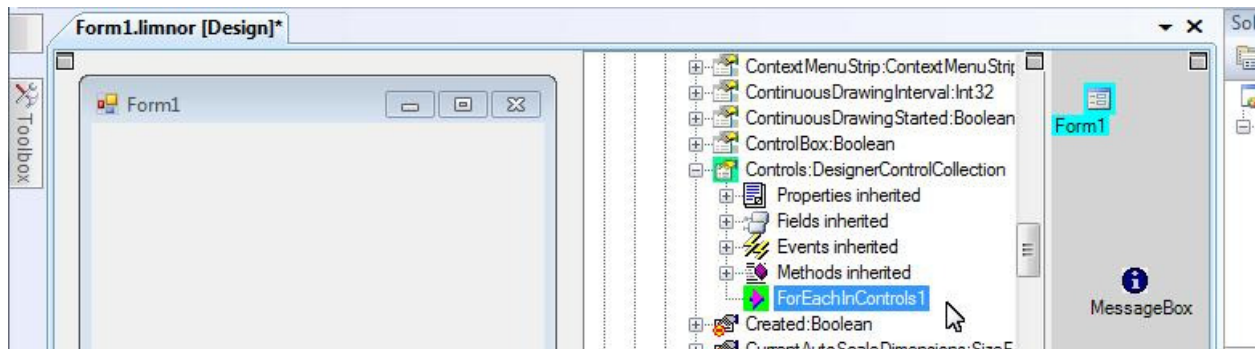
Click OK to finish creating this action:



The action appears in the Action Pane. We may add more actions if needed. For this sample we just use this one action:

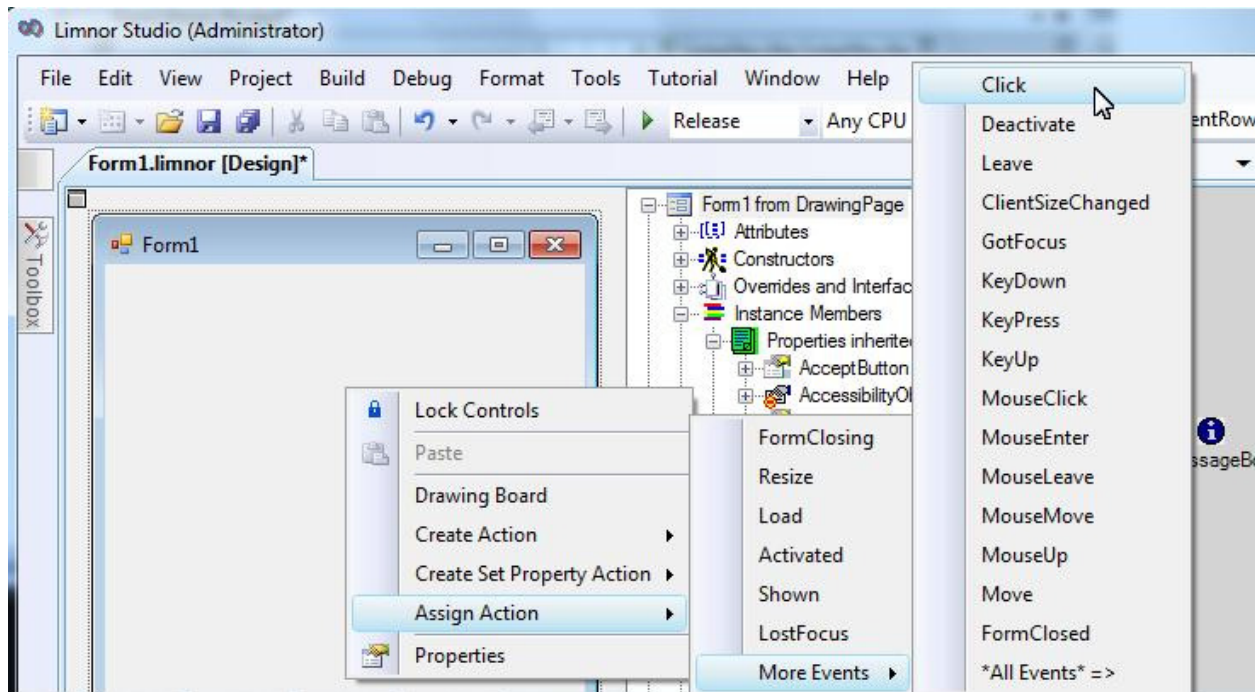


This “act on each item” action is created:

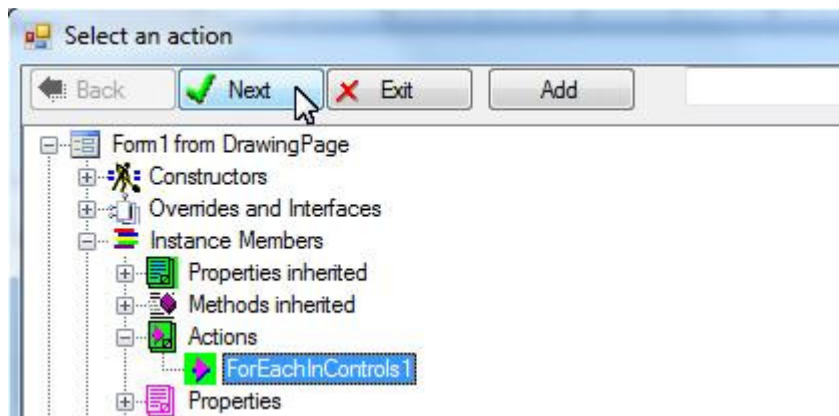


This action will show the name of each control on the form one by one.

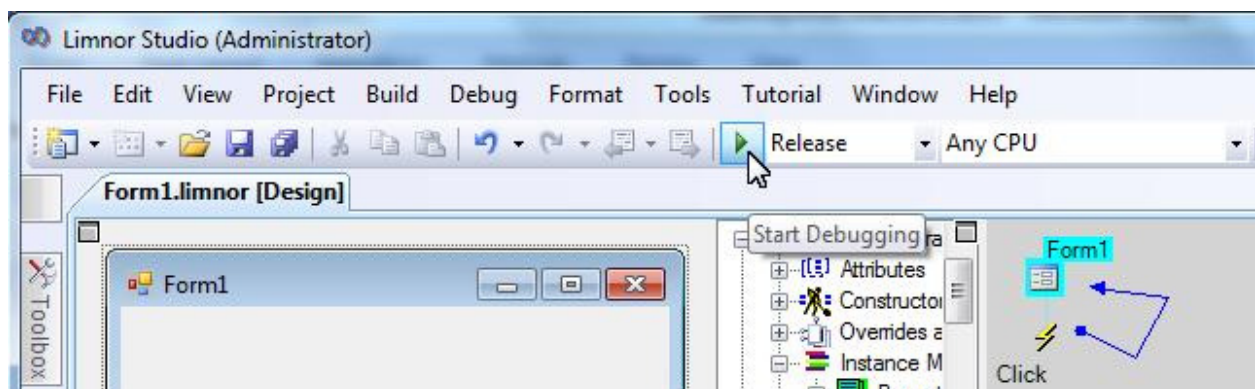
To execute this action, we need to assign it to an event. We assign it to the Click event of the form. Right-click the form; choose “Assign action”; choose “Click” event:



Select the action and click “Next”:



We may test the application:

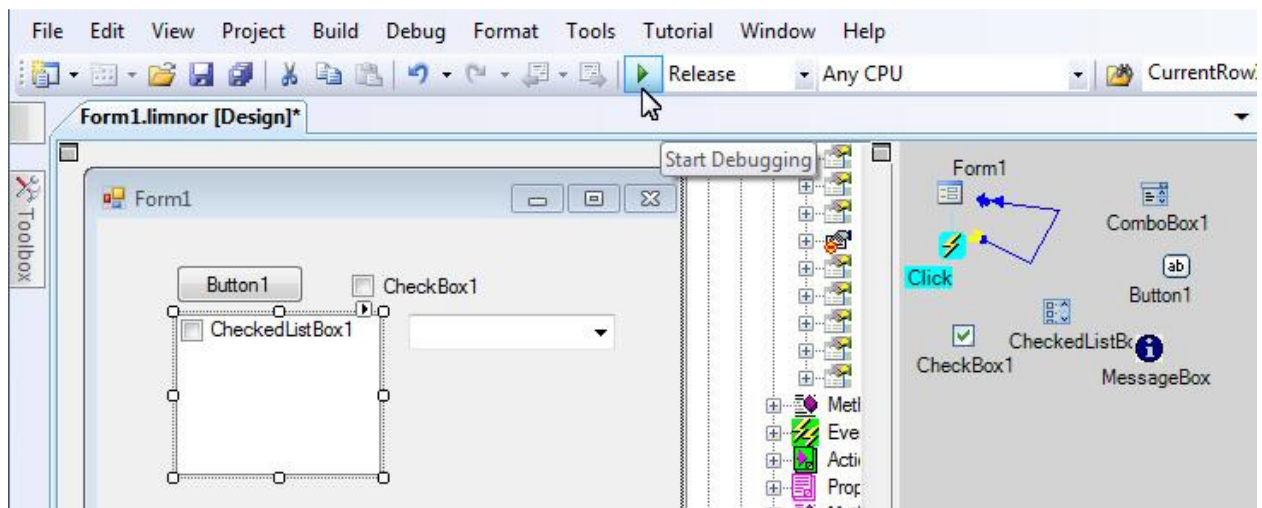


The form appears:

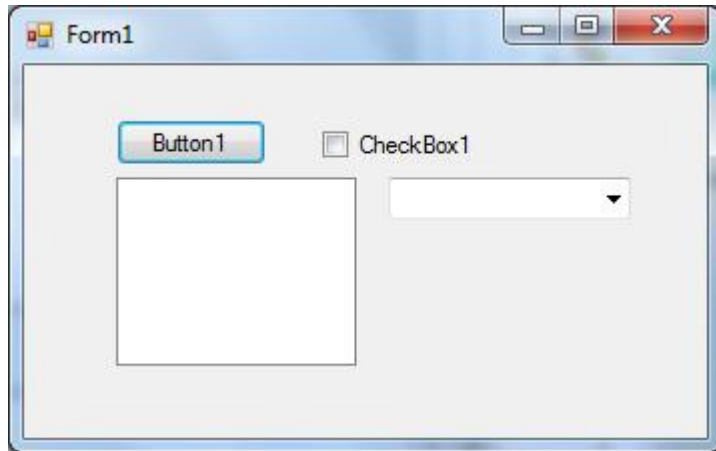


Click the form. Nothing happens. It is because there is not a control on the form. The Controls property is empty.

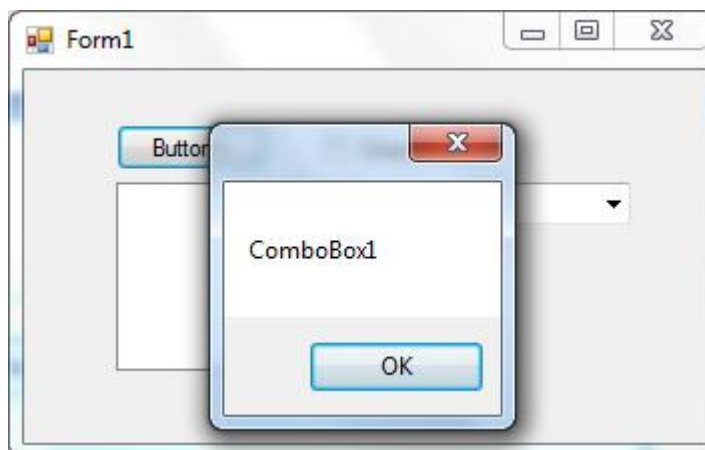
Let's go back to the design mode and randomly drop some controls from the Toolbox to the form and test it again:



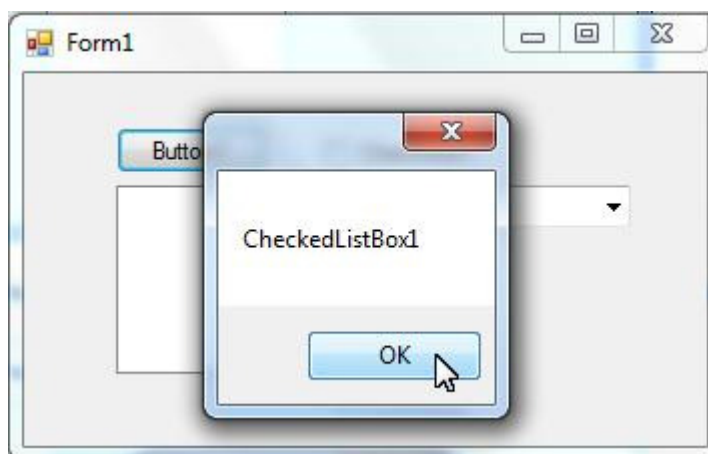
The form appears:



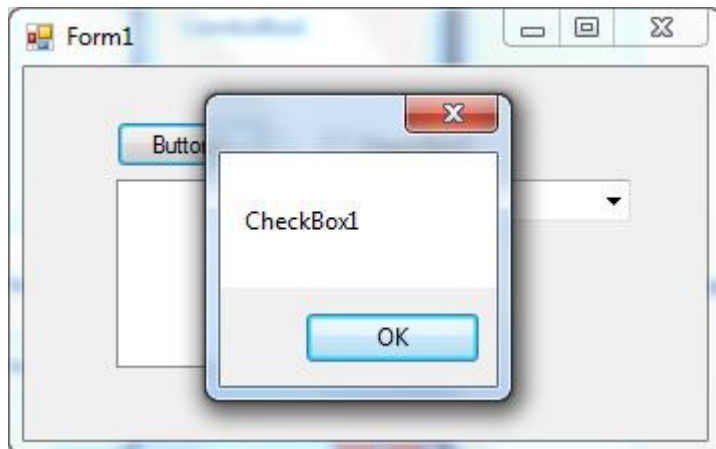
Click the form. This time we see a message box appears show a control name:



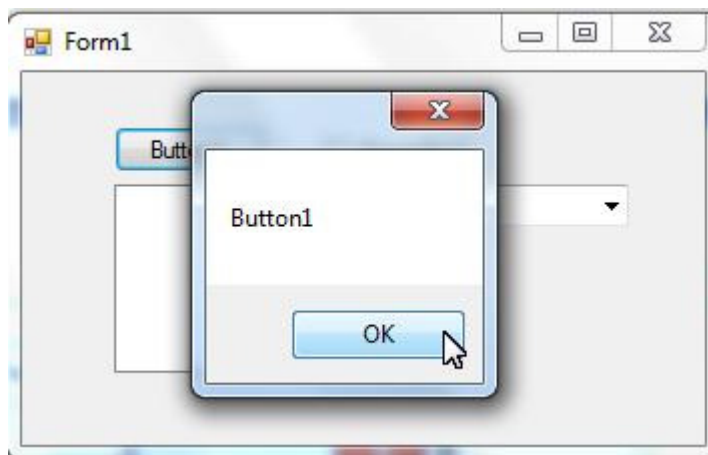
Click the OK button of the message box. Another message box appears showing another control name:



Click OK again, another control name appears:



Click OK again, another control name appears:



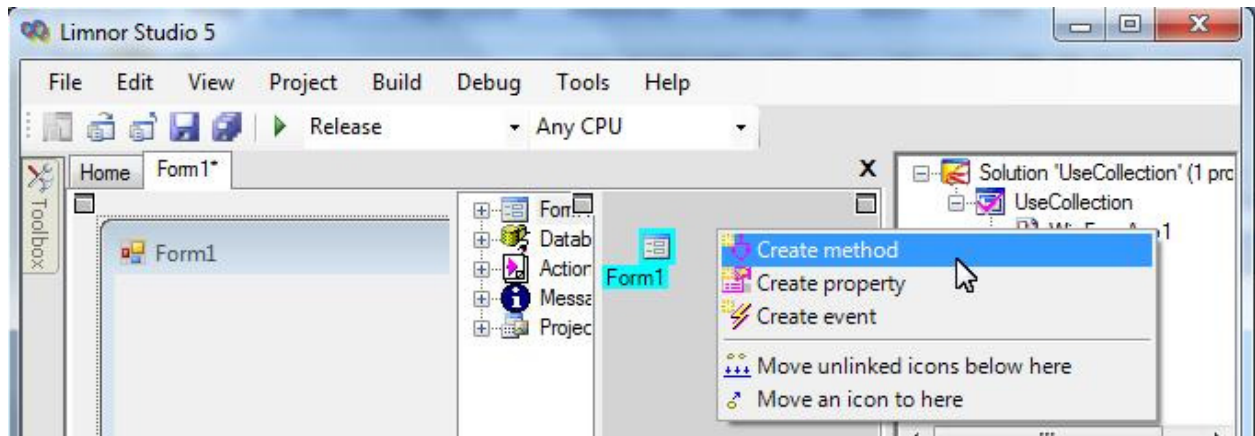
Click OK again. This time no more message box appears because it has gone through all the controls in the Controls collection.

Use Array and Collection in a Method

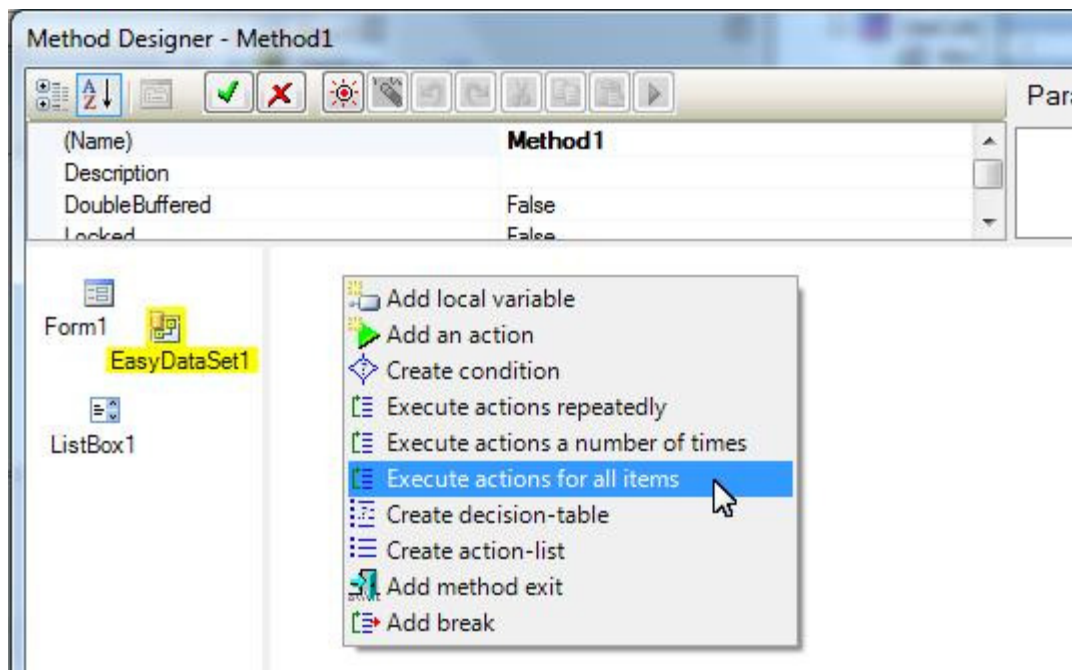
We use an EasyDataSet to get data from a database. We want to use each row of data in the EasyDataSet to form a string and add it to a list box. Going through each row of data demonstrates the skill of “going through all items”. Each row holds many cells. We use value of each cell to form a string. It demonstrates the skill of “getting specific items”.


Create a method

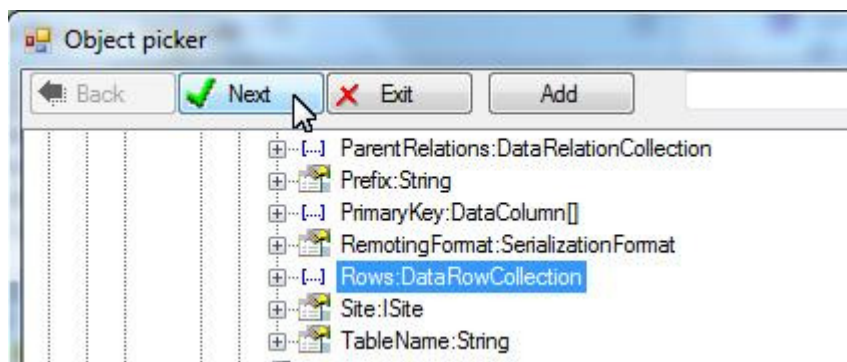
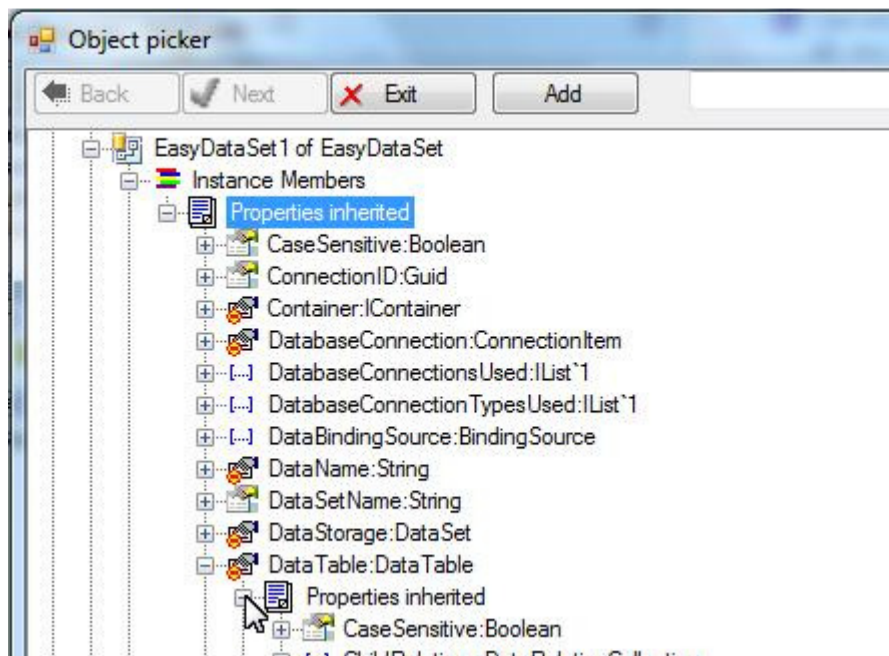
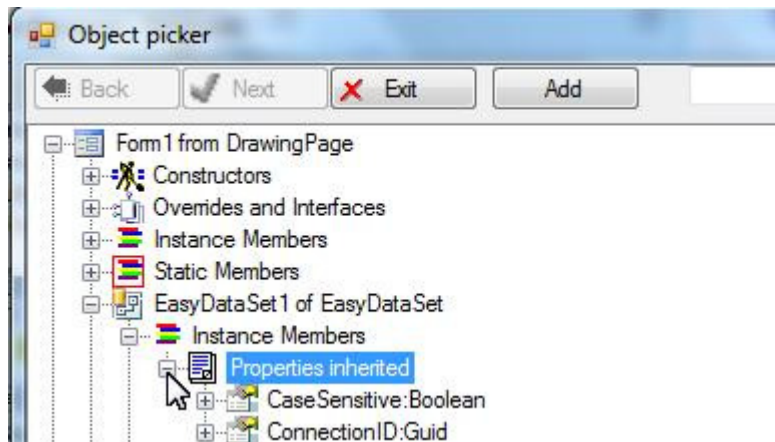
The actions are to be created inside a method. So, let's create a method.



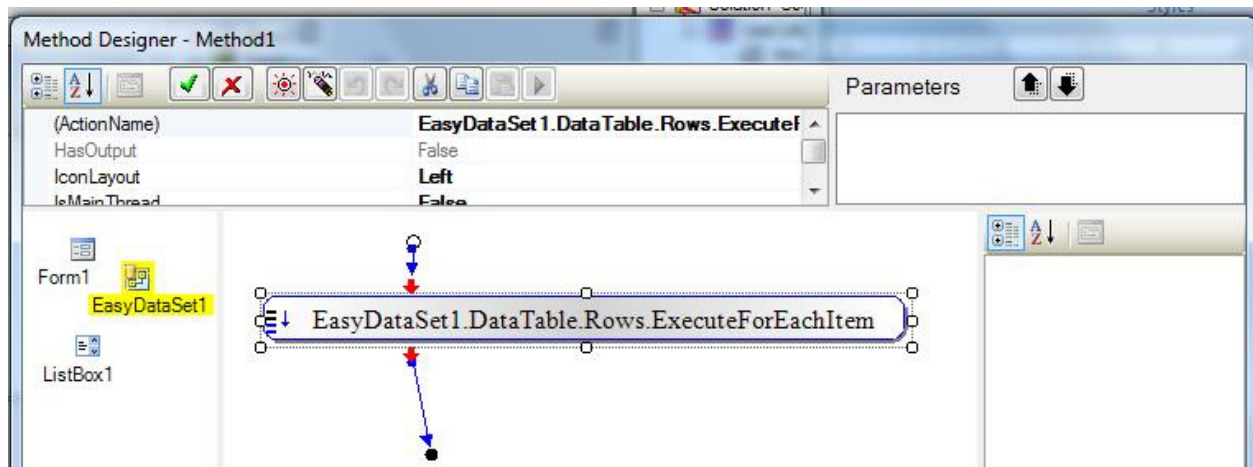
Create Action “Execute actions for all items”



It shows a dialogue box for selecting an array, a list, or a collection. A property with icon  is such a data. We need to find the Rows of the EasyDataSet:

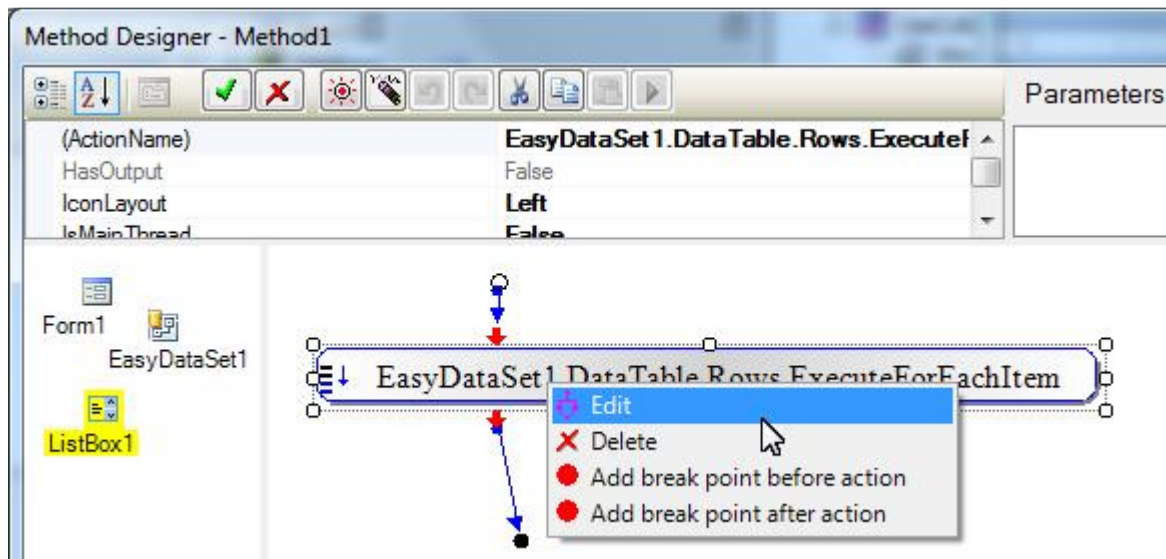


An “Execute for each item” action is created:

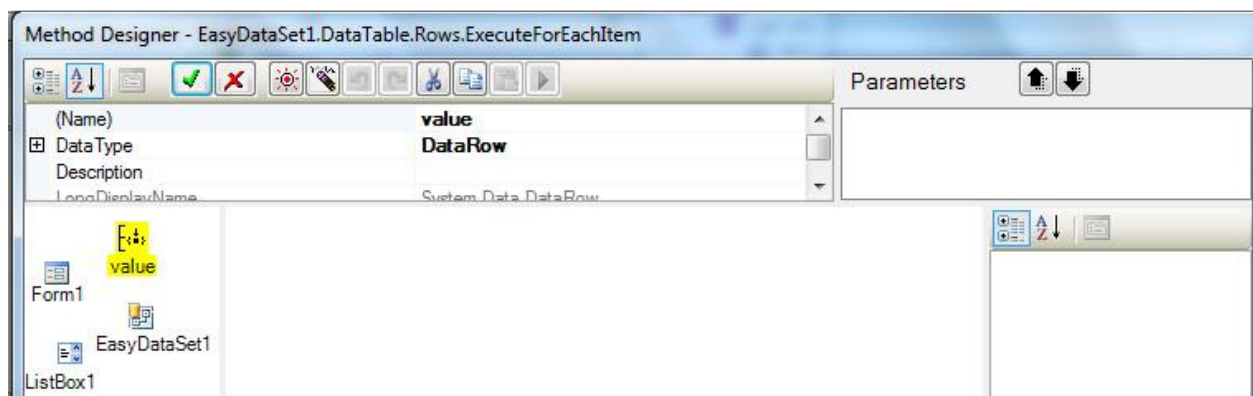


Specify actions to be executed for each item

To specify actions to be executed for each item, right-click the action, choose "Edit":



A Method Editor appears for specifying actions to be executed for each item:



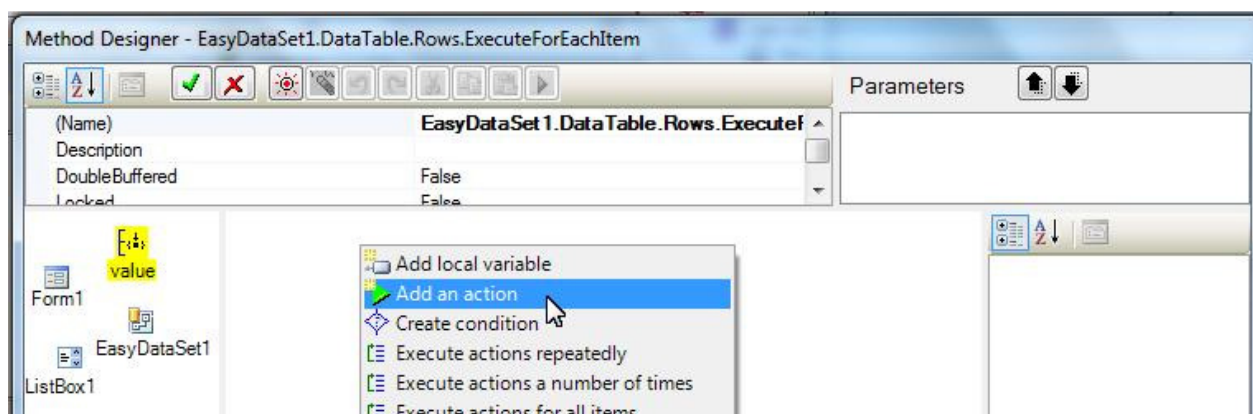
We have kept mentioning “for each item”; the “value” icon in the Method Editor represents the item. In our example, the “value” icon represents a row of the Rows collection in the EasyDataSet.

When action “EasyDataSet1.DataTable.Rows.ExecuteForEachItem” is executed, actions we added to the above Method Editor will be executed repeatedly for each row. For the first time of execution, “value” represents the first row; for the second time of execution, “value” represents the second row; and so on.

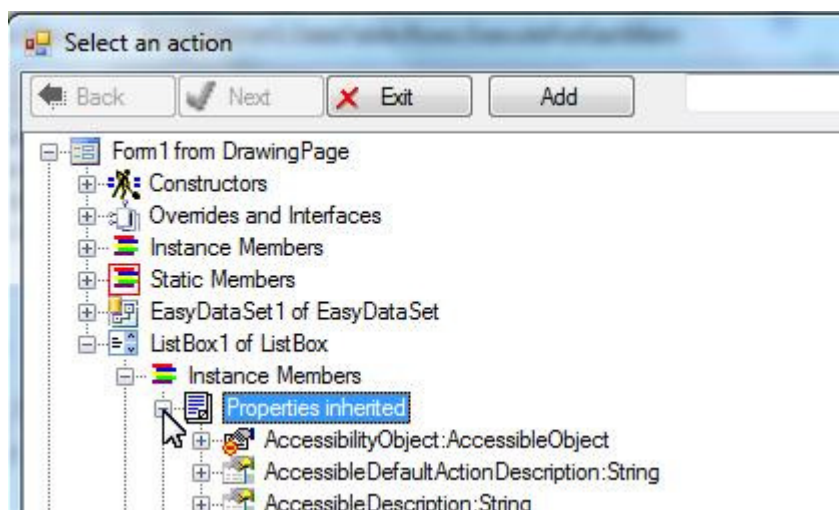
Now let’s add actions to the Method Editor. For this sample we append an item to a list box. The item is formed by columns of the row represented by “value” icon.

Access specific items

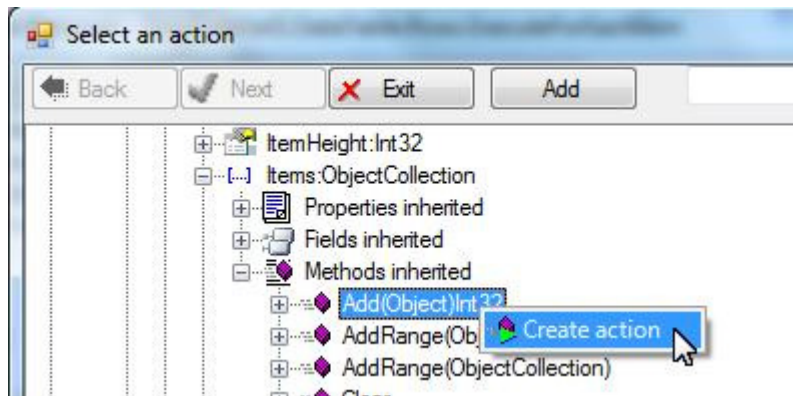
Create a new action for appending an item to the list box:



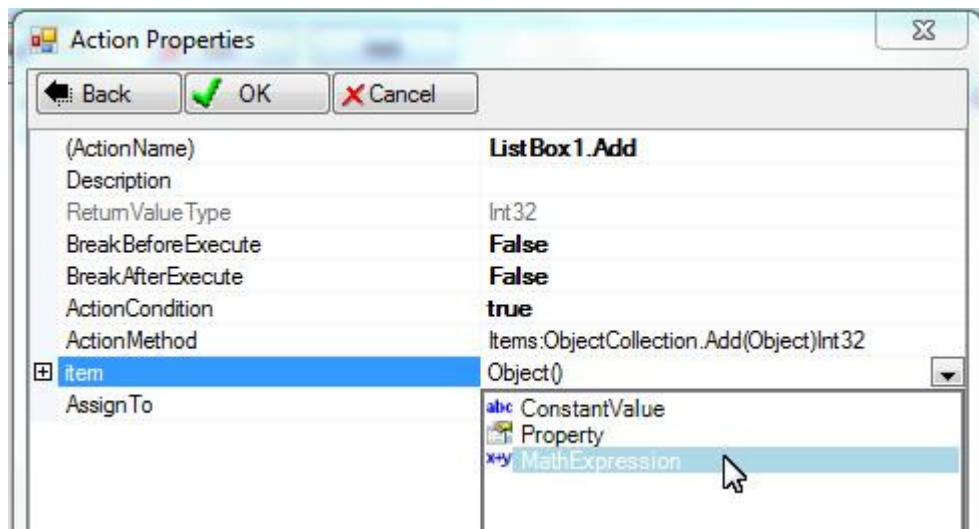
Find the Items property of the list box:



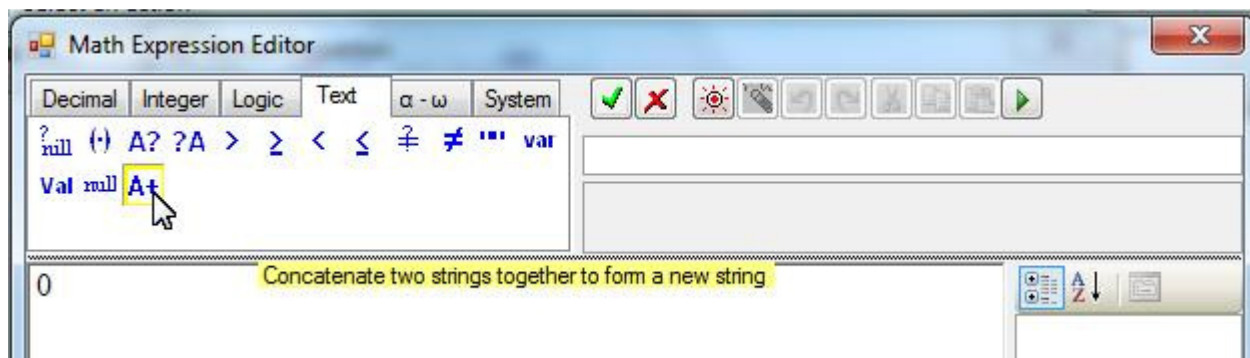
The Items property of the List Box has a method “Add”. Right-click “Add” method and choose “Create action”:



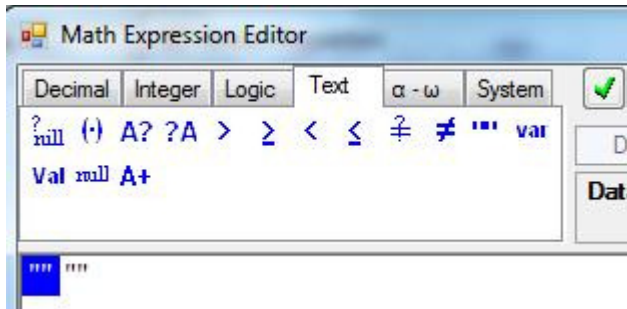
“item” parameter of the action is the item to be appended to the list box. Choose “Math Expression” to form a string using the columns from the row:



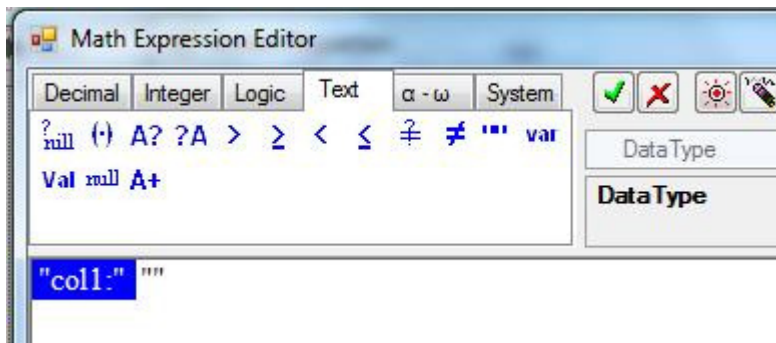
Click A+ to form string concatenation:



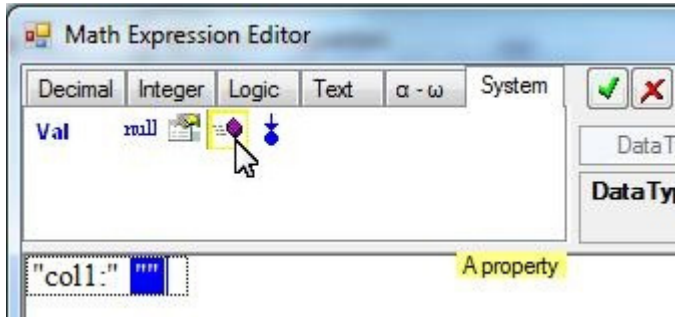
Select the first element:



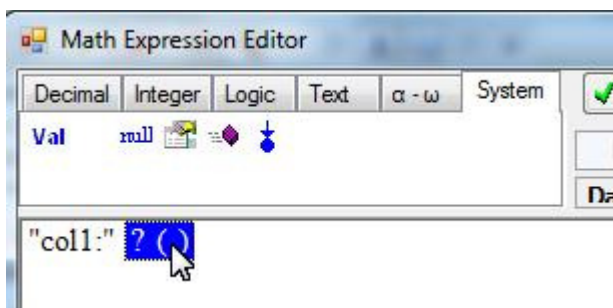
Type col1:



Select the second element and click the Method icon:

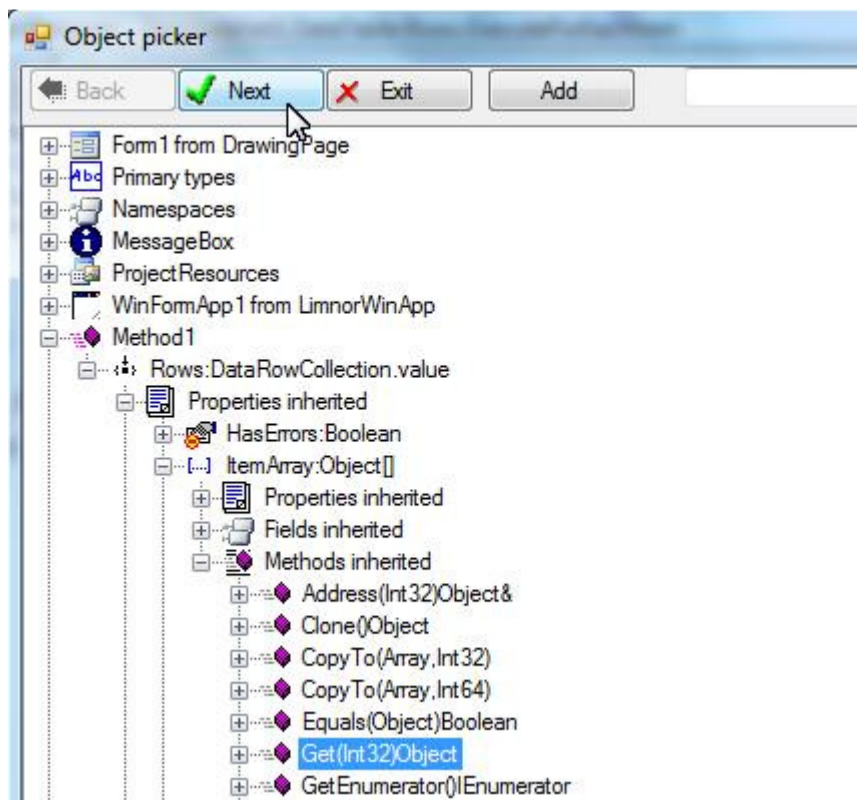


Double-click the Method element:

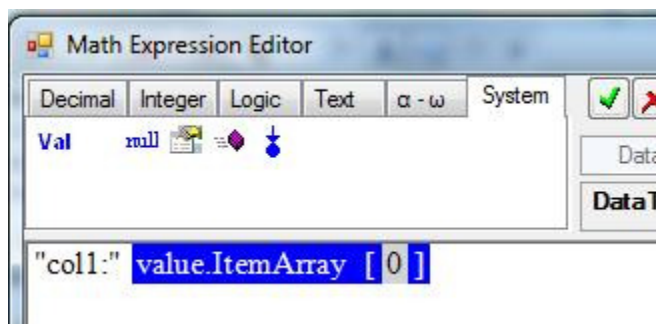


The column values of a row are in a property named ItemArray. It is an array of objects, as its icon [\[...\]](#) indicated. We are not going to use it by going through all its items; instead we are going to access its

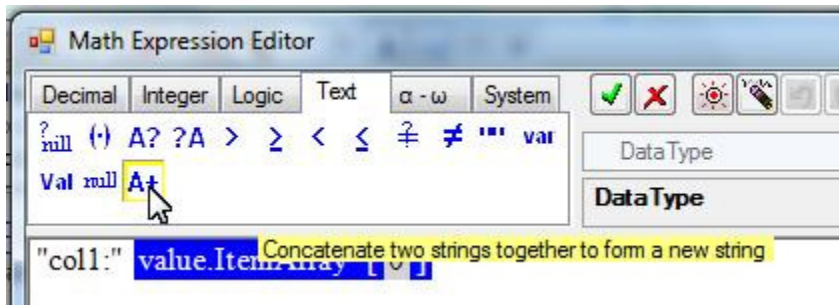
items individually. To do it we need to use its “Get” method. Its “Get” method uses an index to indicate which item we want to get. 0 indicates the first item; 1 indicates the second item; and so on.



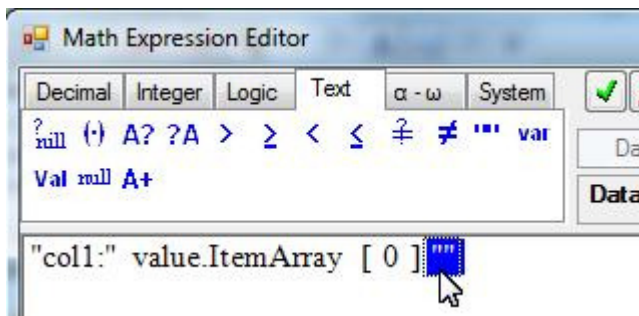
By default, 0 is used as the index:



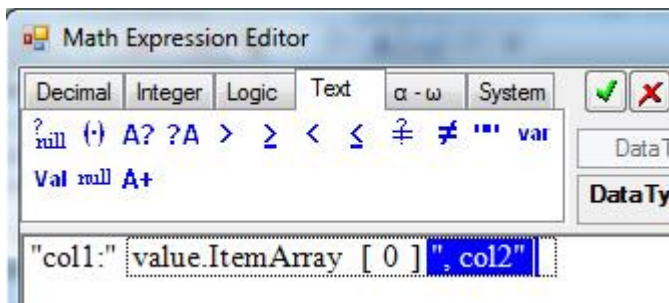
Click A+ to add more elements:



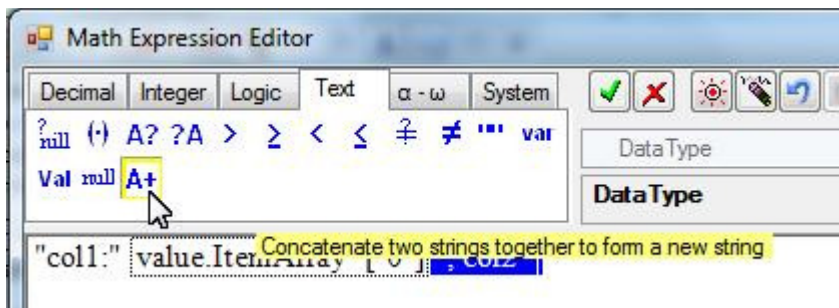
Select the new element:



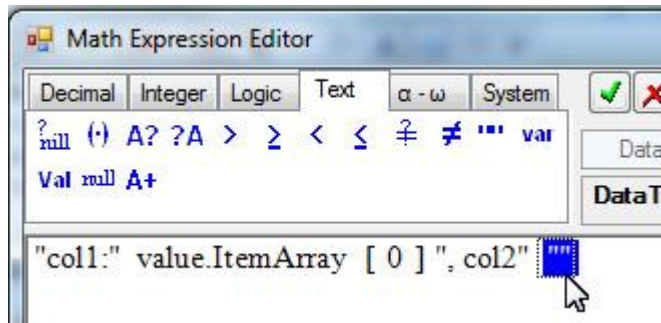
Type , col2:



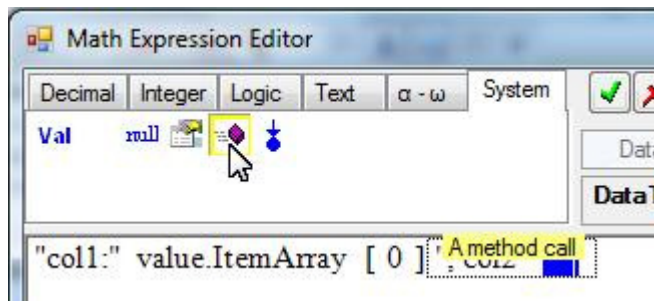
Click A+ again to add another element:



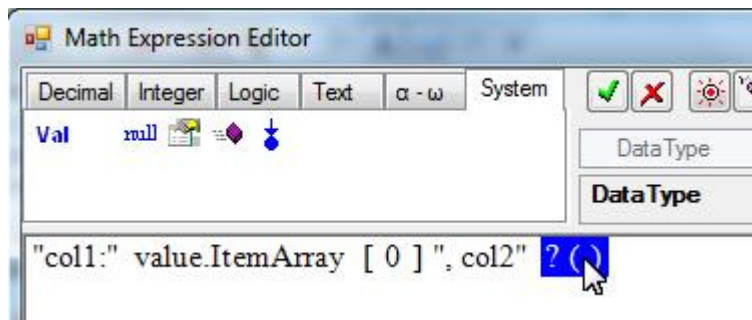
Select the new element:



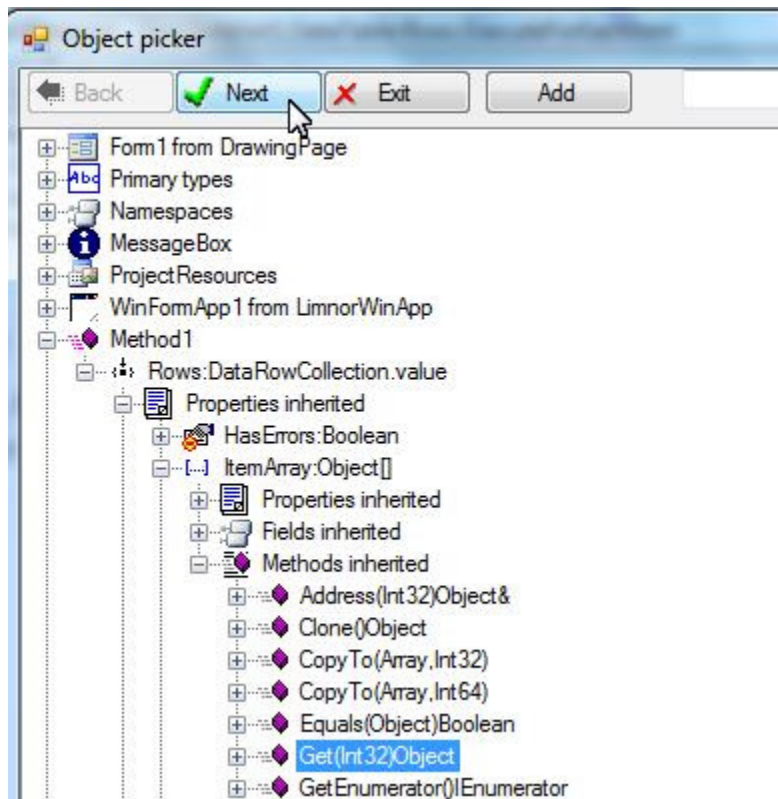
Click the Method icon:



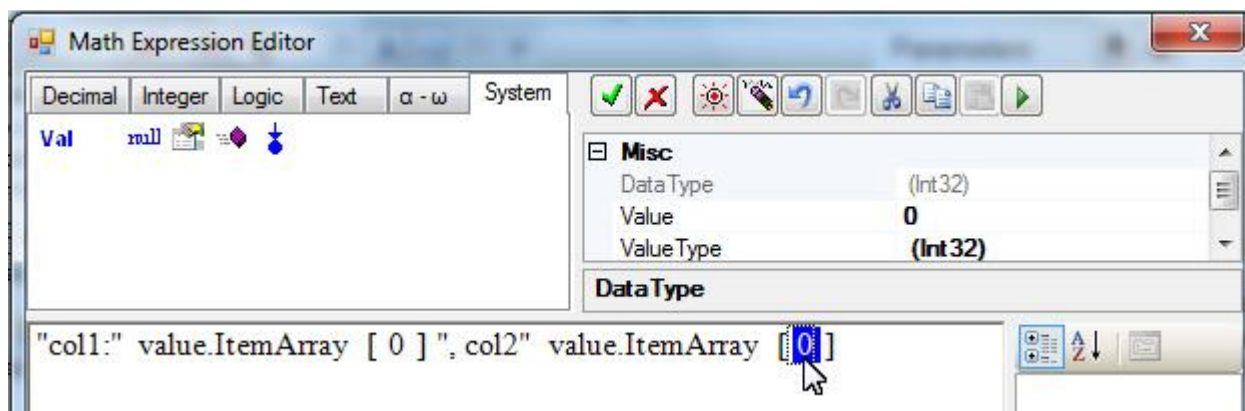
Double-click the Method element:



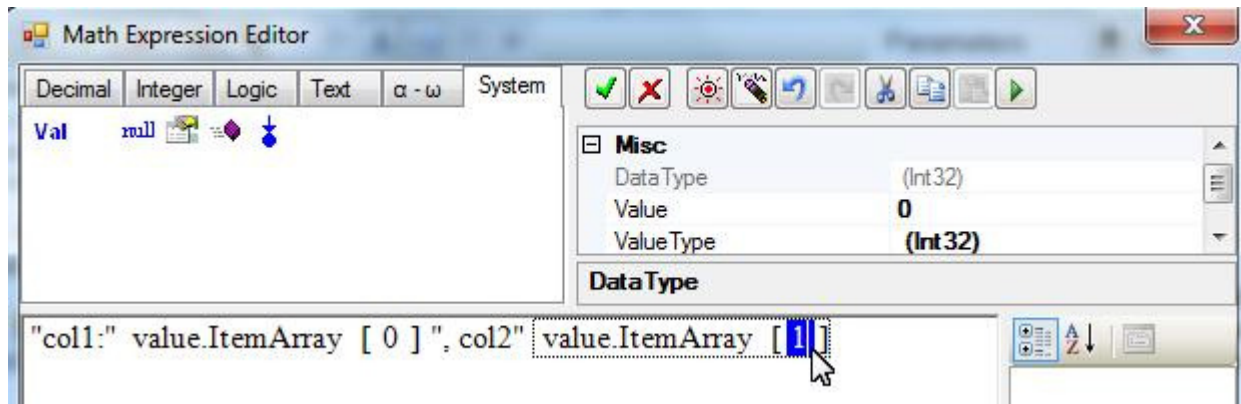
Select the Get method again:



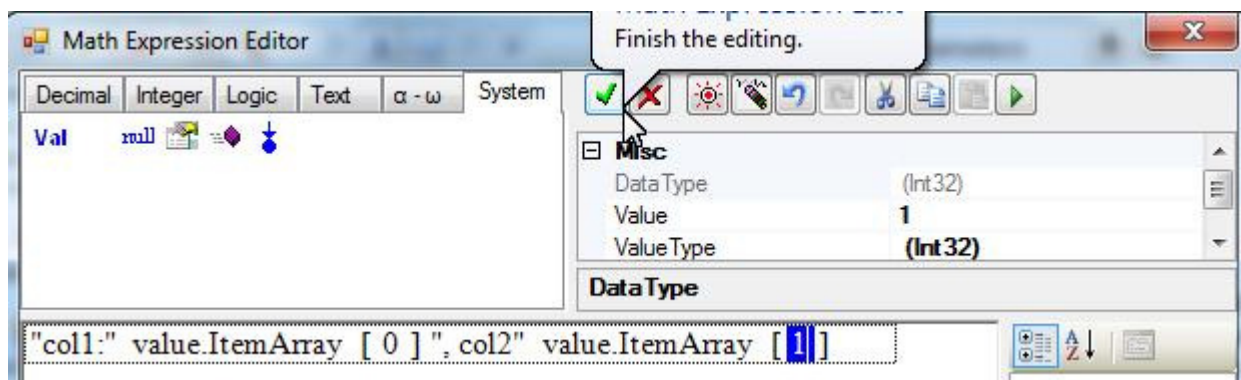
Select the index of the method:



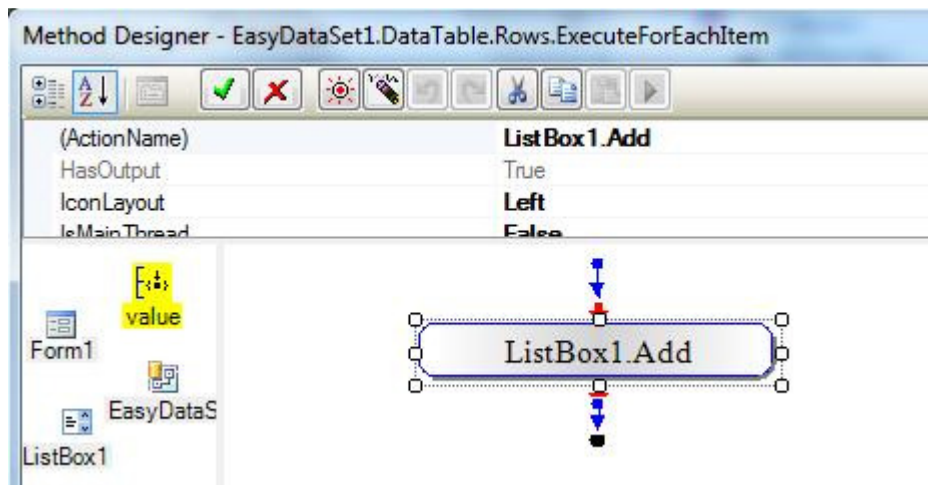
Type 1:



This sample shows getting the first and second items from an array:

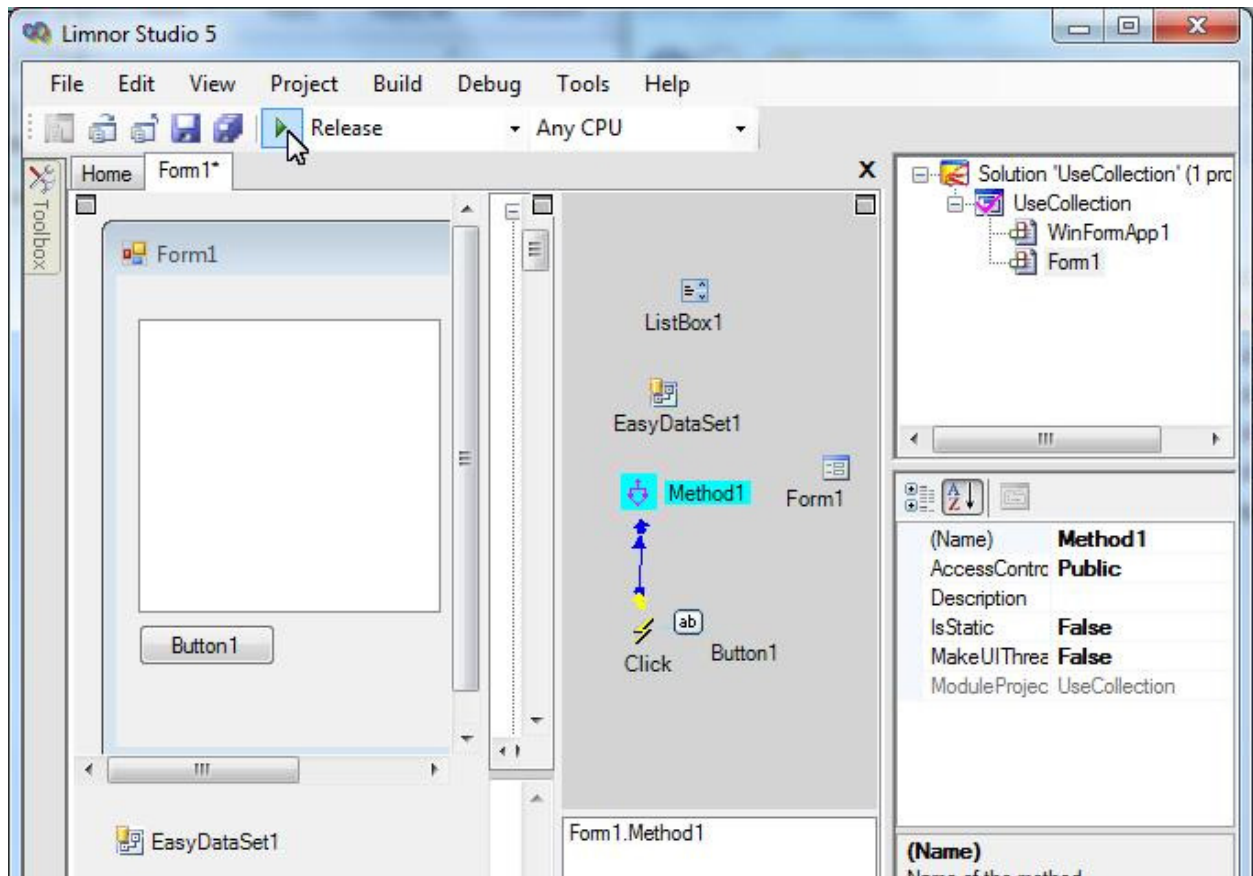


The action appears in the Method Editor:

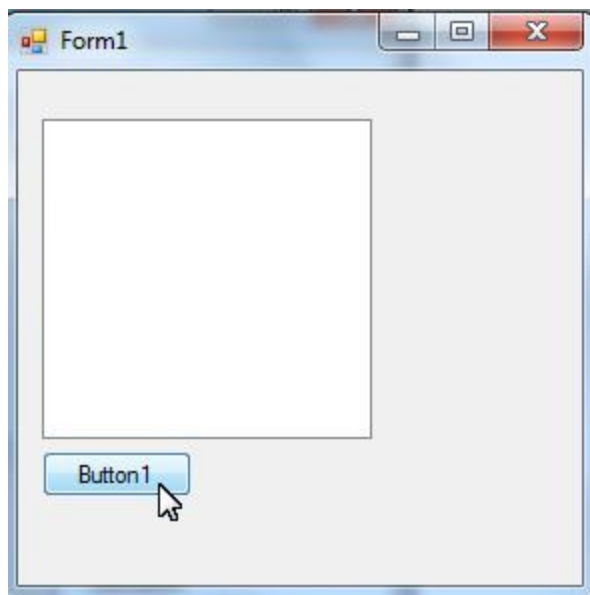


Test

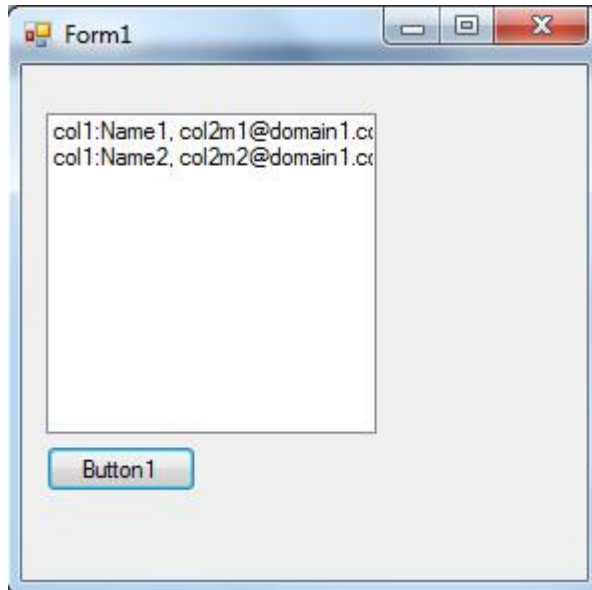
We may test our programming sample now. Create an action using Method1. Assign this action to a button. Click Run button:



Click the button:



EasyDataSet1 retrieves two records from the sample database. We see two items appear in the list box; each item is formed by values from corresponding record:



We see that a colon follows col1 but not col2. This is a mistake we made when forming the item by forming a math expression. A previous screenshot shows that we missed a colon after col2.