Limnor Studio – User's Guide

Part – V

Method Editor

Contents

1 Introduction to Method

A method consists of one or more actions, the action can be linked in various ways forming execution paths (control-flow) according to required programming logic.

There are 3 types of methods:

- Methods for actions
  Actions can be created only using such methods. Such a method is only executed through the actions using the method, or in an expression.
- Property getters and setters
  - Getters
    A getter belongs to a property. It is executed when the property value is retrieved.
  - Setters
    A setter belongs to a property. It is executed when the property is assigned a value.
- Event handler methods
  An event handler method belongs to an event. It is executed when the event occurs.

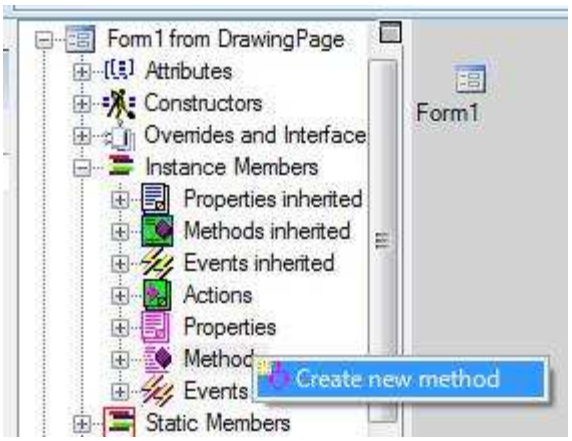
These 3 types of methods are explained in more details below. The Method Editor is used for editing all types of methods.

1.1 Action Methods

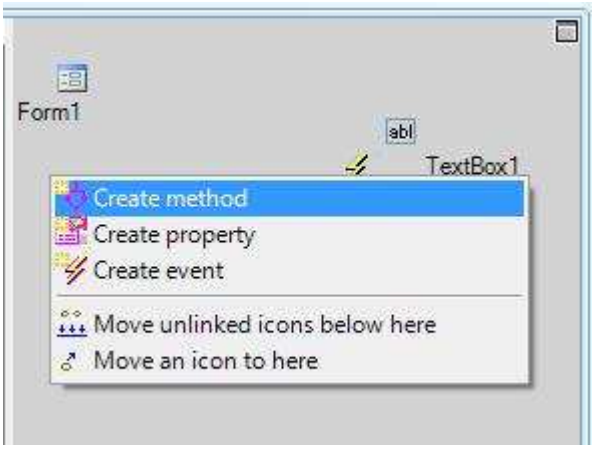
An action method represents a task capability of a class. To execute a method an action must be created using the method. The action is assigned to an event. When the event occurs the action is executed.

- The simplest method is a list of actions.
- A method may have local variables
- A method may have parameters. The parameter values are specified when actions are created using the method.
- Multiple threads can only be created by using methods
- Complex action execution path can be created within a method. For example, branching, looping, recursion, etc.

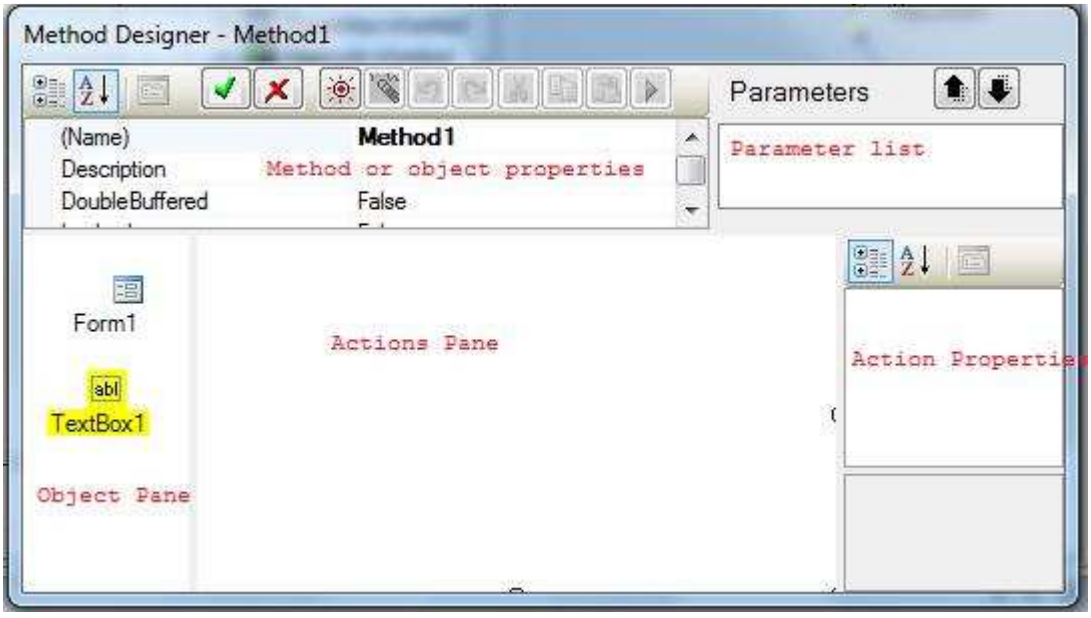
To create a new method via the Object Explorer right-click Methods, choose “Create new method”



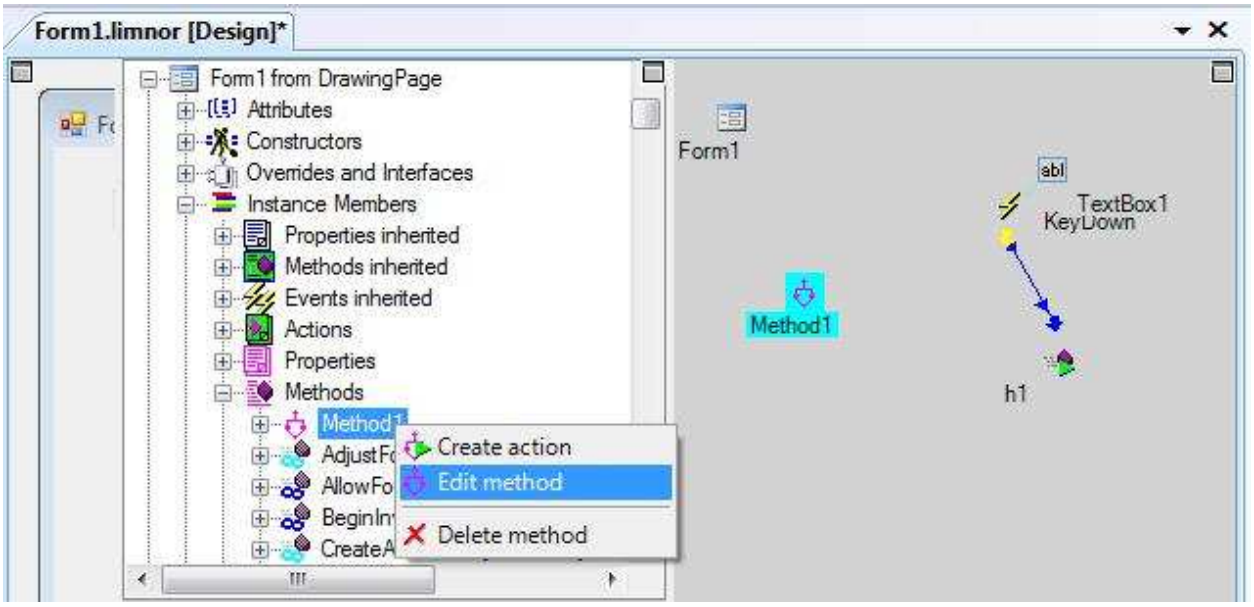
To create a new method via the Event Path, right-click and choose “Create method”

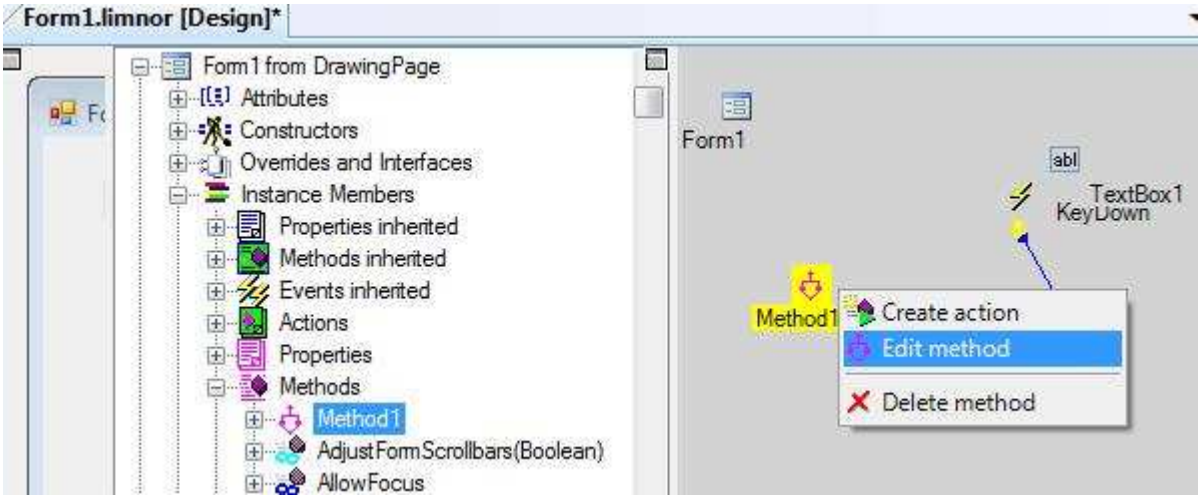


The Method Editor appears to develop the new method:



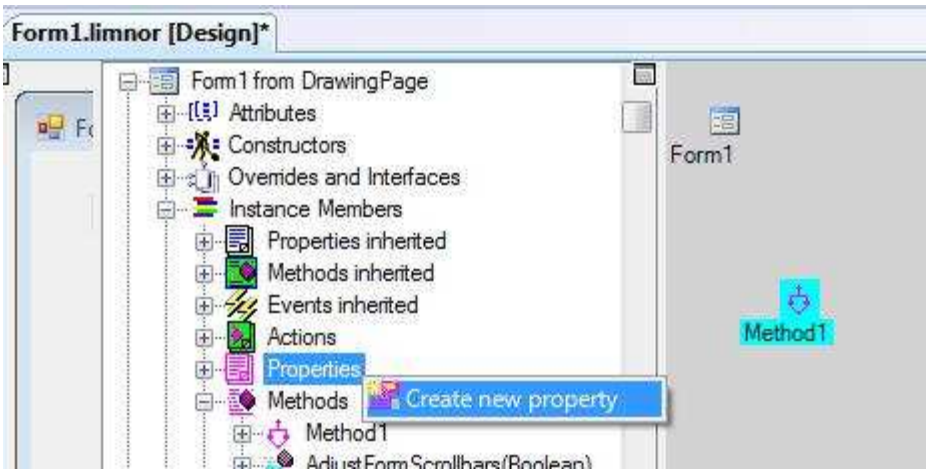
To modify an existing method, right-click the method, choose “Edit method”:

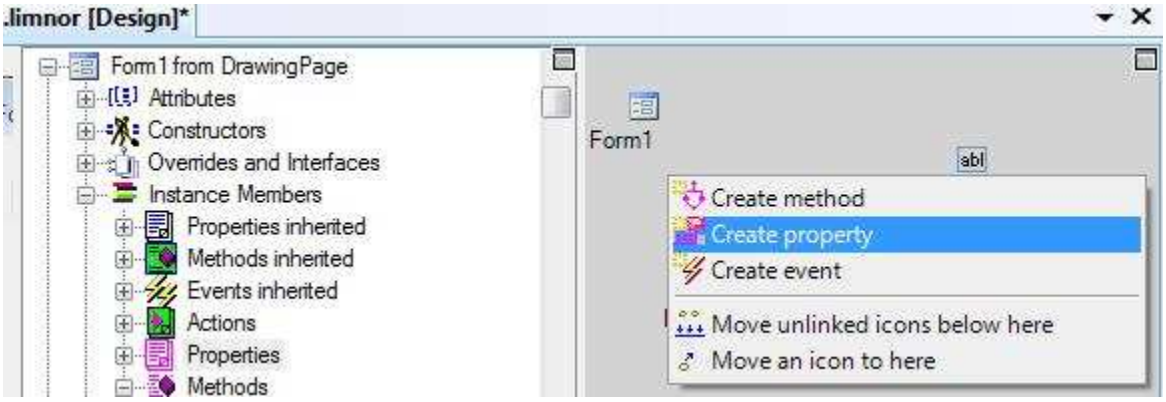




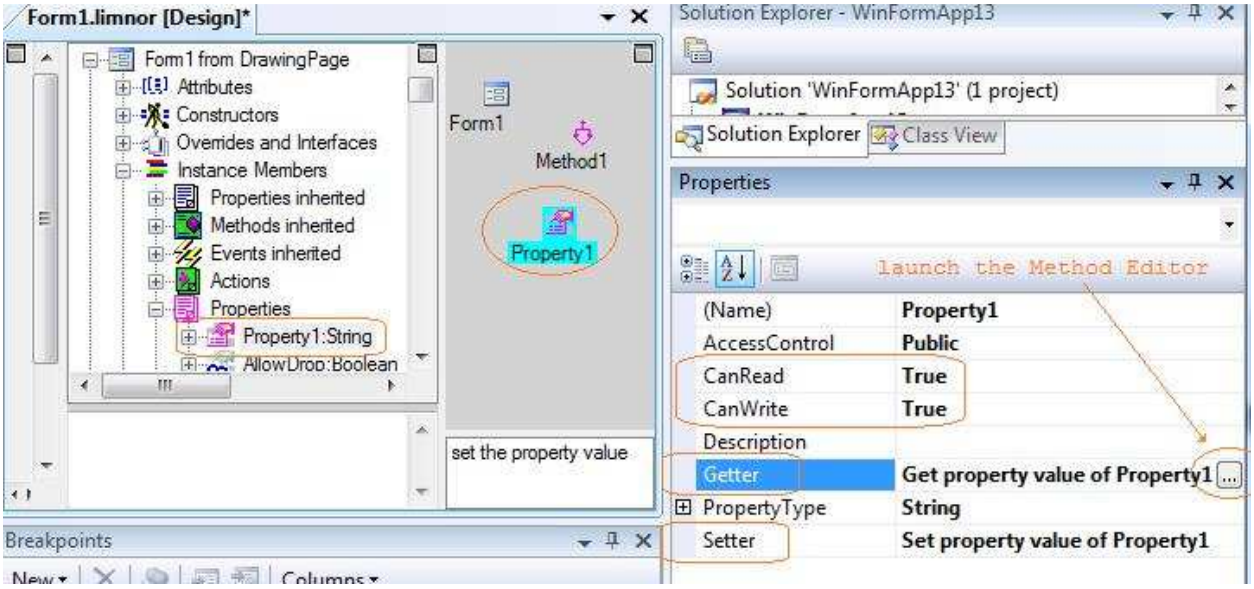
1.2 Special Methods: Property Getter and Setter

When creating a property, two special methods are automatically created.





If the property can be read from then a Getter method is created for returning the property value. If the property can be written to then a Setter method is created for assigning the value to the property.



The default Getter method has one action which returns the property value.

The default Setter method has one action which assigns the value to the property.

Getter and Setter methods are not to be used to create actions. Getter is executed when the property is read from. Setter is executed when a value is assigned to the property.

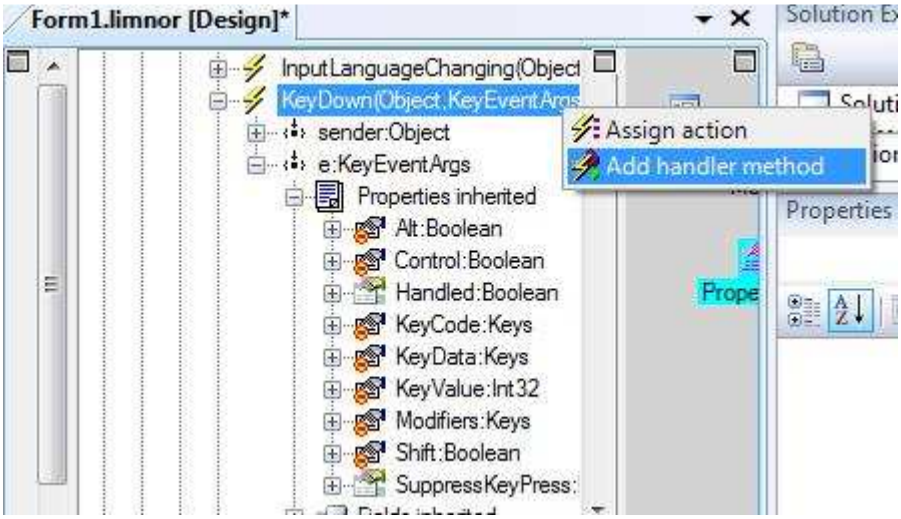
Getter and Setter can be edited, using the Method Editor, to modify its default contents.

1.3 Special Methods: Event Handler Methods

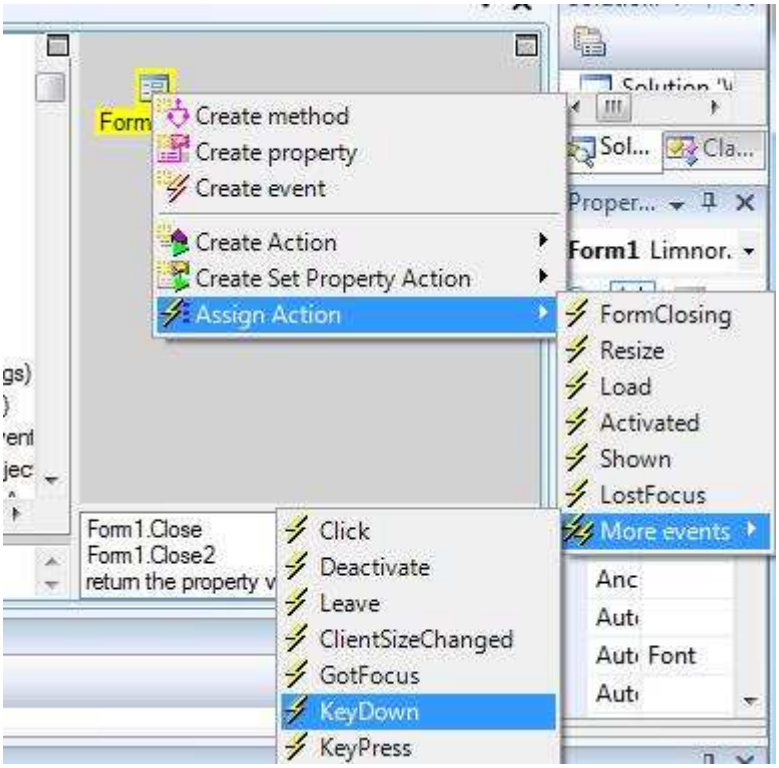
In many events useful information comes with the events. For example, a mouse move event comes with information about the mouse pointer position. To access such information, an event handler method can be created.

You cannot create actions from an event handler method. An event handler method will be executed when the event occurs.

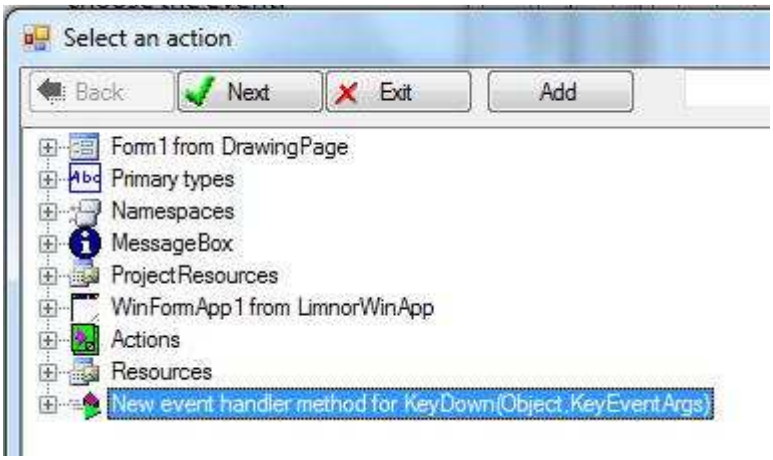
To create an event handler method via the Object Explorer, right-click the event, choose “Add handler method”:



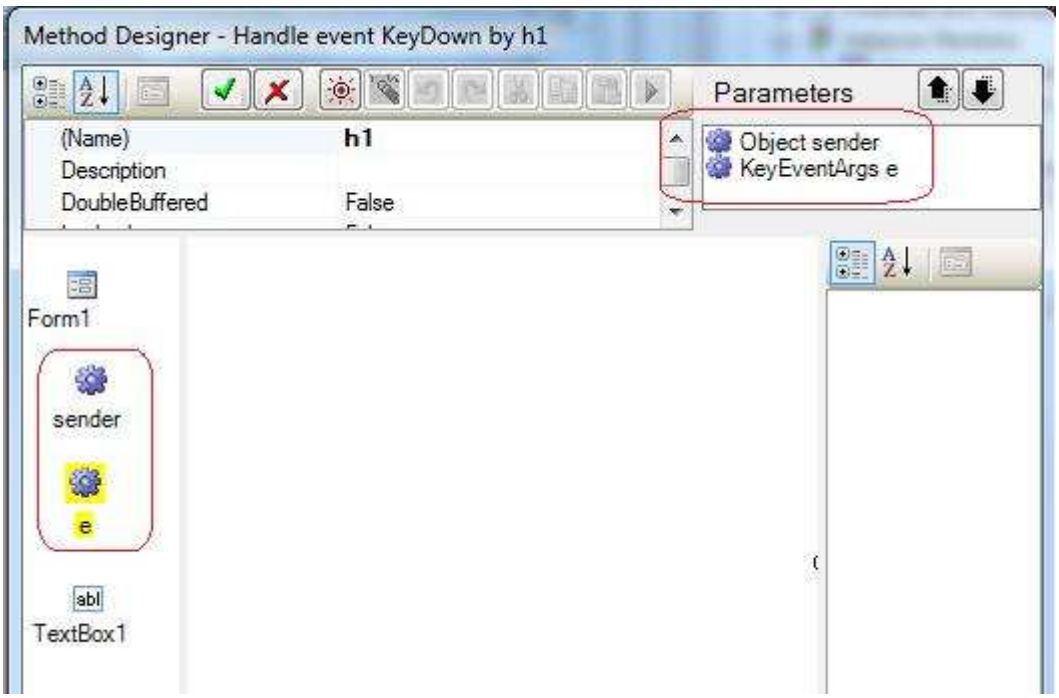
To create an event handler via the Event Path, Right-click the icon for the object, choose “Assign Action”, choose the target event:



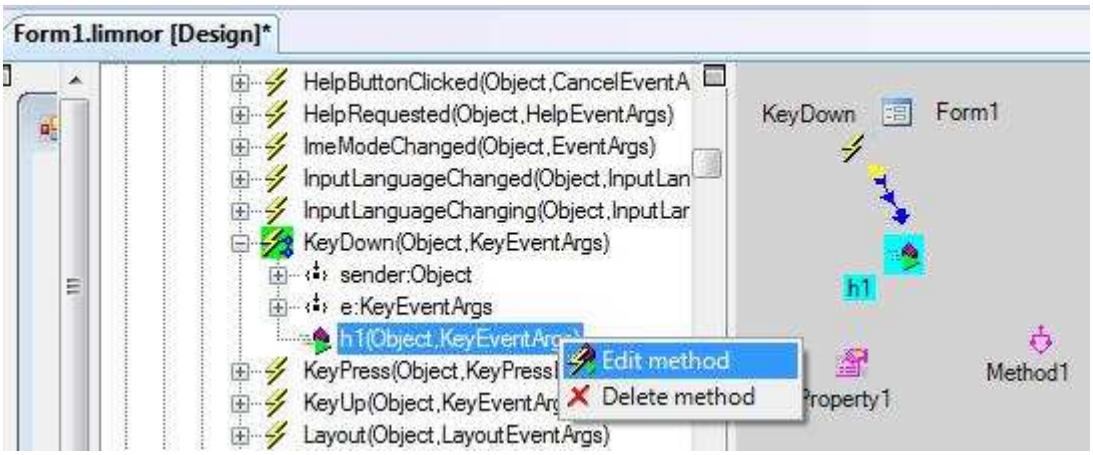
A dialogue box appears for selecting actions. Select “New event handler for …”:

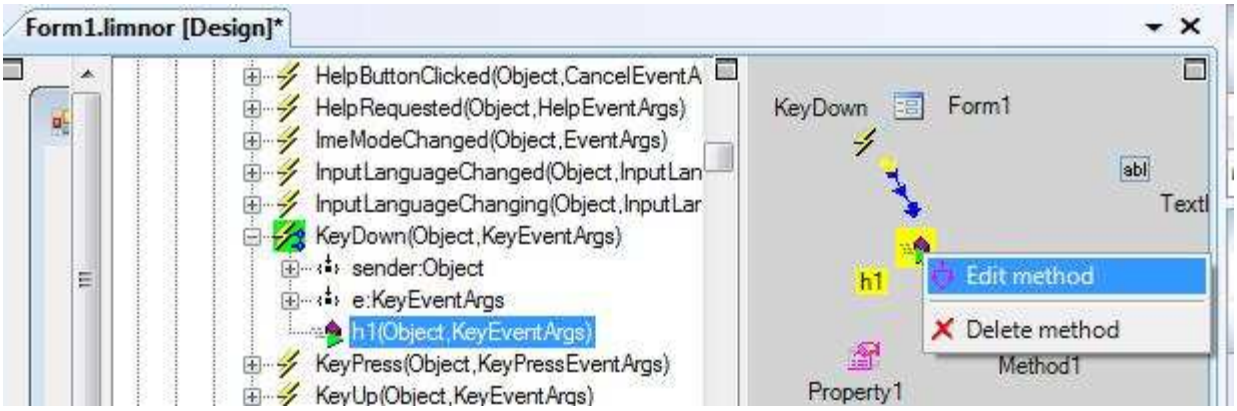


The Method Editor appears. Note that the event parameters are the parameters for the event handler method:



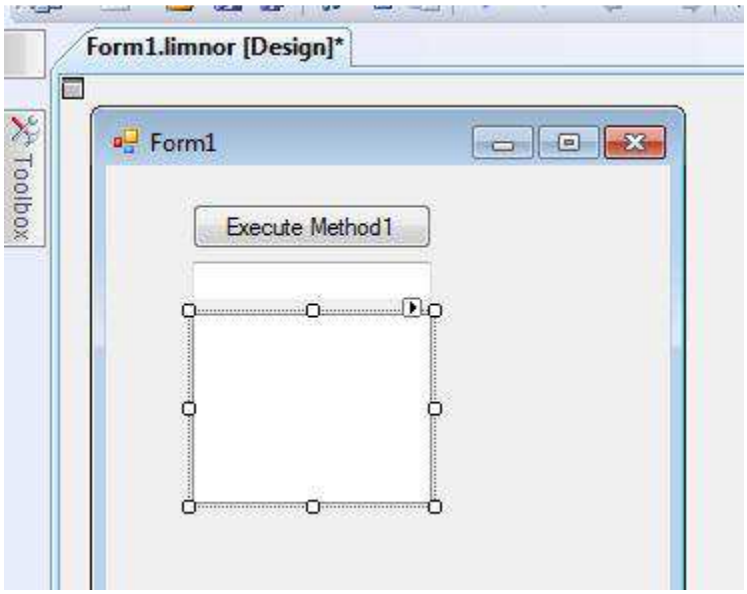
To edit an existing event handler method, right-click it and choose “Edit method”:





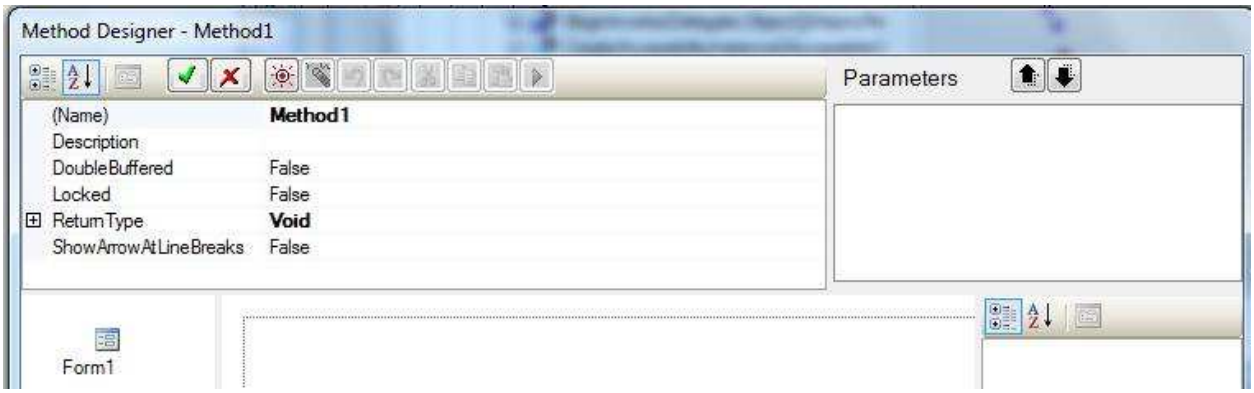
2 Method Properties

Before we going to method editing, let's add a button, a text box and a list box to the form for the sample method editing to use.



2.1 Basic Properties

Click in the action pane of the Method Editor, the properties of the method are displayed on top:

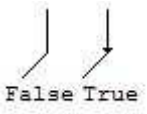


Name – method name for a normal method. This property is not available for special methods.

Description – description for the method. It will be included as comment in C# source code after compiling.

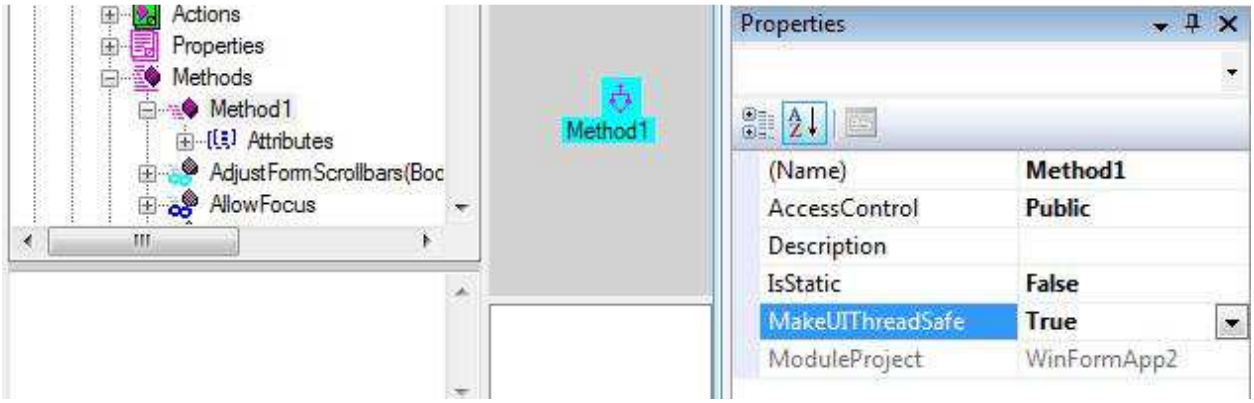
ReturnType – it is the data type for the return value of this method. "Void" means no return value. This property is not available for special methods. If ReturnType is not "Void" then the method can be used in an expression.

ShowArrawAtLineBreaks – actions within the method can be linked by segmented lines. At the joints of line segments, if this property is True then arrows are displayed to show the action execution sequence:



2.2 Other Properties

Close the Method Editor; select the method. Other method properties are displayed in the PropertyGrid:



AccessControl – It can be following values:

Public: the method can be used by any objects

Protected: the method can only be used by classes derived from the current class

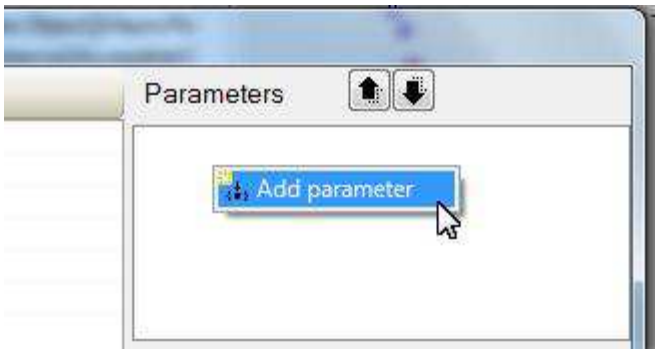
Private: the method can only be used by the current class

IsStatic – Indicates whether this method is a static method.

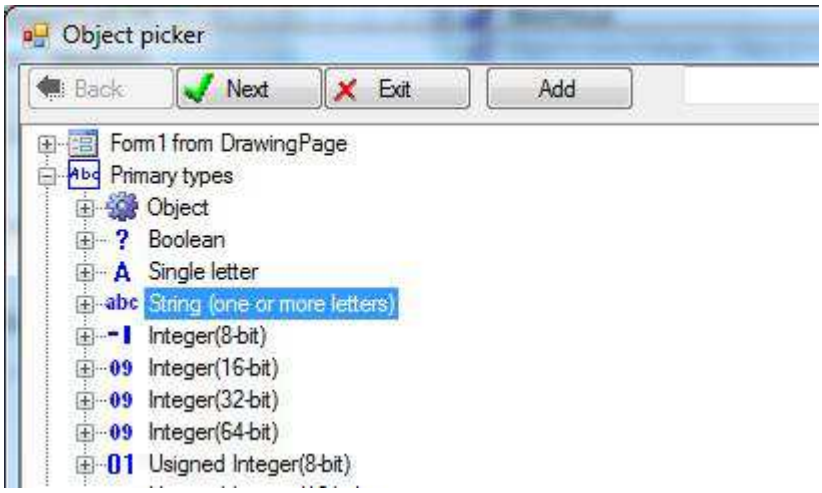
MakeUIThreadSafe – This property gives a hint to the compiler. If actions within this method modifies UI elements and this method is anticipated to be used in multi-threaded parallel execution situations then this property should be set to True. If this property is True then the compiler will try to analysis the UI-modifying actions and put those actions into the UI thread. UI-modifying actions must be executed within UI-thread. The program will crash if this rule is not followed.

3 Method Parameters

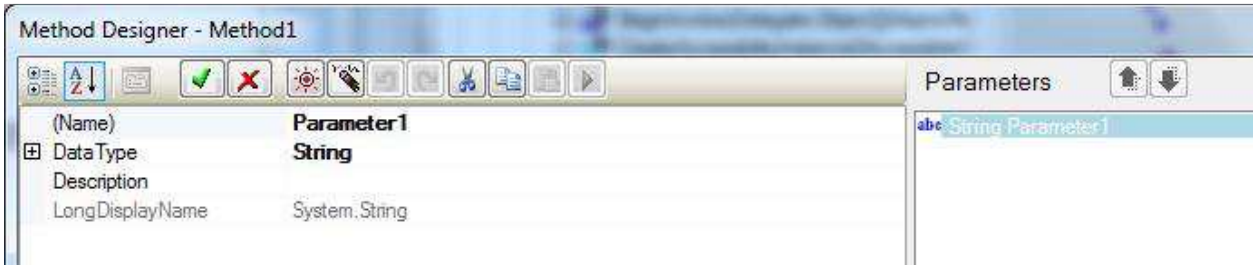
To add a parameter to the method, right-click the parameter list, choose “Add”



Select a data type for the new parameter:



Select the new parameter to see/modify its properties:

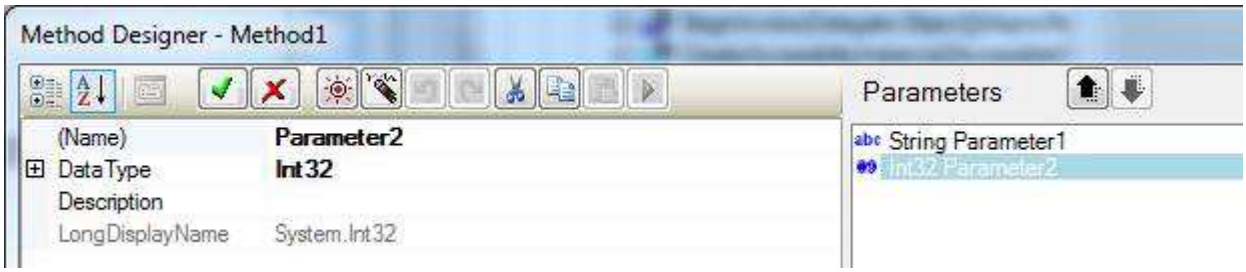


Name – the parameter name

DataType – the data type of the parameter

Description – description of the parameter.

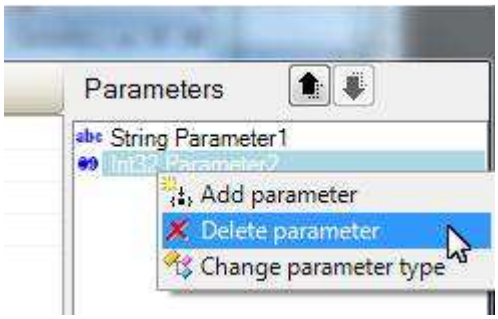
More parameters can be added if needed:



Click  to move the selected parameter up in the list.

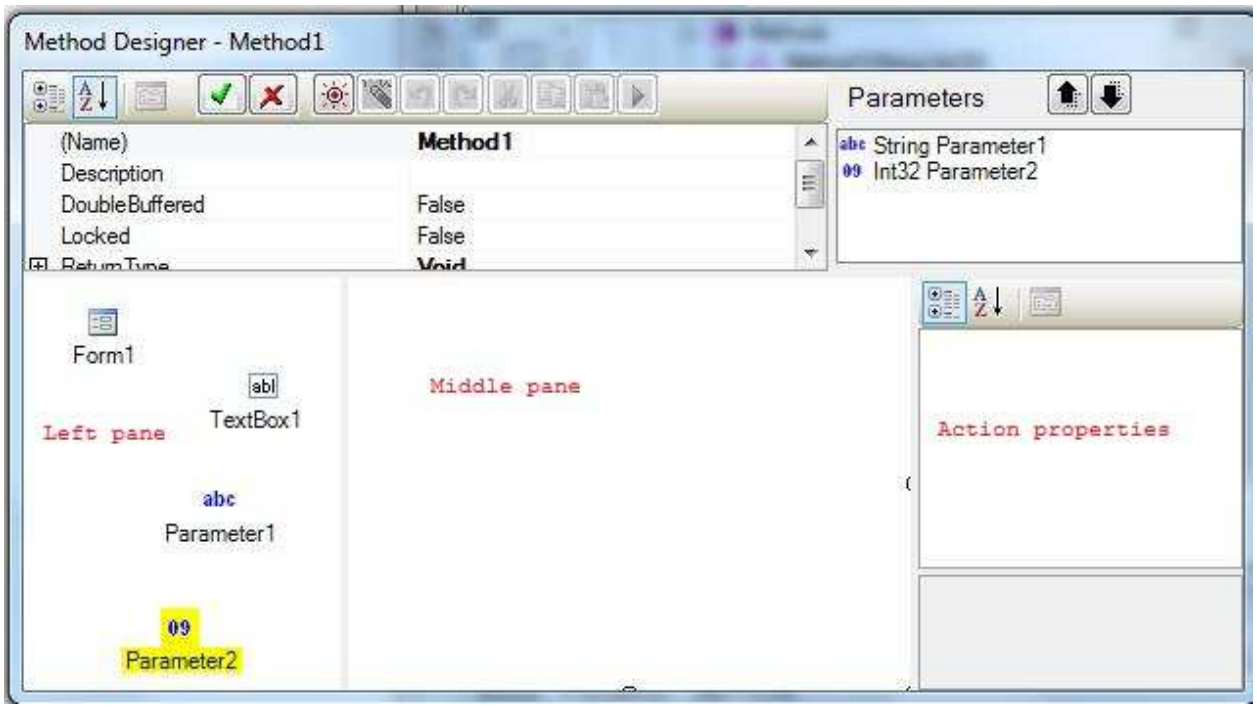
Click  to move the selected parameter down in the list.

To remove a parameter, select it and right-click it. Choose “Delete parameter”:

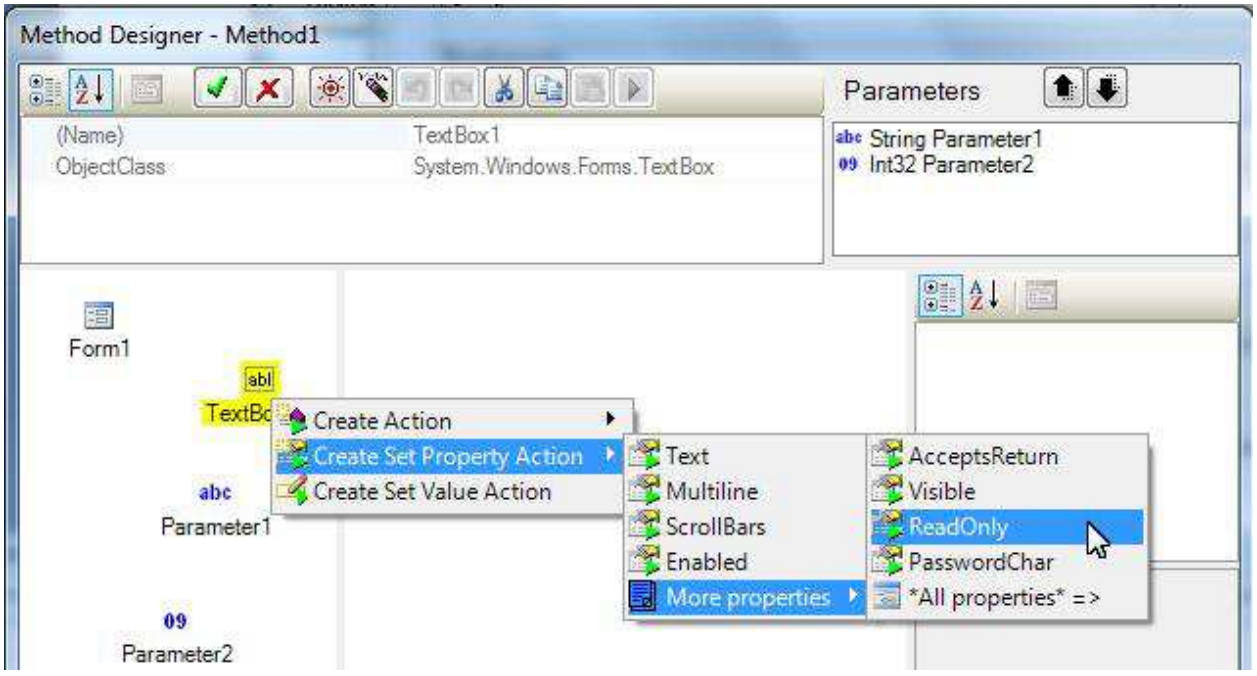


4 Object Icons

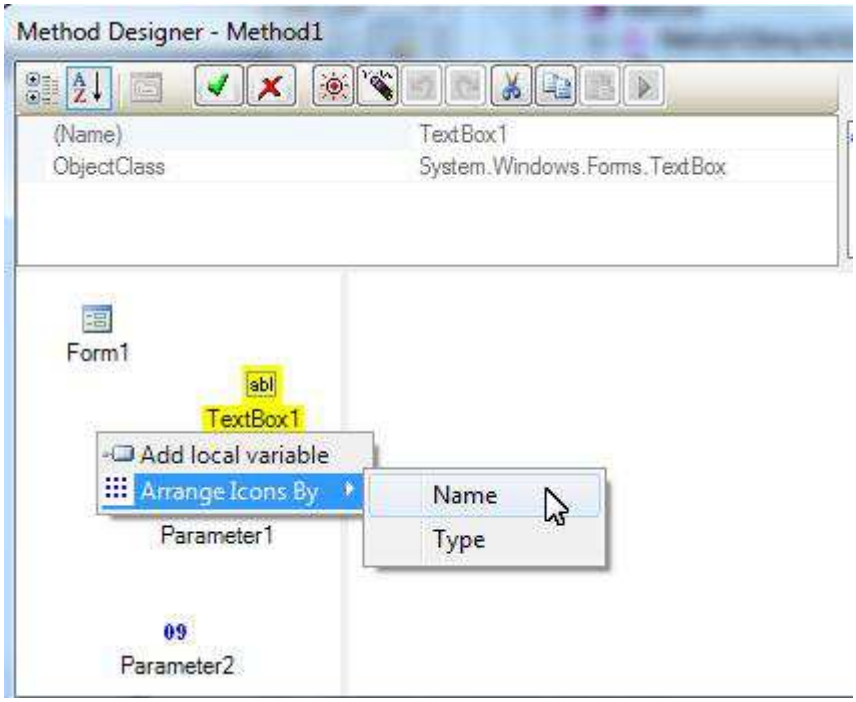
Each component and method parameter is represented by an icon in the object pane of the method editor:



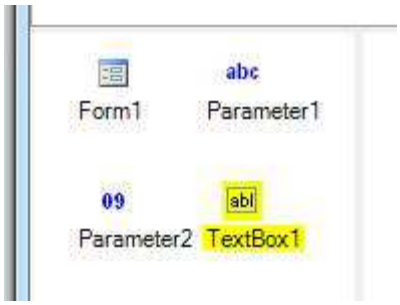
Actions can be created for each component by right-clicking it:



Icons can be re-arranged by Name or by Type, by right-clicking the Object pane:



For example, re-arranged by component names:

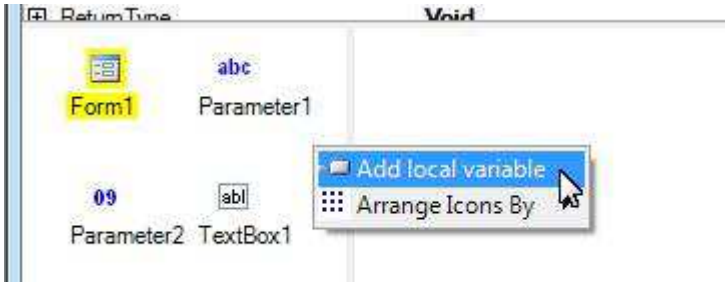


5 Variables

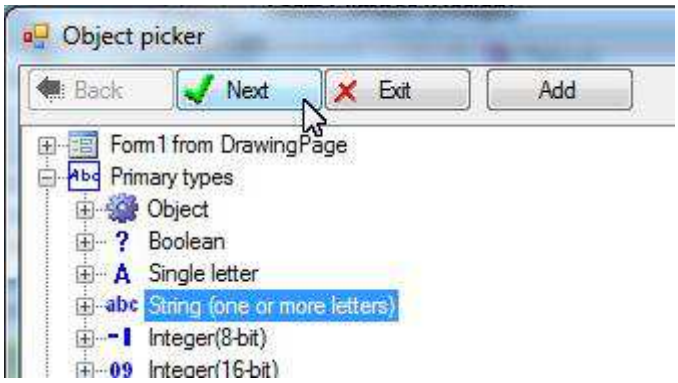
Variables can be used in a method to help passing data between actions. Variables act as temporary storage places to hold data generated from some actions and accessed by other actions. Unlike object properties, variables are gone once the method completes its execution.

5.1 Create variable explicitly

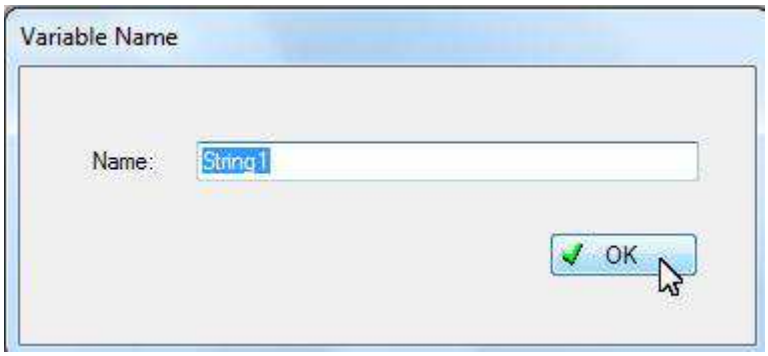
To create a new variable, right-click the object pane or action pane, choose “Add local variable”:



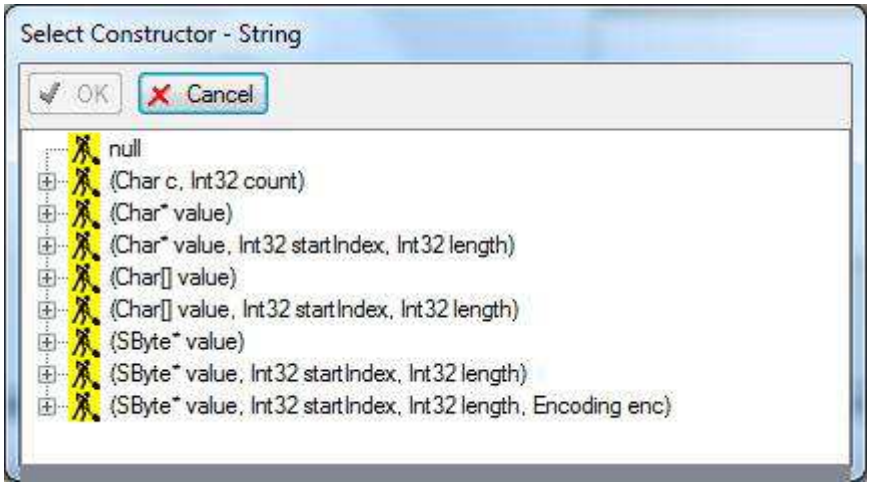
Select data type for the variable:



Give a variable name:



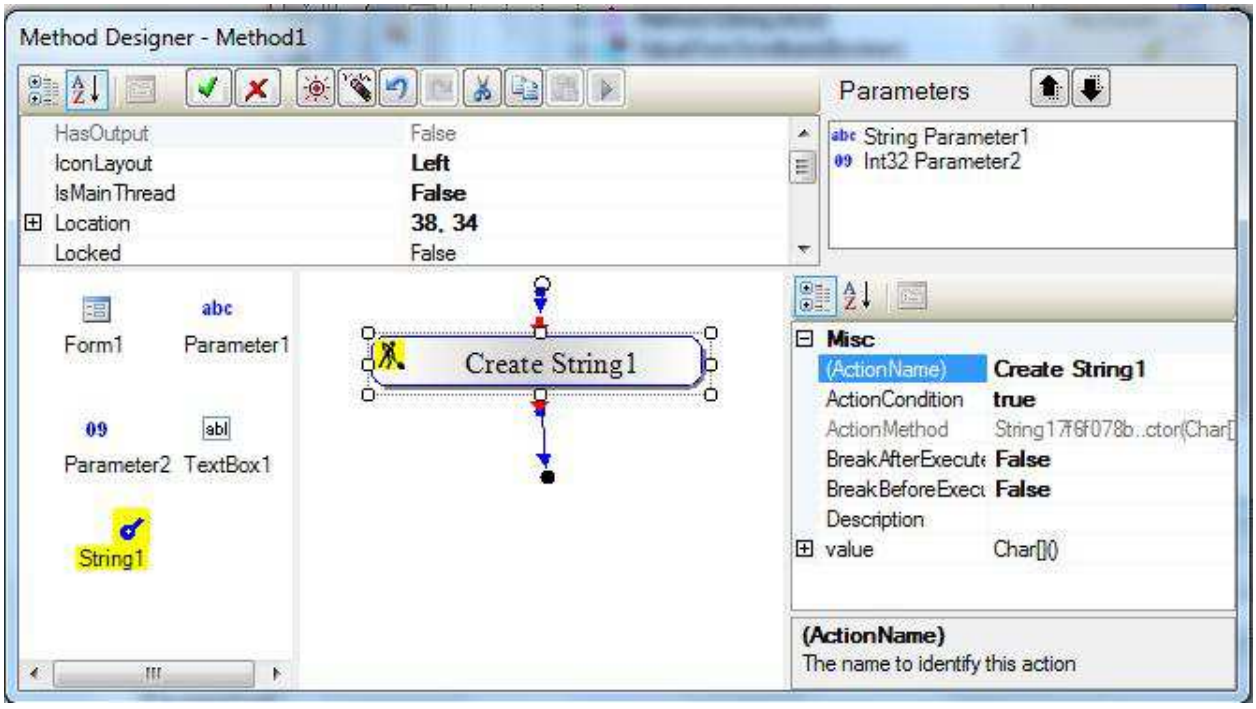
If a class is selected as the data type, for example, a String, then the constructors of the class are listed for selection:



Select null if we do not want to instantiate the variable (we want to use an action to instantiate it later).

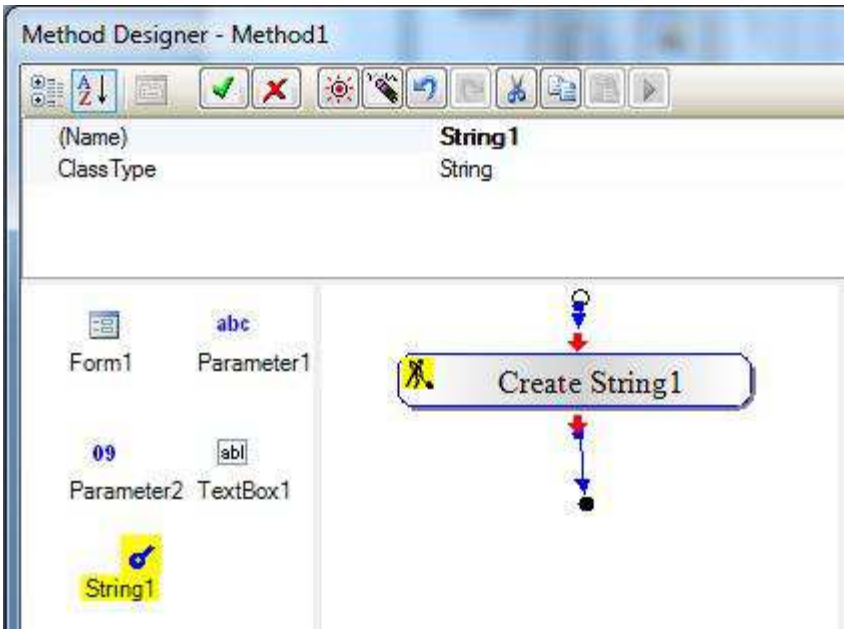
If a constructor is selected then an action using the constructor will be added to the method. We may set the constructor parameter values for the action.

Suppose we choose (Char[] value). This constructor takes an array of characters as the parameters. An action using the constructor is added to the method:



Note that the new variable appears in the object pane represented by an icon.

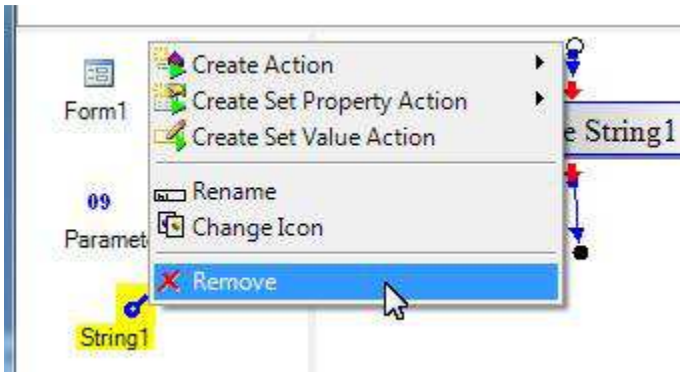
Select the icon for the variable, and the properties for the variable are displayed on top.



Name – the name of the variable.

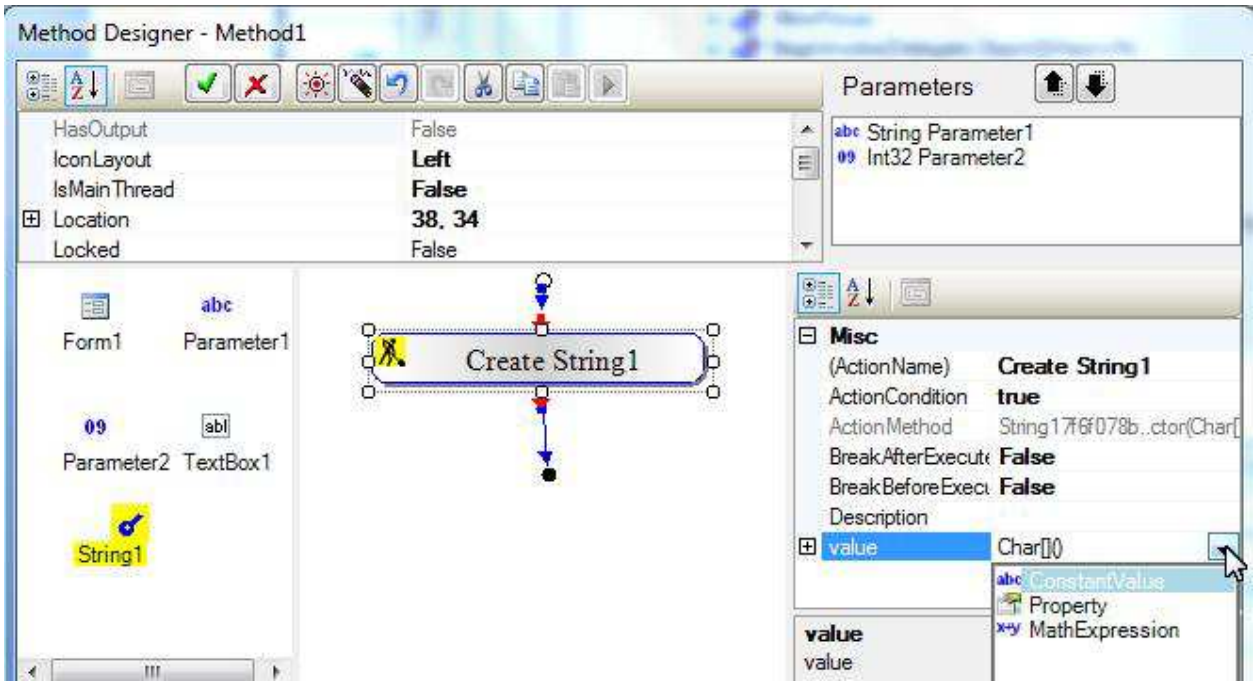
ClassType – the data type of the variable. Be very careful about changing the data type. For example, changing String1 to be an integer will make the action "Create String1" to fail because an integer does not have a constructor which takes an array of characters as parameters.

To remove a variable, right-click the icon of it, choose "Remove":

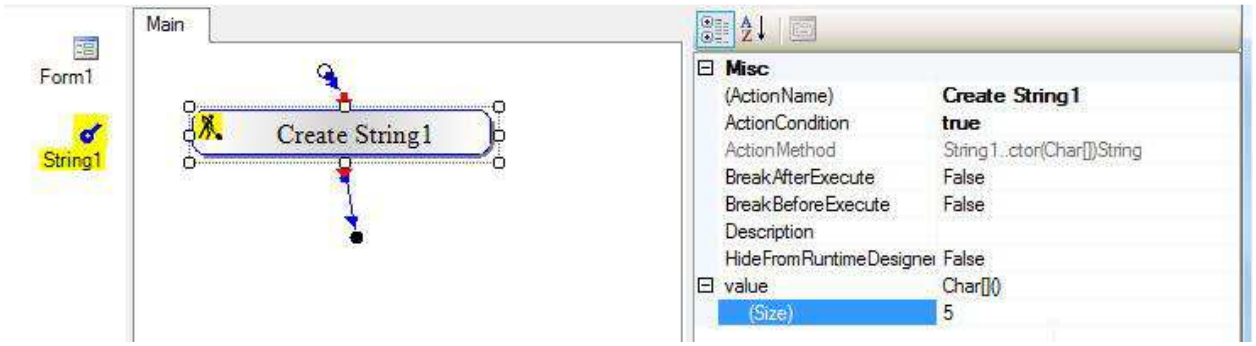


But all actions using the variable must be removed first before the variable can be removed.

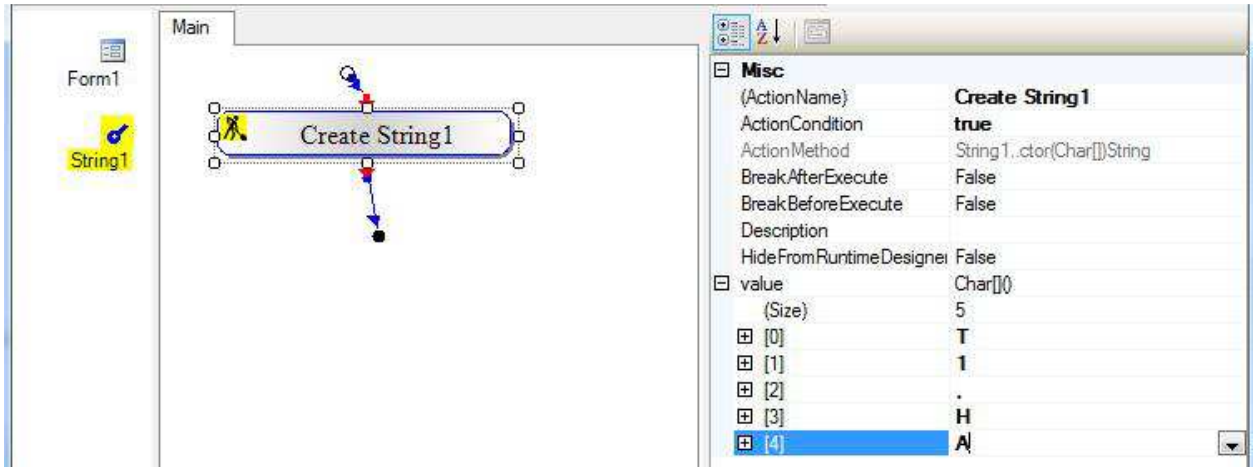
Editing an action of constructor is the same as editing all other actions. For each of its parameters we may provide its value with a constant, a property or an expression:



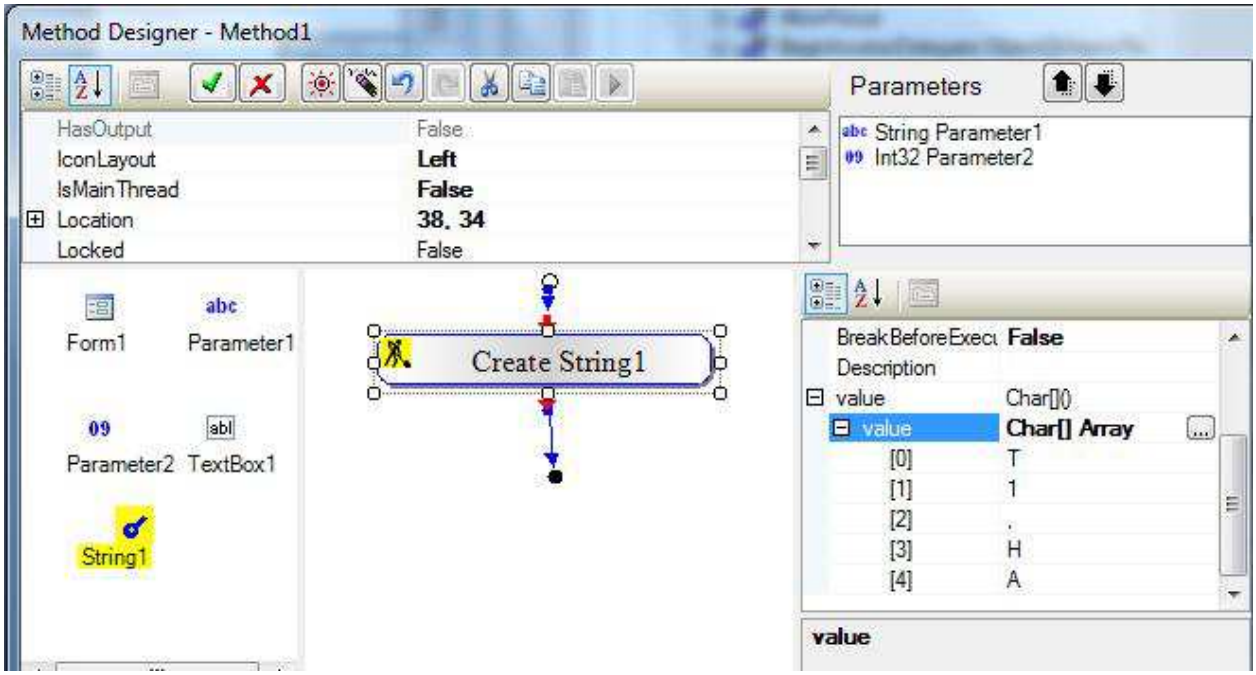
Suppose we want to use a constant. Expand value property; set the (Size) of the character array:



Set a character to each array item:



After executing the action, String1 becomes “T1,HA”

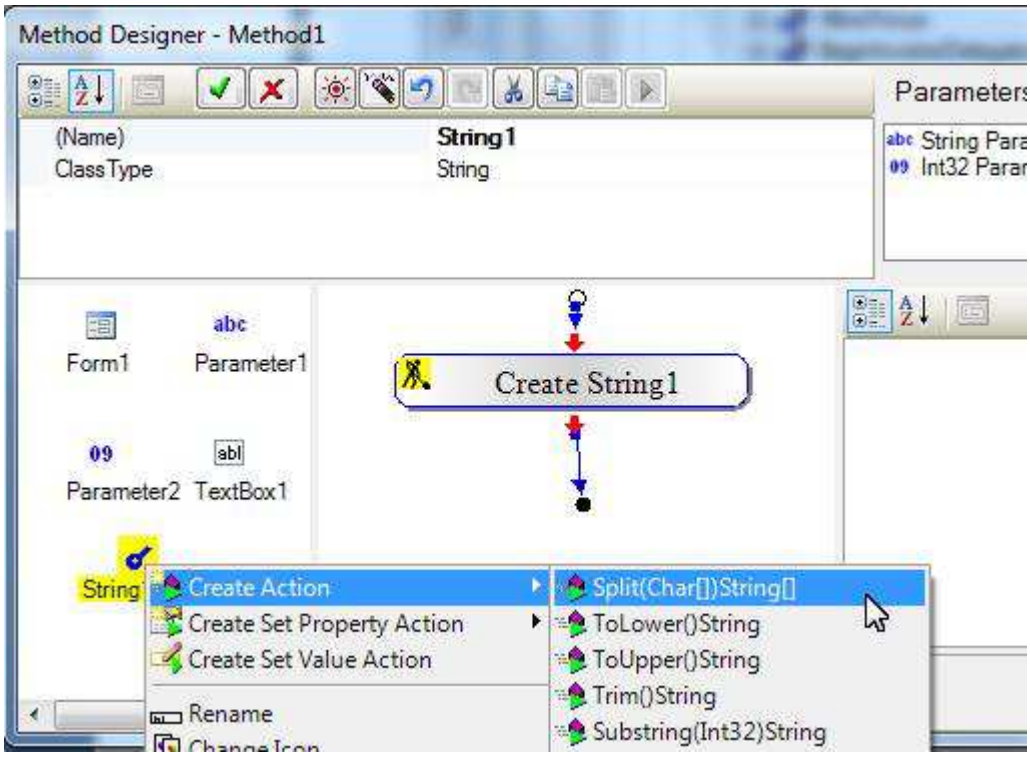


5.2 Create variable as action output

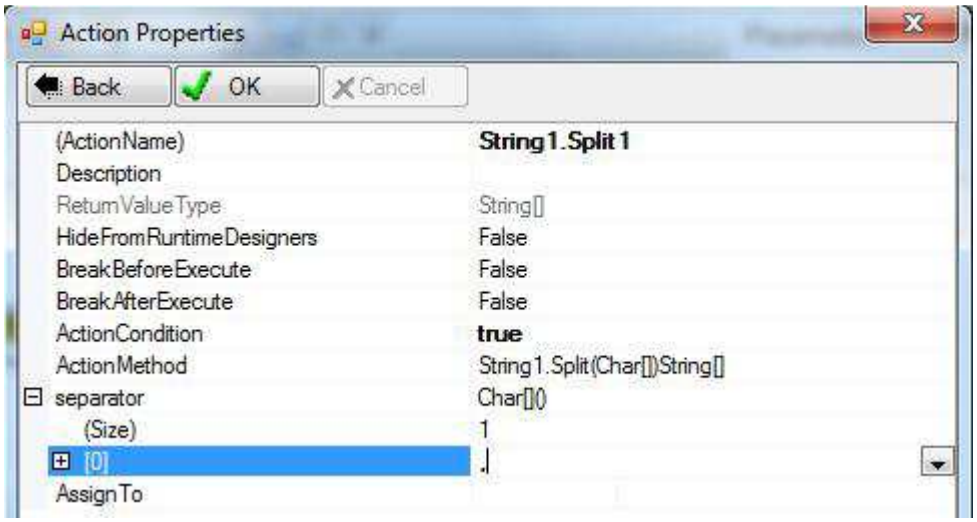
If an action returns a value then the action will have an “AssignTo” property. Setting of this property allows creating a new variable to store the action return value.

For example, a String class has a Split method which splits the string into a string array. The Split method returns a string array.

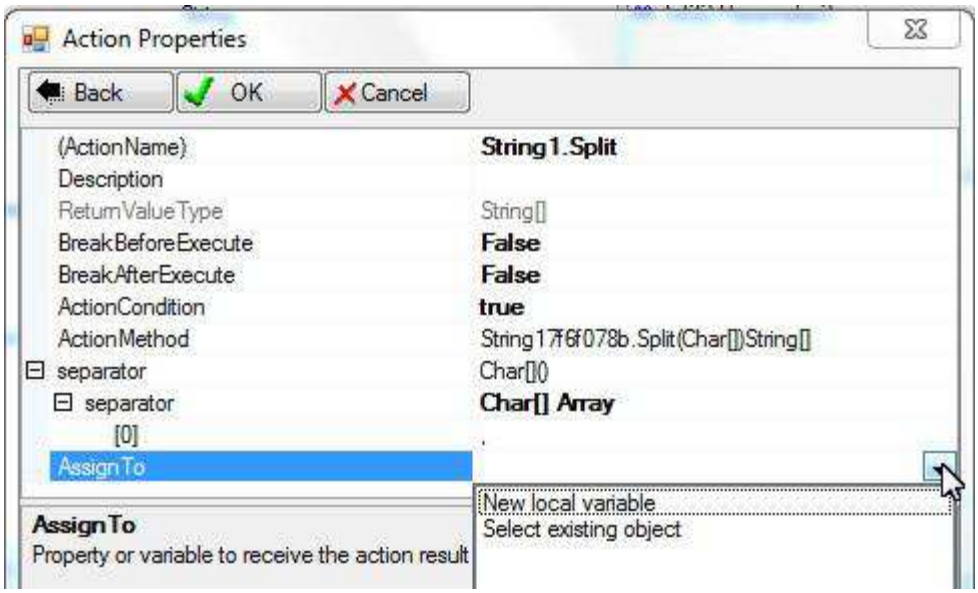
Let’s create a Split method by right-clicking the icon for String1 and choose “Create Action”. Choose “Split” method:



The "separator" parameter is a character array listing the string separator characters. We use a constant character array for this parameter. The character array has one item. The item is a comma. We already showed how to enter a character array before. So, we do not go into details:

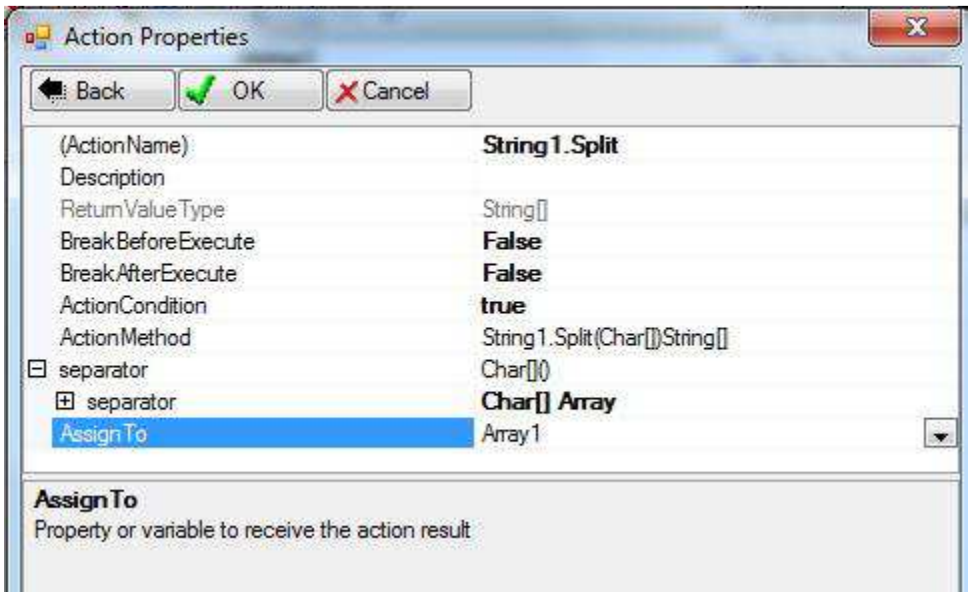


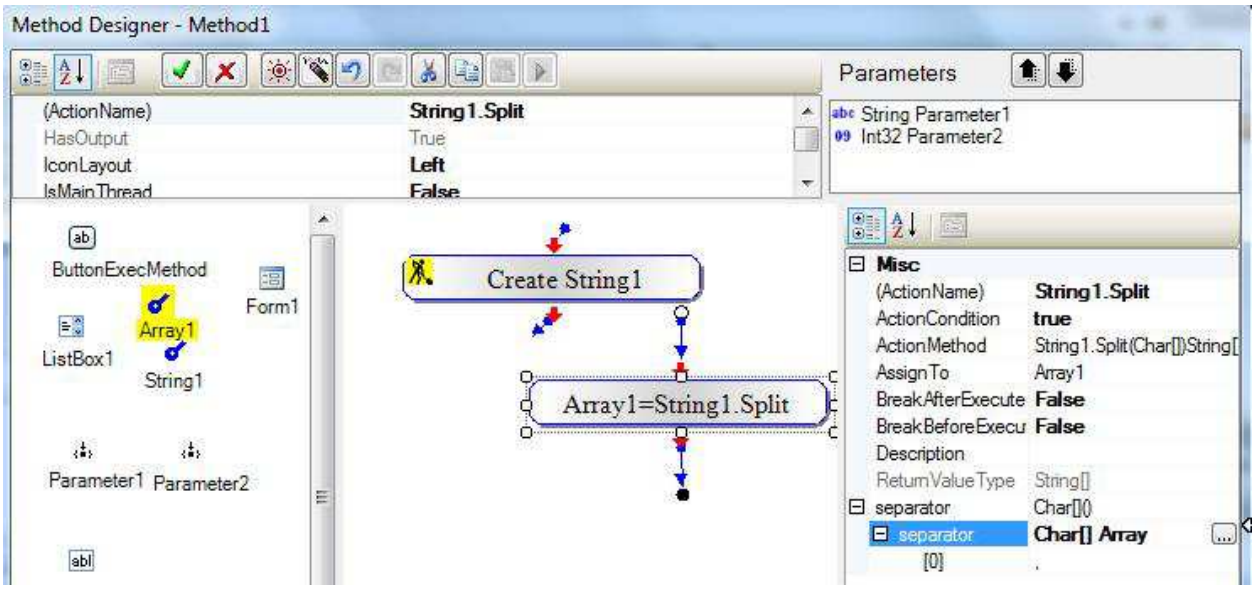
This action has an "AssignTo" property. It allows us to assign the action result, a string array, to a new local variable or to an existing object:



Choose “New local variable”. It asks for a variable name:

A new local variable is automatically created and the action result is assigned to it:



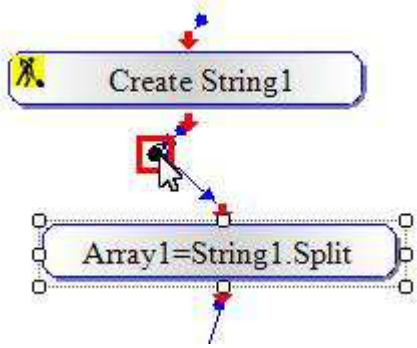


Variable Array1 is a variable generated by the Method Editor. It is a string array. The action String1.Split sets the value of Array1.

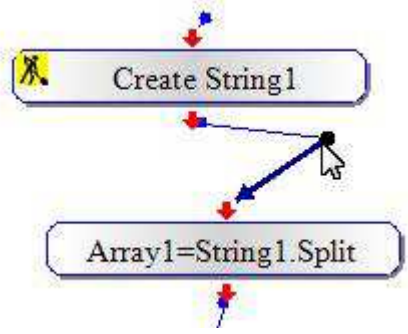
6 Link Actions

6.1 Link actions

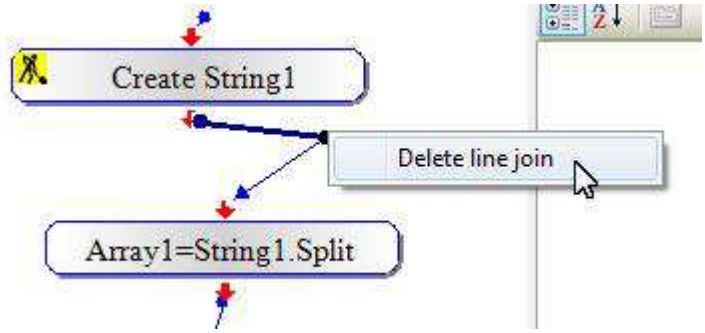
Link the two actions so that correct execution sequence is formed:



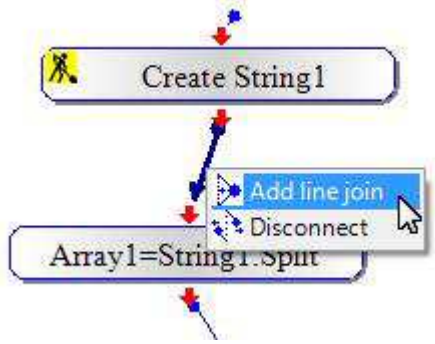
The line joint can be dragged and moved:



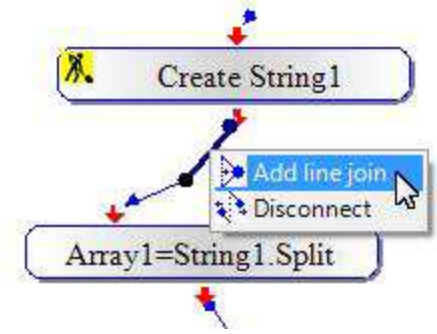
A line joint can be removed by right-clicking it and choose “Delete line join”

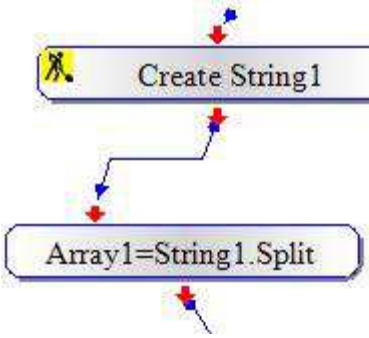


Line joints can be added to lines by moving the mouse pointer over a line until the line is highlighted. Right-click the line; choose “Add line join”:

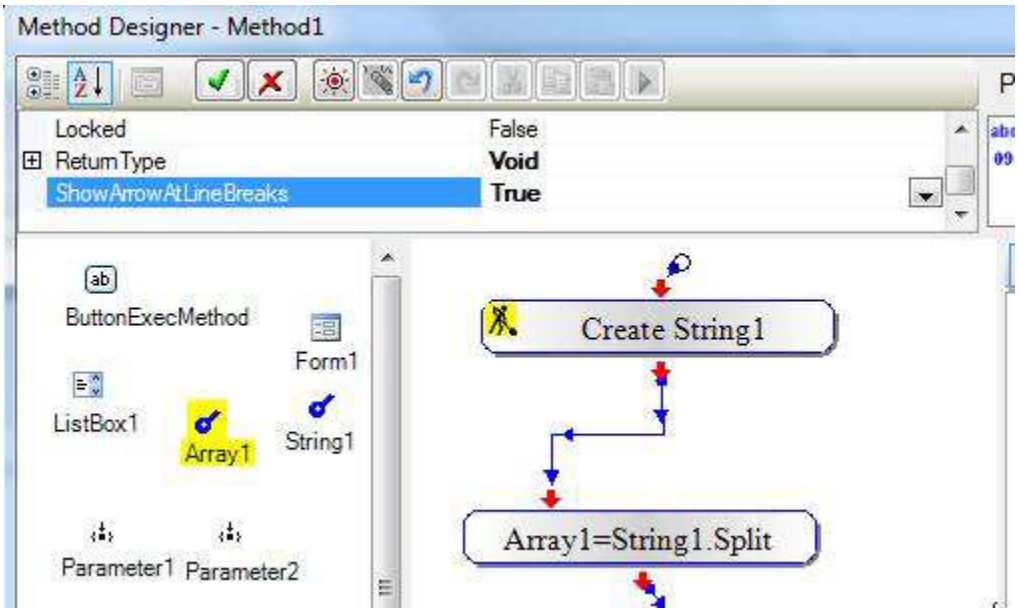


More joints can be added:





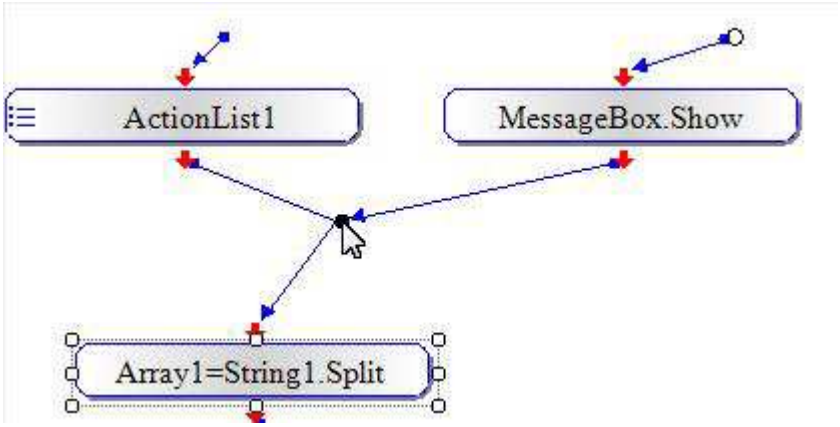
When two actions are linked, one action executes first and the other action execute next. Set the method property ShowArrowAtLineBreaks to True so that arrows are displayed at line joints indicating which action executes first.



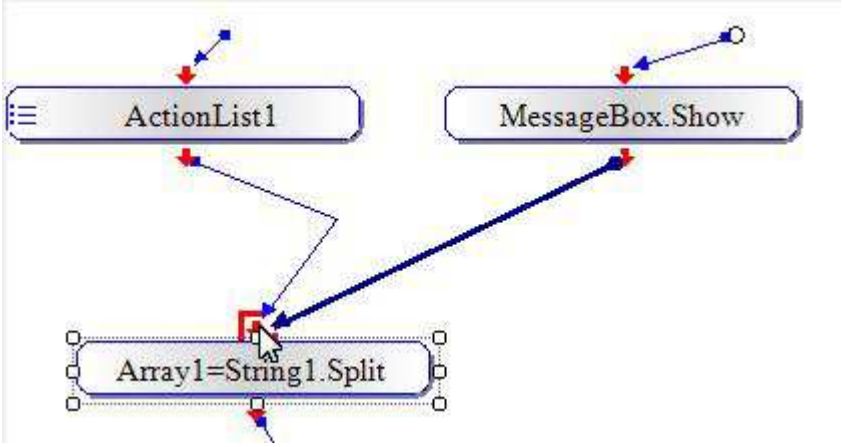
6.2 Link more than one output to one input

More than one output can be linked to the same input. But the outputs have to be dragged to the root of the input, not in the middle.

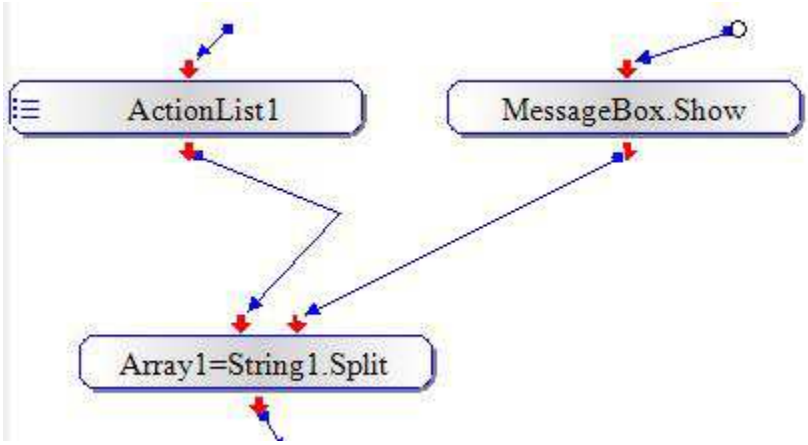
The output cannot be dropped to a middle point in the link line. As shown in the following figure, there is not a red box appearing for indicating that it can be dropped:



The output can be dropped to the root of an input, as a red box indicating that it can be dropped:

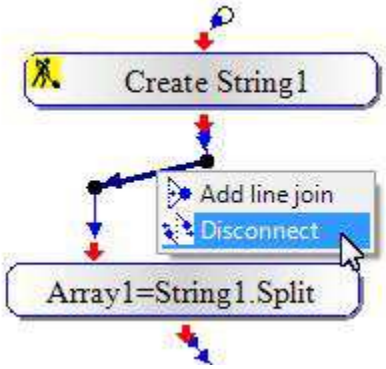


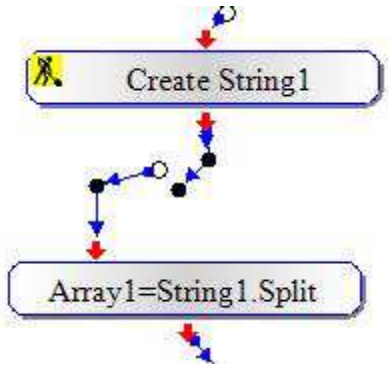
Once an output is dropped to an input, one more input is created. Inputs can be dragged and moved:



6.3 Delink Actions

Actions can be disconnected. Right-click a line; choose “Disconnect”:

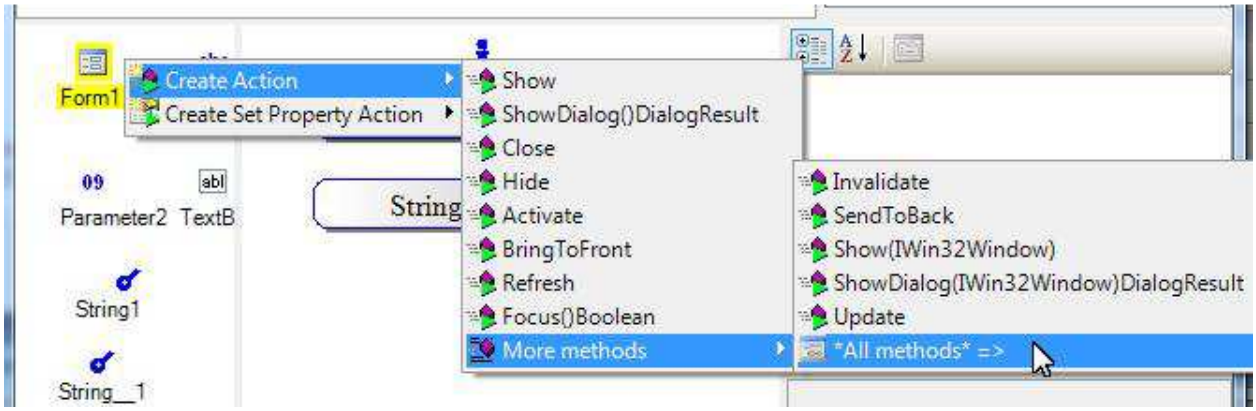




Not linked actions are executed in parallel at the same time in different threads.

7 Icons as Action Executer

Right-click an icon, the context menu allows us to create method-execute action or set-property action.

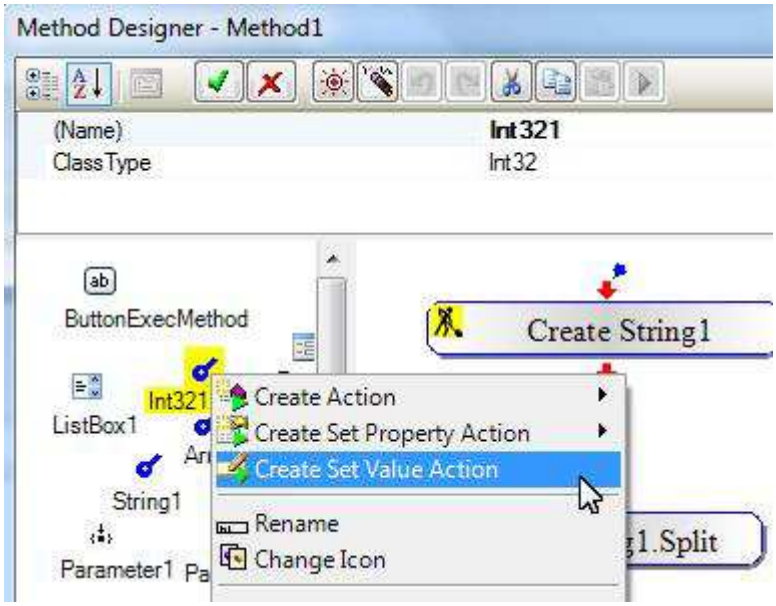


Previously we created an action String1.Split in such a way.

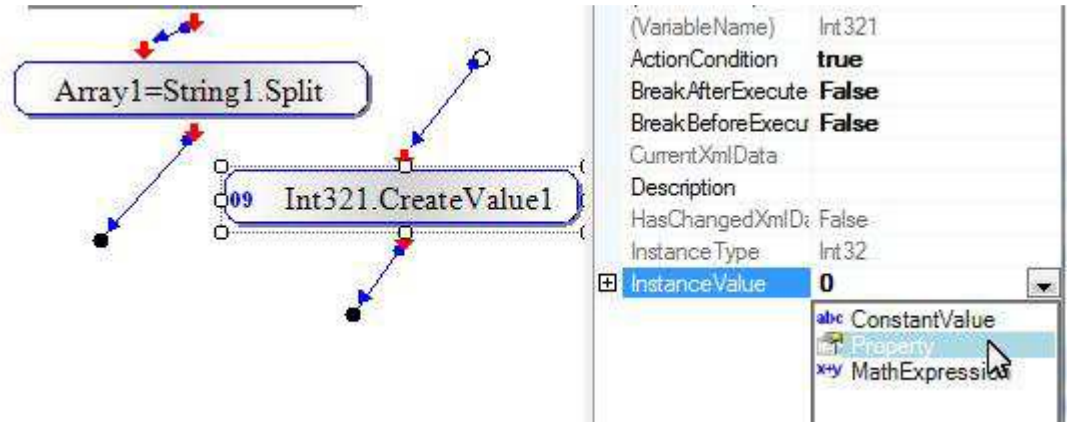
8 Assignment Action

An action can be used to set value to a variable.

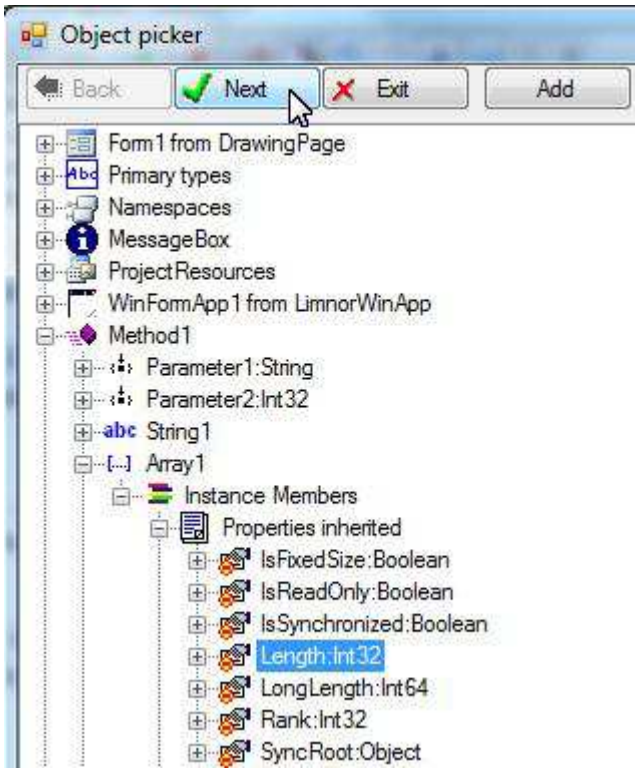
Let's add an integer variable. Right-click the variable icon, choose "Create Set Value Action":



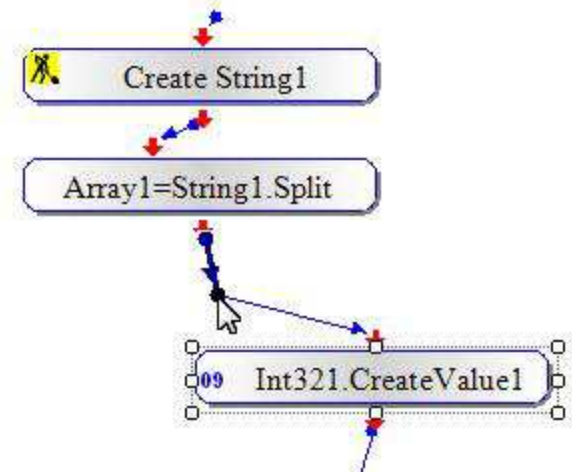
A new action is added to the method. The "InstanceValue" property of the action represents the value to be assigned to the variable. In this example we assign the Length property of the array Array1 to this variable. Select "Property" for it:



Select the Length property of the array variable Array1:



The Length property is used in the action. Link the action:



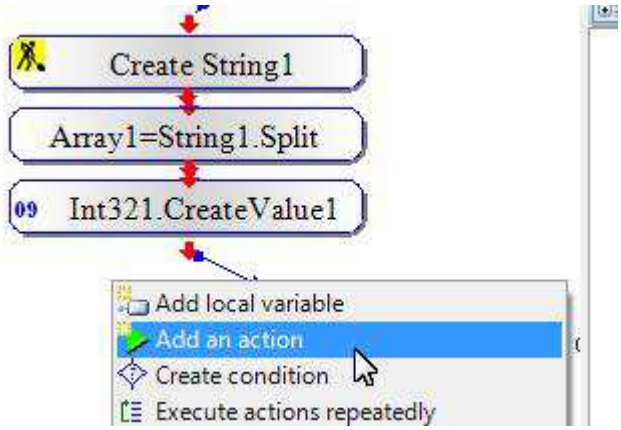
9 Action Executer in Deep Hierarchy

If action executer is represented by an icon then the action can be created by right-clicking the icon, as we have done previously.

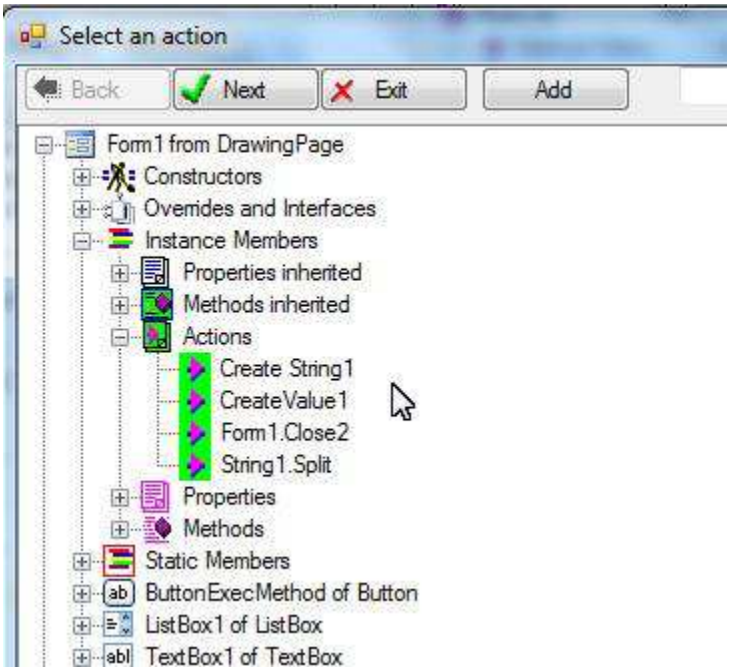
If an action executer is deep in object hierarchy then the Object Explorer can be used to locate the action executer for creating the action.

For example, a ListBox component does not have a method to add items to the list. The ListBox has an Items property representing all items in the list. The Items property has an Add method which is for adding new items to the list box. So, the Items property is the action executer.

To create such an action, right-click the action pane, choose “Add action”:

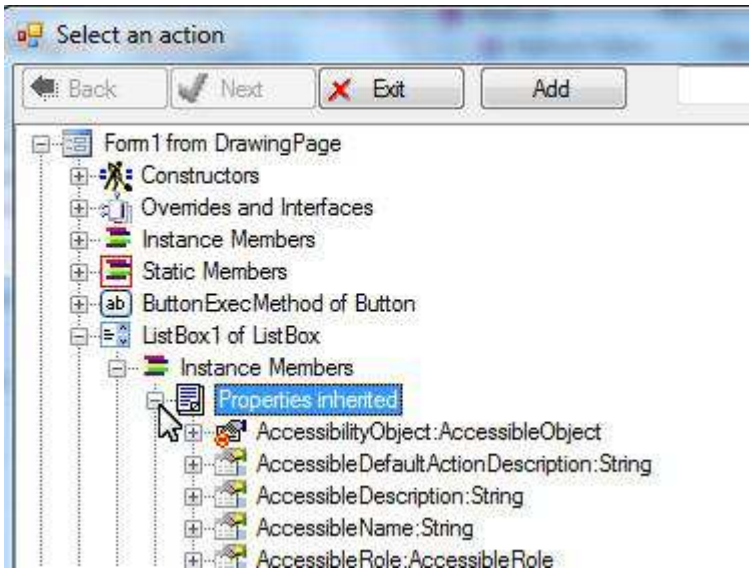


The “Select action” dialogue box appears for selecting an existing action. New actions can also be created at this time.

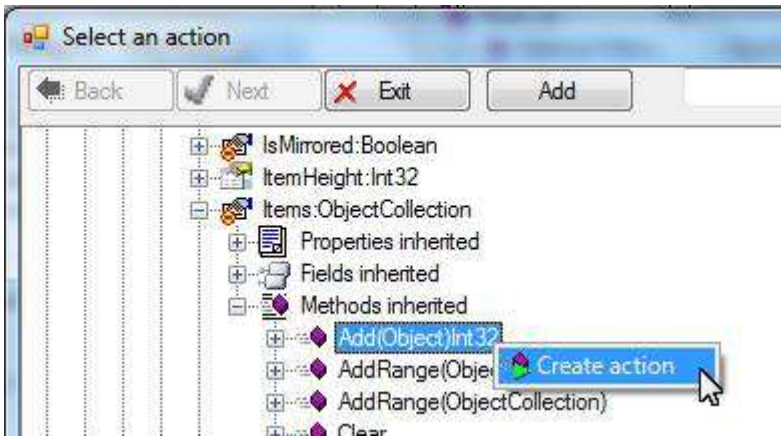


To create a new action, first locate the action executer. In this case it is the Items property of the list box.

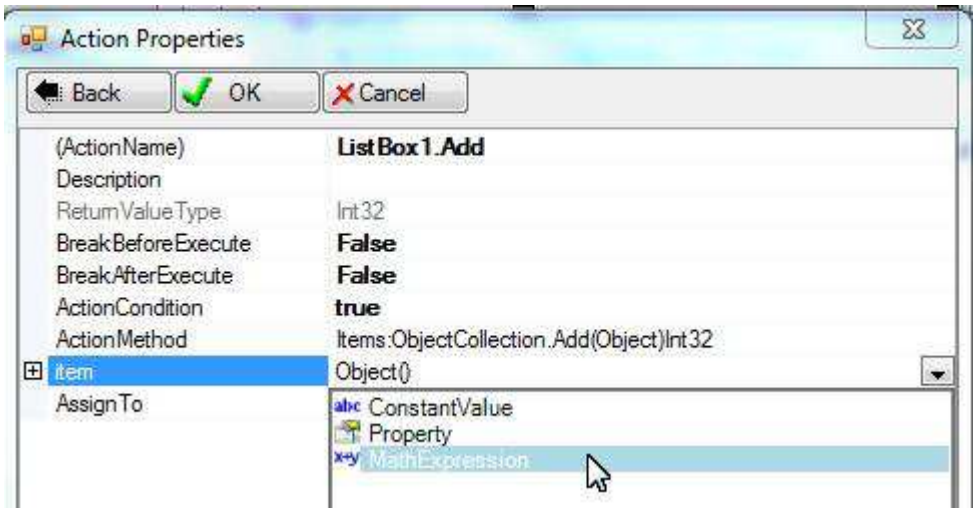
Expand “Properties inherited” node of ListBox1:



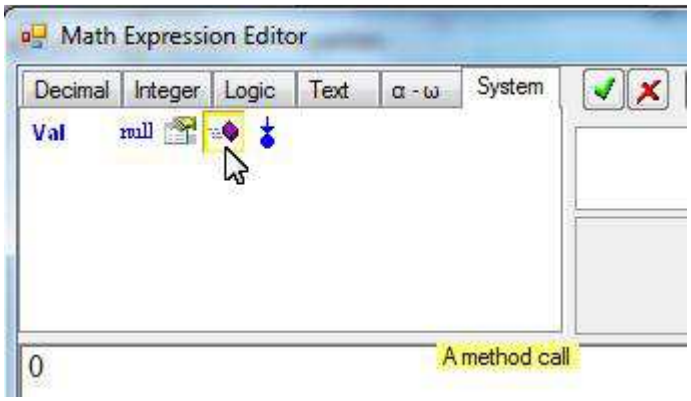
Scroll down to find Items property. Right-click its Add method. Select “Create action”.



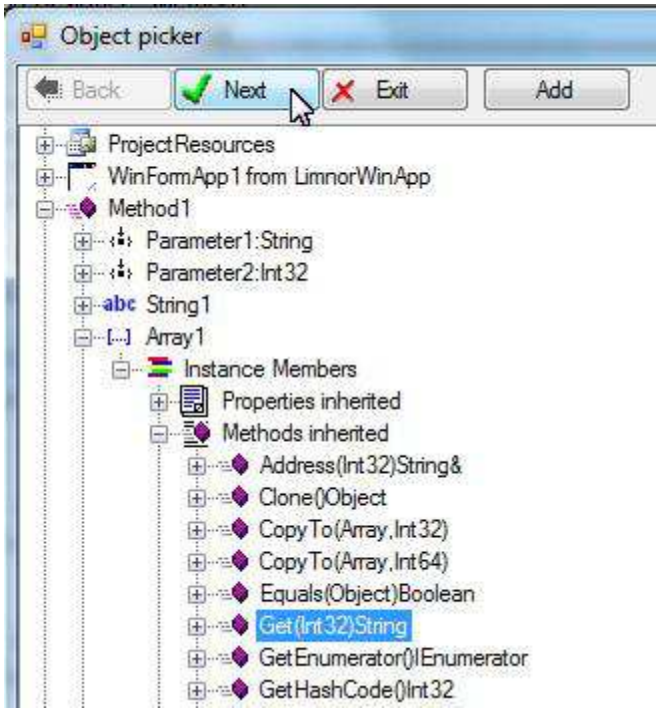
The Item property of the action represents the object to be added to the list box. We want to add the first array item of the array variable Array1 to the list box. Math Expression can be used to retrieve array items:



Click the method icon to use a method to retrieve array item:

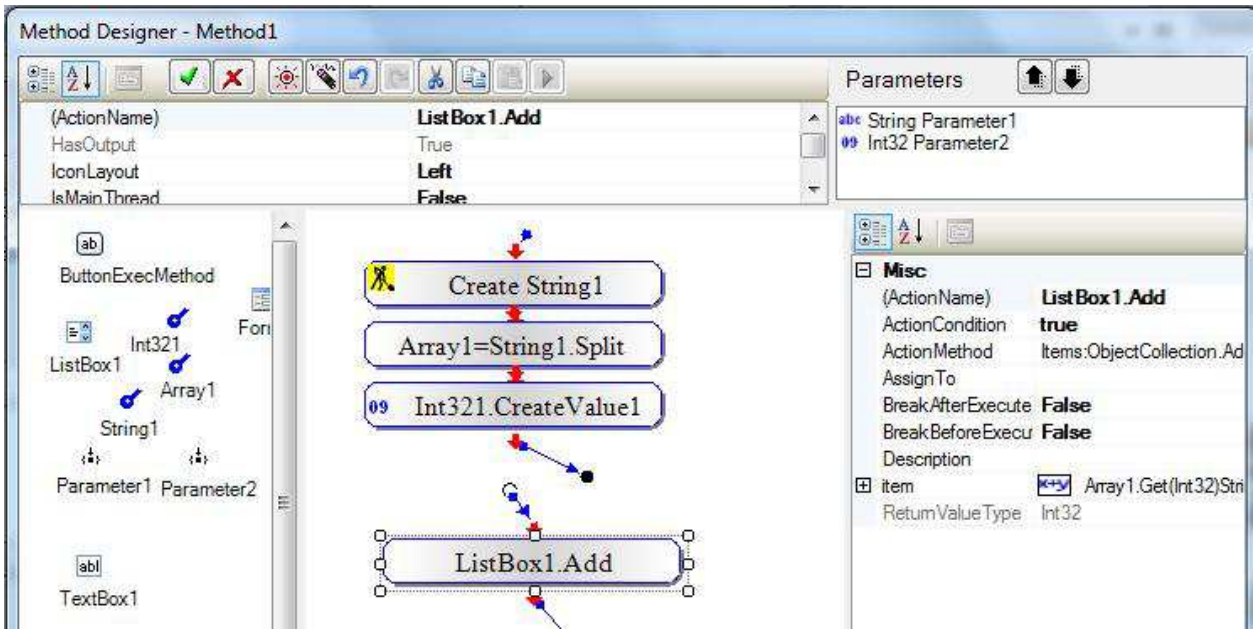


Select the method “Get(Int32) string” of the string array Array1:

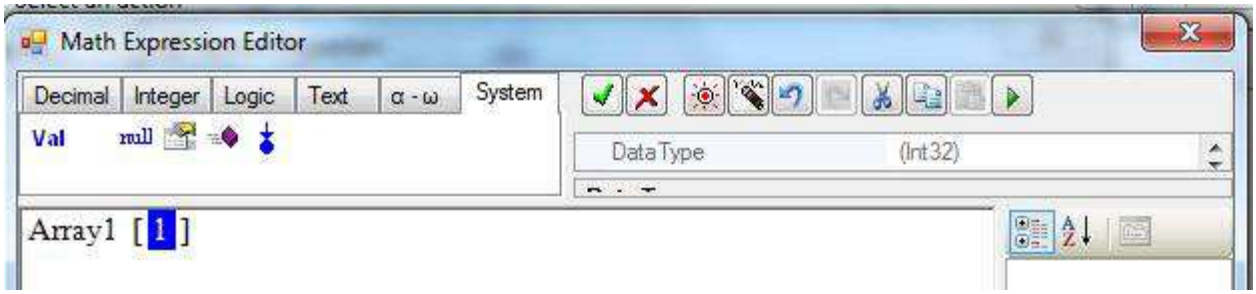


Value 0 represents the first array item:

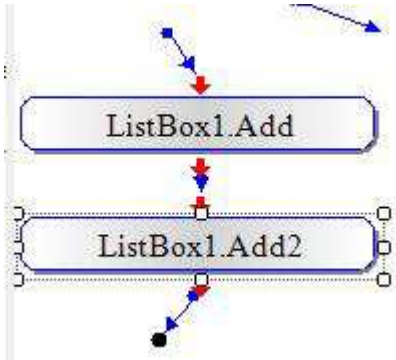
The action appears in the method:



Add another add item action. This time use 1 for the index to get array item. 1 indicating the second array item:



Link the two actions together:

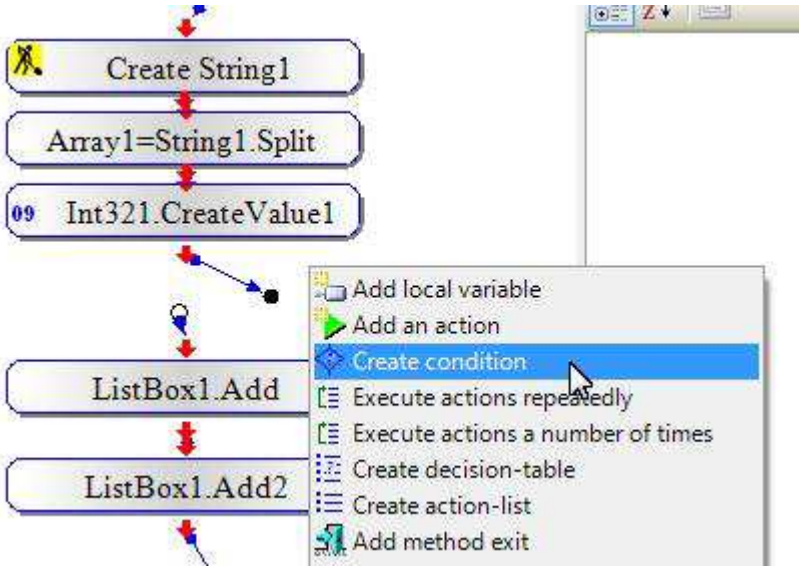


If the string array has 100 items, we do not want to create 100 actions to add each item to the list box. Later we will describe using one action to handle all items.

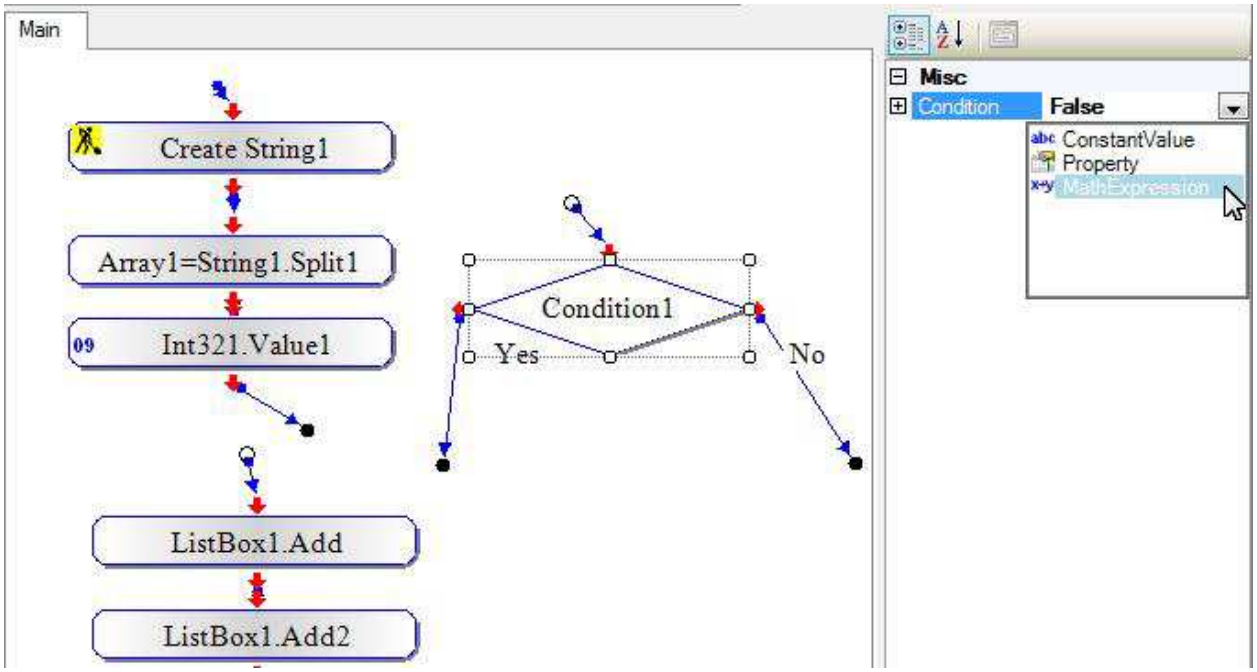
10 Execution Branching

A Condition Action may route action executions to different branches based on an expression evaluation.

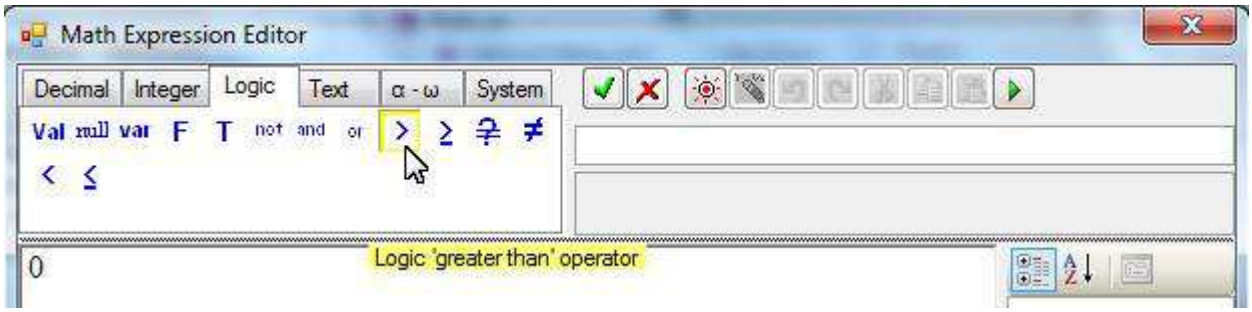
Right-click the action pane, choose “Create condition”:



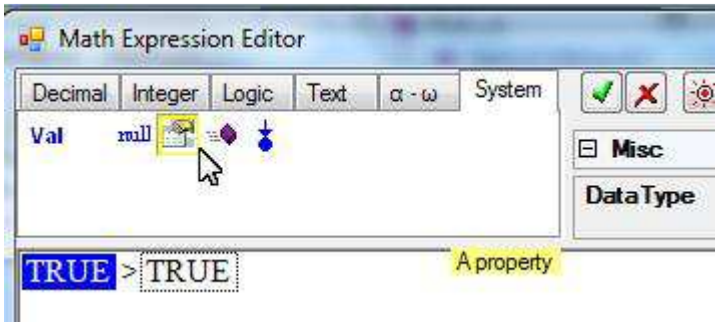
The condition we want to use is that the array has more than 1 item. Select Math Expression to build the condition:



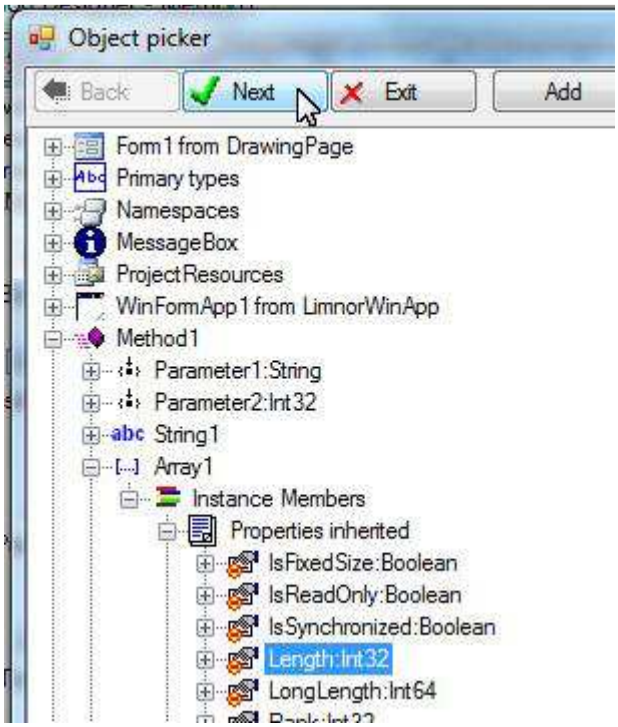
Click ›:



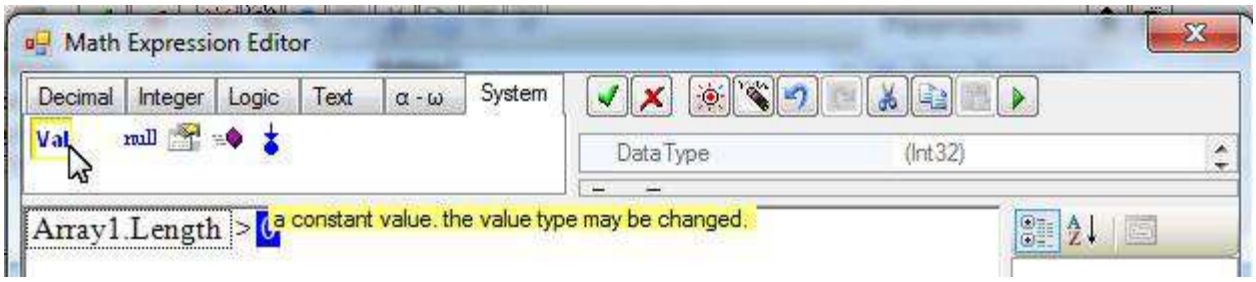
Select the first element, click the property icon:



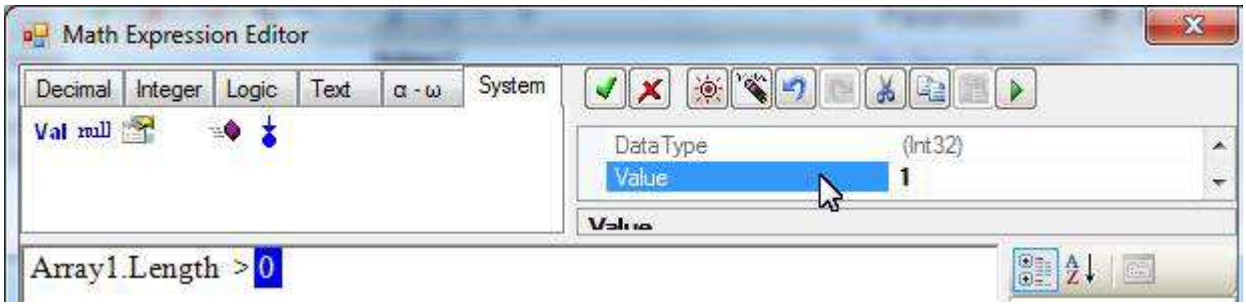
Select the Length property of the string array Array1:



Select the second element, click Val:

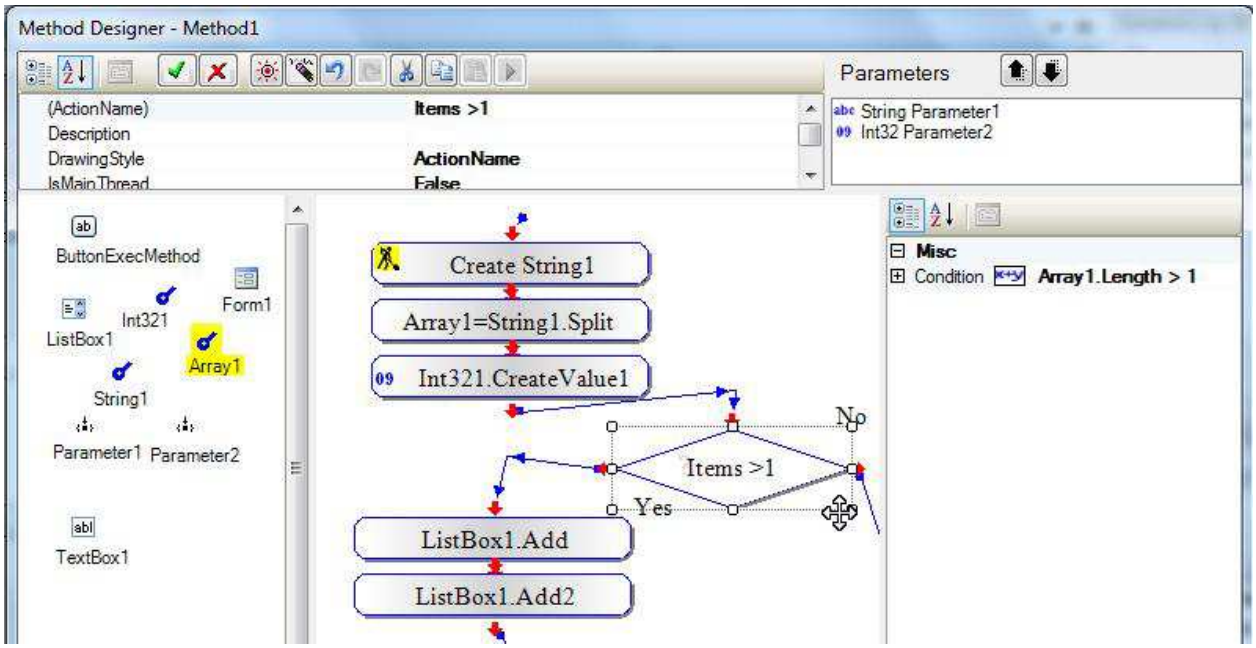


Set the value to 1:



This is the math expression for the condition:

Rename the condition action. Link it to other actions:

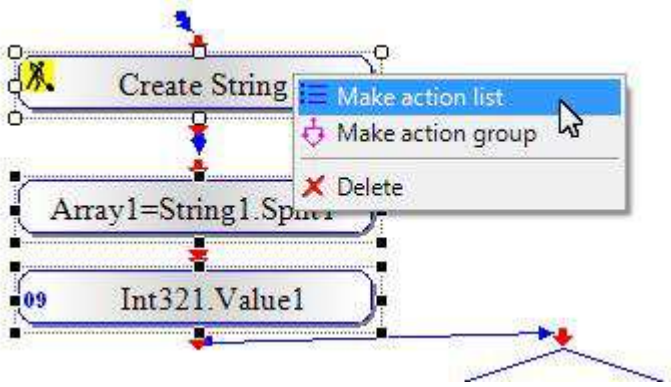


11 Action List

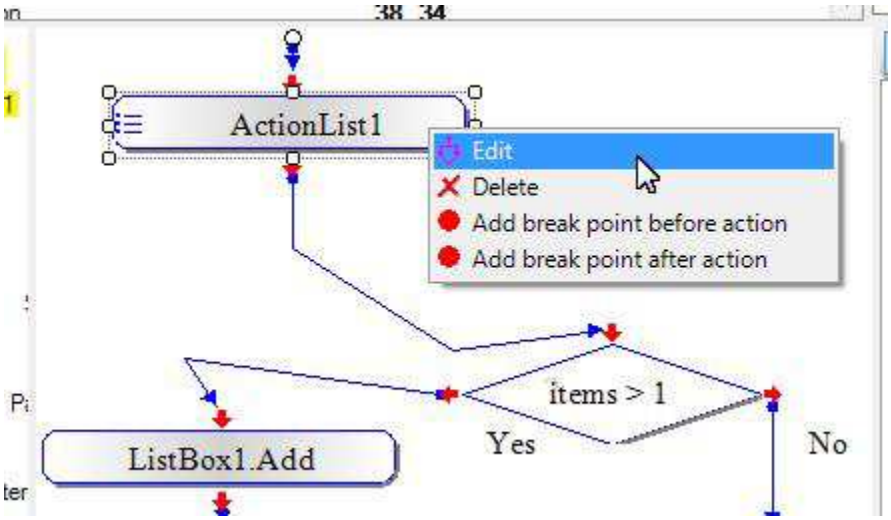
11.1 Group existing actions into one single action

The Method Editor is getting clouded with more and more actions added. We may put a single branch of actions into one single action list to save the screen space.

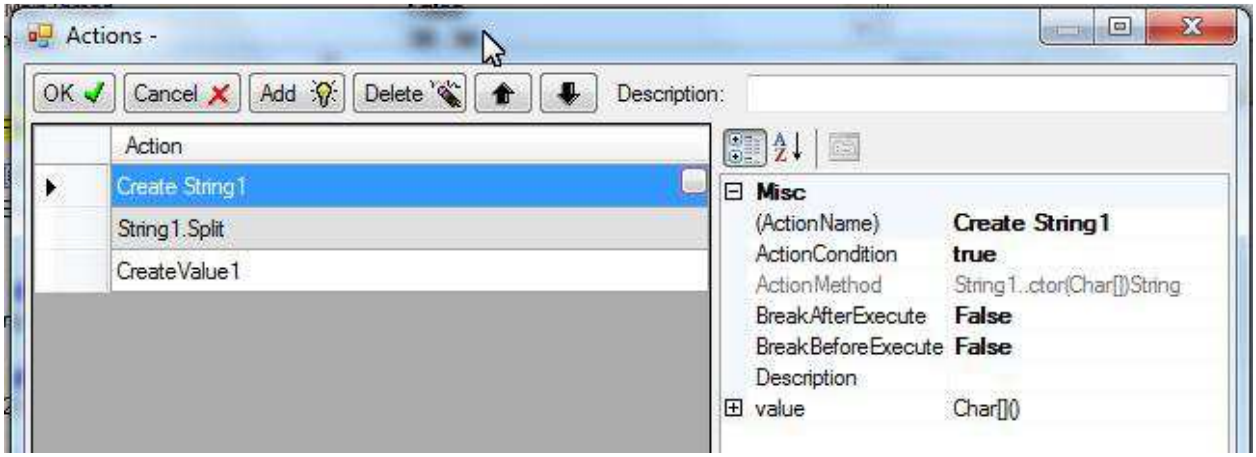
Select the actions we want to put into a list, right-click the selection, choose “Make action list”:



A new action list is created in place of the selected actions. It can be edited by right-clicked and choose “Edit”:



Each action can be edited by selecting it and modify its properties:



Add -- add an action to the list.

Delete -- delete selected action

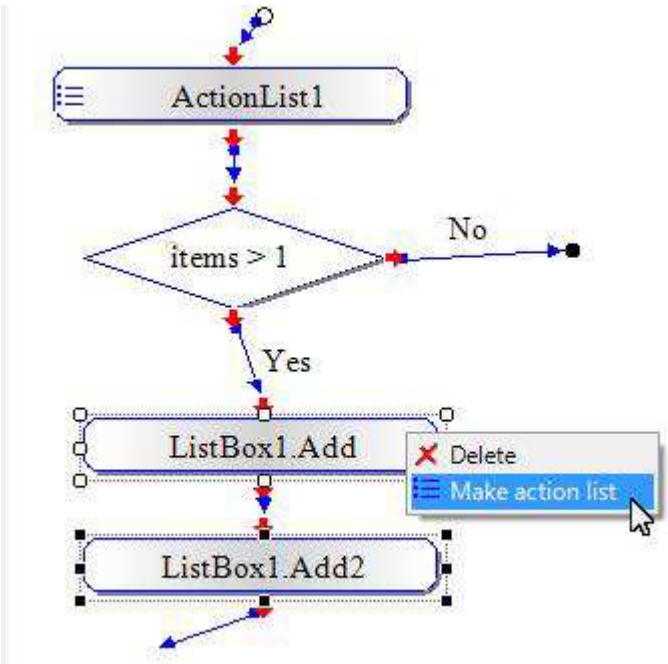
-- move selected action up

-- move selected action down

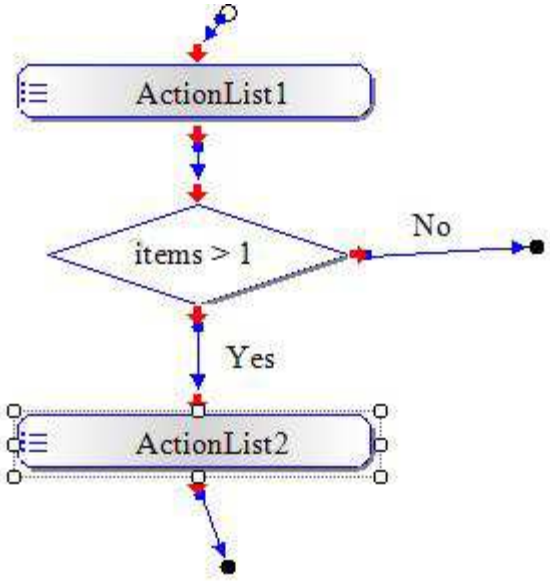
Cancel -- Cancel editing and close the dialogue box

OK -- accept the editing and close the dialogue box

Create another action list:

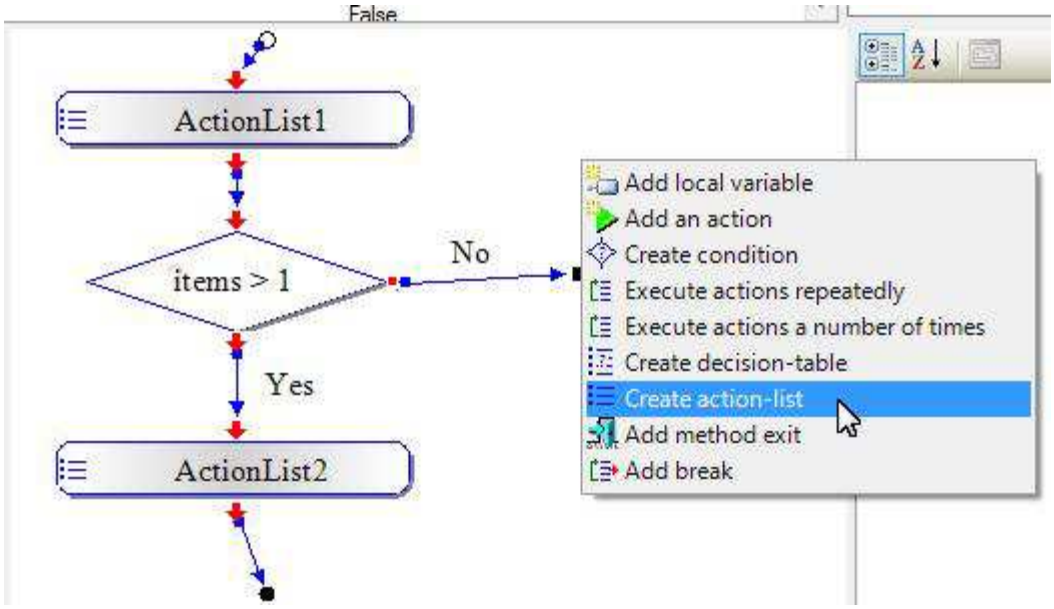


The method is now much clear:



11.2 Explicitly create action list

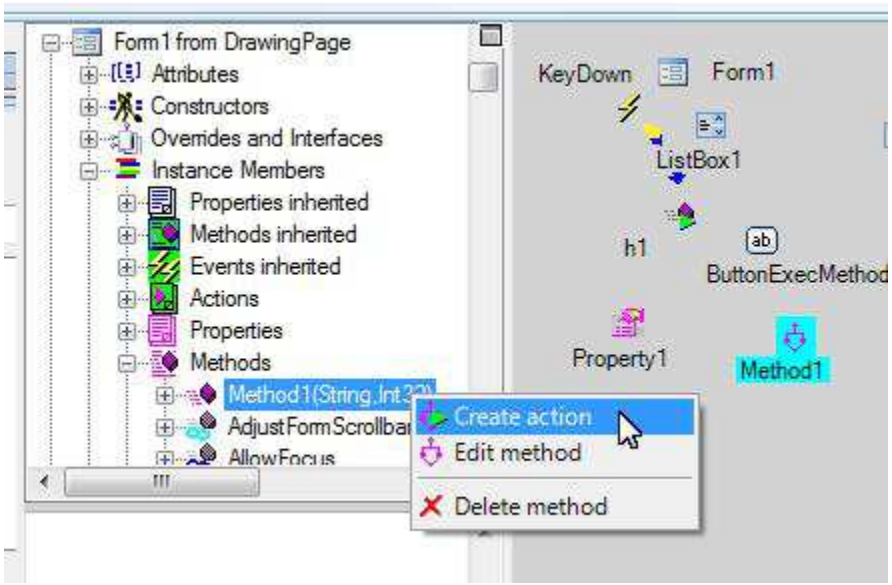
An action list may also be created by right-clicking the action pane and choose “Create decision table”:

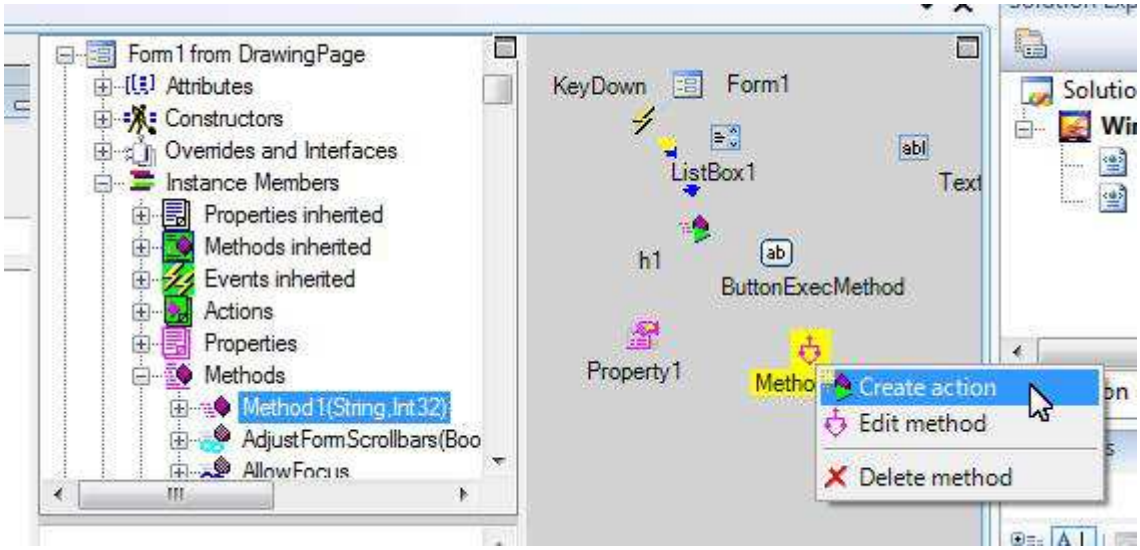


Once we have an action list action we may add actions to it.

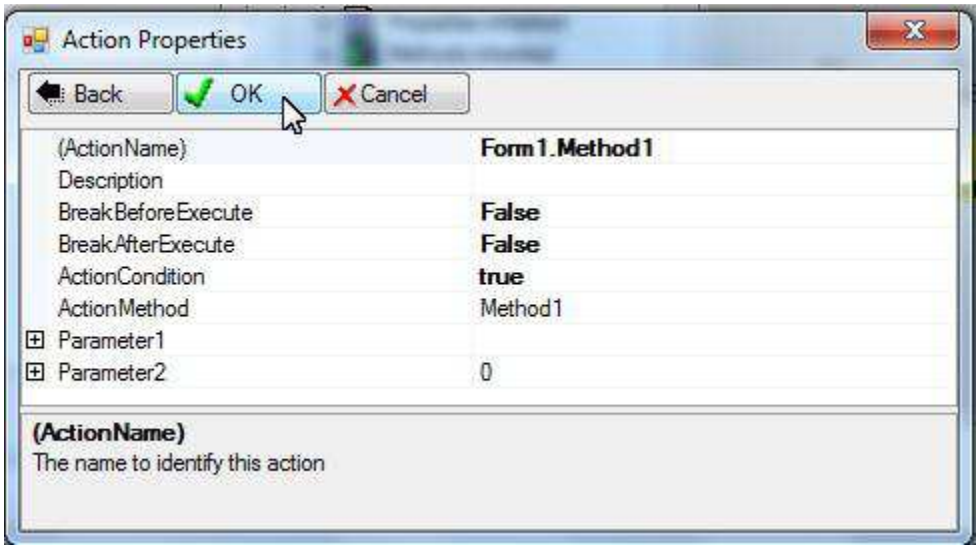
11.3 Test

Using a method we created is the same as using a method from library. Right-click the method, choose "Create action":

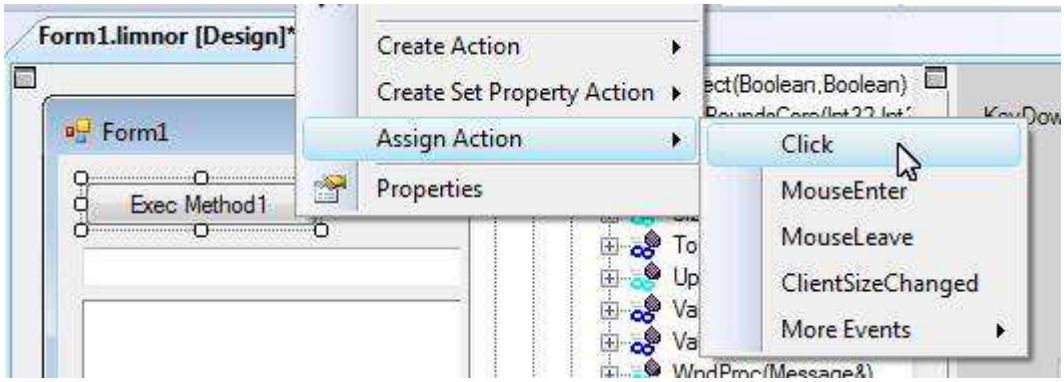


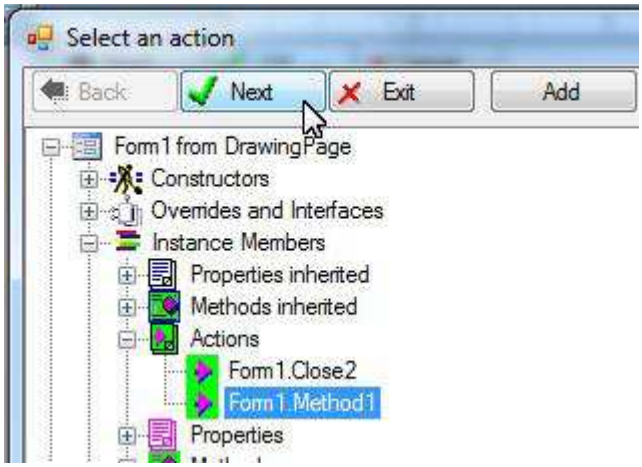


We created a bad method because we created two method parameters but they were not actually used in the method. So, we do not need to provide values for parameter1 and parameter2. We will later create a more practical method sample which uses parameter values.



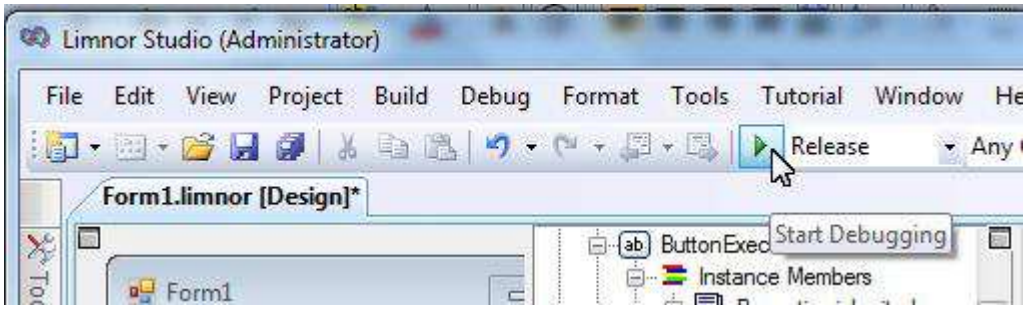
Let's assign this action to the Click event of a button:



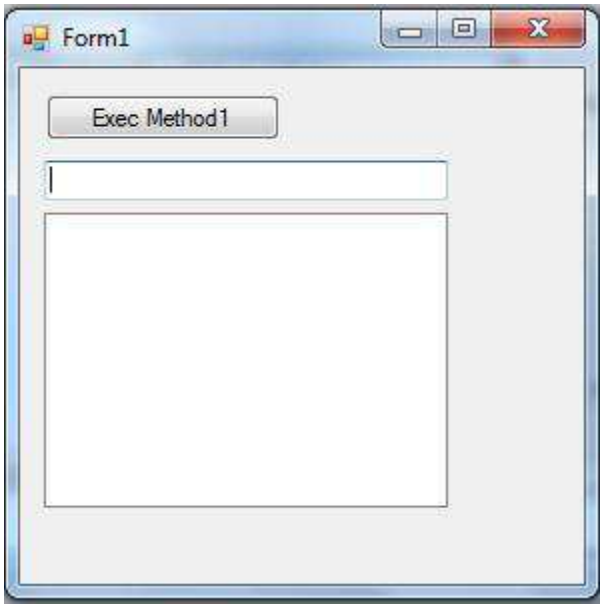


The action is assigned to the button.

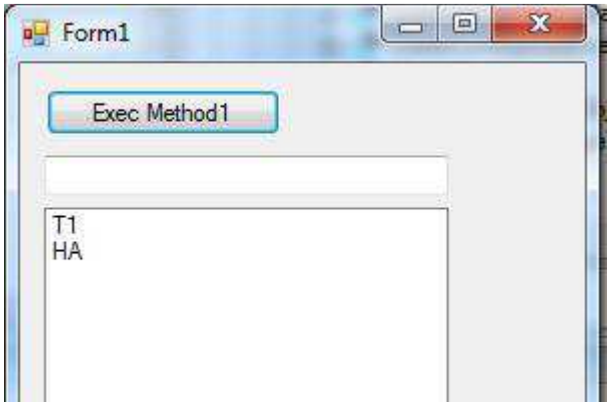
Compile and debug the program:



The program runs:



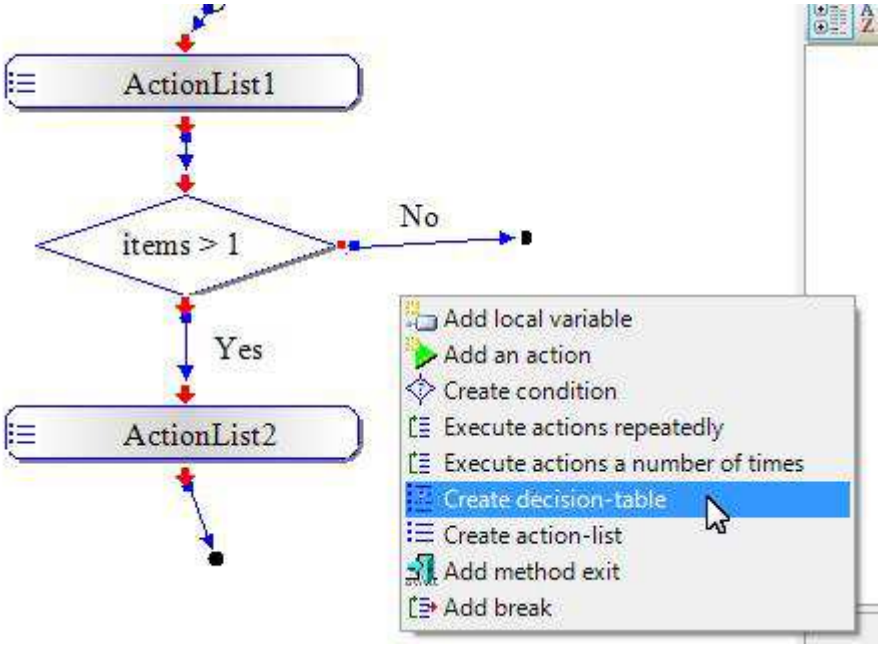
Click the button. The array items are added to the list box:



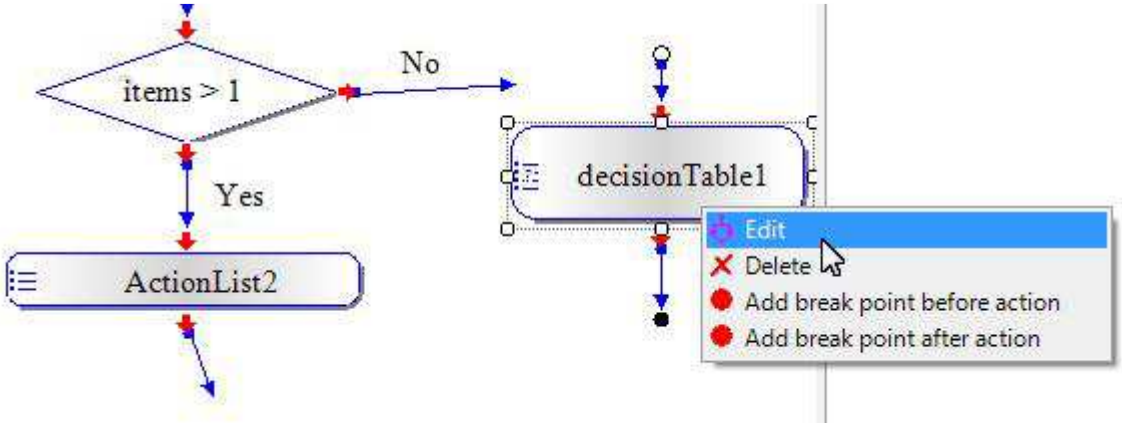
12 Decision Table

A decision table is a list of {condition, actions} pair. When a decision table is executed, condition in each pair is evaluated. If a True condition is found then the corresponding actions are executed and the rest of the pairs are ignored.

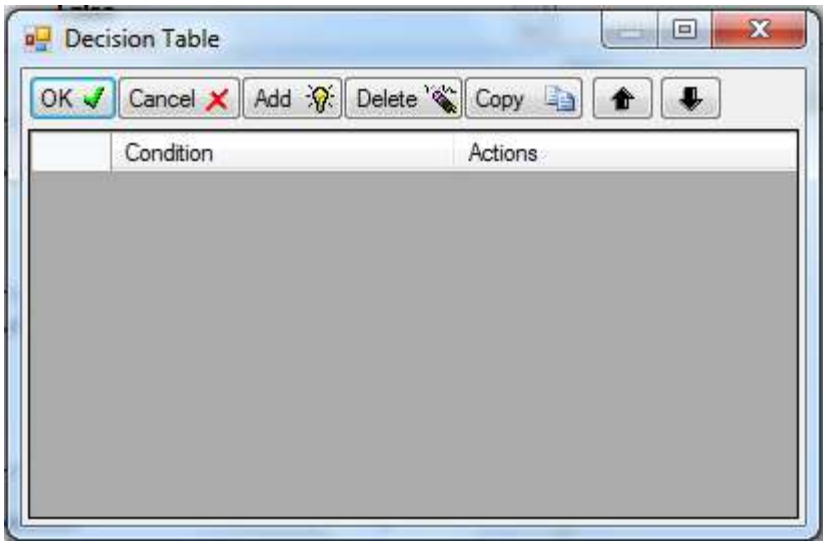
To create a decision table, right-click the action pane, choose “Create decision table”:



A decision table action is created. Right-click it and choose “Edit” to edit its {condition, actions} pairs:

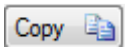


The Decision Table editor appears:



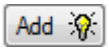
 -- Add a new {condition,actions} pair

 -- Remove selected {condition,actions} pair

 -- Add a new {condition,actions} pair by duplicating selected {condition,actions} pair

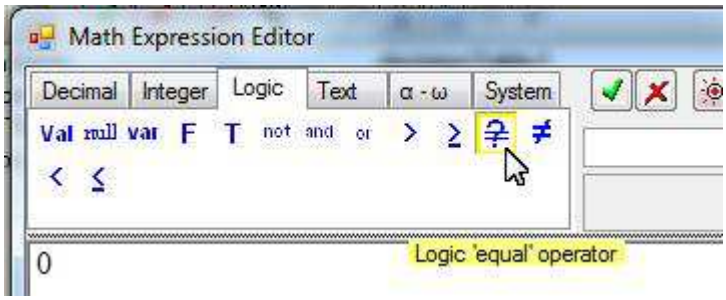
 -- Move the selected {condition,actions} pair up in the list so that it will be evaluated/executed earlier.

 -- Move the selected {condition,actions} pair down in the list so that it will be evaluated/executed later

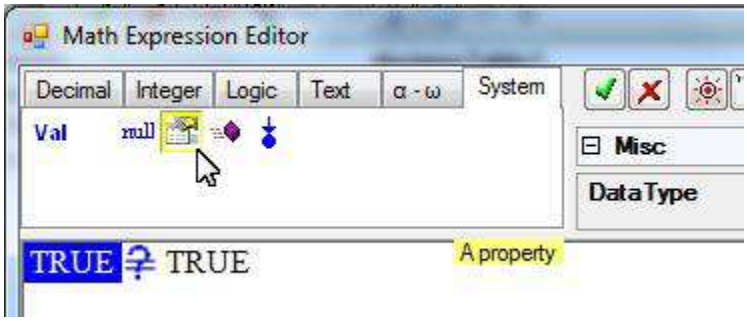
Let's click  to add a new {condition,actions} pair.

With the Condition column selected, click  to enter condition for the pair:

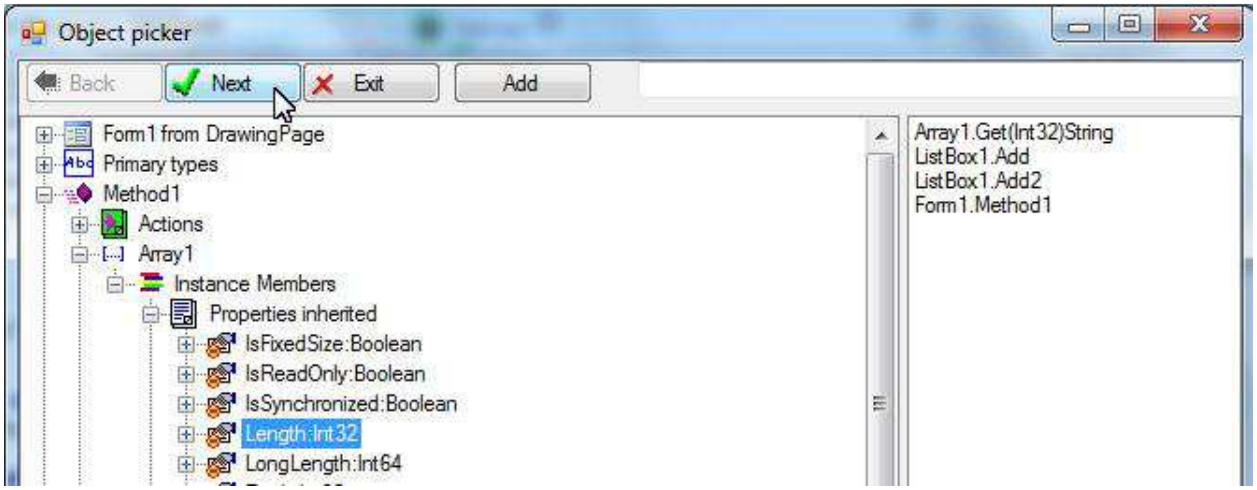
Suppose we want the condition to be that the array is empty. That is, the Length property of the array is 0. Click the equality checking icon:



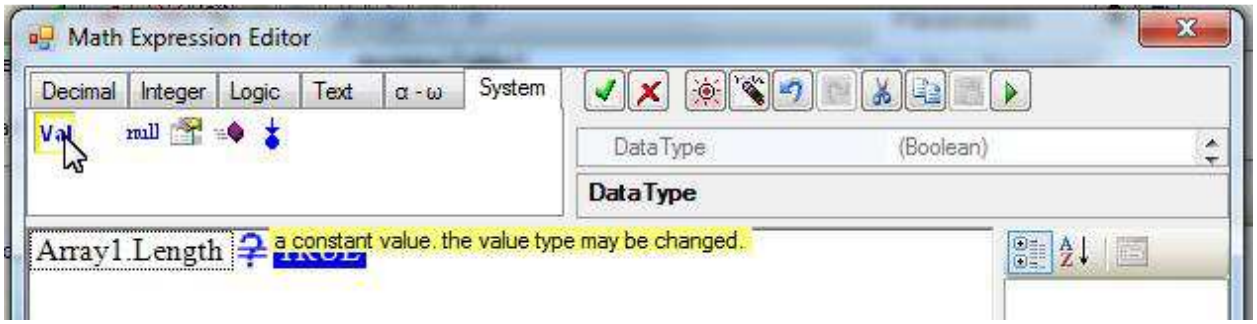
Select the first element; click the Property icon for selecting the Length property:



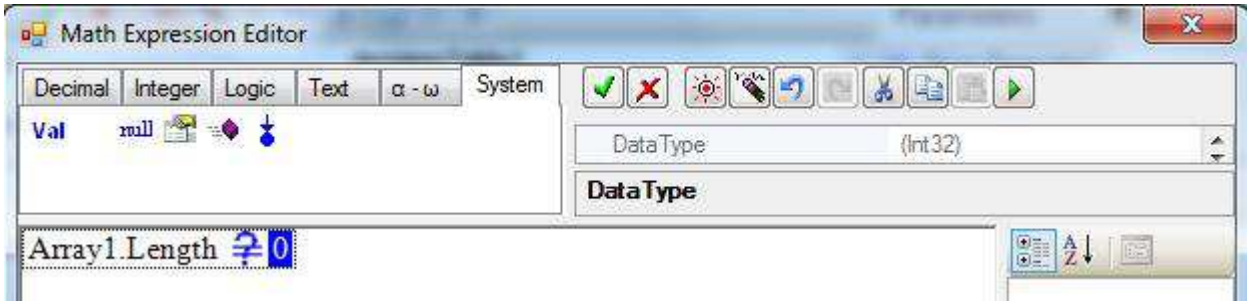
Select the Length property of the array variable:



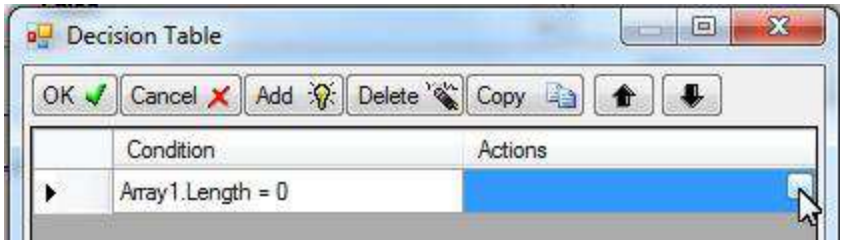
Select the second element; click the integer value icon for entering an integer value:



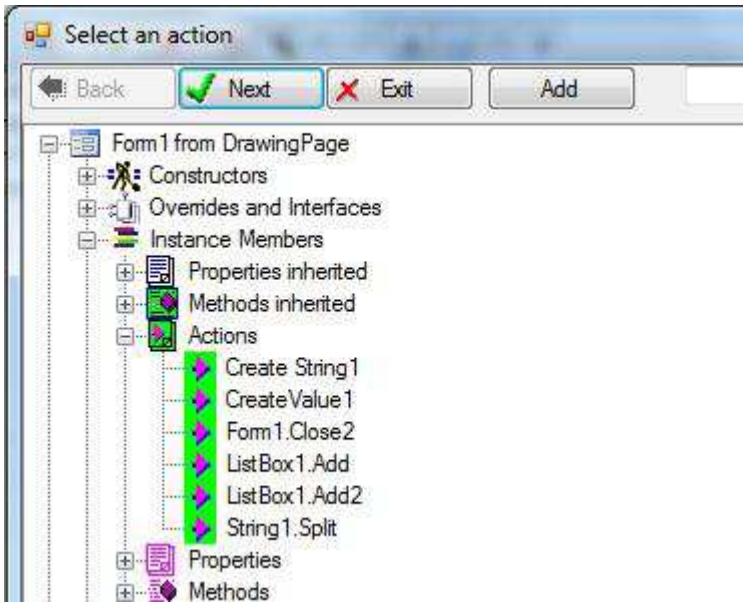
0 is the value we want:



With the Actions column selected, click to enter actions for the pair:

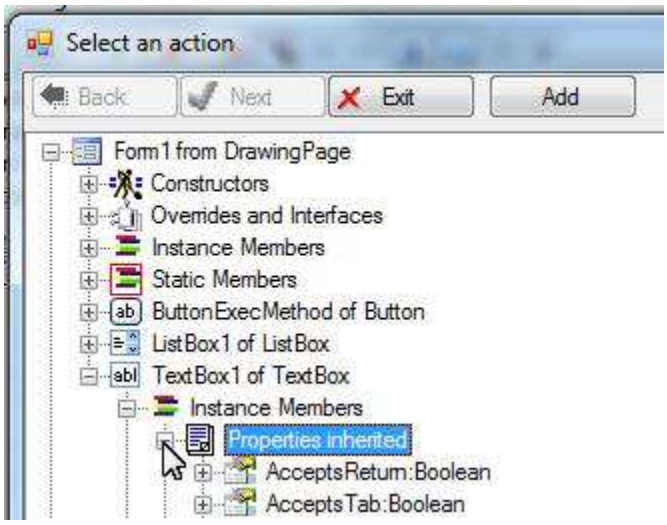


The "Select action" dialogue box appears.

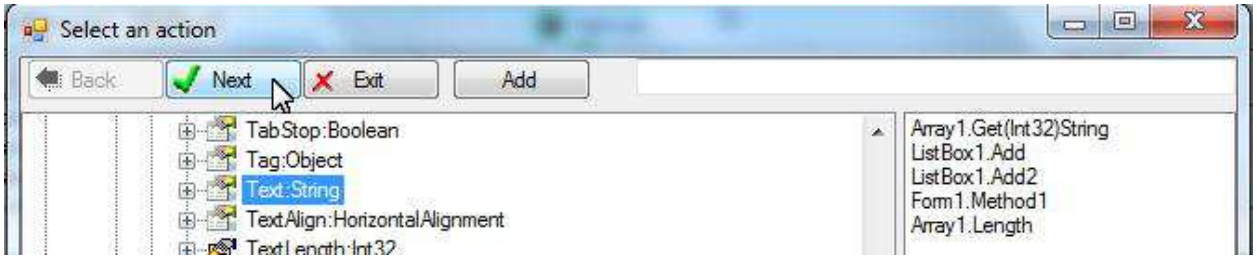


We may select one or more existing actions. We may also create new actions.

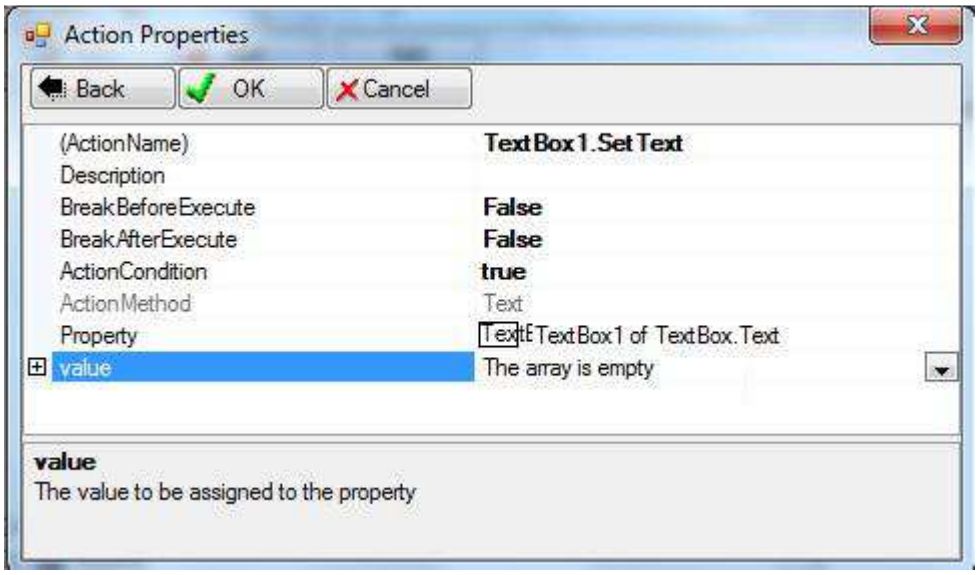
Let's create a new action of setting the Text property of the text box. Expand node "Properties inherited" of the Text Box:



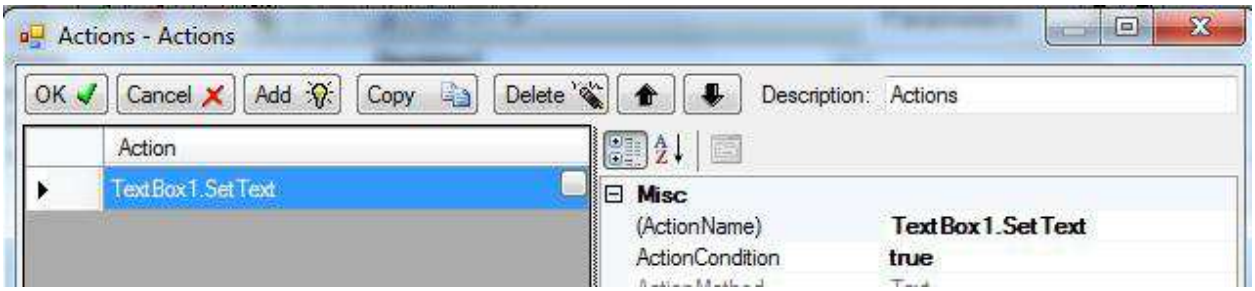
Scroll down and find the Text property. Select the Text property and click "Next"



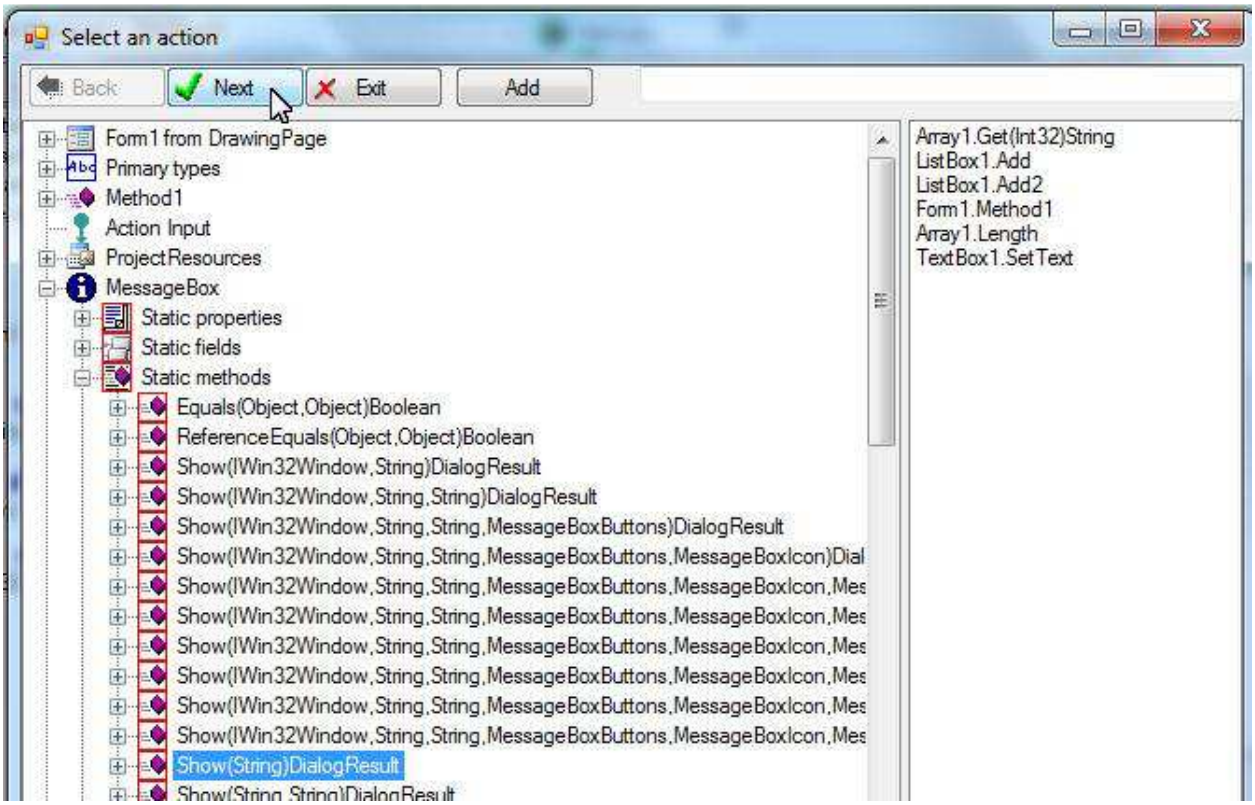
Type some message as the new property value and click OK:



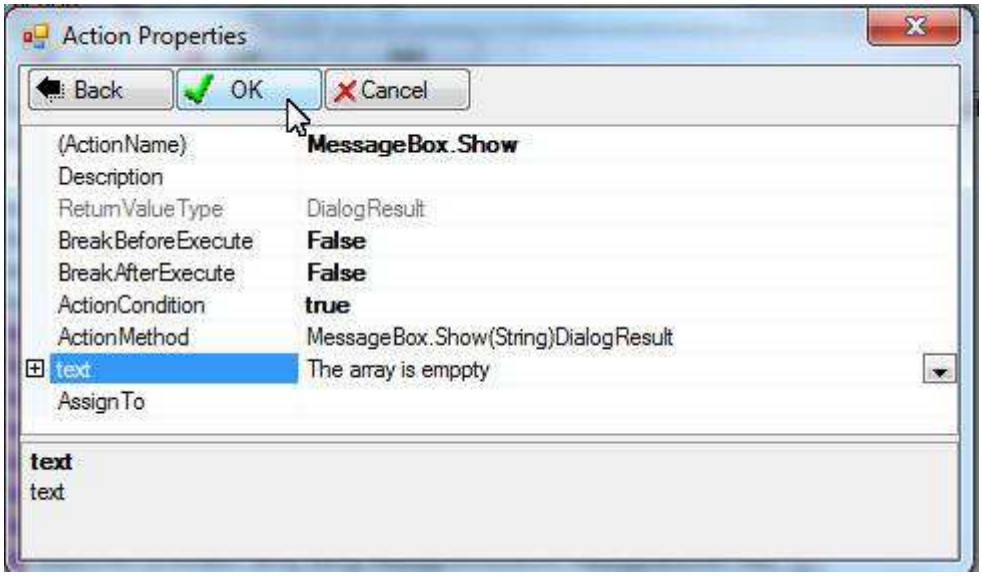
The action is created and added to the action list:



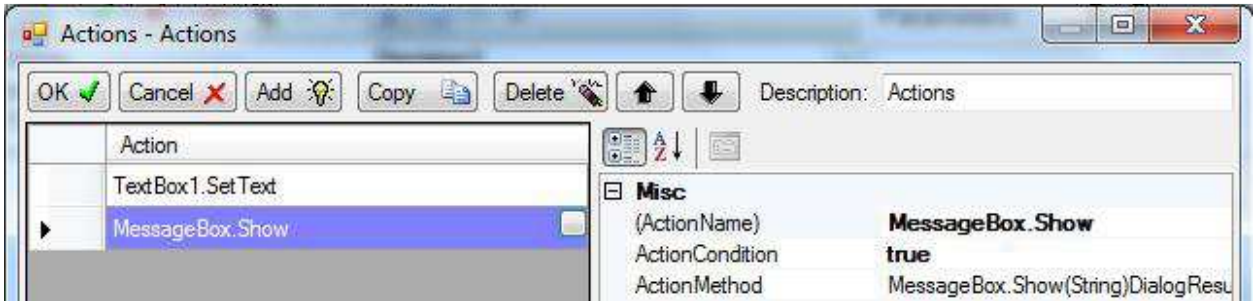
We may create more actions if we want. Let's just create another action to show that we can do it. Click "Add" button; select Show method of the MessageBox; click "Next":



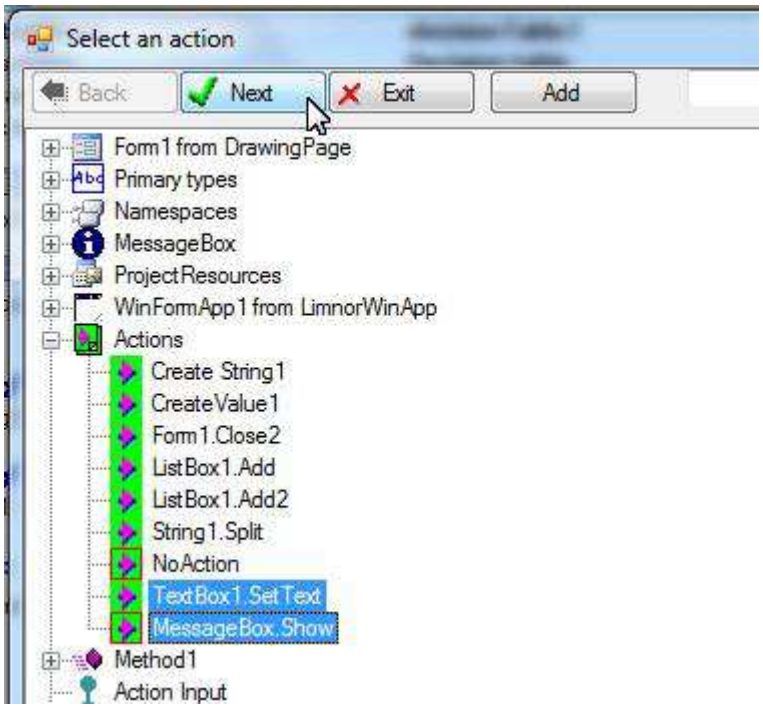
Type some message for the Show action and click OK:



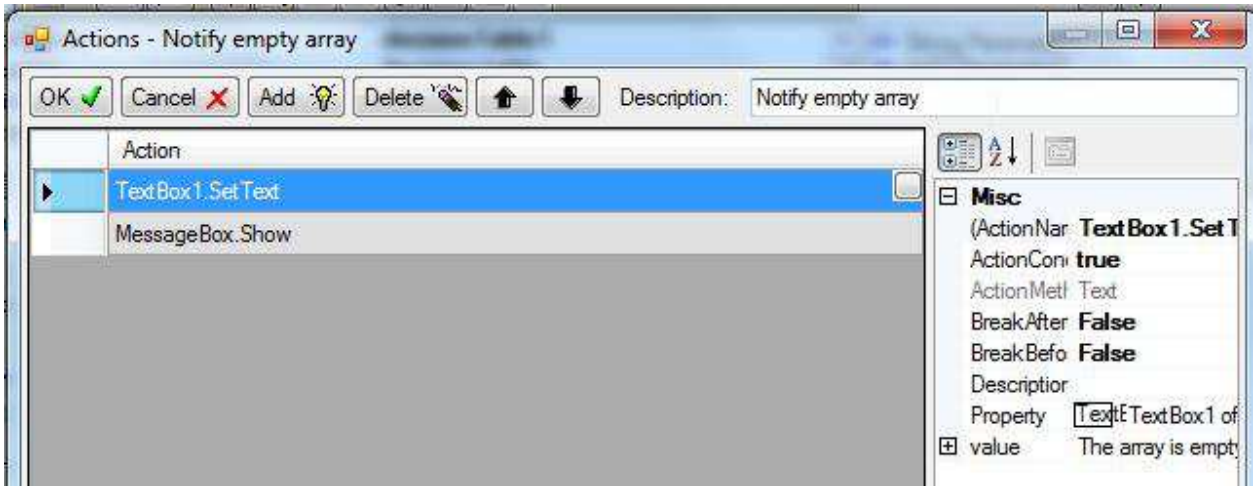
The new action is created and added to the action list:



We may also add existing actions to the action list. On clicking “Add” button, all actions can be found under Actions node. Select the actions we want for our action list and click “Next”:

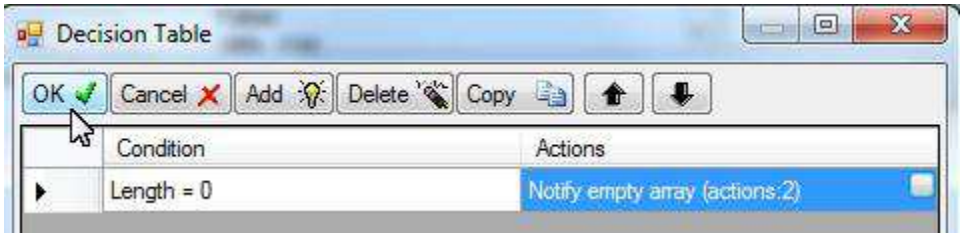


We may give the action list a description:

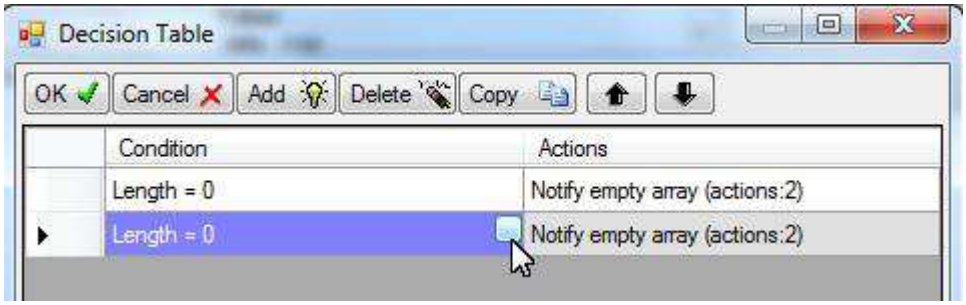


Each action can be modified by selecting an action and modify its properties.

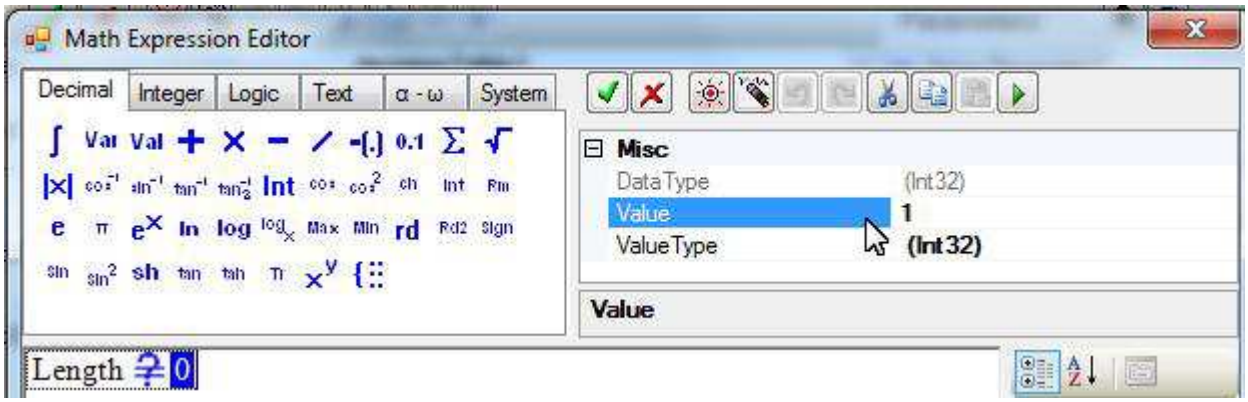
Click OK. We have our first {condition, actions} pair completed:



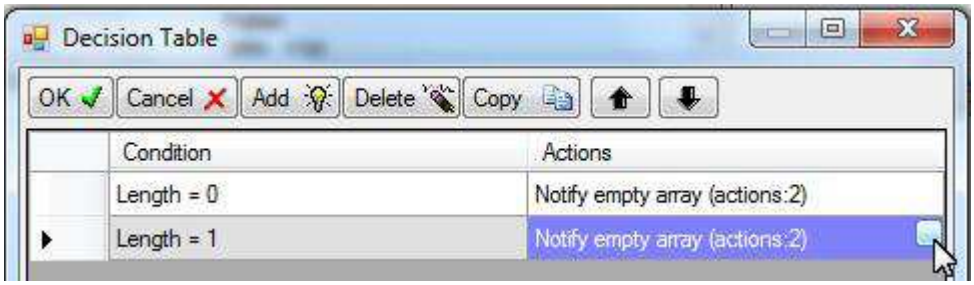
We may add more {condition, actions} pairs. If a new {condition, actions} pair to be added is similar to an existing one then we may select the existing one and click Copy. Once we have a copy we may modify its condition and actions:



Let's change the comparison value from 0 to 1:



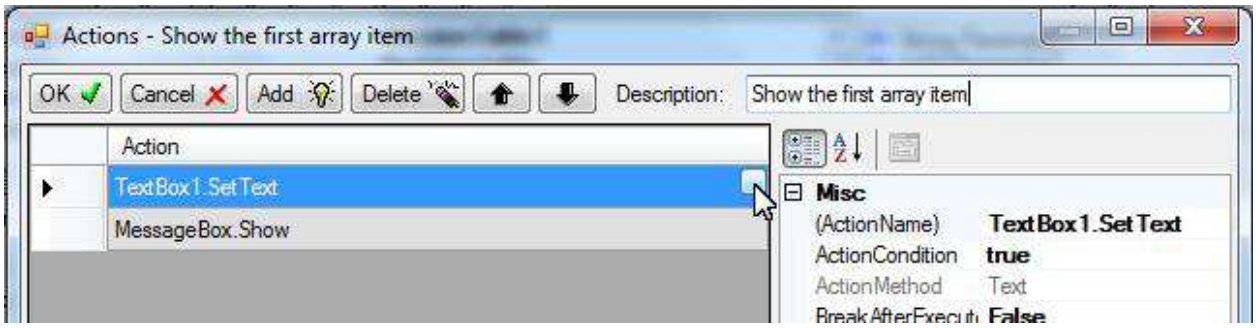
Let's modify the action list:



First, let's change the action list description.

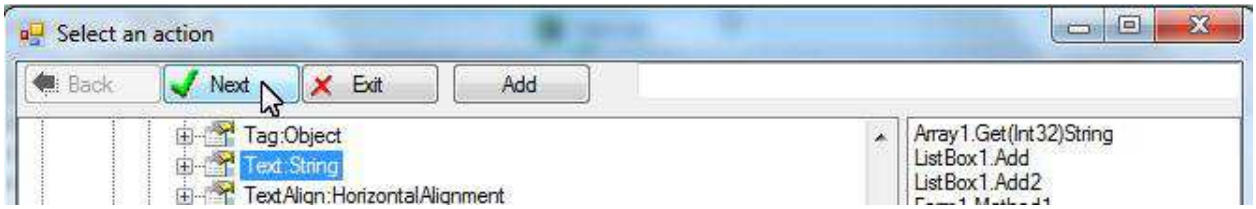
Be very careful about modifying existing actions because other {condition, actions} pairs are also using the actions. Be aware of the fact that if you modify an action then other {condition, actions} pairs will also use modified actions.

We may click Delete to remove actions from the list. We may click to replace selected action:

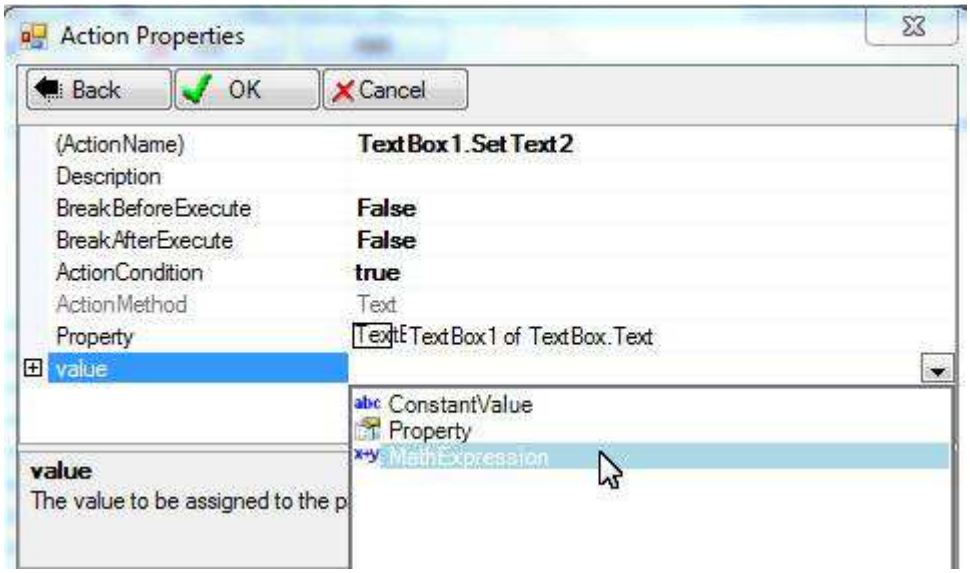


Let's create a new Set Property action for the Text property of the text box. This time we will show the contents of the first array item.

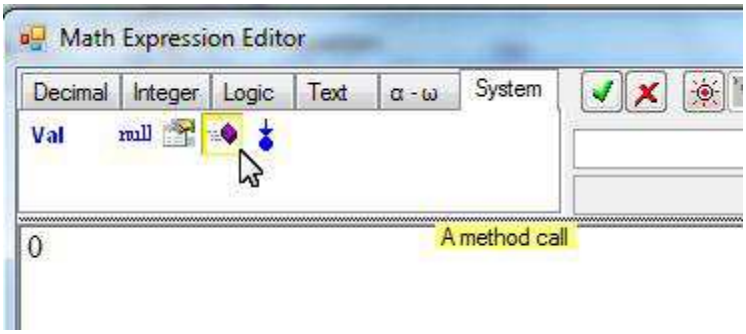
Click to replace action "TextBox1.SetText". Select the Text property of the text box; click "Next":

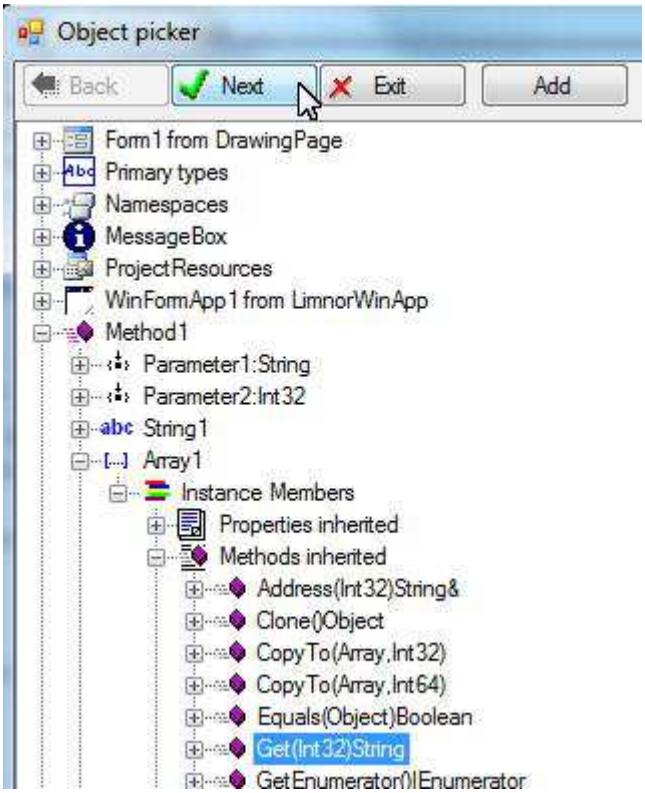


Choose Math Expression for the property value so that we may get array item:

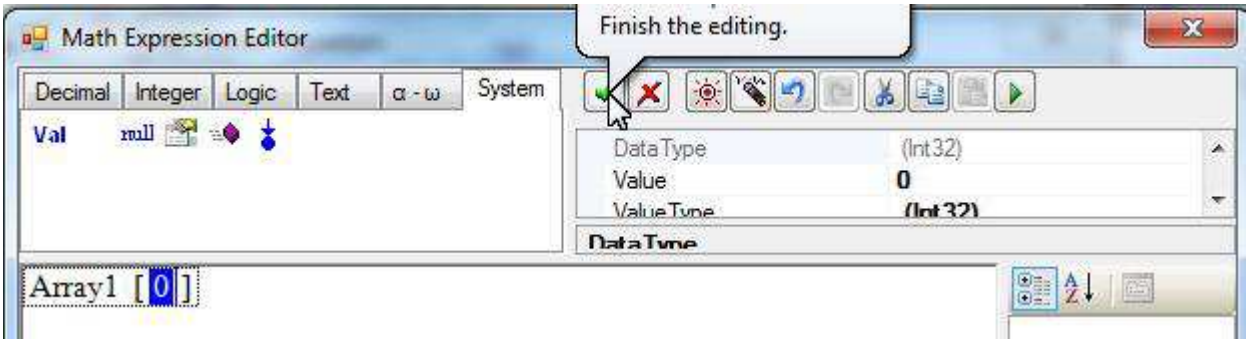


Click the Method icon because array item can be retrieved by calling Get method of the array:

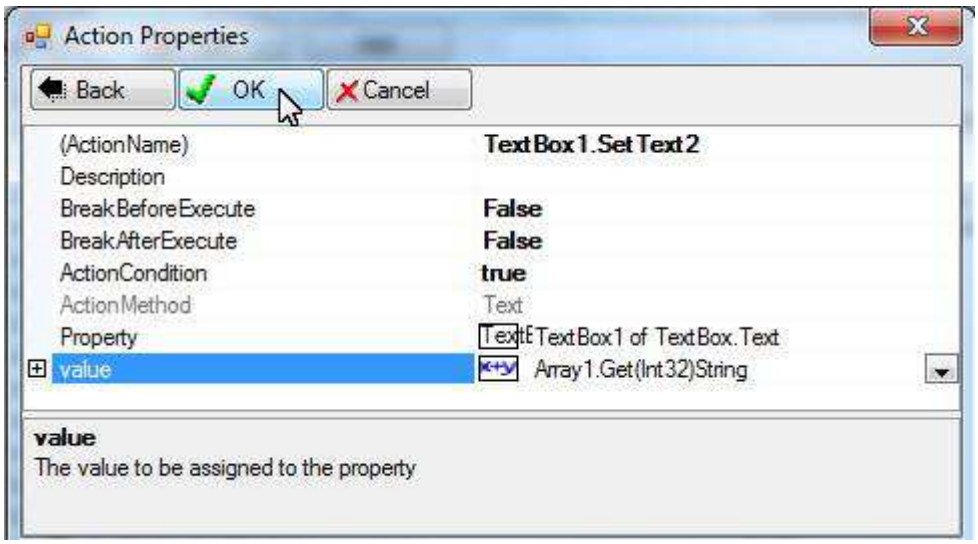




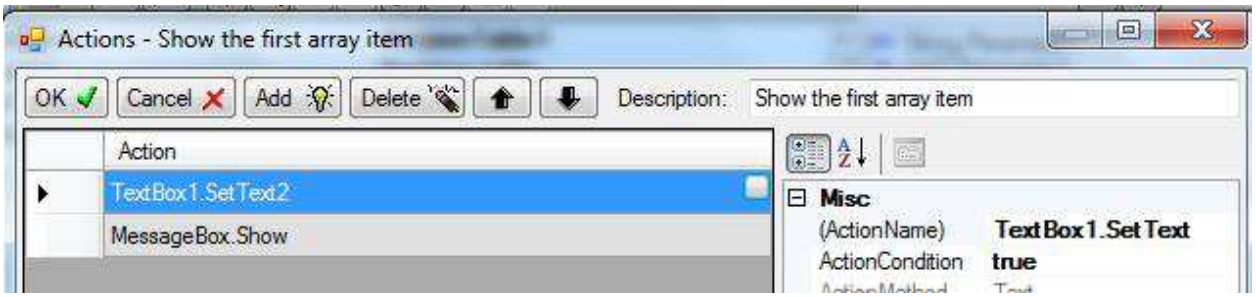
The Get method appears in the expression. By default its parameter is 0 which happen to be what we want because 0 indicates the first array item:



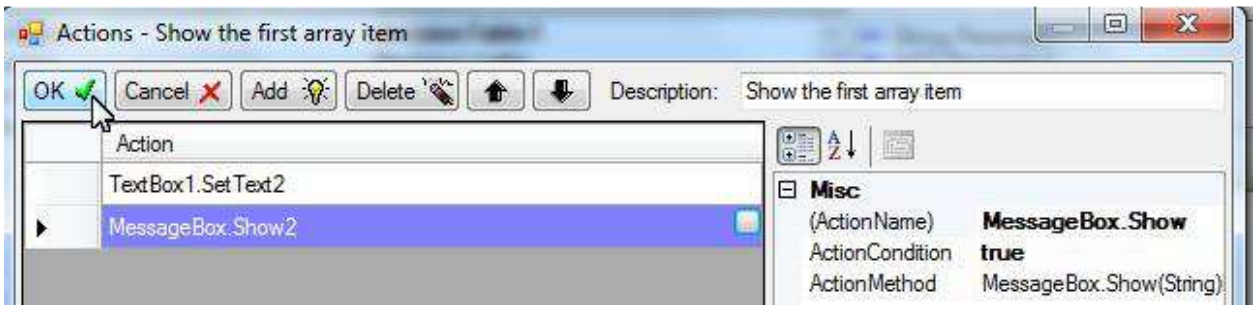
Click OK to finish creating this action:



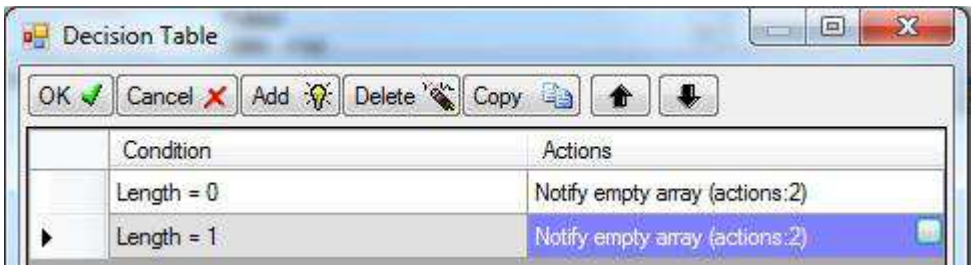
The new action appears in the action list, replacing the original action:



We may do the same for the other action. We are not going into details.

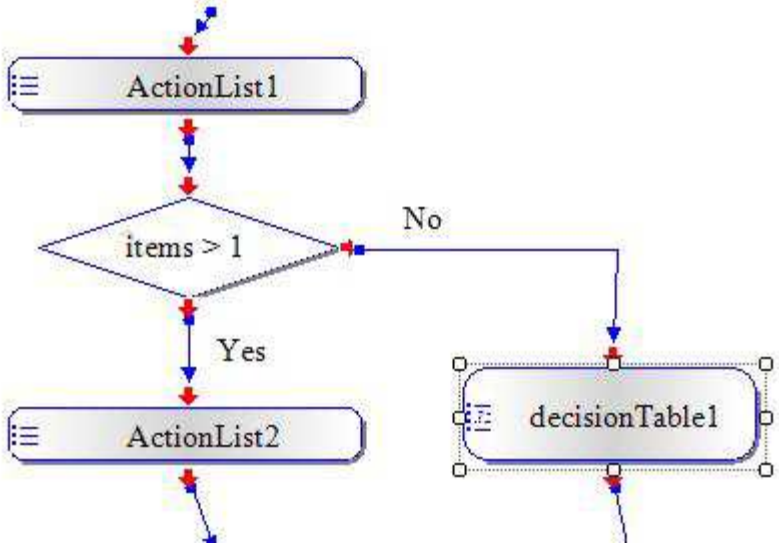


Now our decision table has two {condition, actions} pairs:



We may add more {condition, actions} pairs into the decision table.

Link the decision table action to the other actions in the method:



Remember that even more than one {condition, actions} evaluates True only the actions for the first matching {condition, actions} pair will be executed. The rest of {condition, actions} pairs will be ignored.

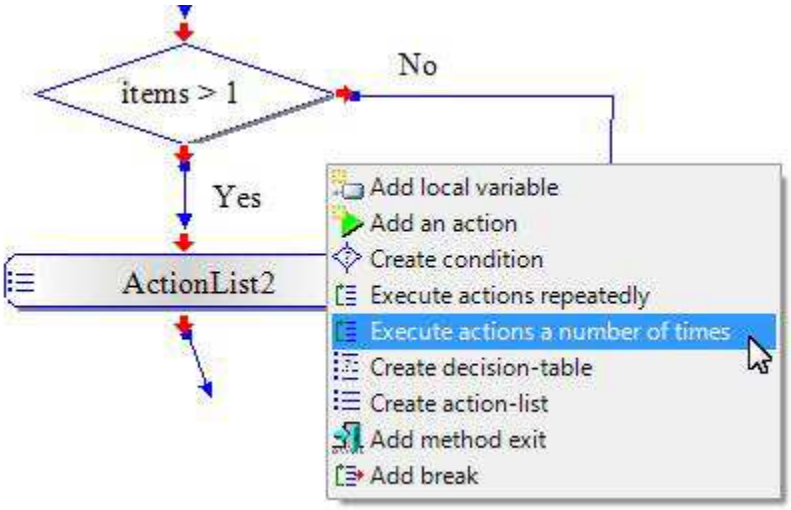
13 Execute Actions a Number of Times

Suppose we want to add all items of a string array to a list box. Previous we used two actions to add two array items to a list box. If the array has many items then this is not a good approach. In this User's Guide we describe 3 other approaches. The purpose is to introduce these approaches, not to find a best approach for a specific task.

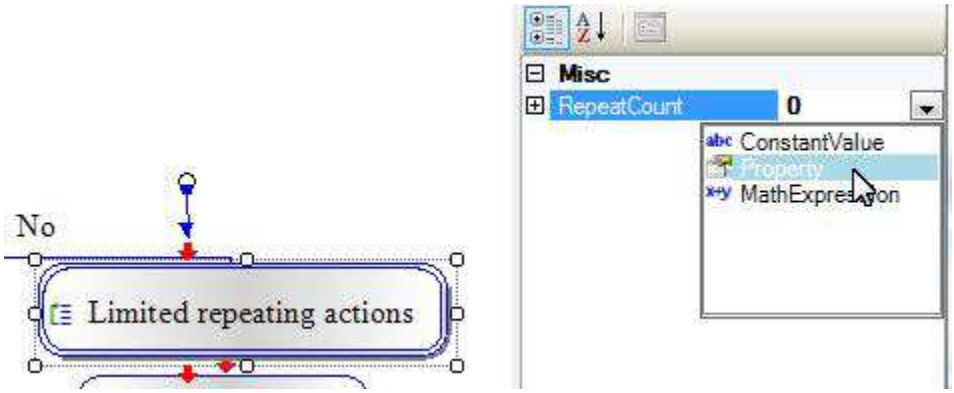
13.1 Create Limited Number of Repeated Execution

This section introduces "Limited Number of Repeated Execution".

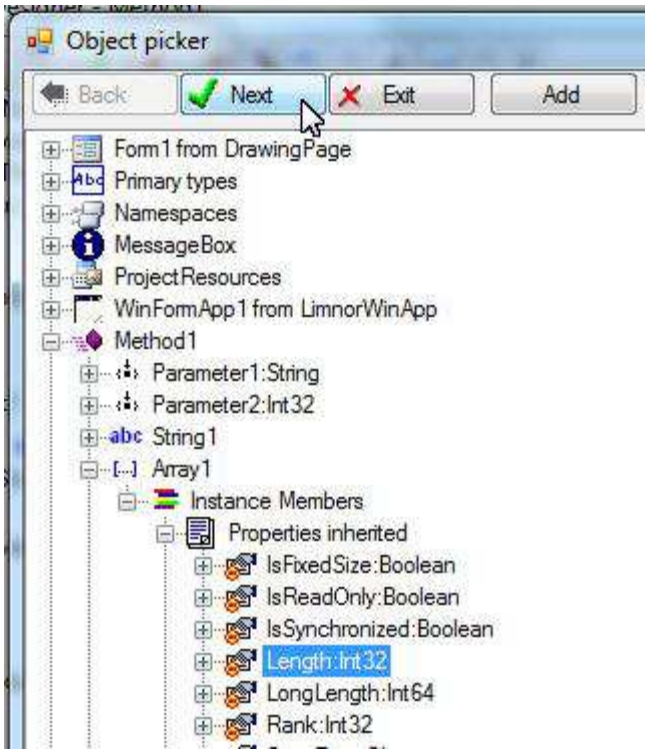
Right-click the Action Pane, choose "Execute actions a number of times"



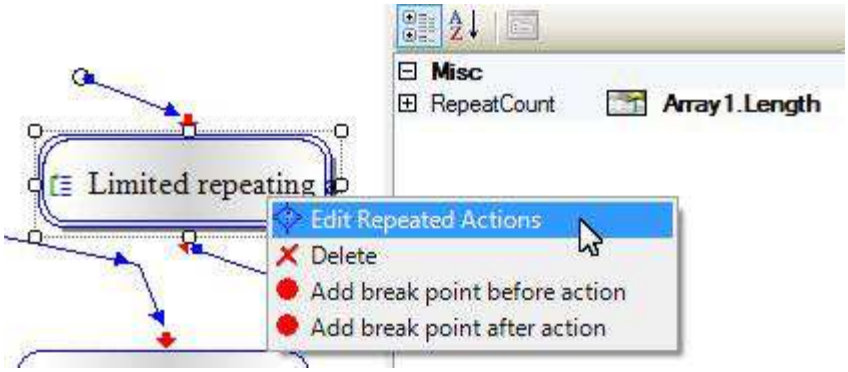
A new "Limited repeating actions" appears. Its RepeatCount property indicates the number of times its actions will executed. We may choose "Property" to set this number to the Length of the array:



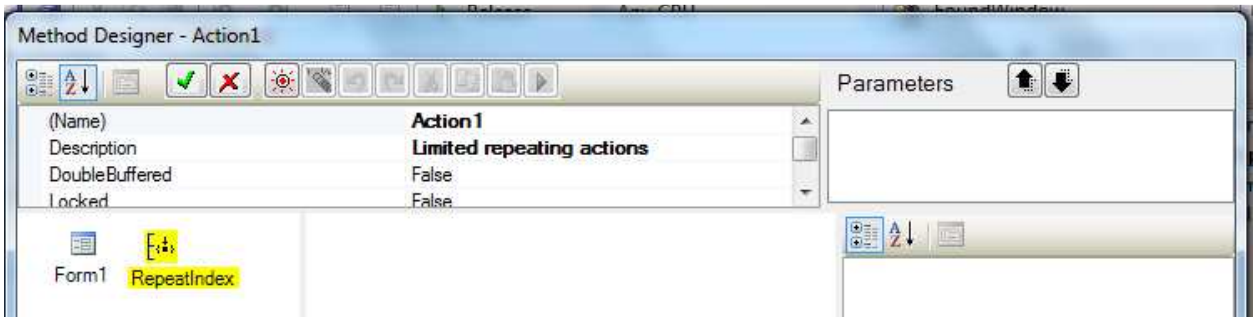
Select the Length property of the array:



Right-click it and choose “Edit Repeated Actions”



A new Method Editor appears for adding actions to be executed:

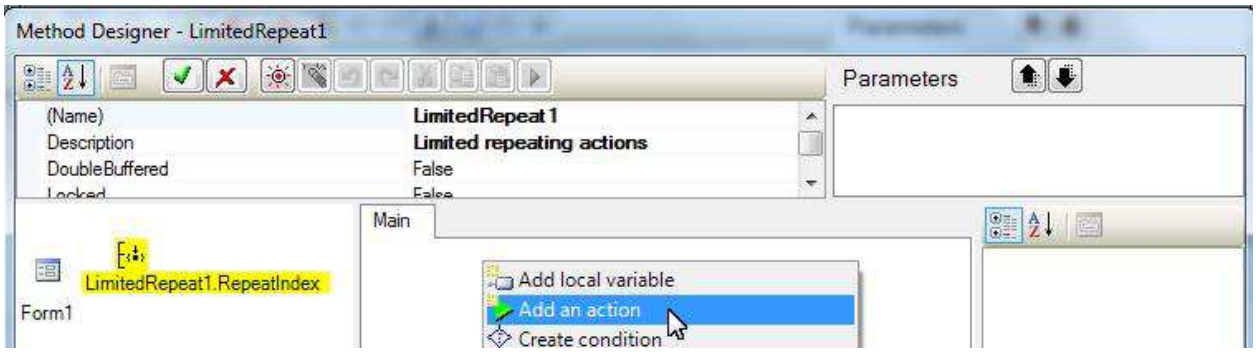


Note the icon 

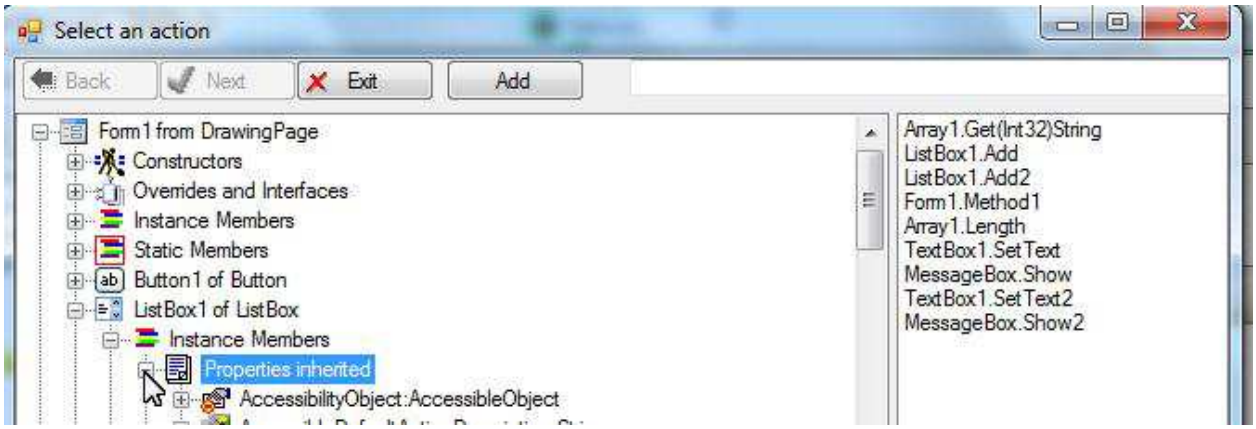
. It represents the number of times of the execution. 0 indicates it is the first time execution; 1 indicates the second time execution; 2 indicates the third time execution, and so on.

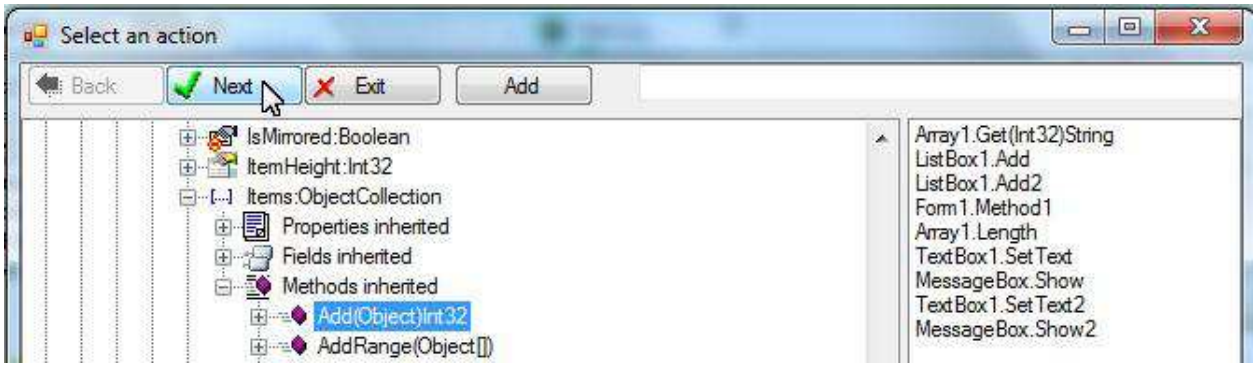
We may use this RepeatIndex as the index for getting the array item. For the first time execution we get the first array item; the second time execution we get the second array item, and so on.

Let's add an action to add array item to the list box. Right-click the Action Pane, choose "Add an action":

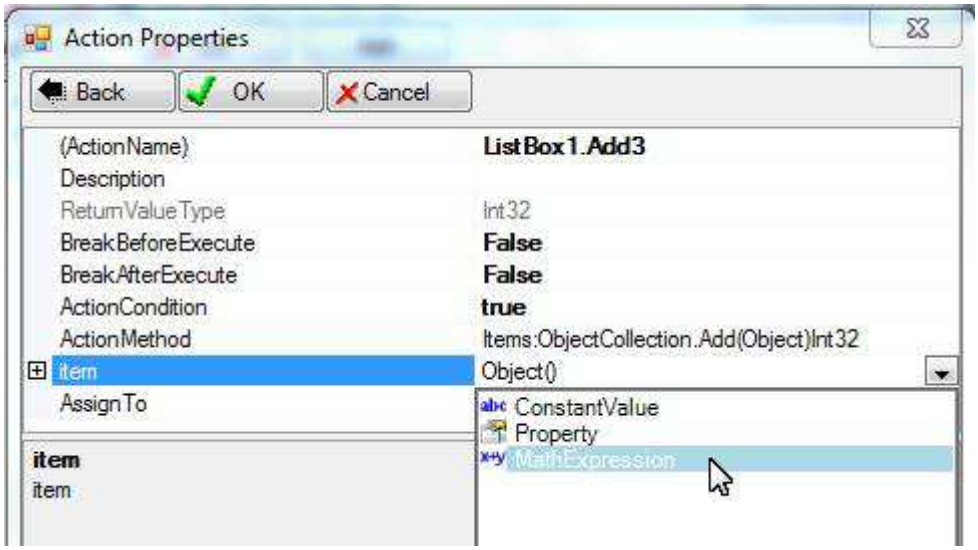


Select the Add method of the Items property of the list box, and click "Next":

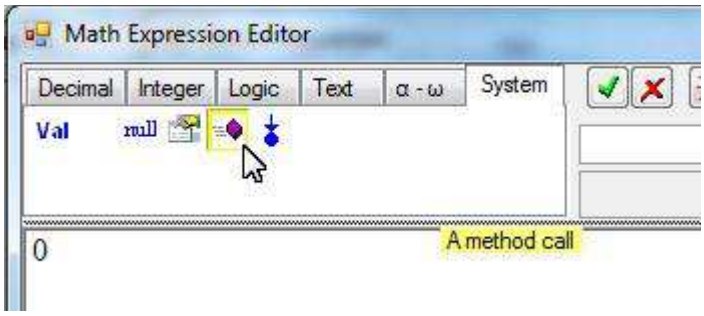




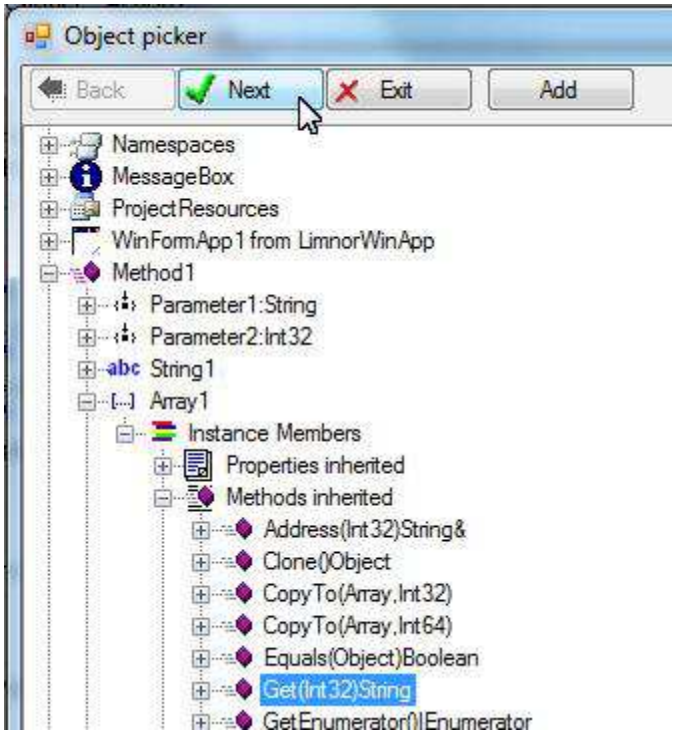
Select Math Expression to call Get method of the array to get array item:



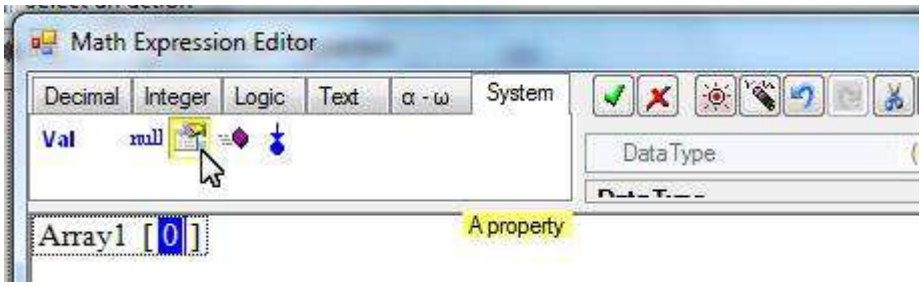
Click the Method icon:



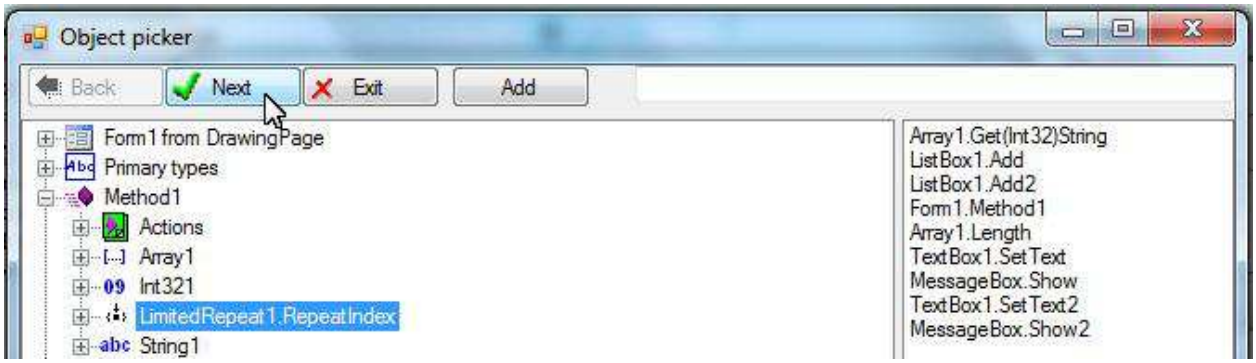
Select the Get method of the array:



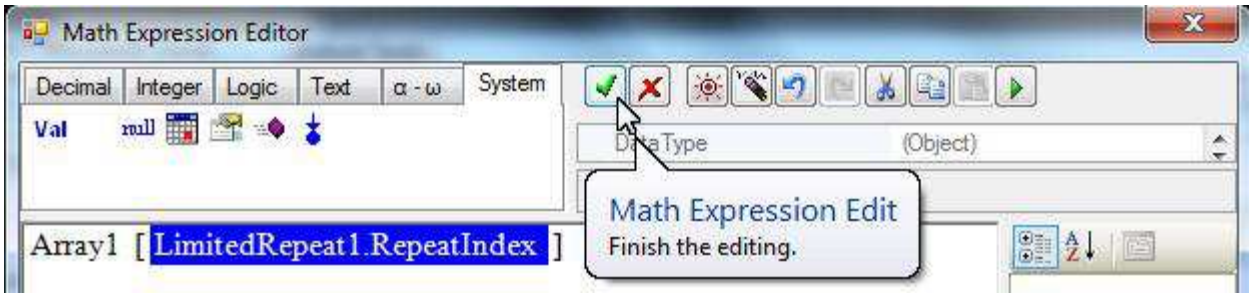
Select the method parameter; click the Property icon:



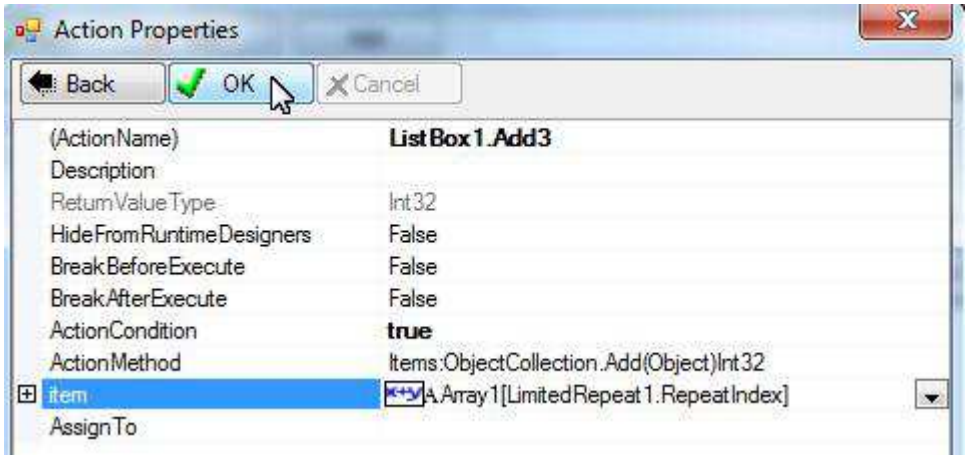
Select the RepeatIndex:



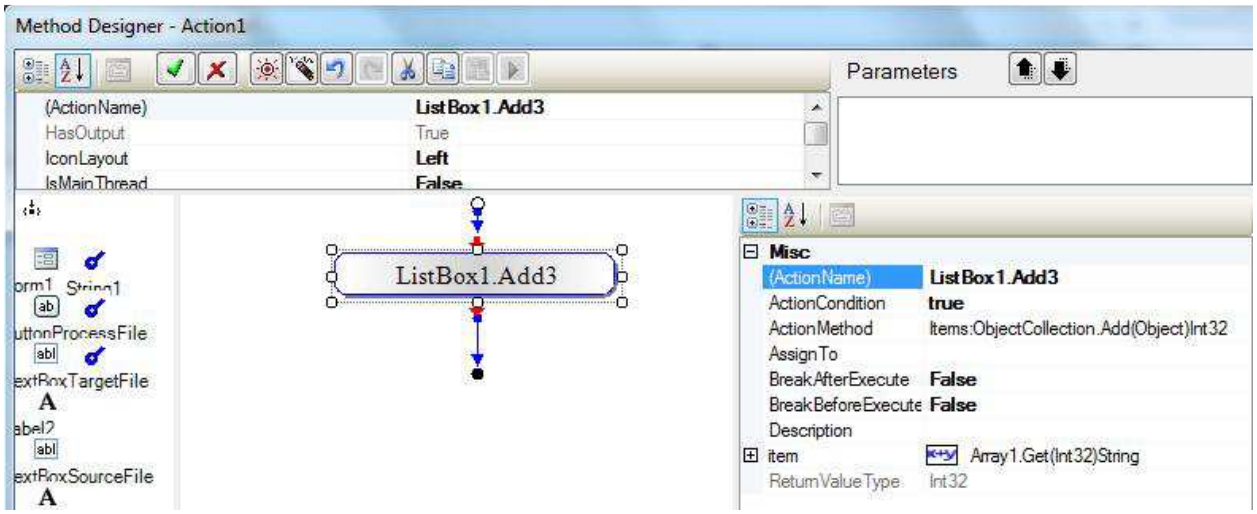
This is the value to be added to the list box. It gets an array item identified by RepeatedIndex.



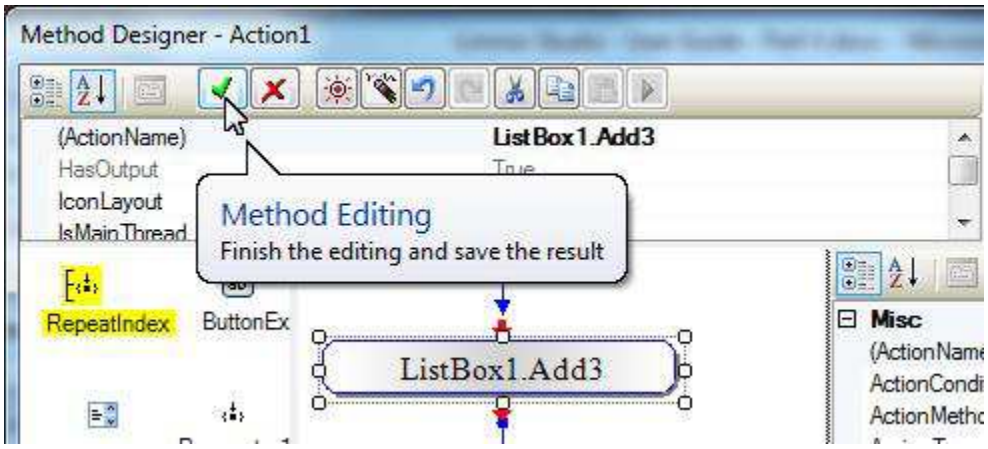
Click Ok to finish creating this action:



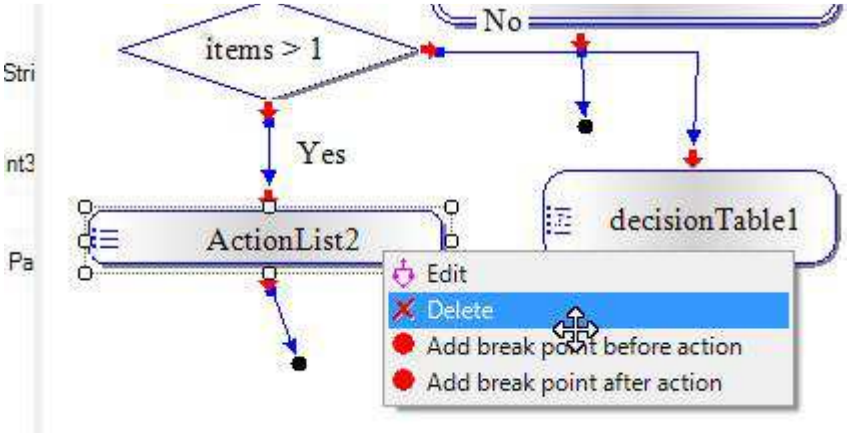
The action appears in the method:



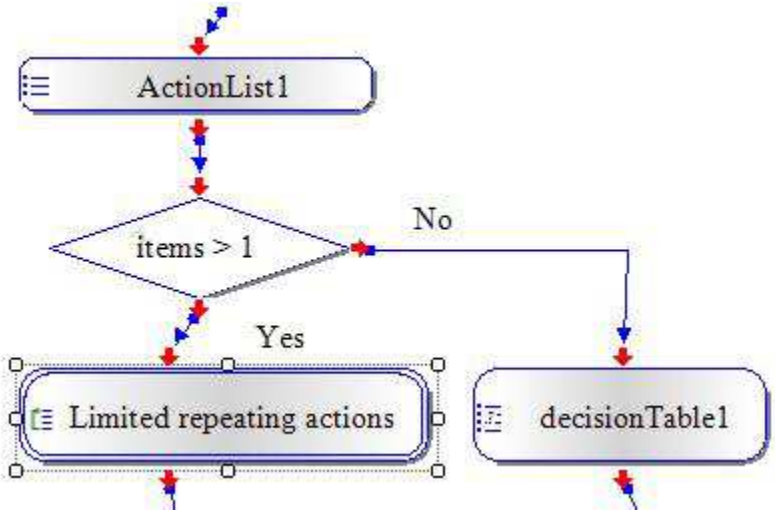
For this sample, we just need this one action.



Let's remove the action list:

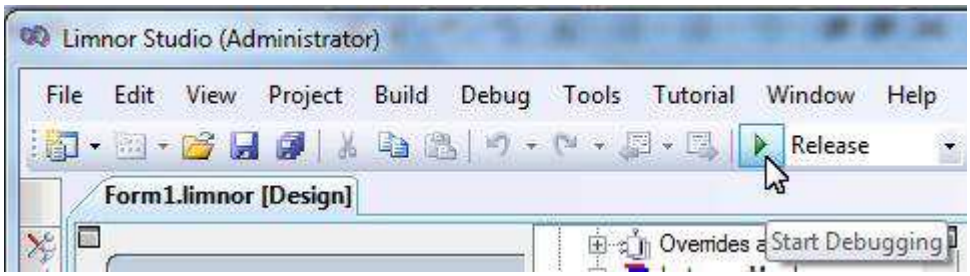


And add the new repeat action in the place:

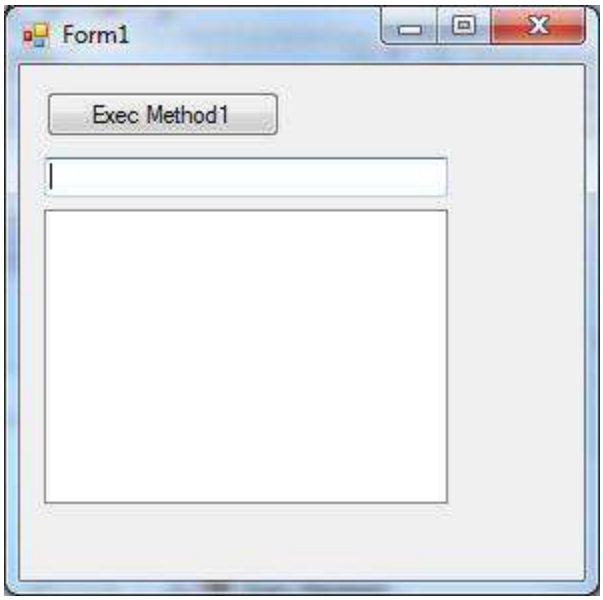


13.2 Test

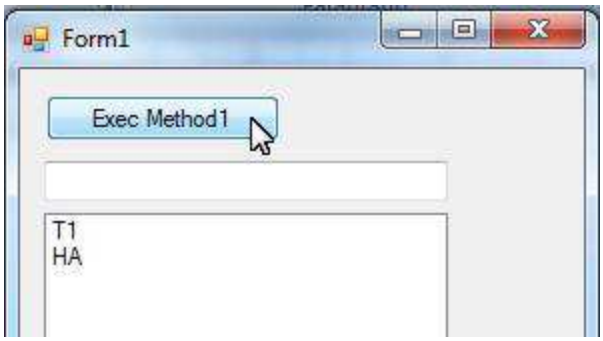
Compile and test the application:



The application runs:



Click the button, the array items are added to the list box:



It looks exactly like the previous sample. The difference is that if the array has 1000 items then it will add all 1000 items into the list box, not just the first 2 items as the previous sample did.

14 Execute Actions repeatedly

In the previous sample, we know how many times the actions should be executed. In that example, we know the number of execution times by the number of array items.

Sometimes we do not know how many times the actions should be executed. Let's example such an example: suppose we want to process such a text file, each line of the file is in such a format:

{month/day/year hour:minute:second},{value1},{value2}

We want to write the file to another file and change each line into the following format

{year}{month{day},{value 1},{value 2}

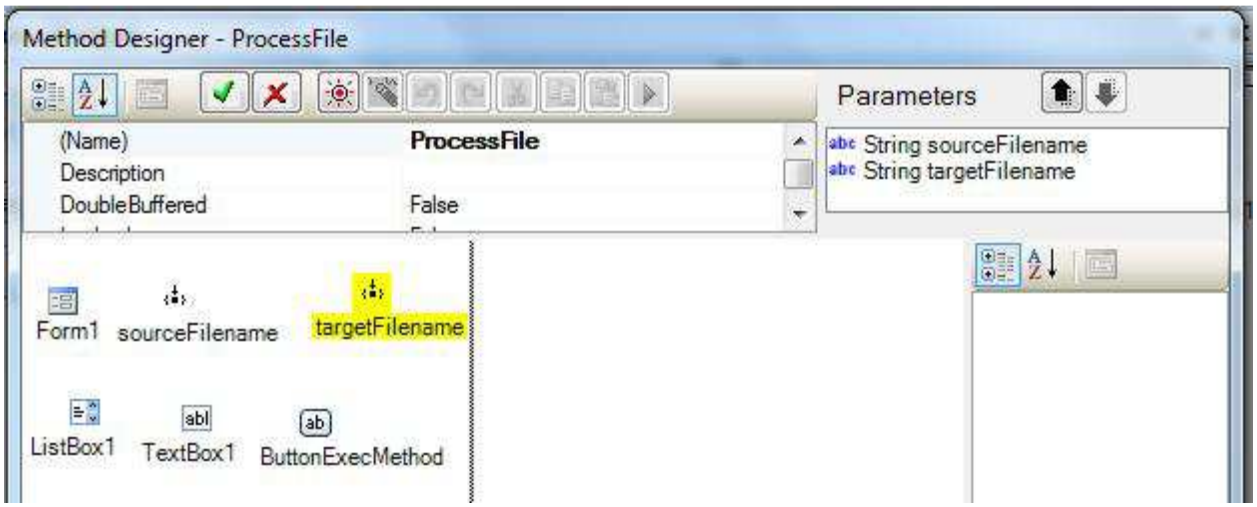
Because we do not know how long the file is we cannot use the previous sample to do it. We may use "repeat actions" to do it.

14.1 Prepare for File Read and Write

14.1.1 Create Method with Parameters

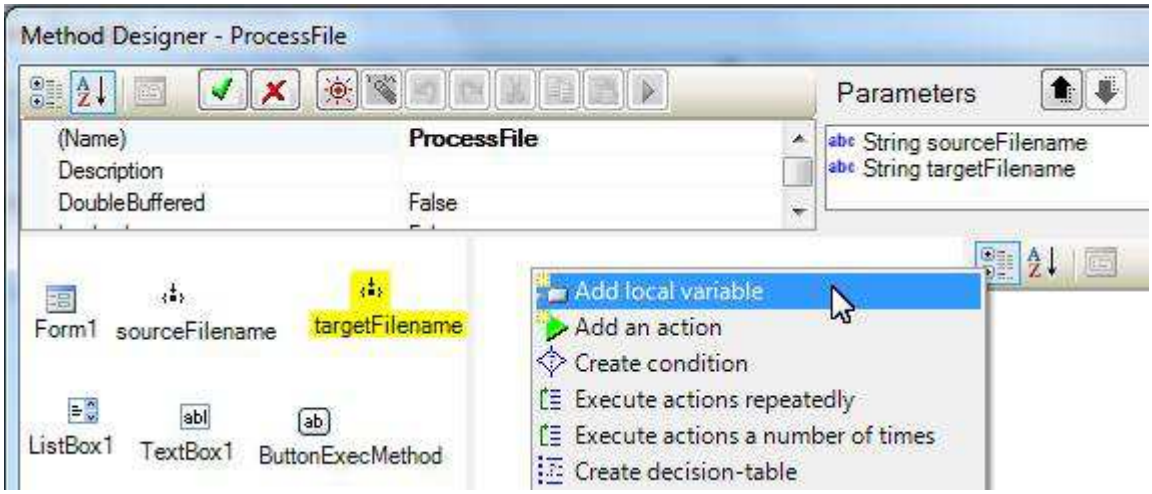
We create a new method to process file. We add two parameters to this method. One parameter is for specifying the name of a file to read from. Another parameter is for specifying the name of a file to write to.

Using parameters gives us the freedom of specifying filenames when creating actions using this method.

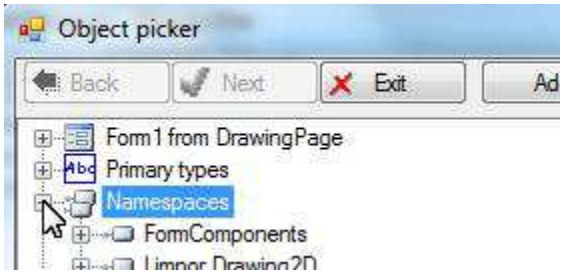


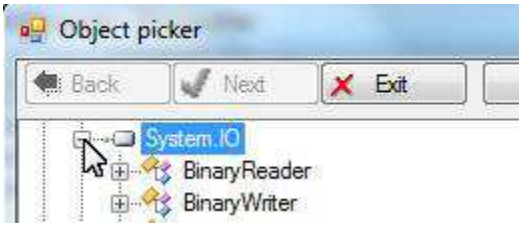
14.1.2 Add StreamReader and StreamWriter

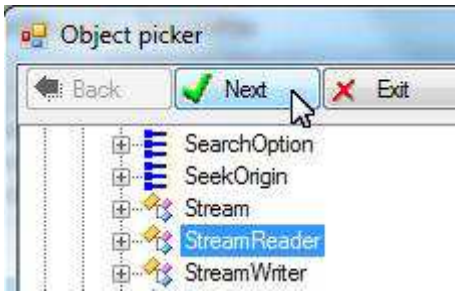
StreamReader class can be used to read a text file line by line. Right-click the Action Pane; choose "Add local variable":



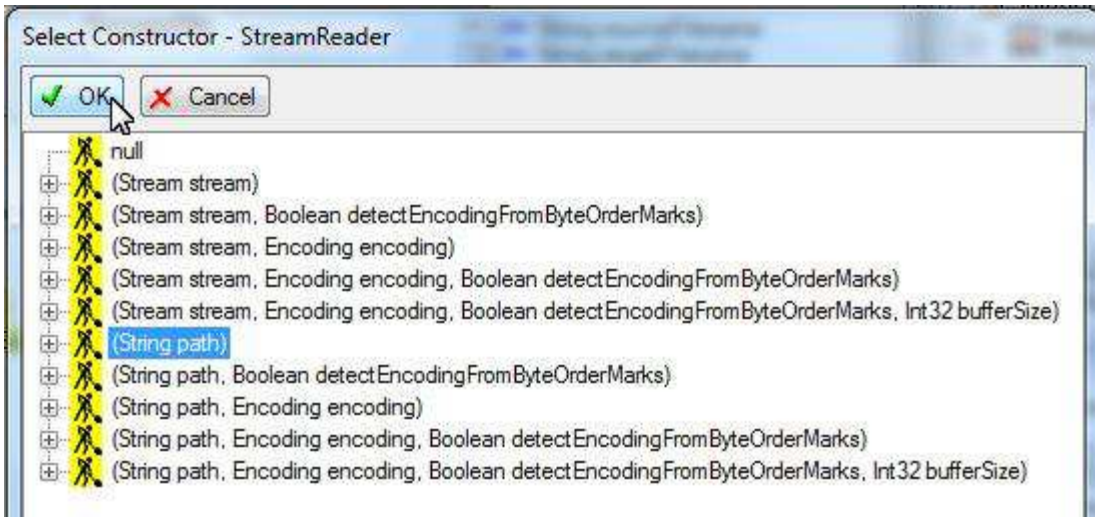
Locate the StreamReader:



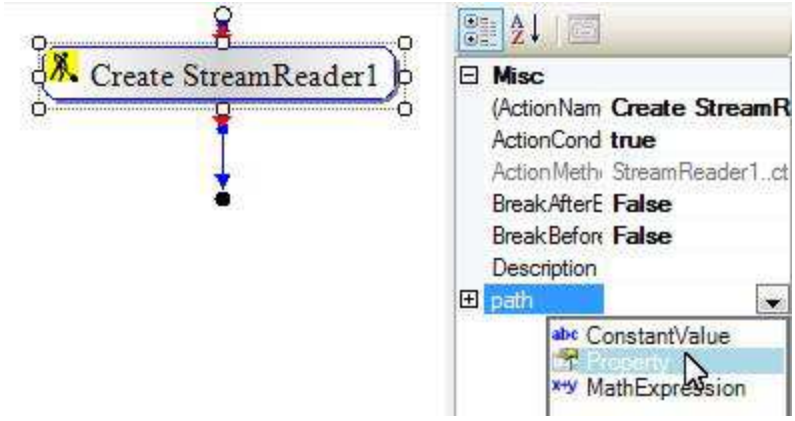


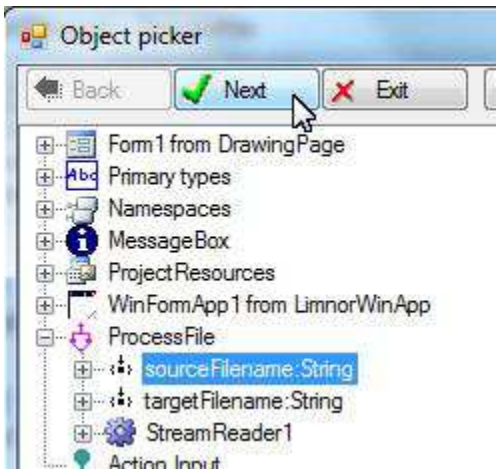


The constructor list appears. Select the simple constructor of (String path)

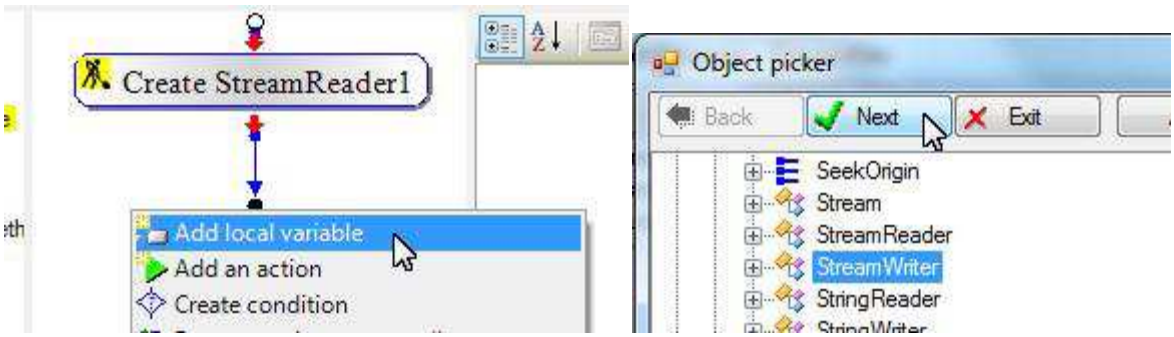


For the path parameter of the constructor, select sourceFilename parameter:

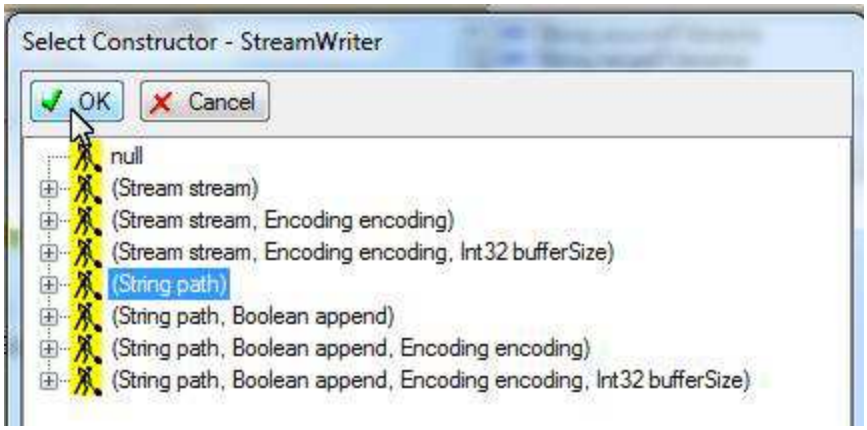




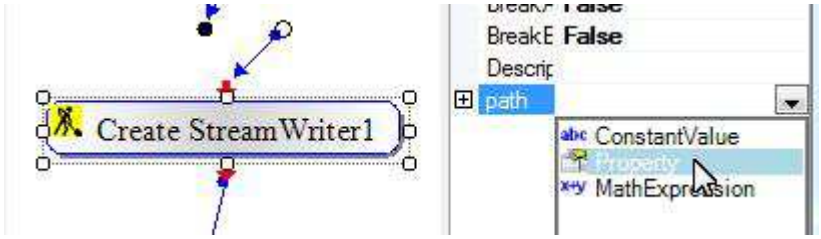
Create another StreamWriter variable:

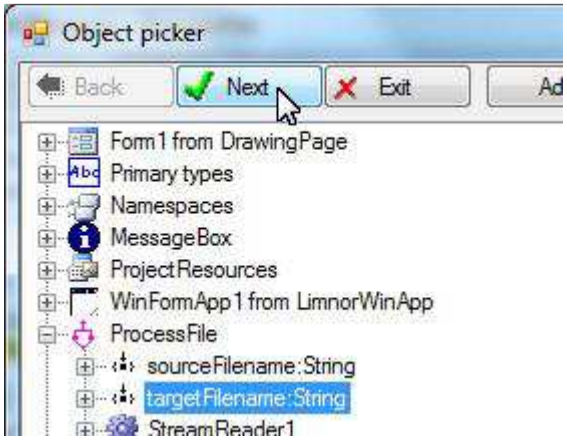


Select constructor (String path) for creating this StreamWriter variable:

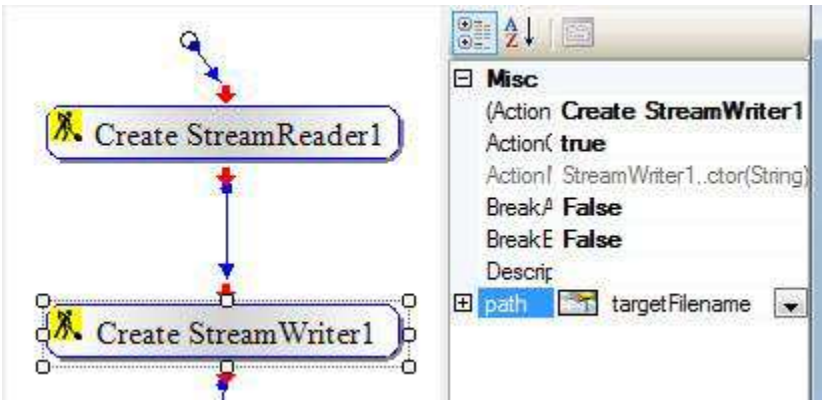


For the path parameter of this constructor, select the targetFilename parameter:





Link the two actions:



14.2 Repeatedly Execute Actions

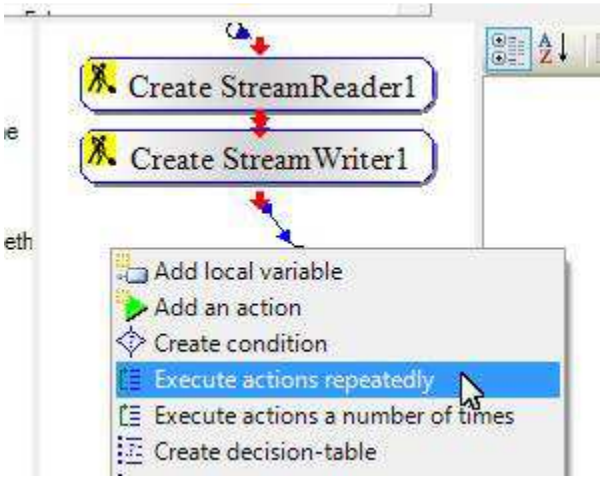
For each line of the source file, we want to do the following:

1. Reformat the text
2. Write to the target file.

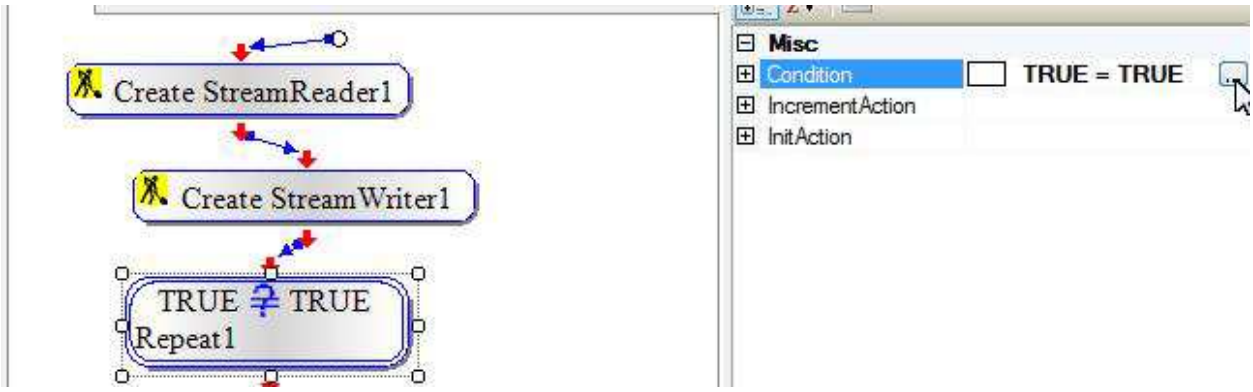
The above actions should be repeated for all lines.

The ReadStream has a EndOfStream property indicating whether the reading reaches the end of file. We may use this property to know when to finish the file processing.

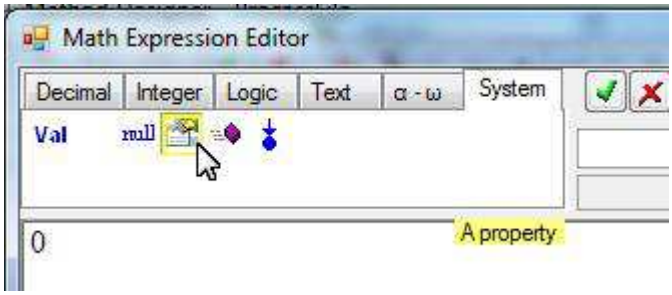
Right-click the Action Pane. Choose “Execute actions repeatedly”:



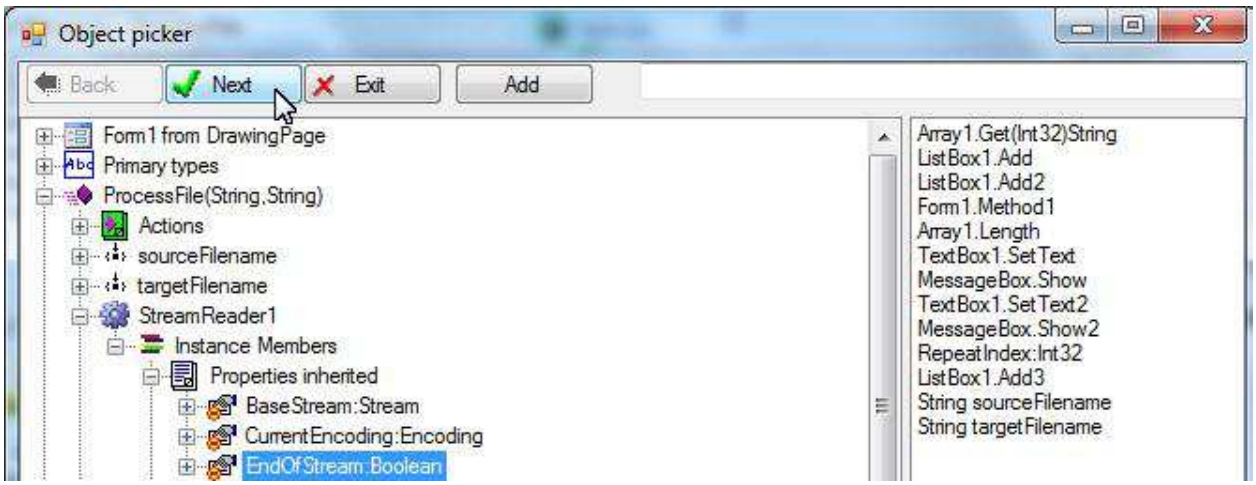
Link the new action to the existing actions. Set its Condition property to specify the conditions for the action to run. As long as the Condition property evaluates True the actions contained inside this action will be executed time and time again.



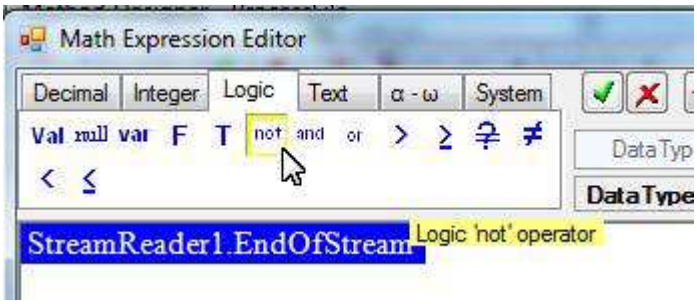
Select the Property icon to use the EndOfStream property:



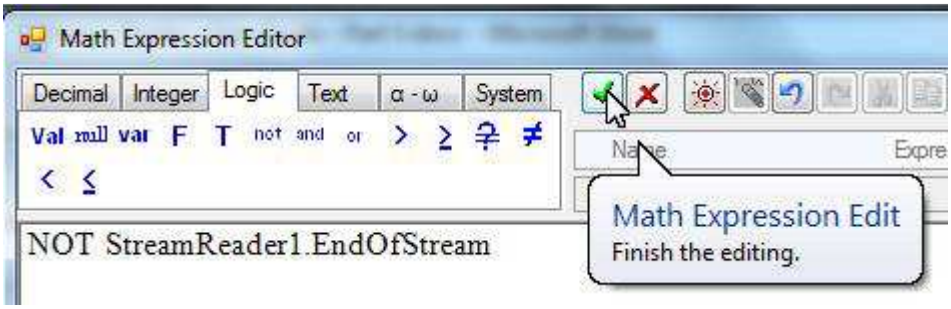
Select the EndOfStream property:

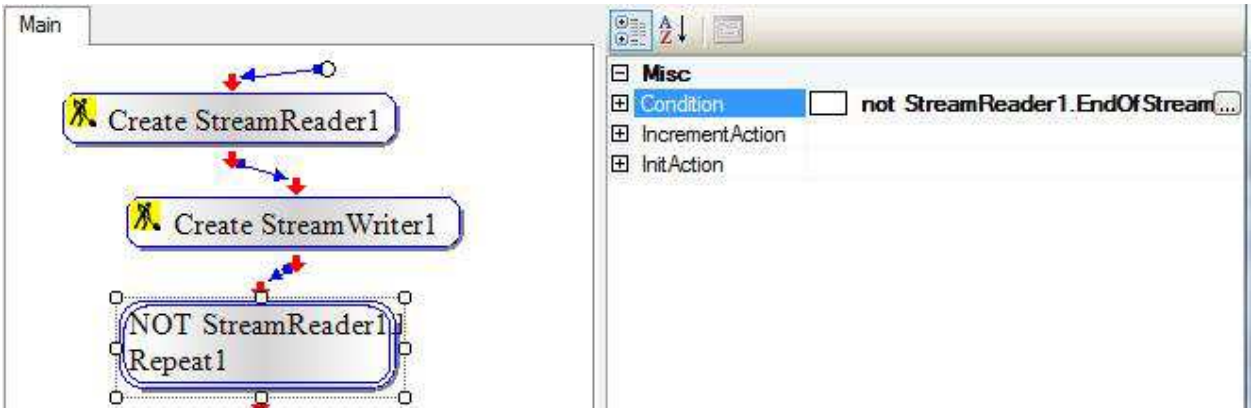


Click not



The expression indicates that the actions will keep running while the end of file is not reached:



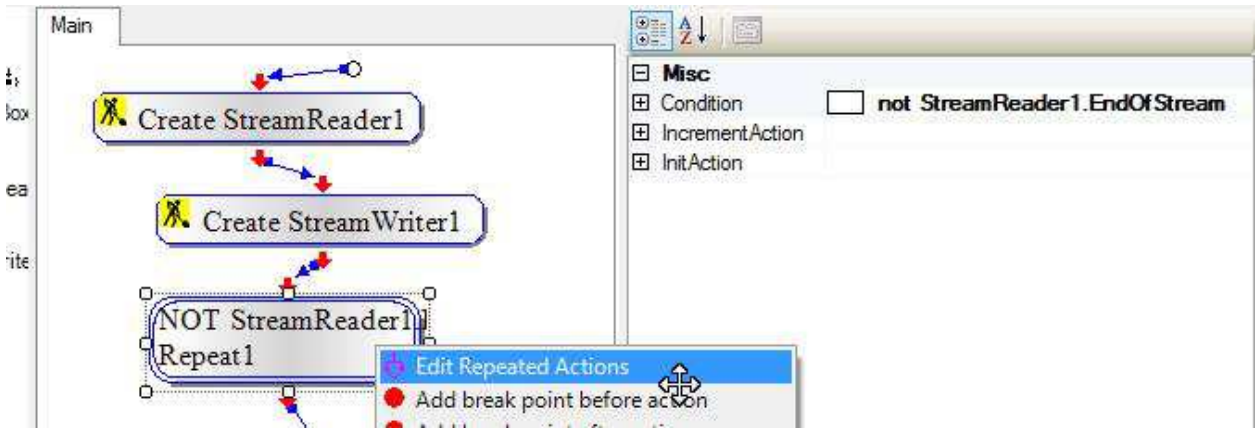


This sample does not use the following properties:

- InitAction
  It is the action to be executed before entering the loop
- IncrementAction
  It is the action to be executed after one repeat.

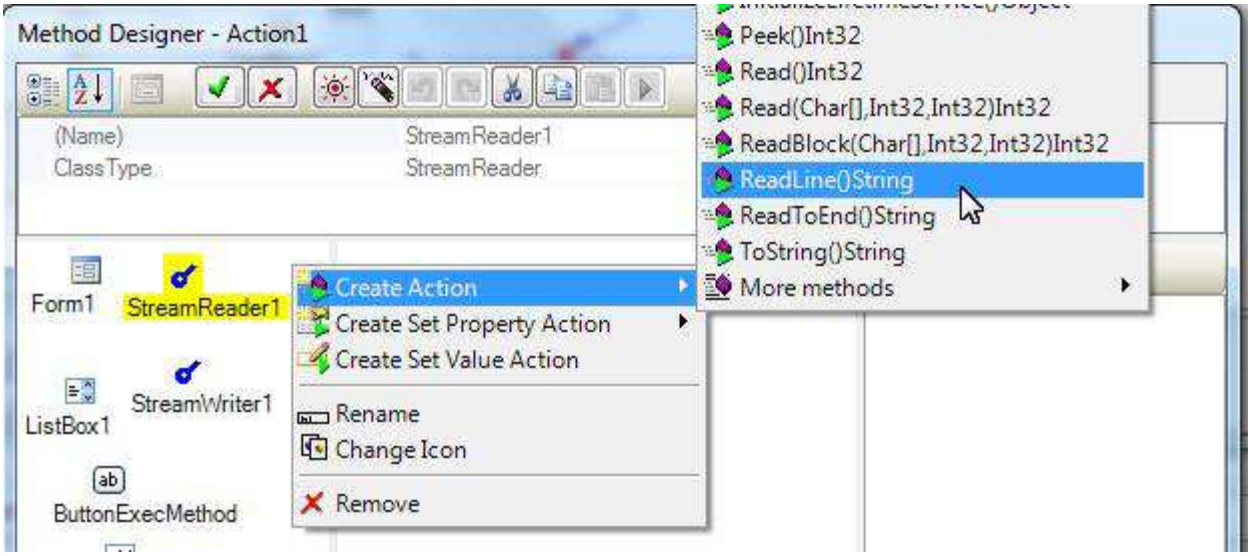
14.3 Add actions to be repeatedly executed

To add actions to be executed, right-click the action, choose “Edit Repeated Actions”:

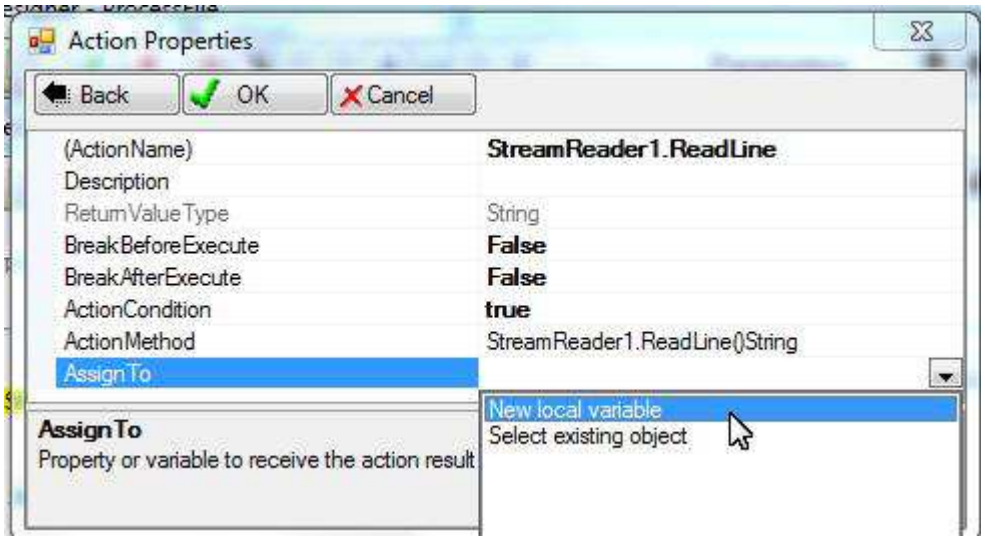


A new Method Editor appears for adding actions to be repeatedly executed.

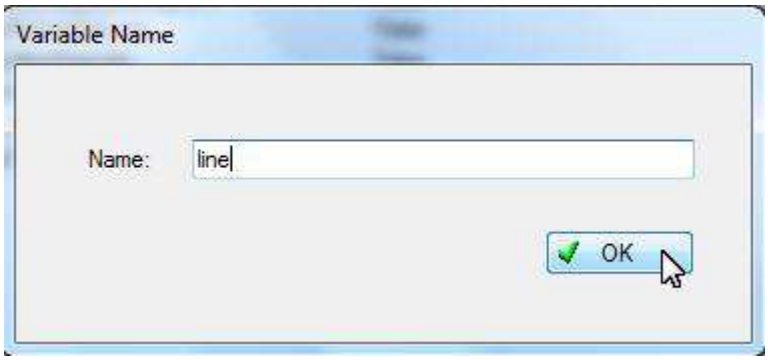
Right-click the StreamReader, choose “Create Action”. Choose “ReadLine”:



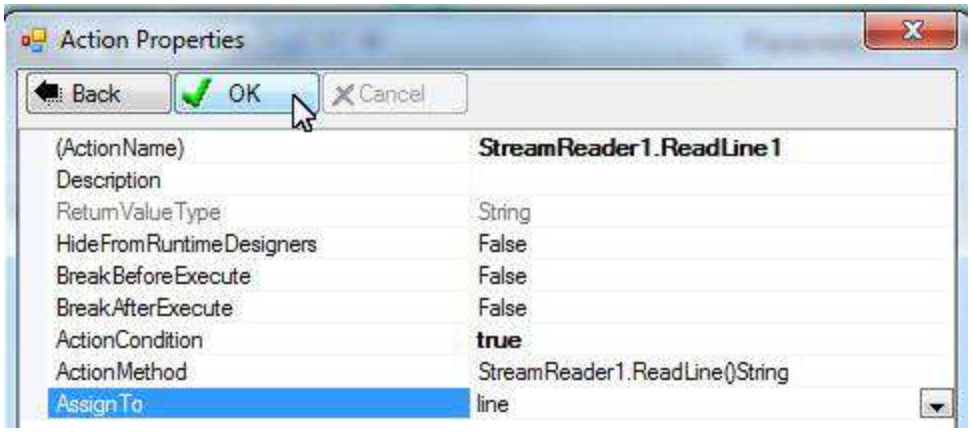
The ReadLine action reads a line of text from the file. We choose “New local variable” for “AssignTo” property of this action. The line of text will be assigned to the new local variable:



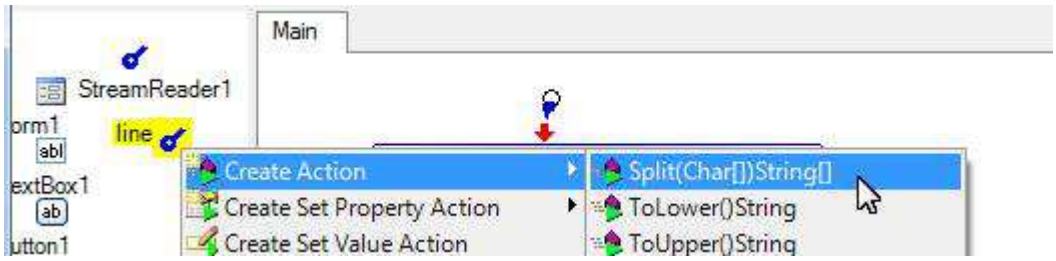
Name the new variable “line”:



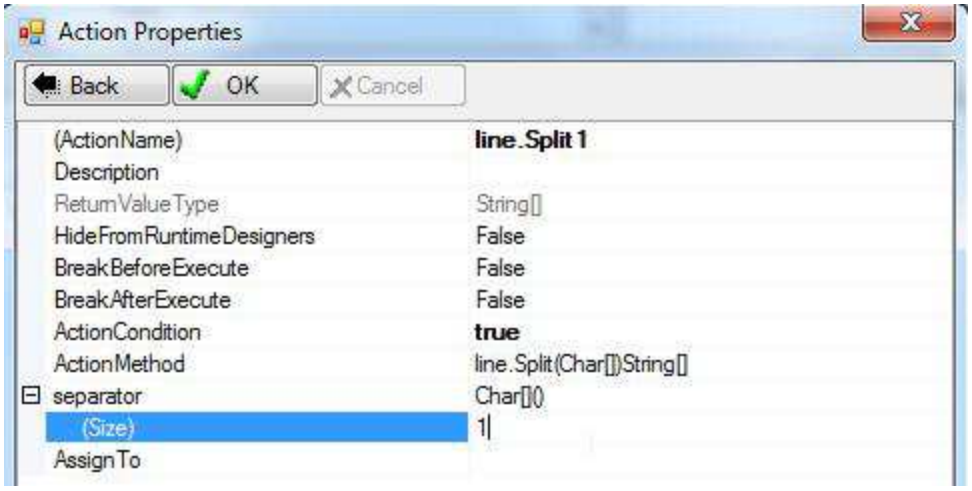
Click OK to finish creating this action:



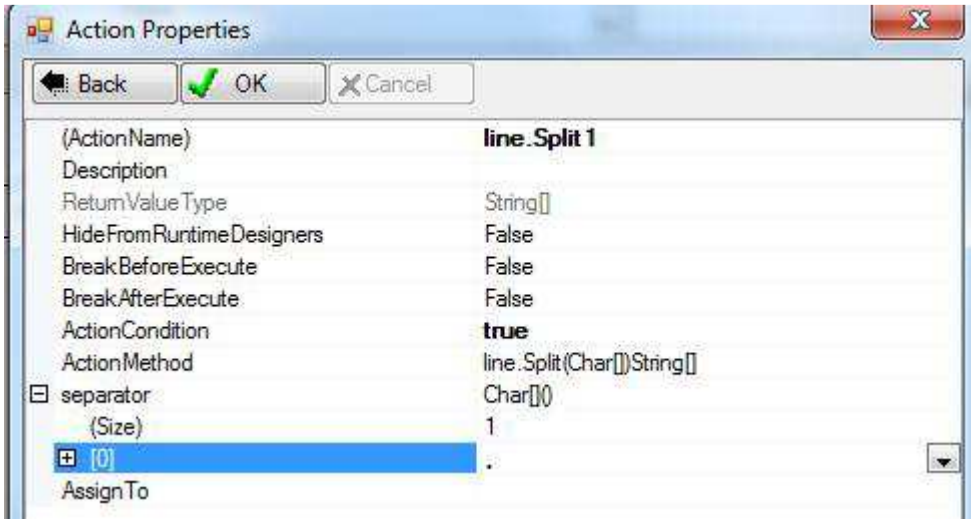
The action is created. The variable, line, is also created, which is a line of text read from the file. Right-click “line”; select “Create Action”; choose “Split” method:



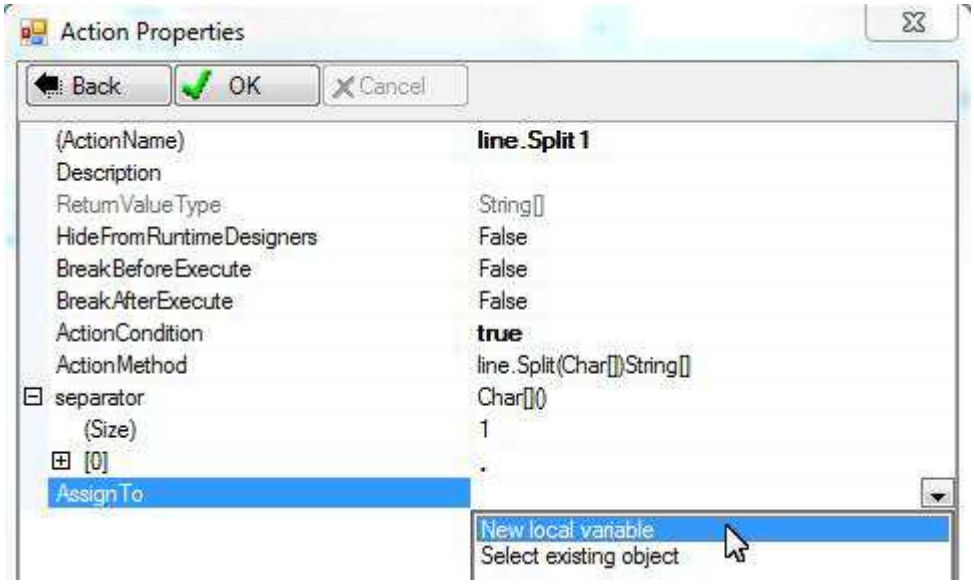
Enter a character array as separators for splitting the string:



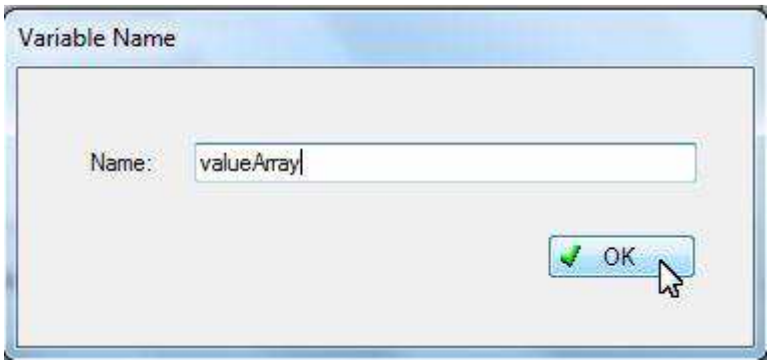
The array has one element which is a comma:



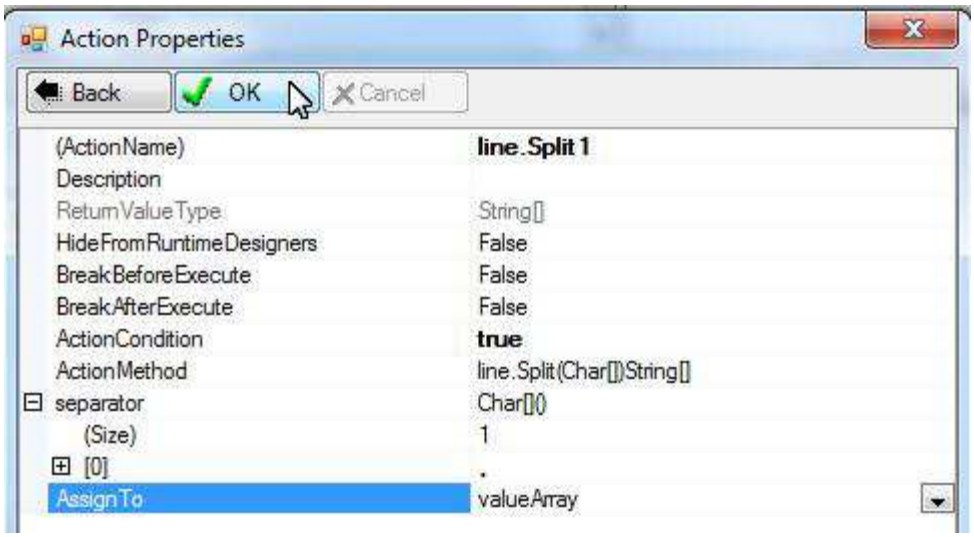
Select “New local variable” for “AssignTo” property:



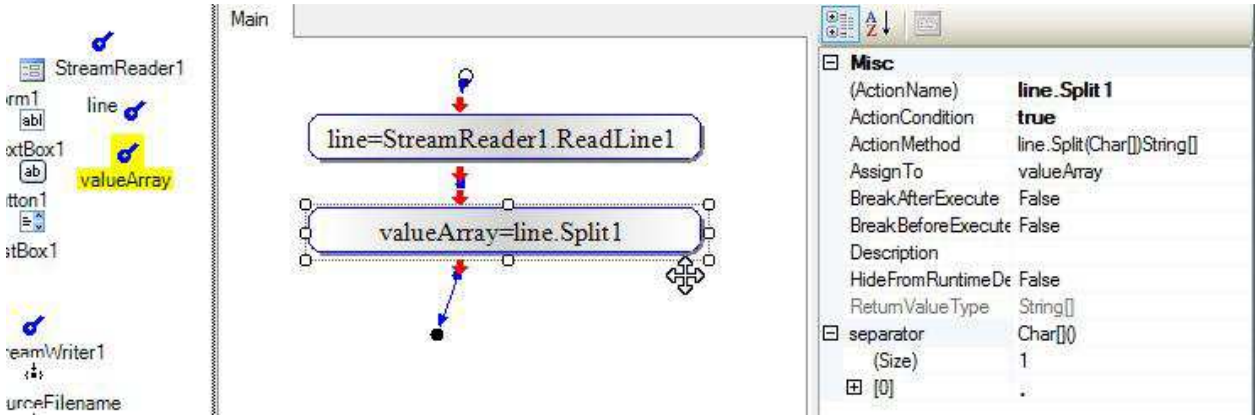
Name the new variable valueArray:



Click OK to finish creating this action:



The new action appears in the Action Pane. We link it with the first action. We can see that an array variable, valueArray, is also created.



Suppose the line of text from the file is

03/21/2001 09:01:33,231,test1

Then it is the contents of variable “line”. “valueArray” is a string array with 3 items:

03/21/2001 09:01:33
231
test1

Now we are supposed to convert the first array item from

{month}/{day}/{year} {hour}:{minute}:{second}

To

{year}{month}{day}

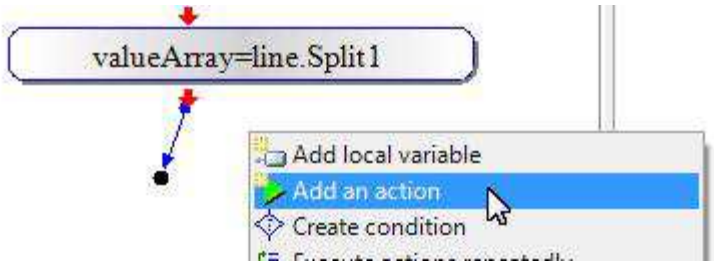
For the sample line should be converted to:

20010321

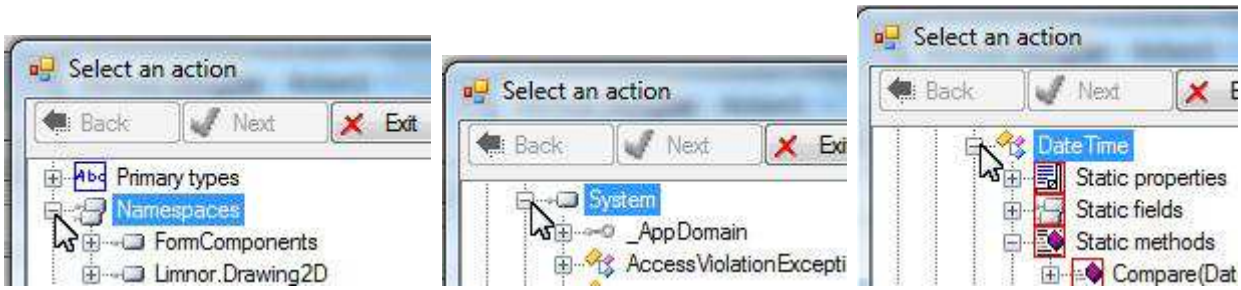
Suppose {month}/{day}/{year} {hour}:{minute}:{second} is the United States date time formatting in the computer. We may use culture “en-US” to parse the first array item into a DateTime value. Then use formatting string “yyyyMMdd” to convert the date time value into the string we want.

Let’s create actions to do the above two conversions.

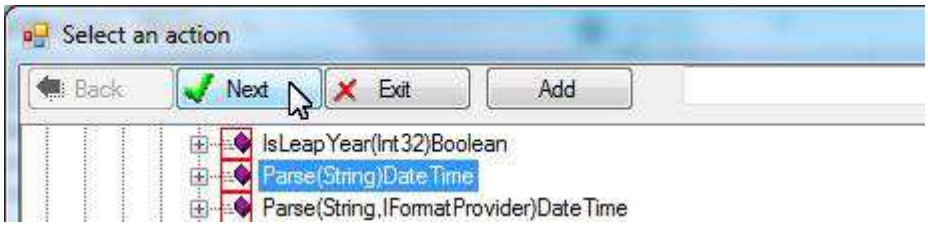
Right-click the Action Pane; choose “Add action”:



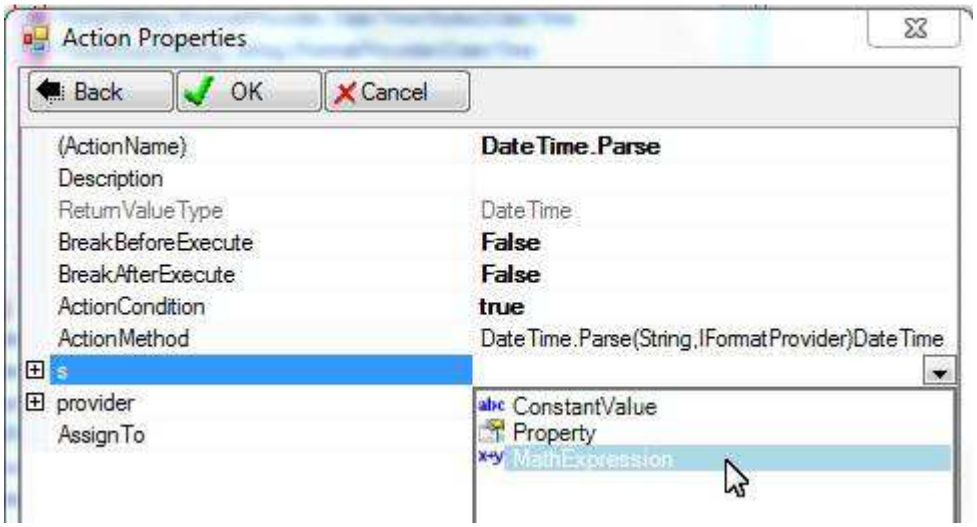
Find the DateTime class:



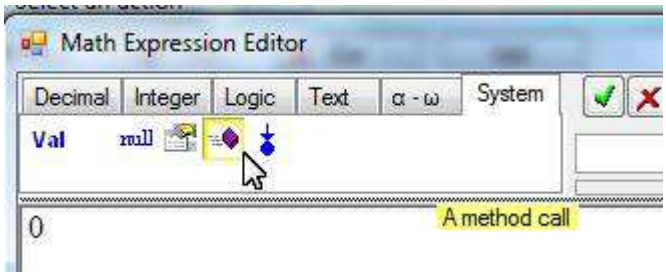
Select its Parse method and click “Next”:



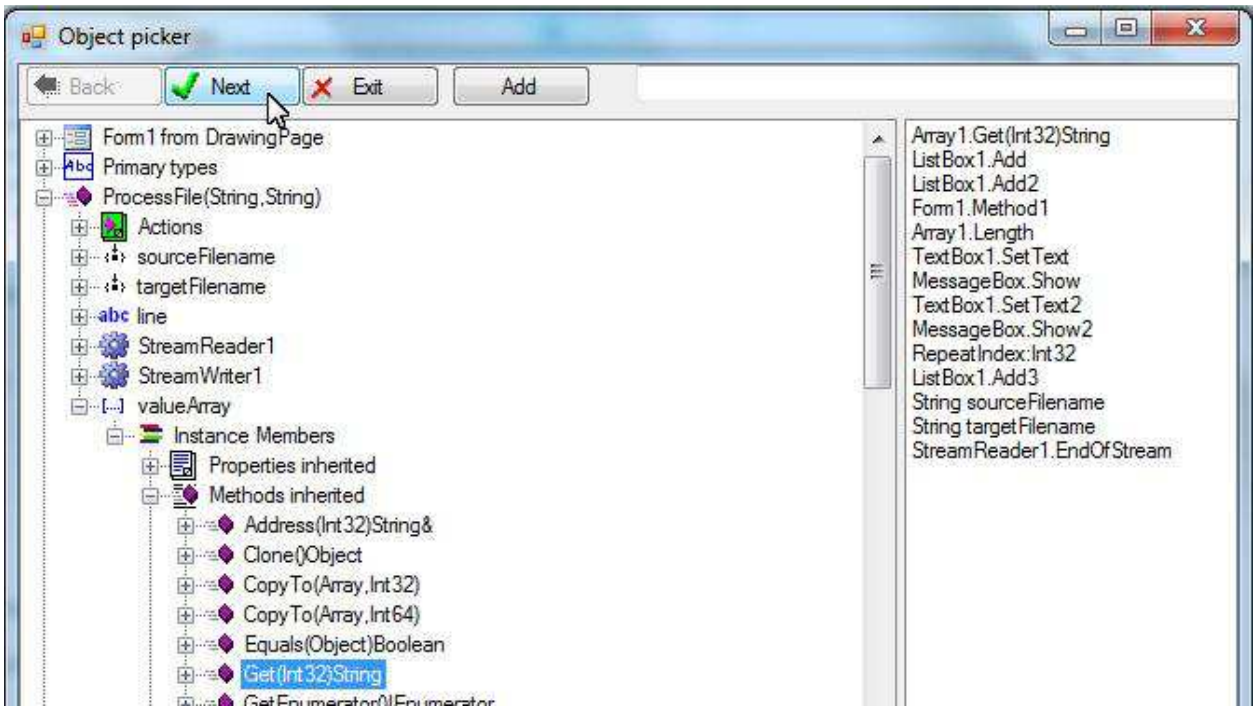
For the parameter s, choose Math Expression to get the first array item:



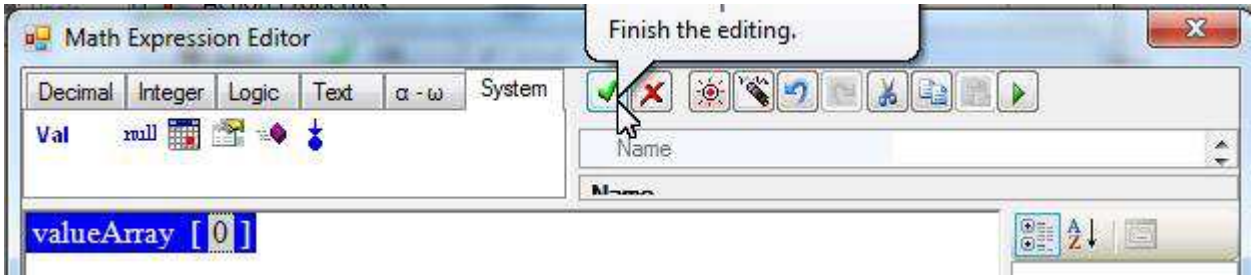
Click the Method icon:



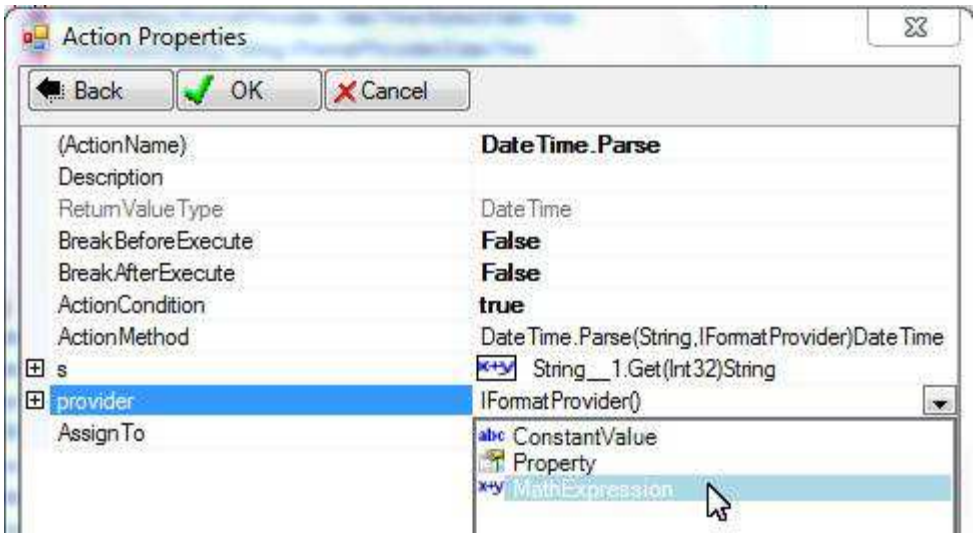
Select the Get method:



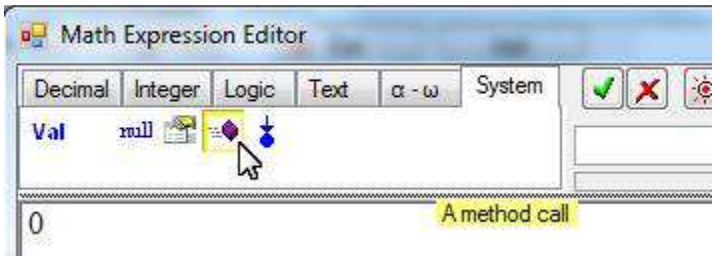
0 indicates the first array item:



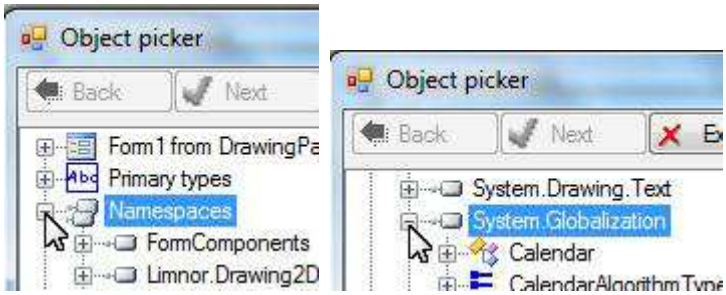
For provider, select Math Expression to get “en-US” culture:



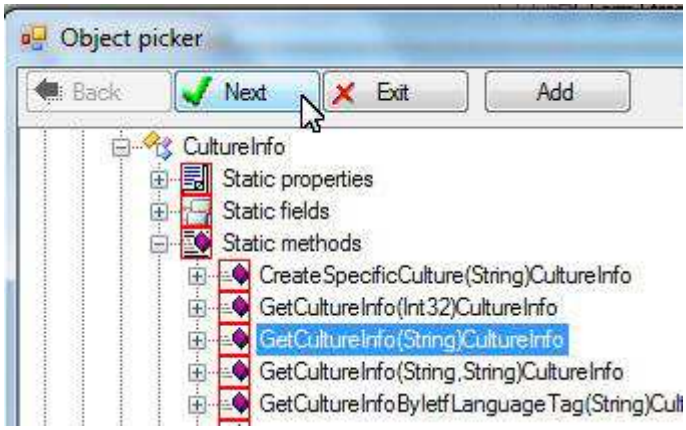
Click the method icon:



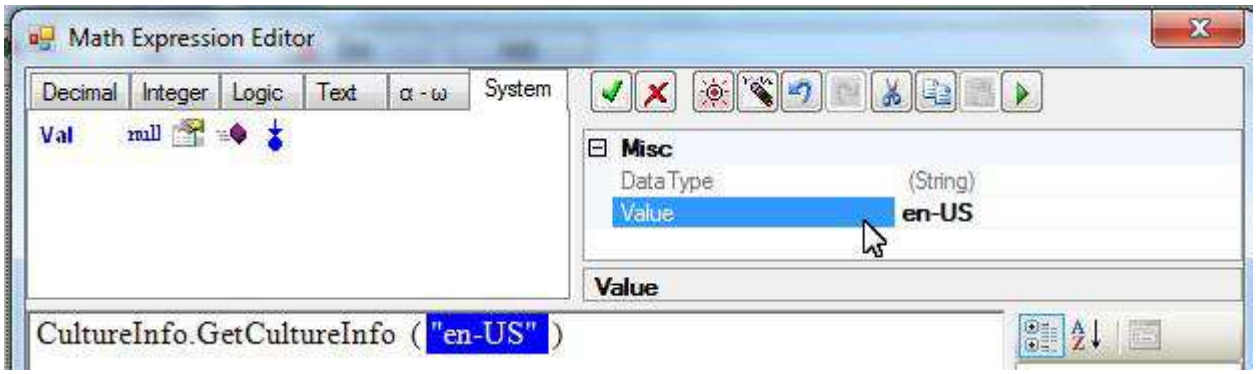
Select System.Globalization:



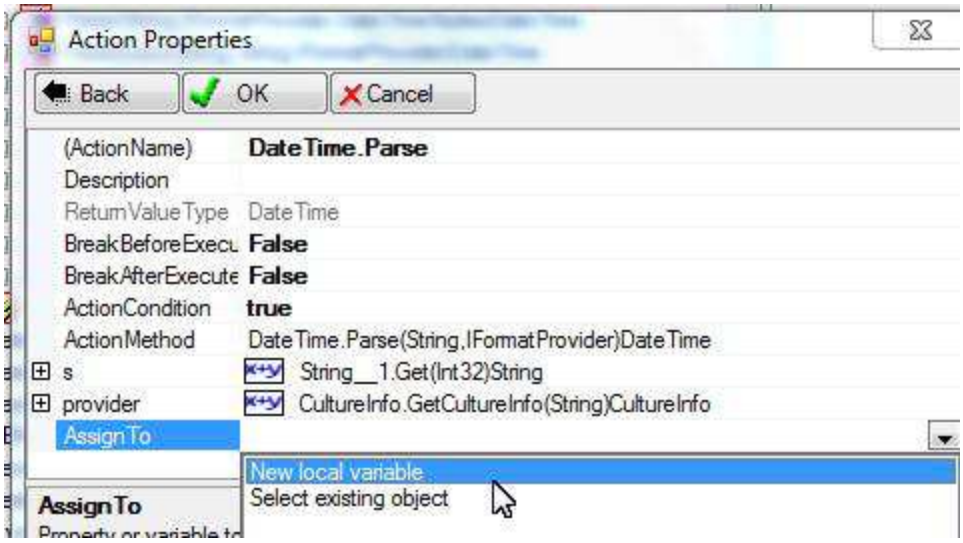
Select GetCultureInfo method of the CultureInfo class:



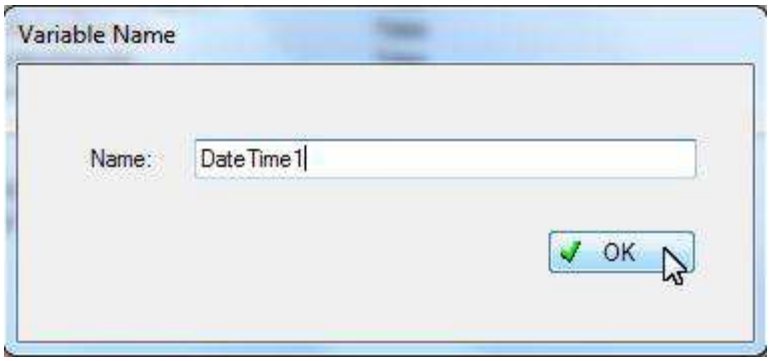
Enter “en-US” as the culture name:



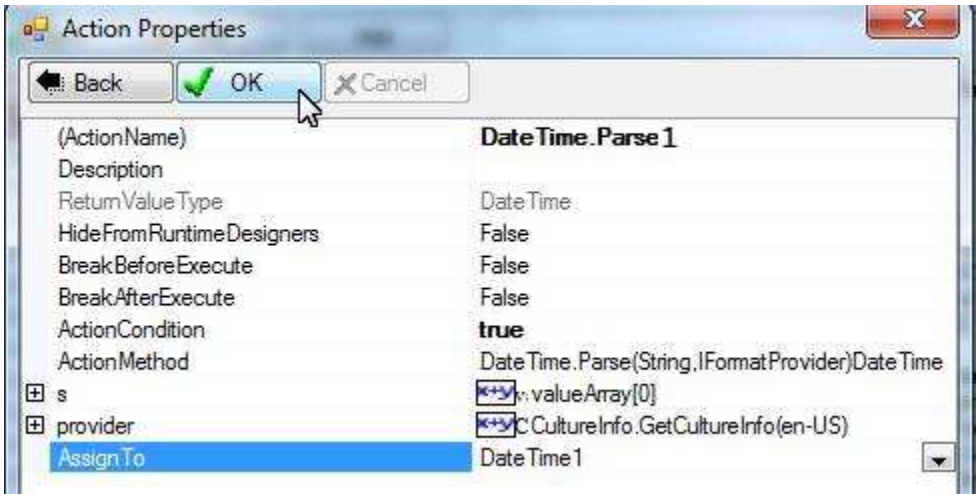
Choose “new local variable” for “AssignTo”:



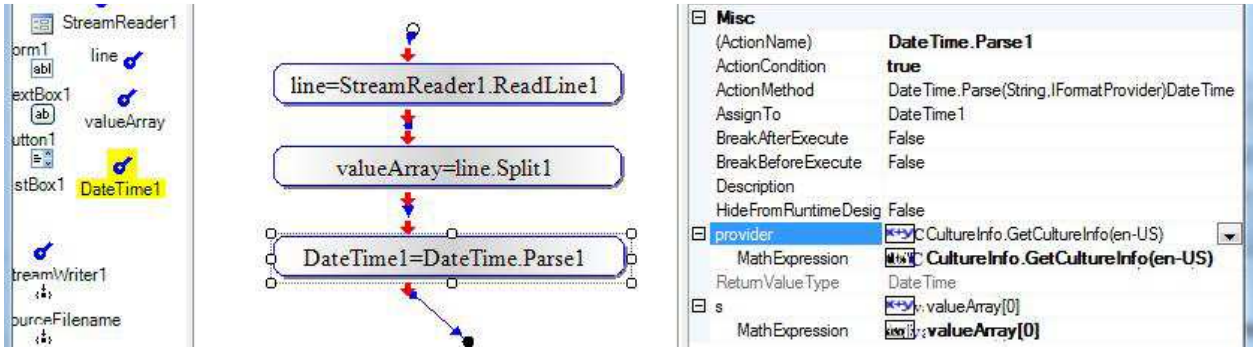
Give a variable name:



Click OK to finish creating this action:

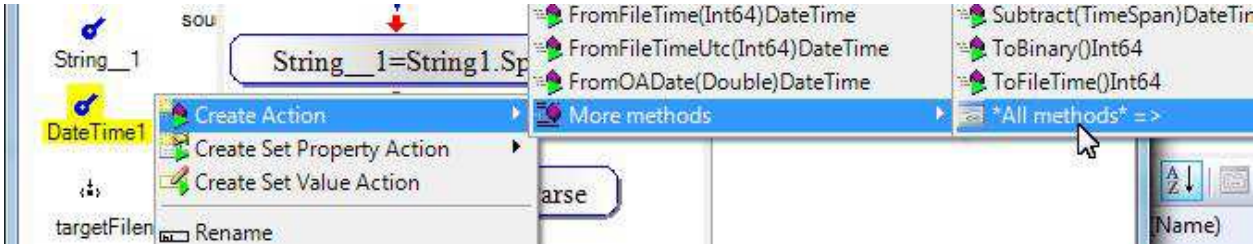


The action appears in the Action Pane. Link the action. The new variable, DateTime1, also appears in the Object Pane:

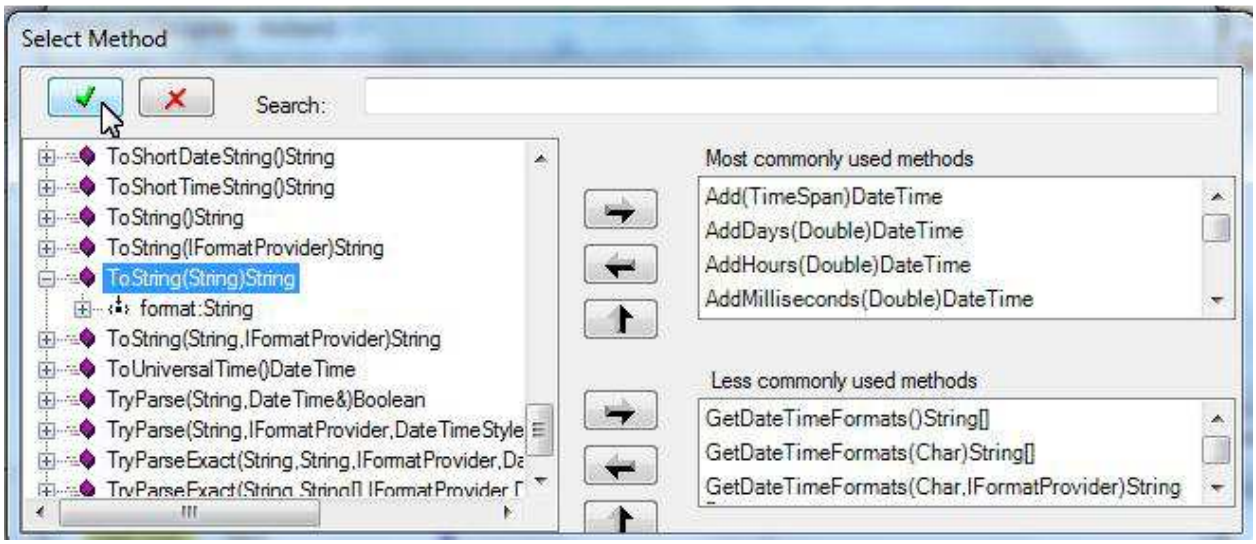


Right-click DateTime1 to make an action to convert it to a string of format yyyyMMdd:

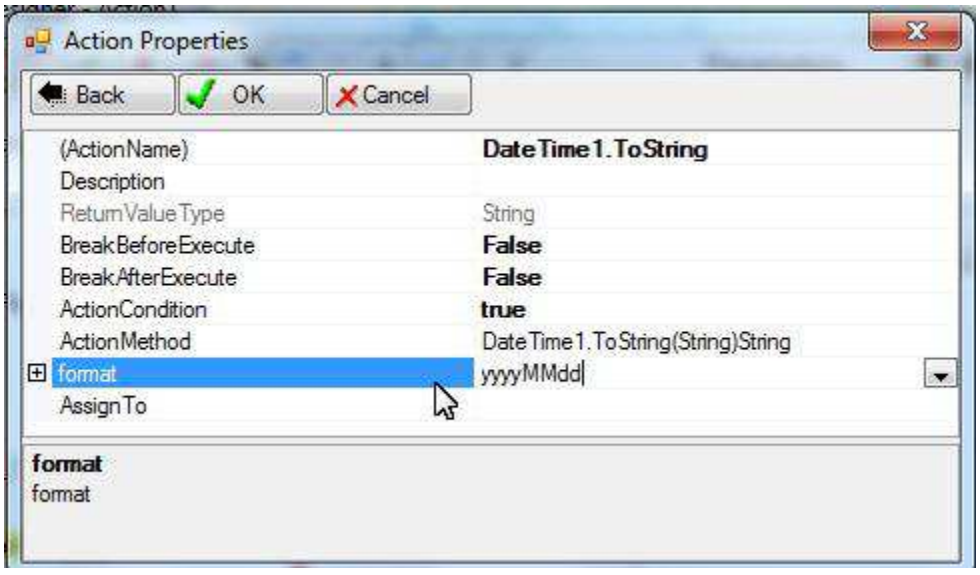
We want to use method ToString. It is not in the list. Choose “All methods” to find it:



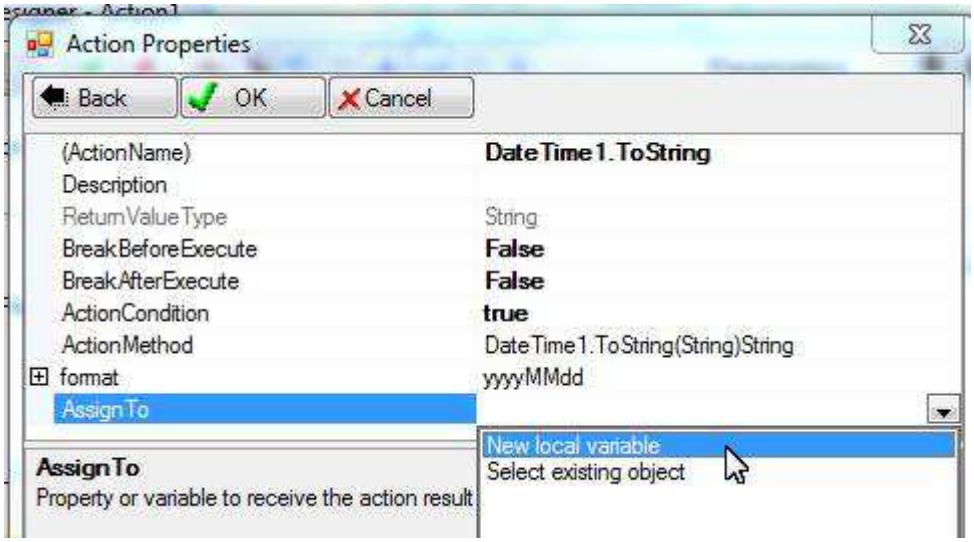
Choose the ToString method:



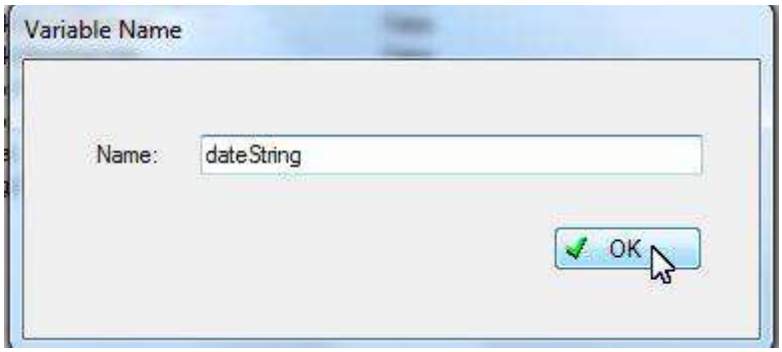
Enter yyyyMMdd as the format:



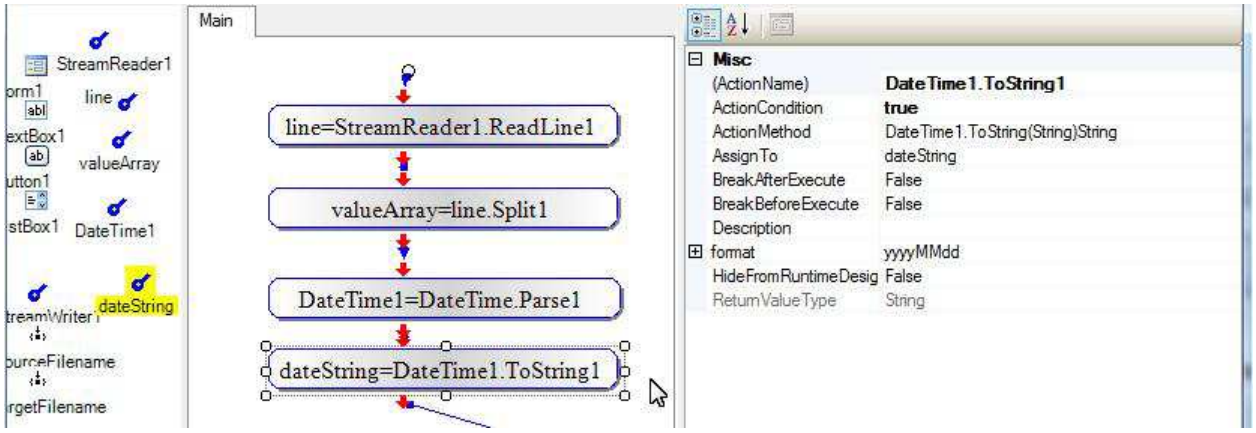
Choose “New local variable” for “AssignTo”:



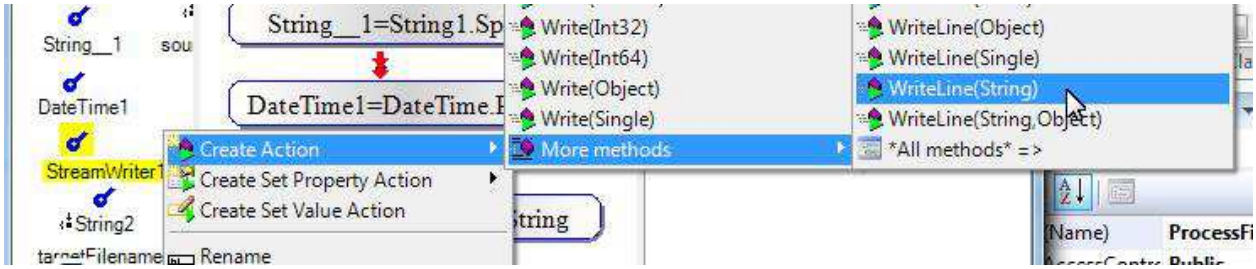
Give a variable name:



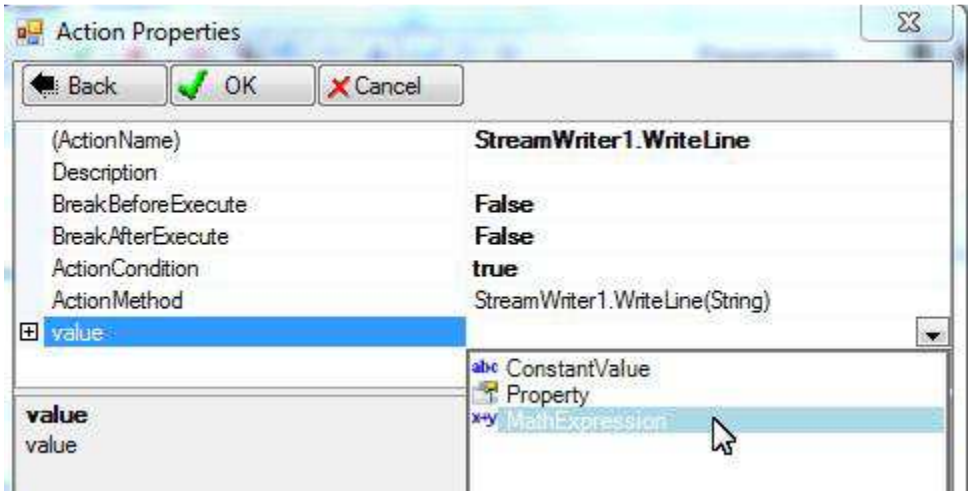
The new action appears in the Action Pane. Link it to others. The new local variable, dateString, also appears in the Object Pane:



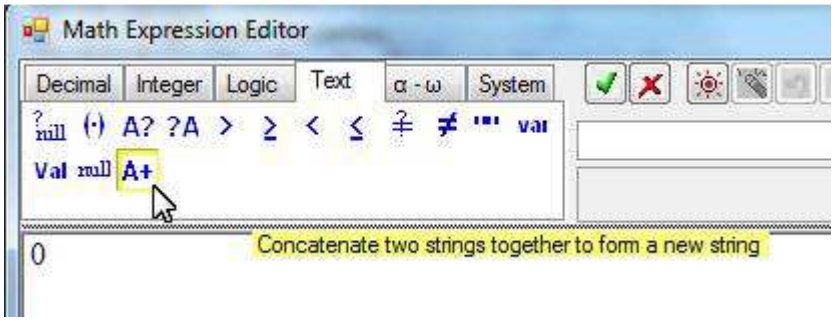
Now we may write the result to file. Right-click StreamWriter1; choose “Create Action”; choose “WriteLine”:



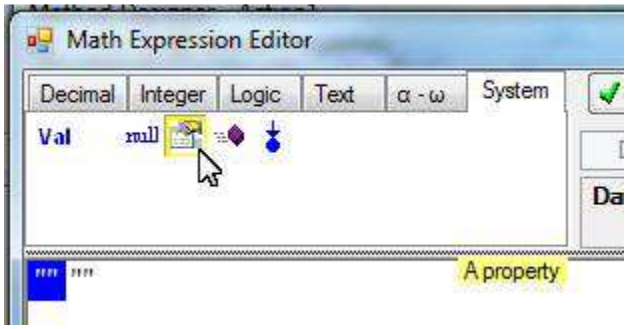
Select Math Expression to form an output line:



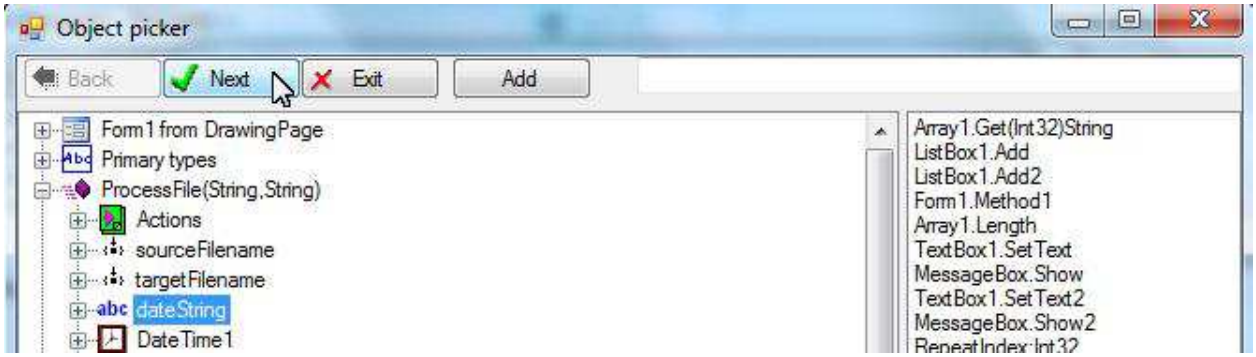
Click A+ to use string concatenation to form a line of text:



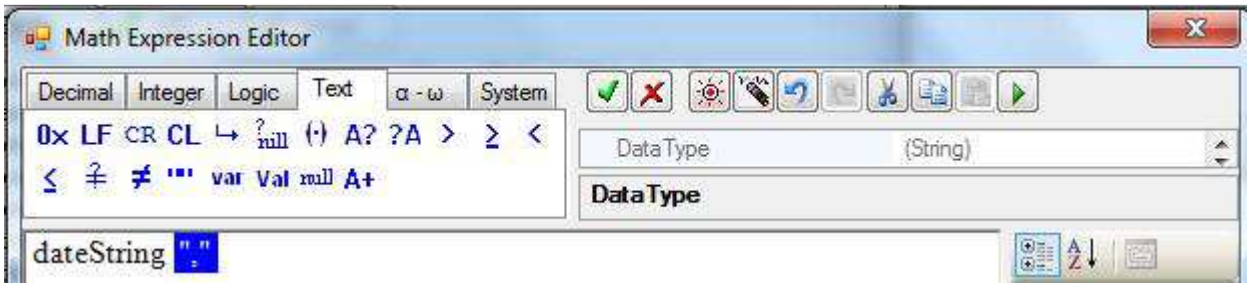
Select the first element; click the Property icon:



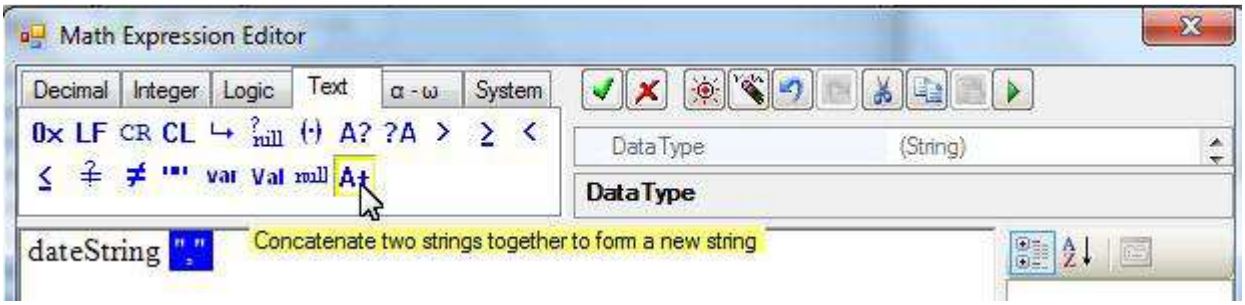
Select dateString:



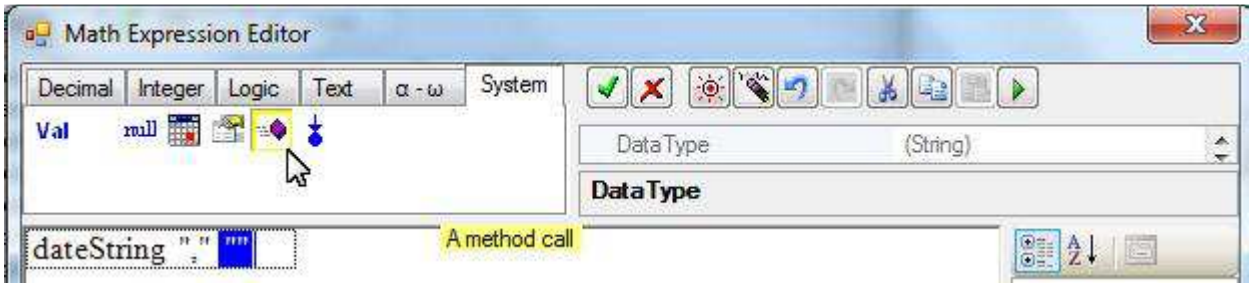
Select the second element, type comma:



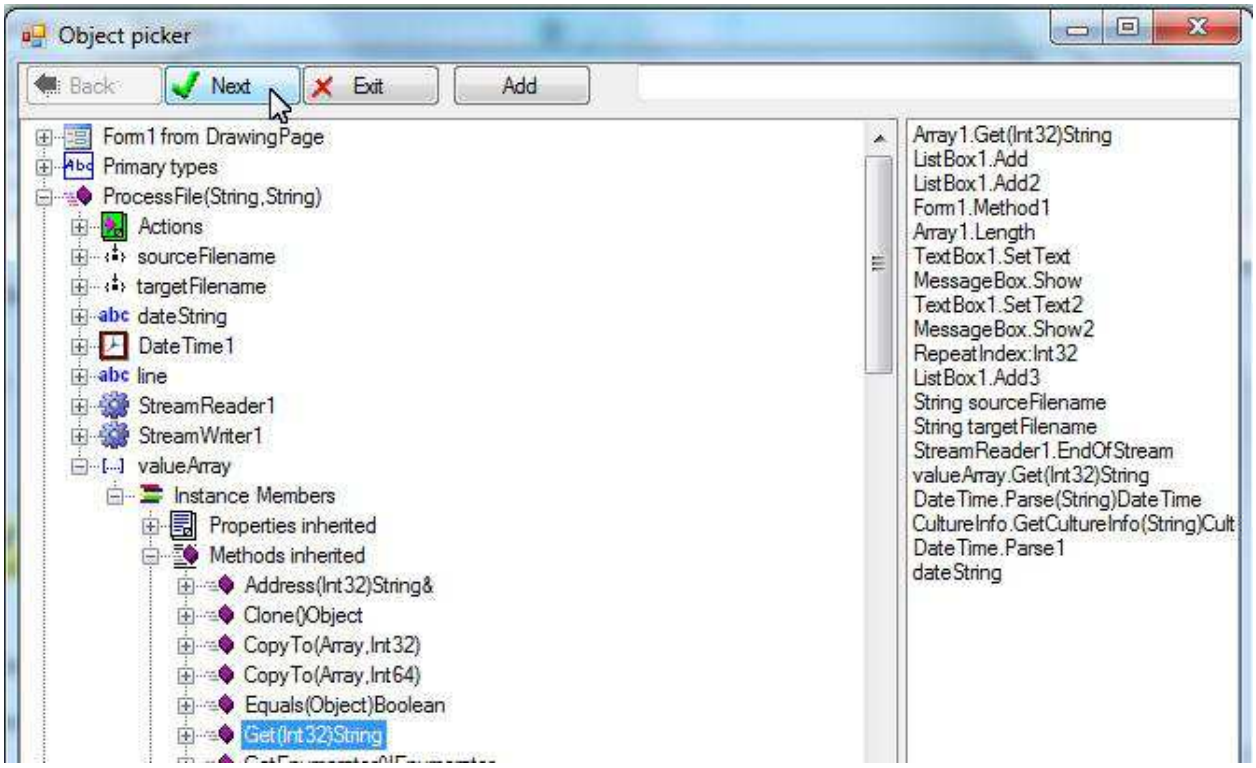
With the last element selected click A+ again:



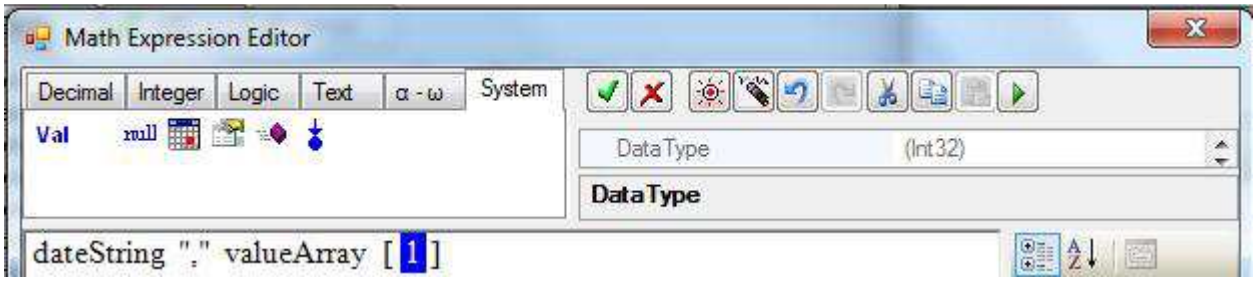
With the last element selected, click the Method icon:



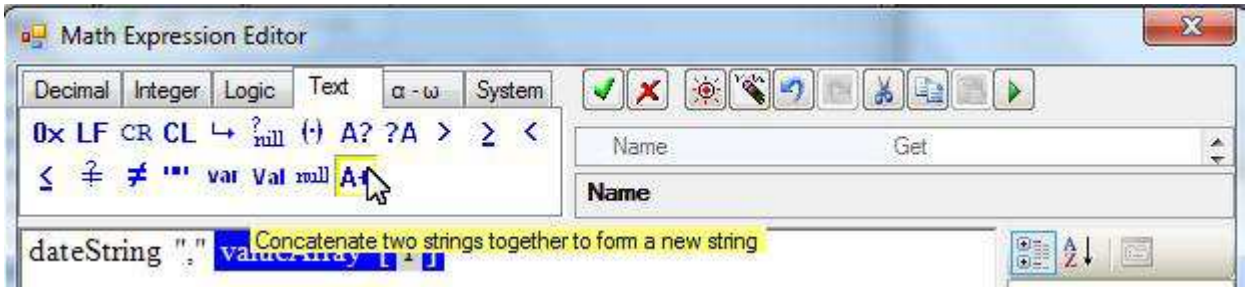
Select the Get method of the valueArray:



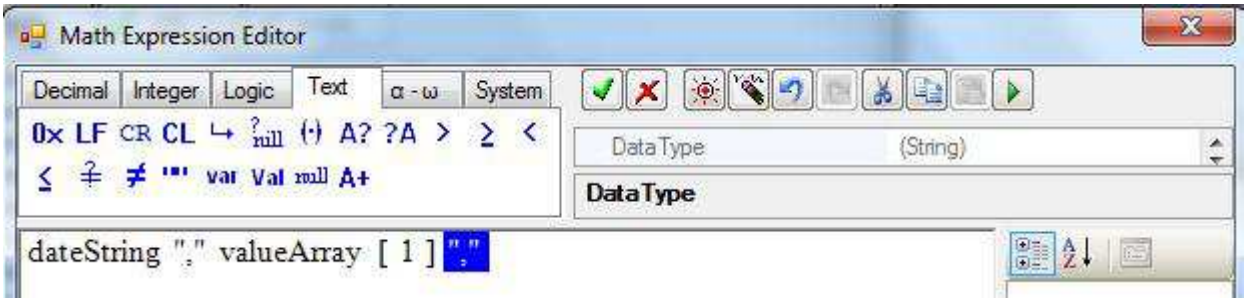
Type 1 for getting the second array item:



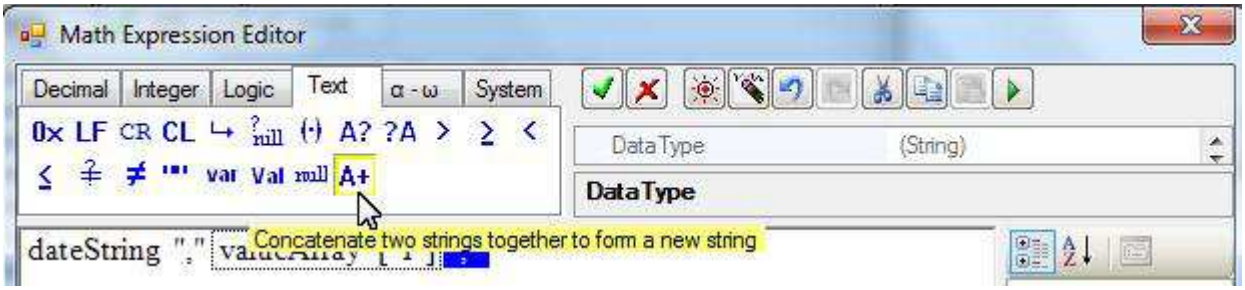
With the valueArray1[1] selected, click A+ again:



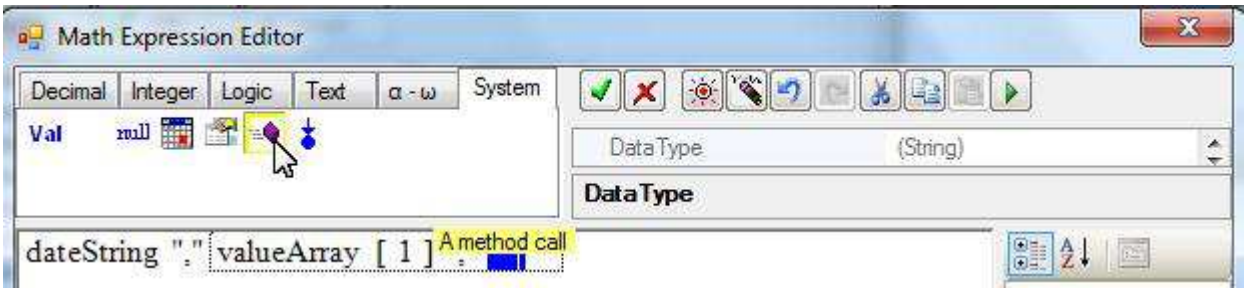
Select the new last element, type comma:



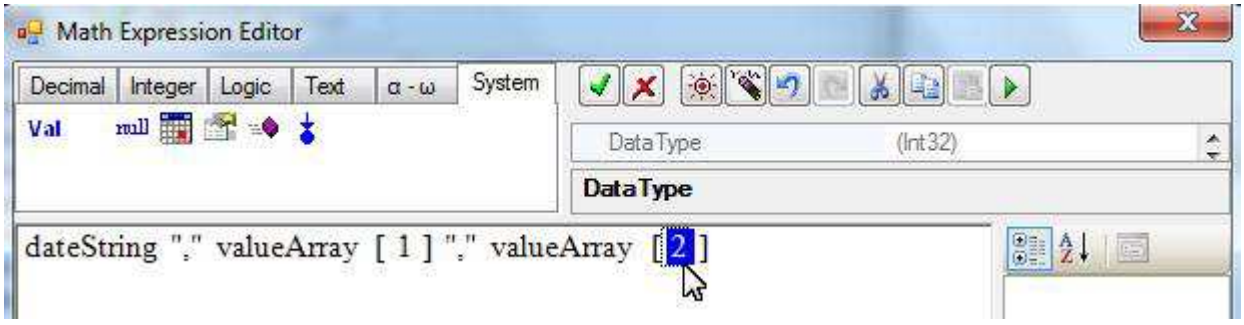
With the last comma selected click A+ again:



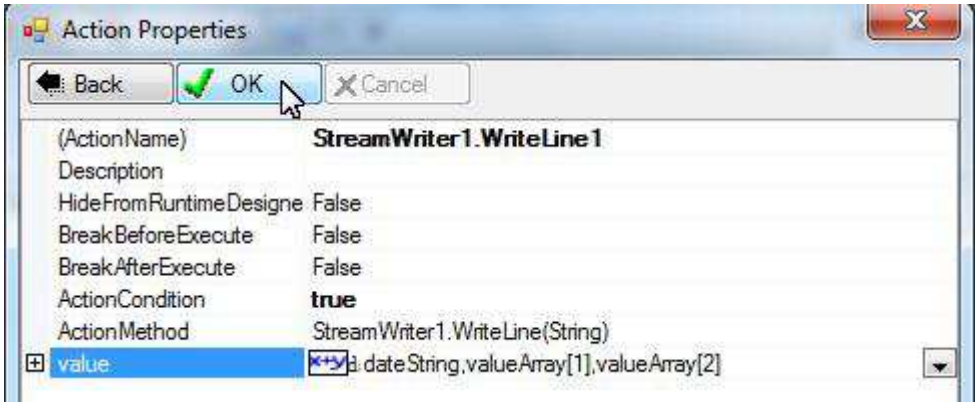
Select the last element, click the Method icon:



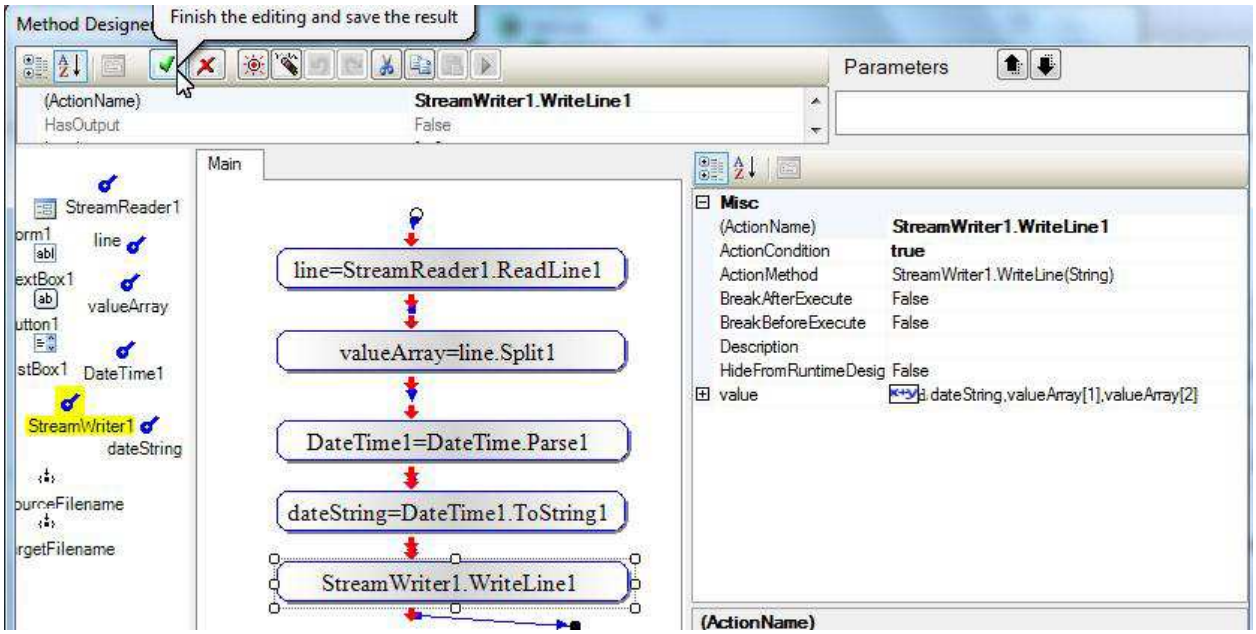
Select the Get method again. Type 2 for the parameter to get the 3rd array item



This is the line to be written to the output file.



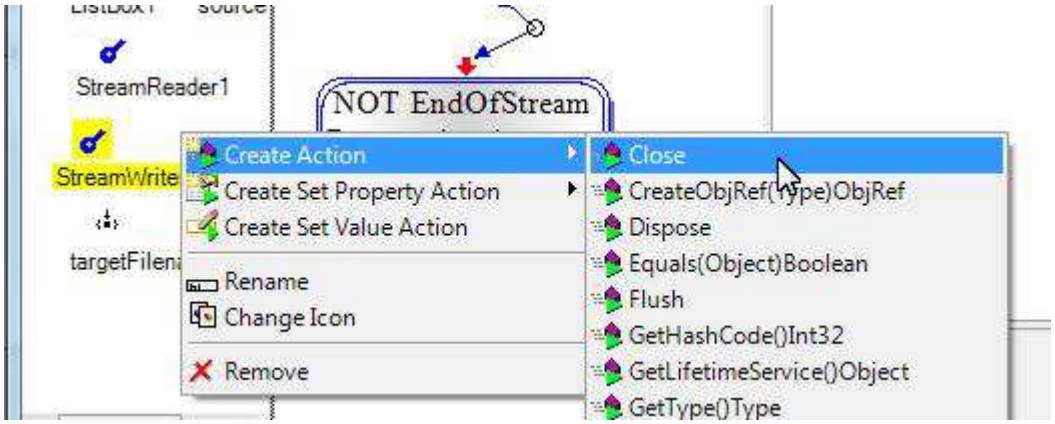
The new action appears in the Action Pane. Link it to others. We are done making the actions for processing one line of the file:



Click  to finish the editing. We are back to the previous Method Editor.

14.4 Close Files

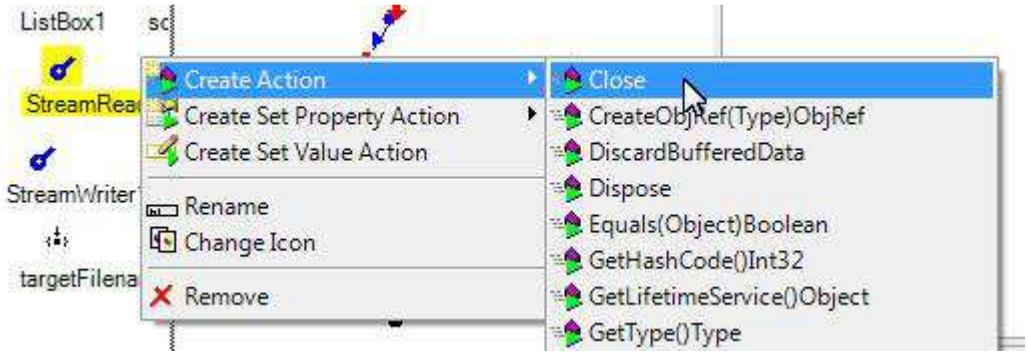
Right-click StreamWriter1, choose “Create Action”. Choose “Close” method:



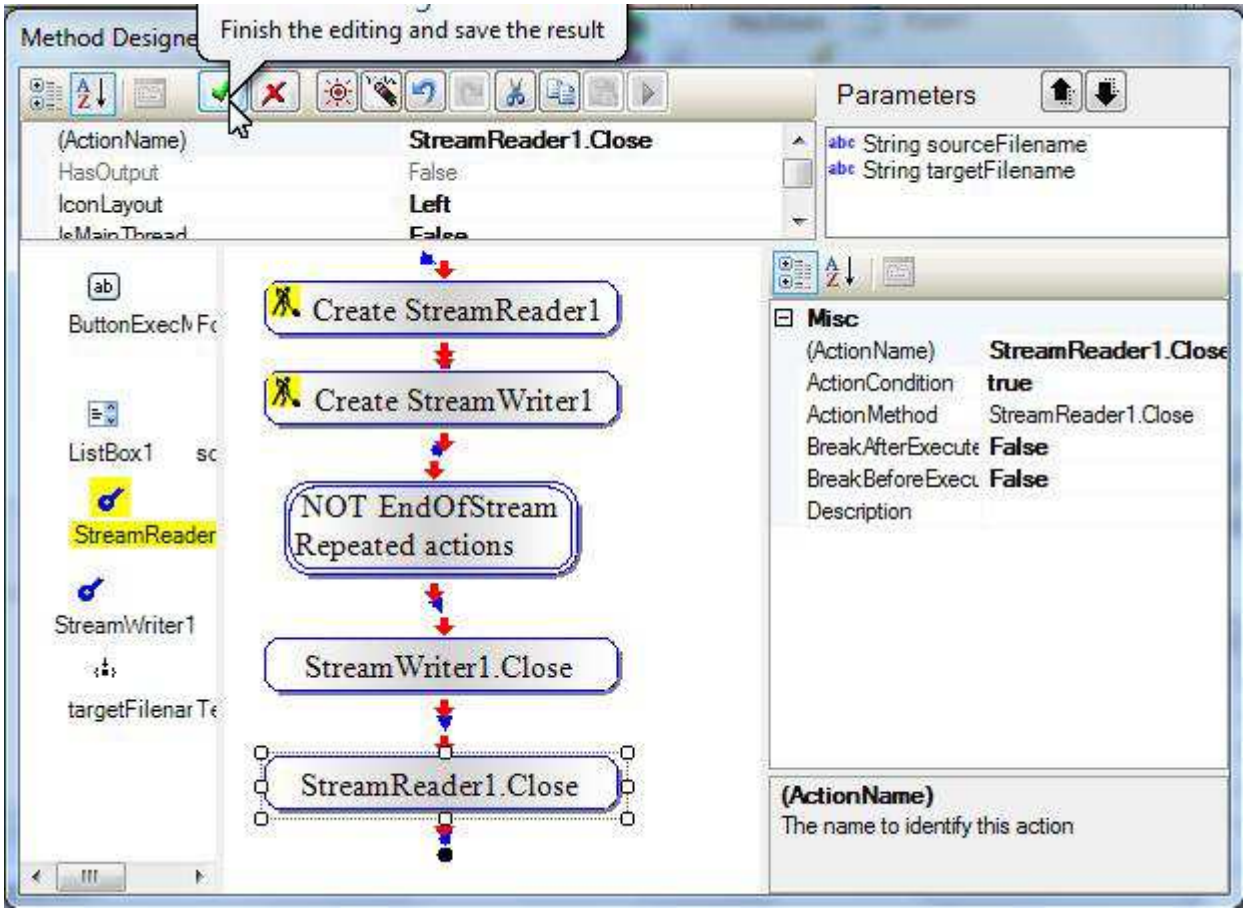
Click OK:

Link the new action.

Also create a Close action for StreamReader1:



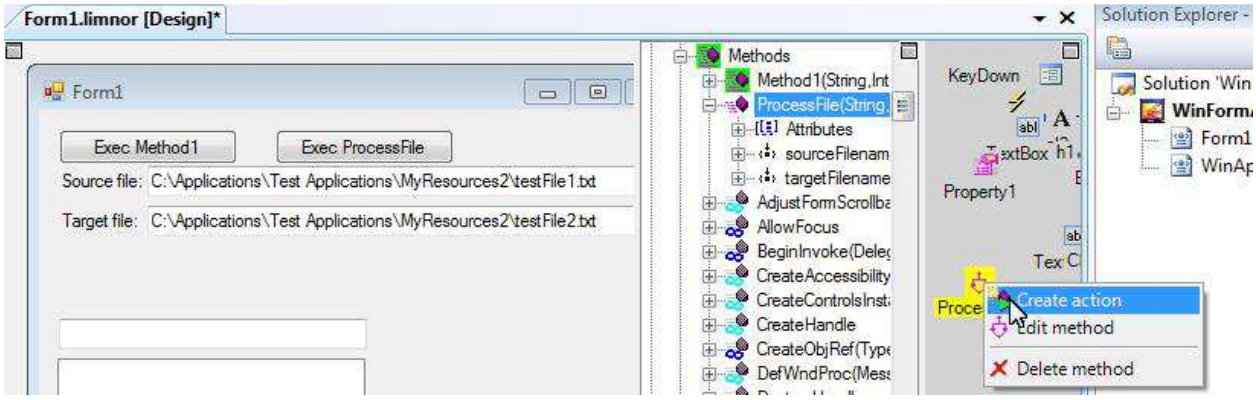
This is the final result of this method:



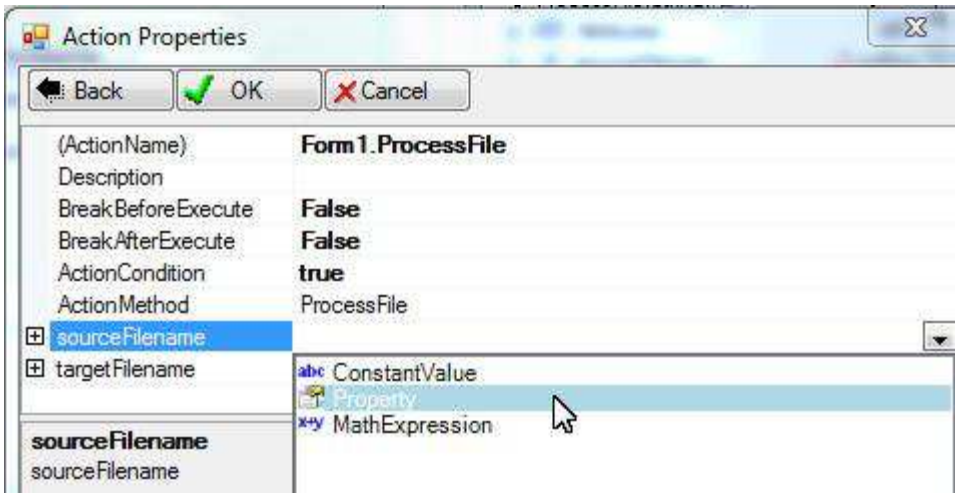
14.5 Test

We use two text boxes to enter the source file name and the target file name.

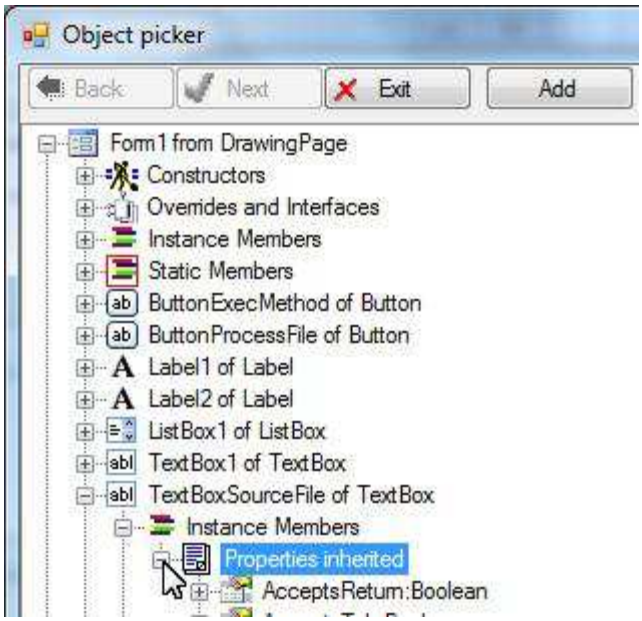
Right-click method ProcessFile; choose “Create Action”:

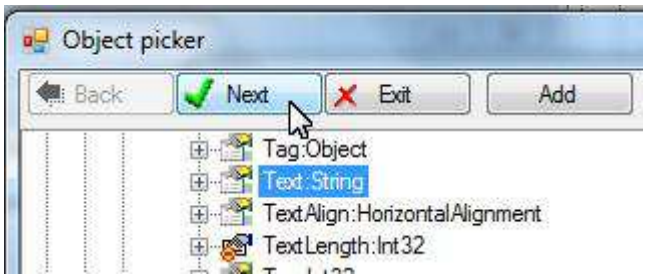


Select “Property” for sourceFilename:

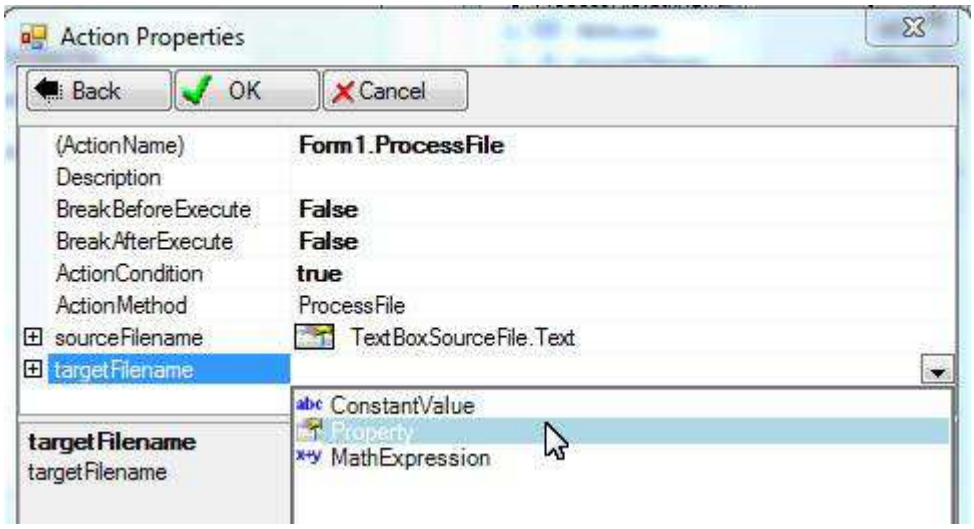


Select Text property of the text box for the source file:

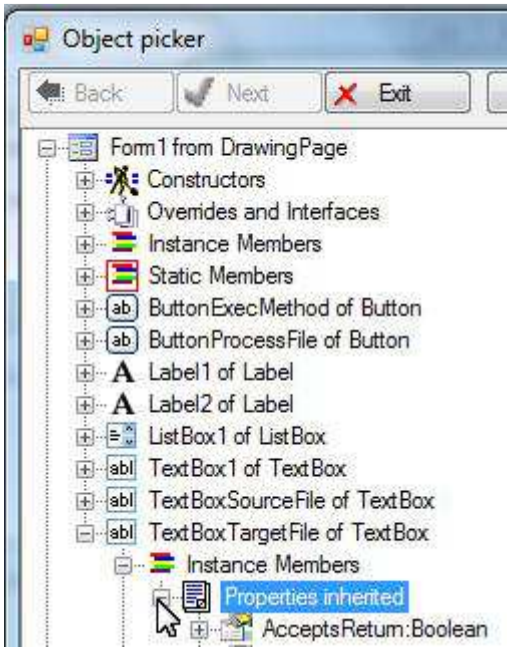


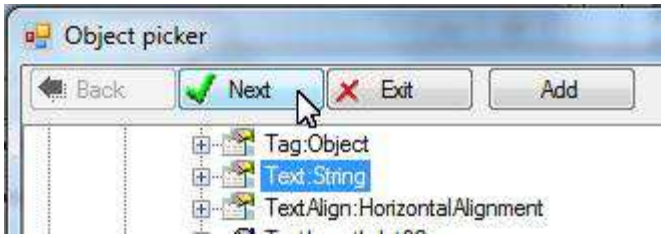


Select “Property” for targetFilename:

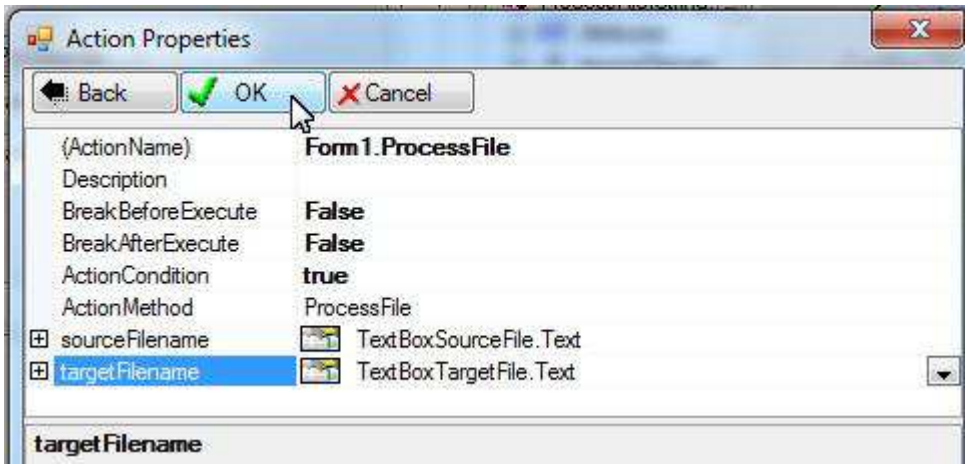


Select Text property of the text box for target file name:

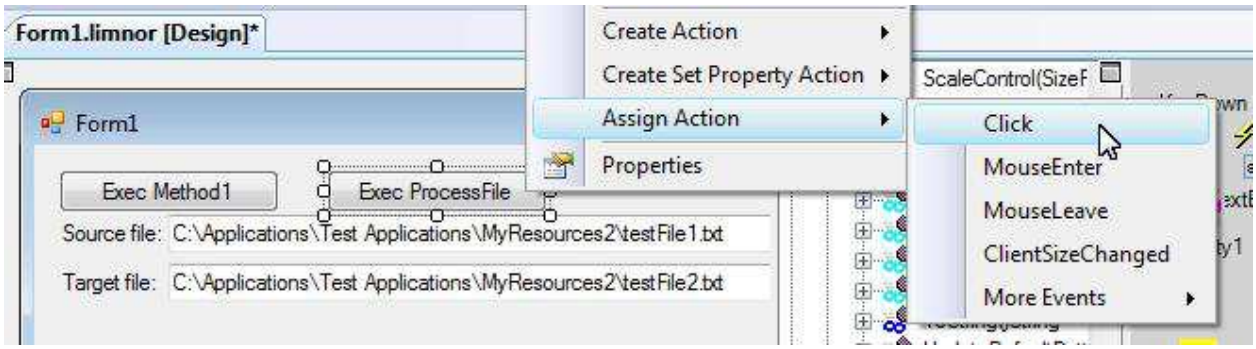




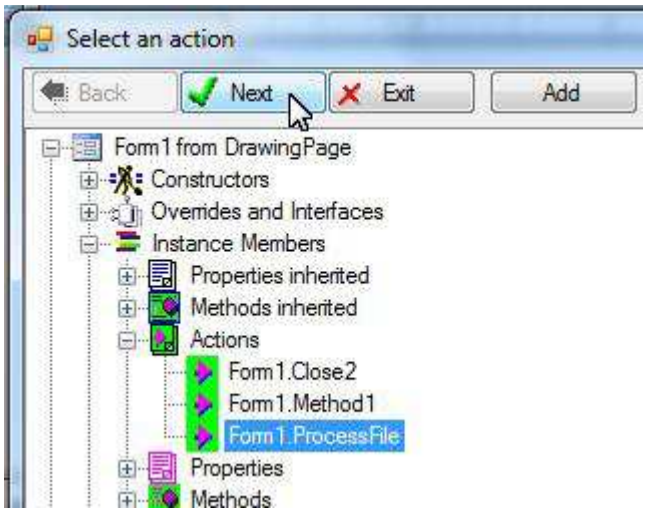
This action reads the file specified by the first text box and writes modified contents to the file specified by the second text box.



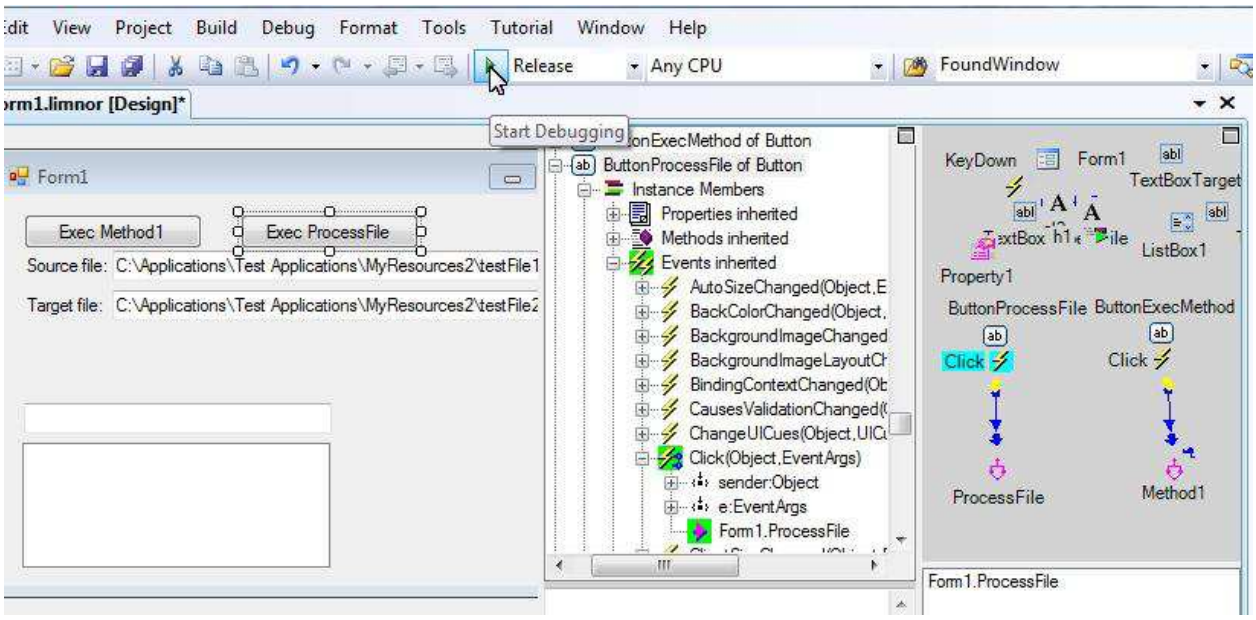
Right-click the second button; choose “Assign Action”; choose Click event:



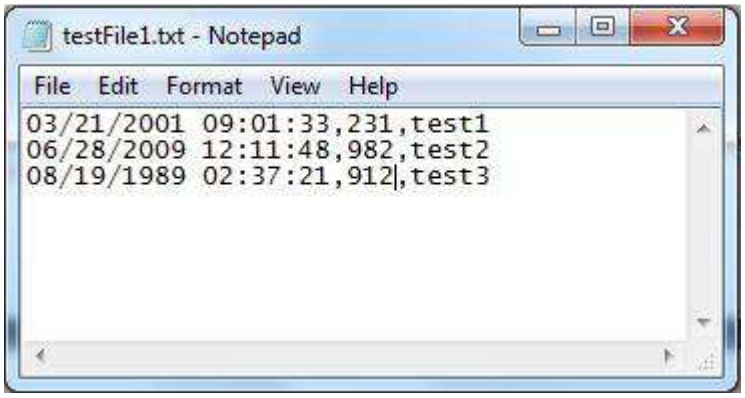
Select the ProcessFile action:



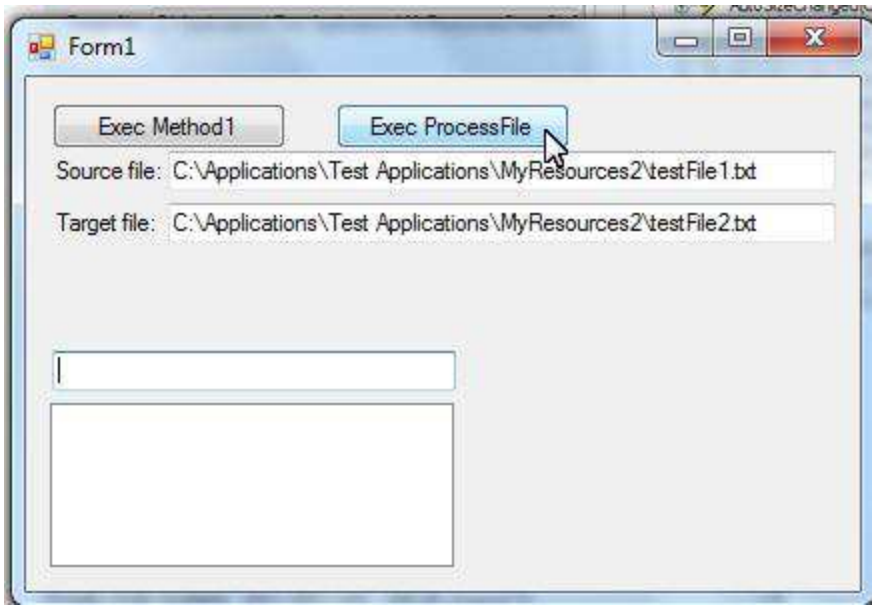
The action is assigned to the Click event of the button. We may test the application:



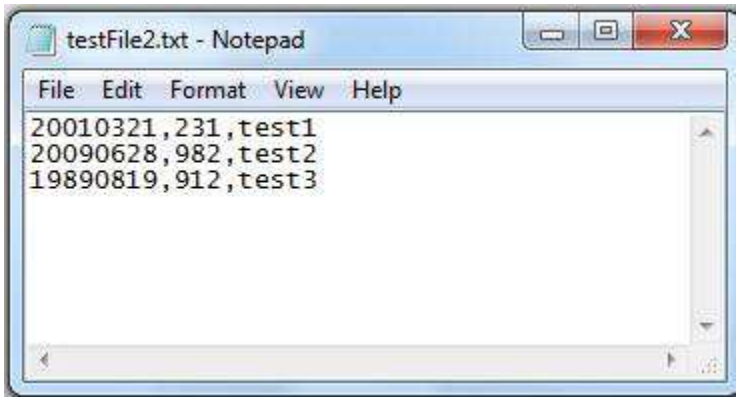
Suppose the source file looks like:



Click the file process button:



We saw the target file was created. Use the Notepad to open it; we saw we got the contents we wanted:



15 ExecuteForEachItem Action

For classes of arrays, lists, and collections, a special method, ExecuteForEachItem, can be used to create actions similar to the "Limited Number of Repeated Execution" action described above.

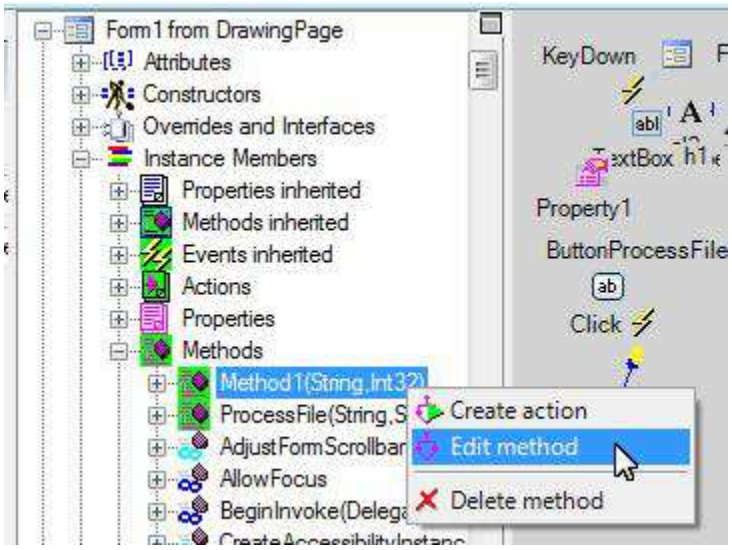
The "Limited Number of Repeated Execution" action is more flexible than ExecuteForEachItem action. But ExecuteForEachItem action is more convenient if it can be used.

1. Do not need to explicitly specify repeat times. If there are 2 items then it will execute 2 times. If there 1000 items then it will execute 1000 times.
2. Do not need to explicitly retrieve item value for each execution. The item value will be available when using the Method Editor to add actions to be executed.

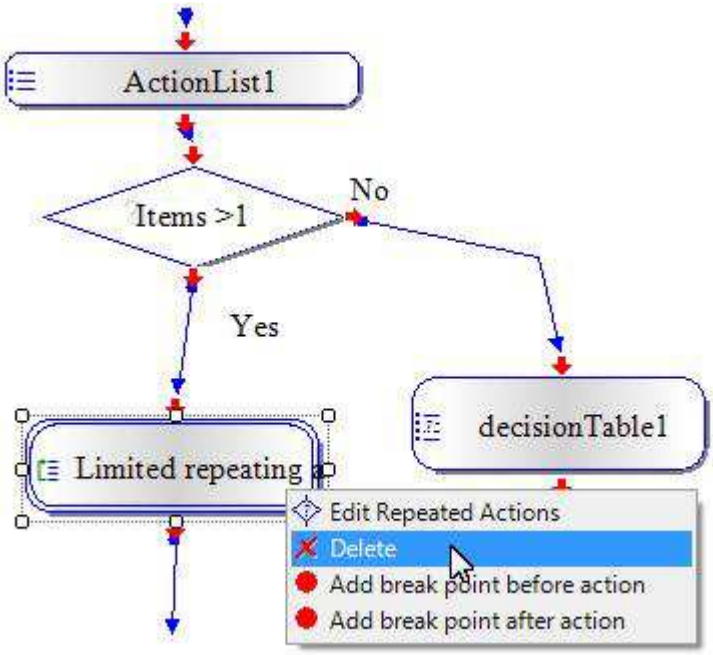
15.1 Create ExecuteForeachItem action by action executer

Let's modify previous example Method1 to use ExecuteForEachItem action.

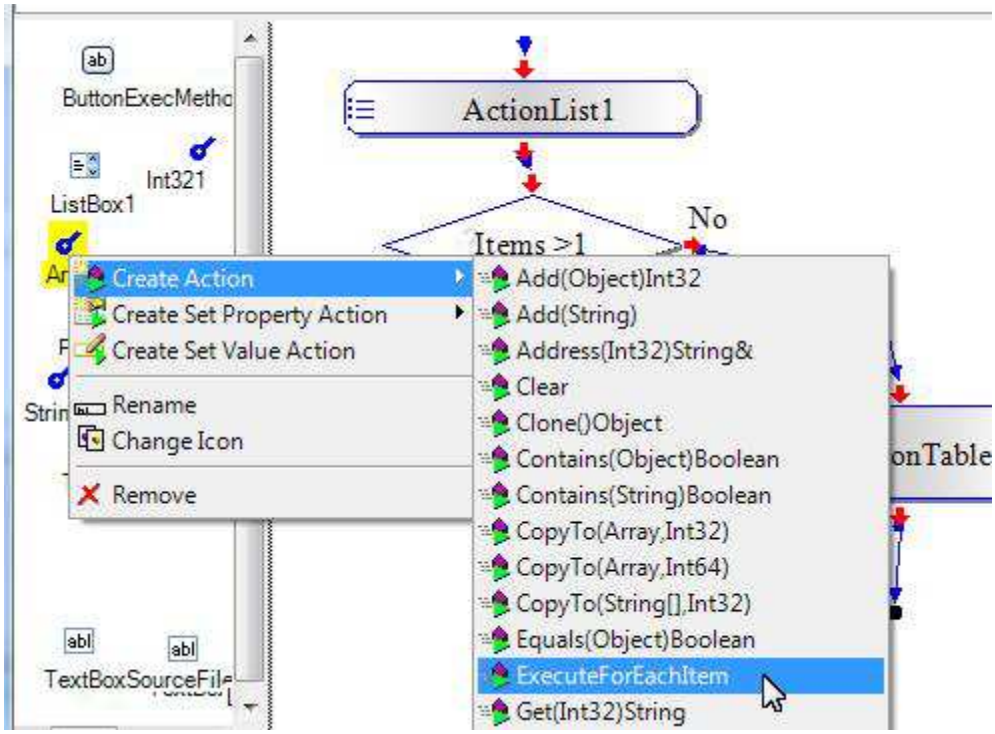
Right-click Method1, click “Edit”:



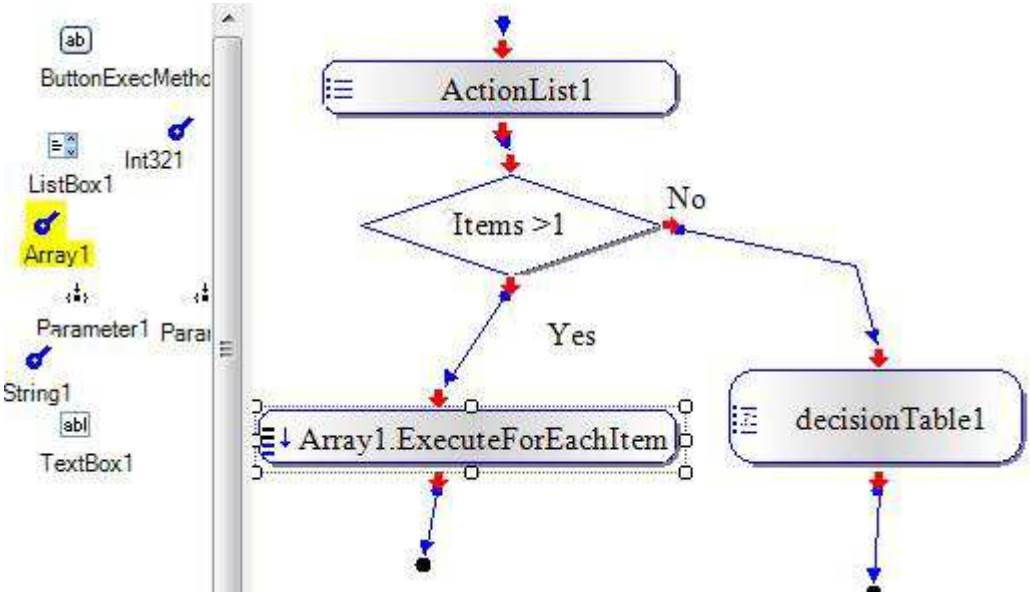
Remove the existing “Limited Number of Execution” action:



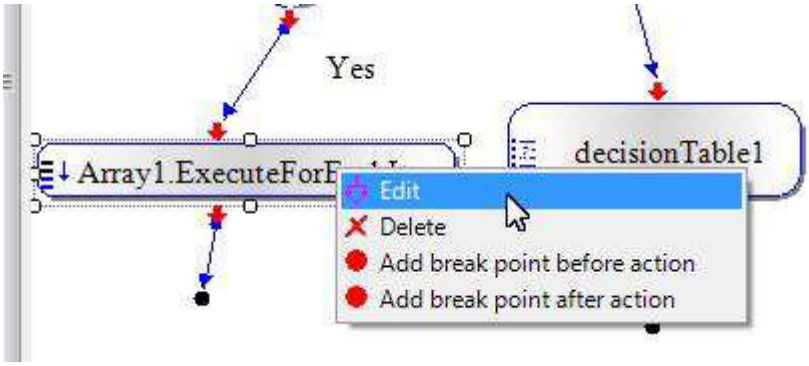
Right-click the array variable icon; choose “Create Action”. Select “ExecuteForEachItem” method:



Link the action to the YES port of the condition action:

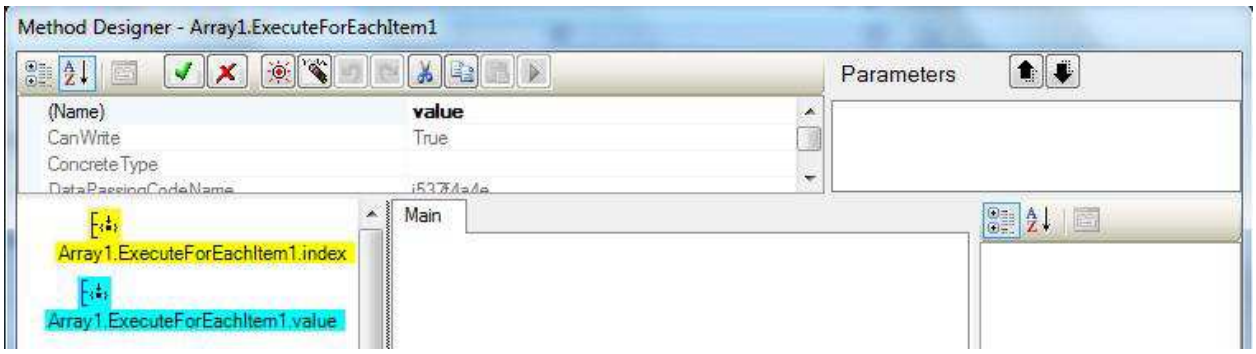


Right-click the action, choose “Edit”

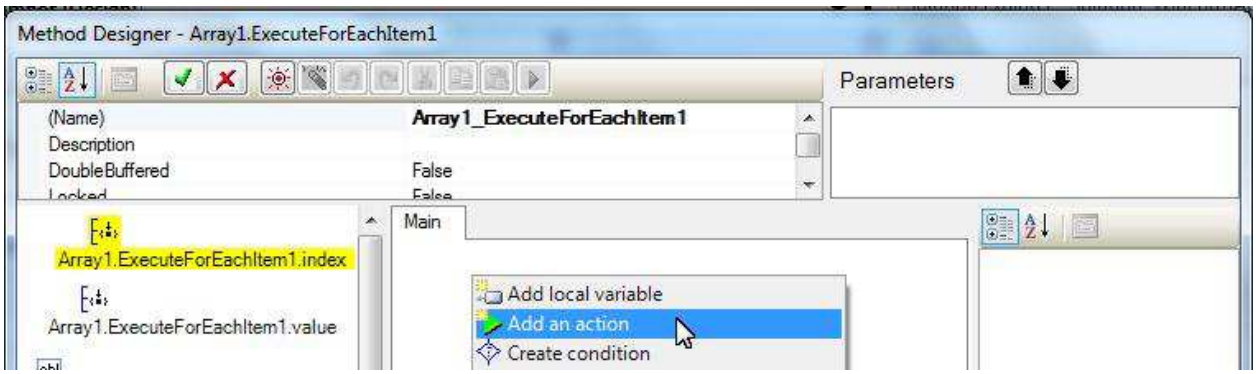


A new Method Editor appears. Note the two new variable icons index and value.

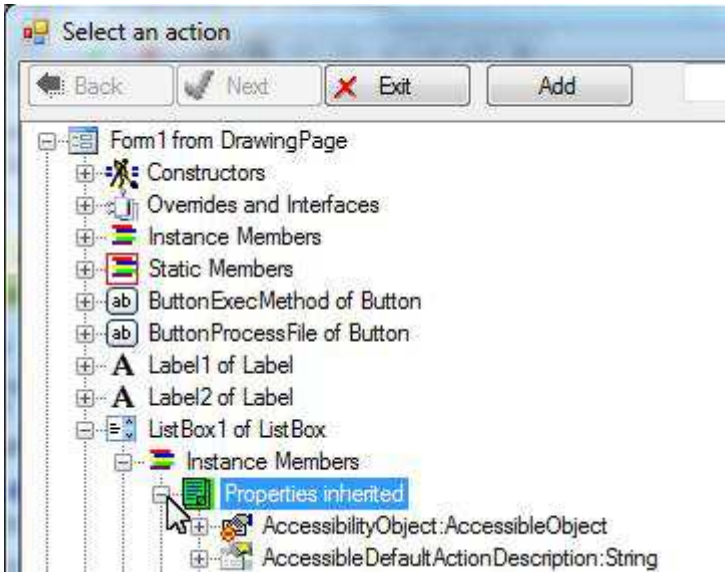
index is an integer. 0 indicates the first time execution and value is the first item; 1 indicates the second time execution and value is the second item; and so on.



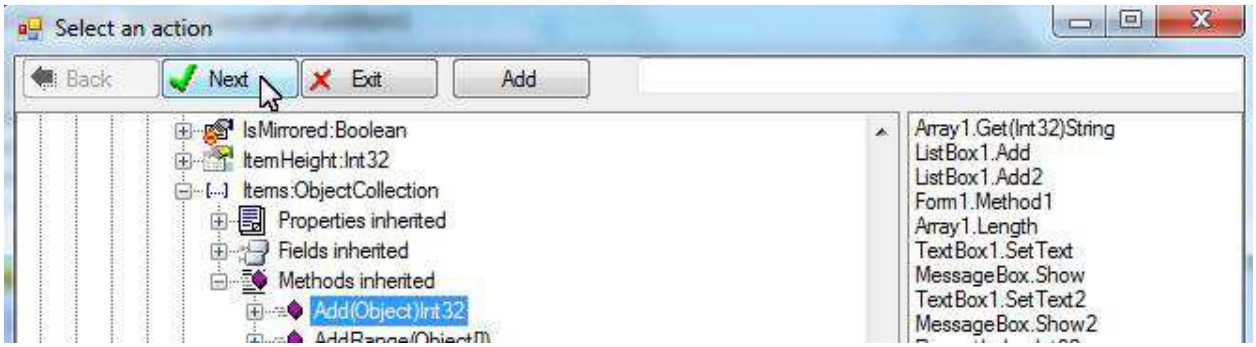
Right-click the Action Pane; choose “Add an action”:



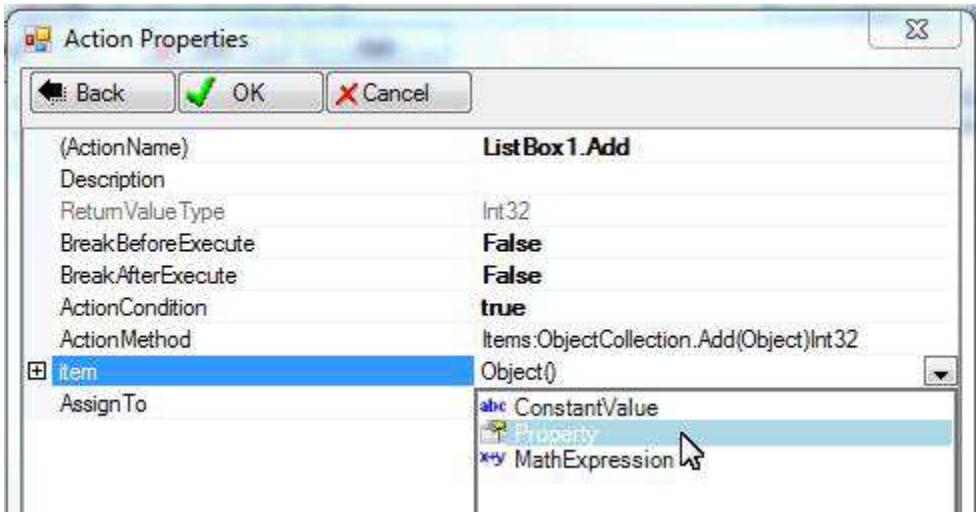
Find the Items property of the list box:



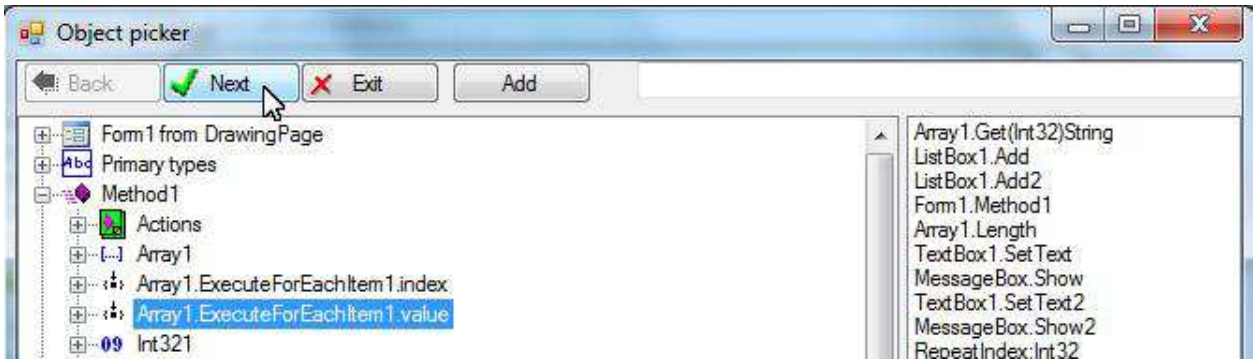
Select the Add method of the Items property of the list box, and click “Next”:



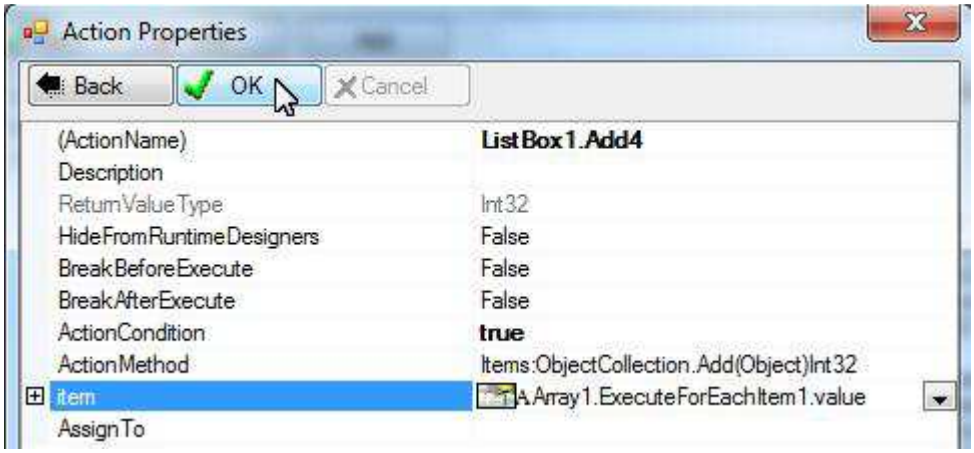
Select “Property” for the item to be added to the list box:



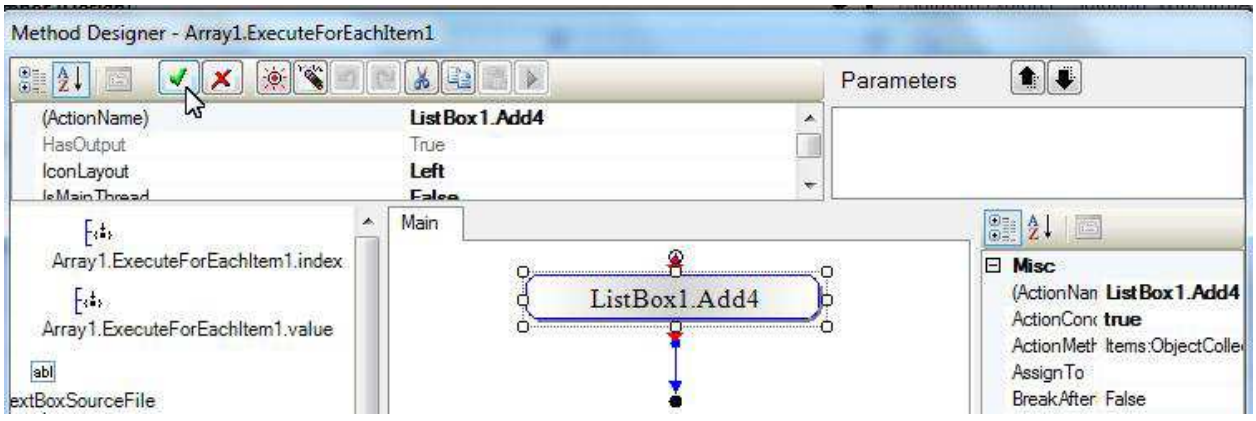
Select “Array1.ExecuteForEachItem1.value” as the item to be added to the list box.



Click OK to finish creating this action. In the first time execution,”Array1.ExecuteForEachItem1.value” is the first item. So, the first item is added to the list box. In the second time execution,”Array1.ExecuteForEachItem1.value” is the second item. So, the second item is added to the list box. And so on. We do not need to explicitly retrieve item from the array as we did using Get method.

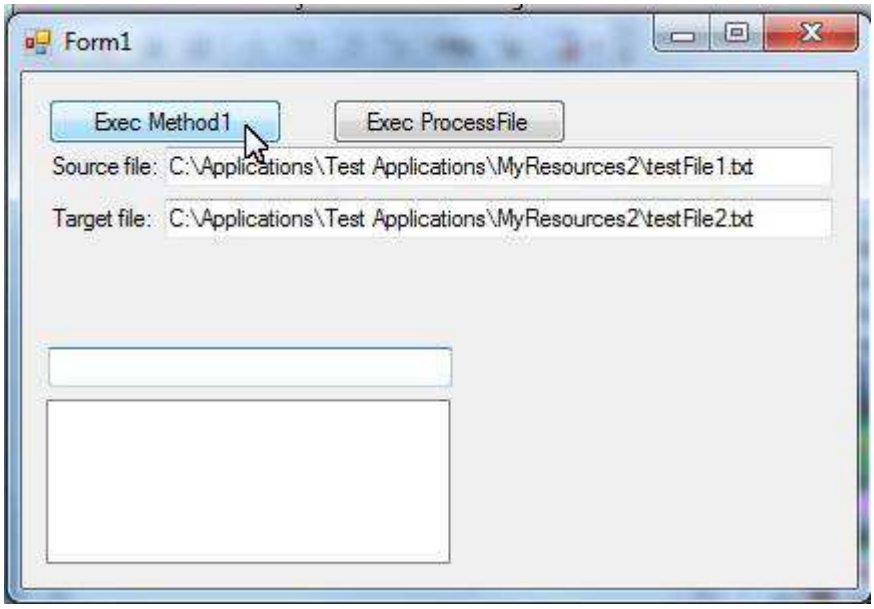


The action appears in the Method Editor:

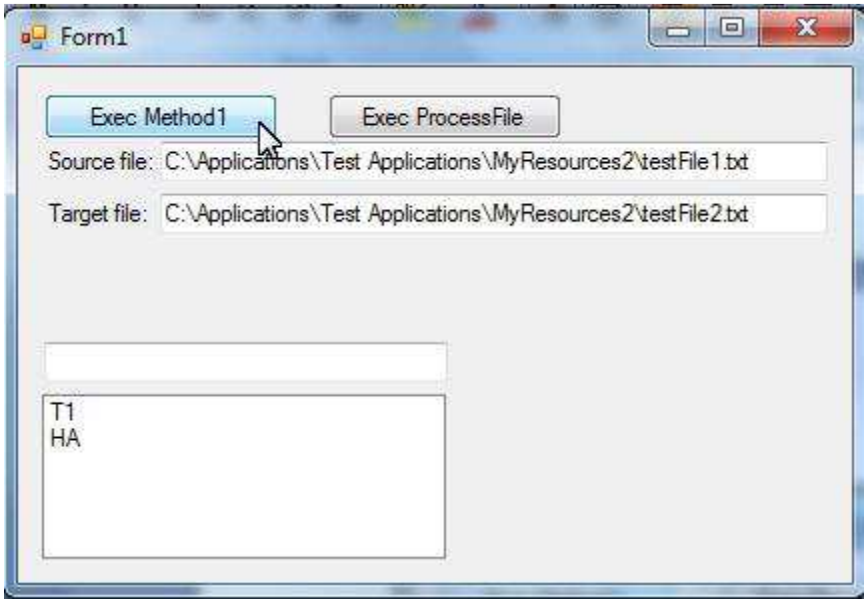


We are done. Close all the Method Editors. We can test it:

Click the button:



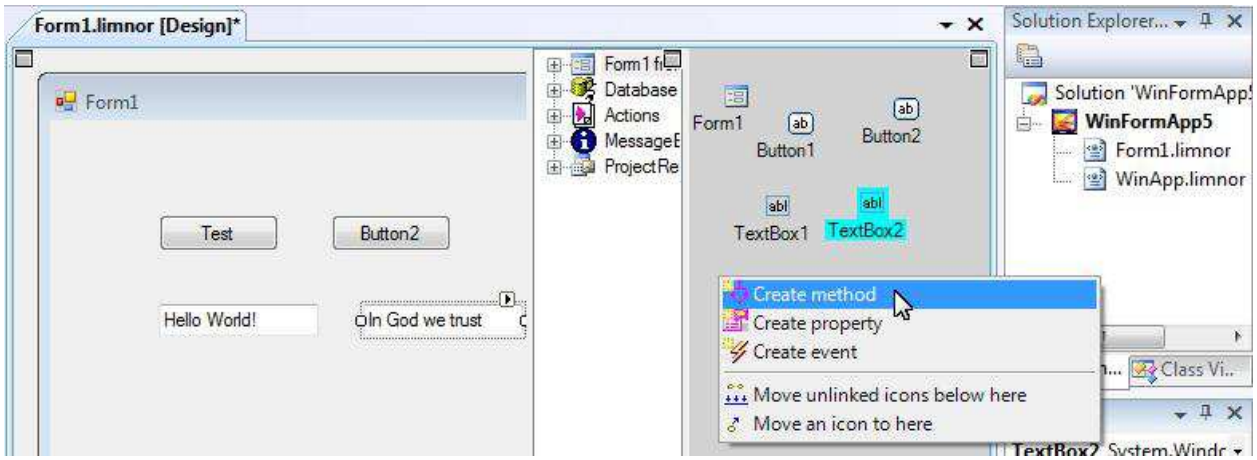
The array items are added to the list box:



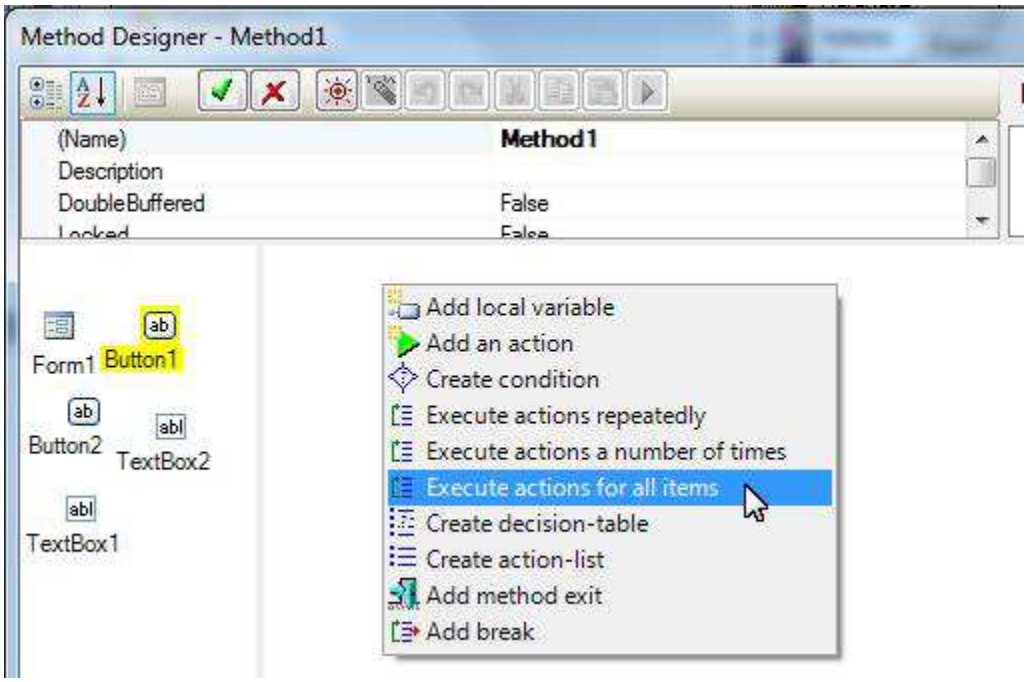
15.2 Create ExecuteForEachItem action from the Action Pane

In the above example, we choose "ExecuteForEachItem" method from an object represented by an icon. In many cases, we want to create an "ExecuteForEachItem" action for an object not in the Object Pane. We can do it by creating such an action from the Action Pane.

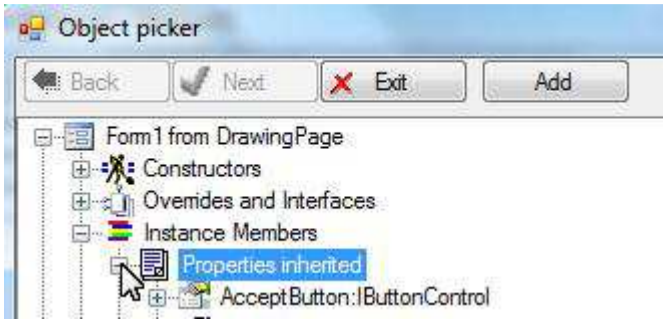
Let's create a new Windows Form Application project and add some text boxes and buttons to the form. We create a new method:

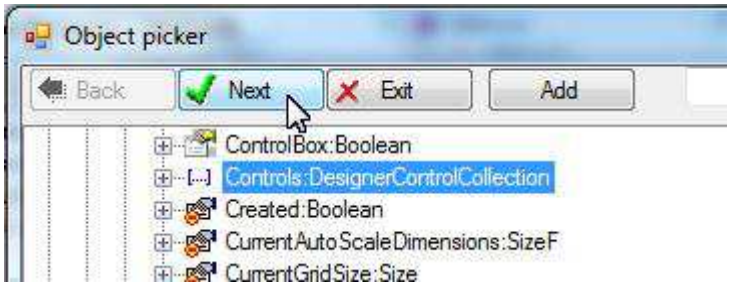


Right-click the Action Pane; choose “Execute actions for all items”:

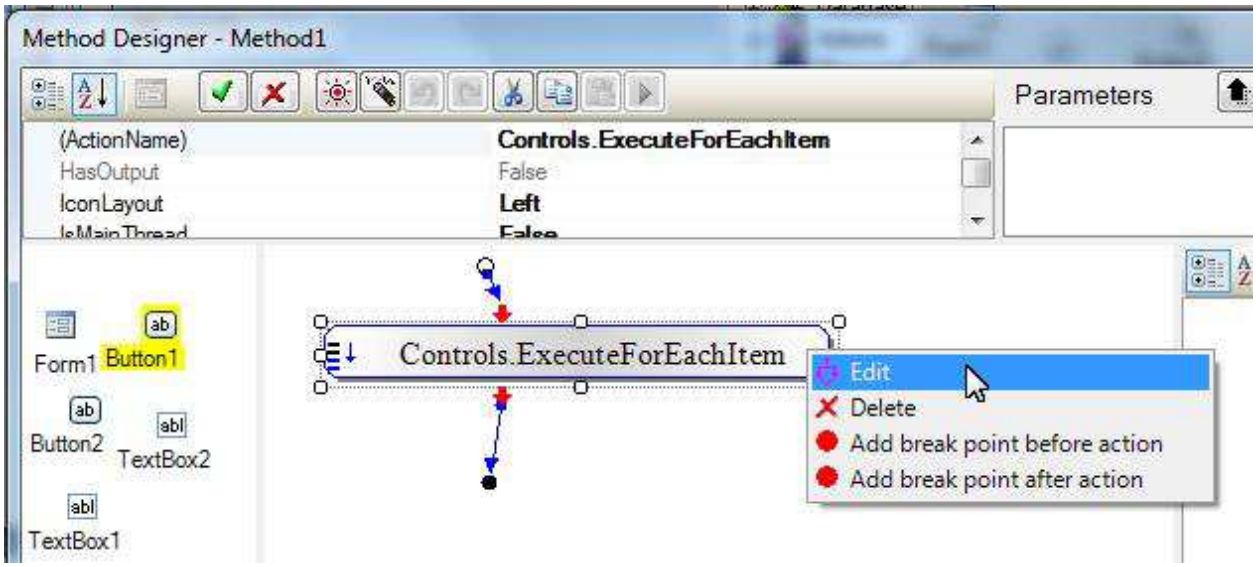


A dialogue box appears for selecting an object which is an array, a list or a collection. If a property is such an object then it has icon [...]. For example, the Controls property of a form is a collection of Controls including all controls in the form. Let’s select the Controls property as an example:



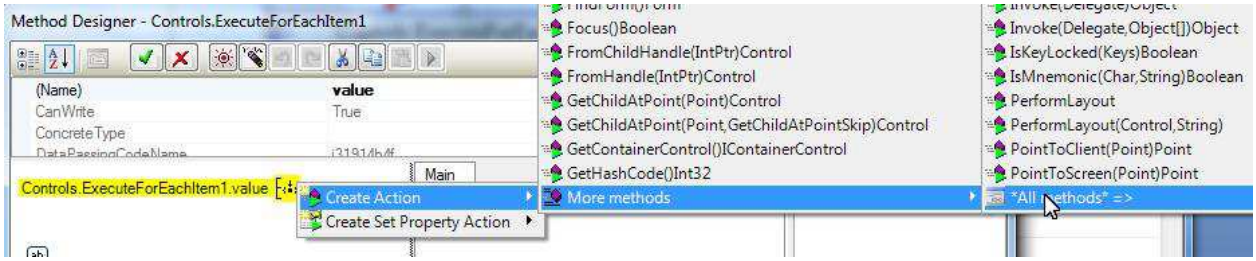


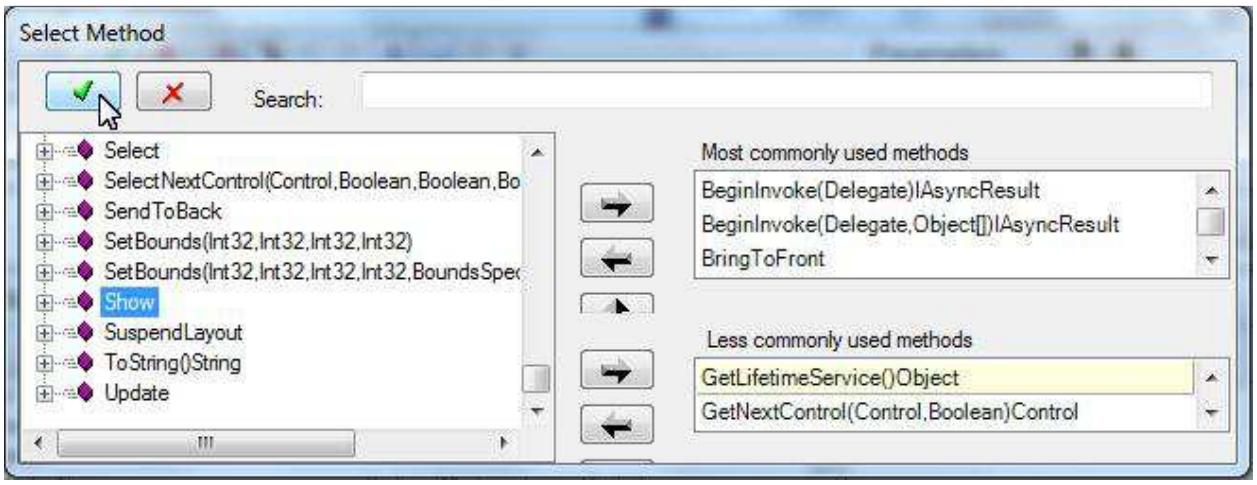
A new "Execute actions for all items" action appears. It will go through all controls in the Controls collection and execute actions. To specify the actions to be execute right-click it and choose "Edit".

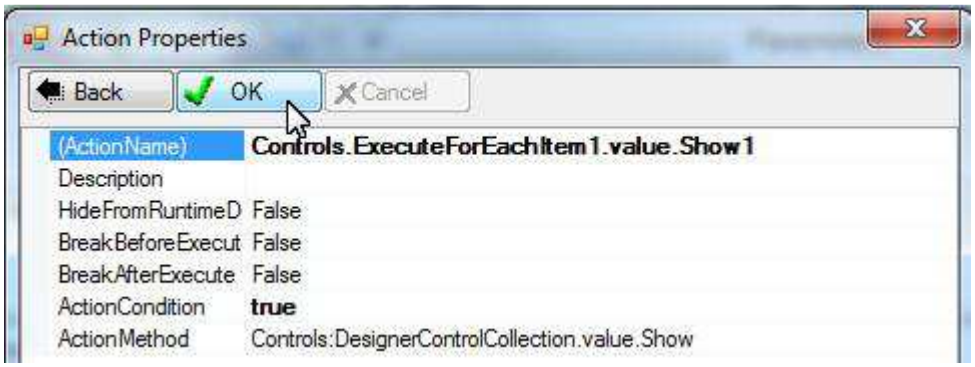


A new Method Editor appears. Note the "Controls.ExecuteForEachItem1.value" icon in the Object Pane. It represents the control to be processed. Suppose Controls property contains 100 controls then the actions specified in this Method Editor will be executed 100 times. For every time of execution the "Controls.ExecuteForEachItem1.value" icon represents a control of the 100 controls.

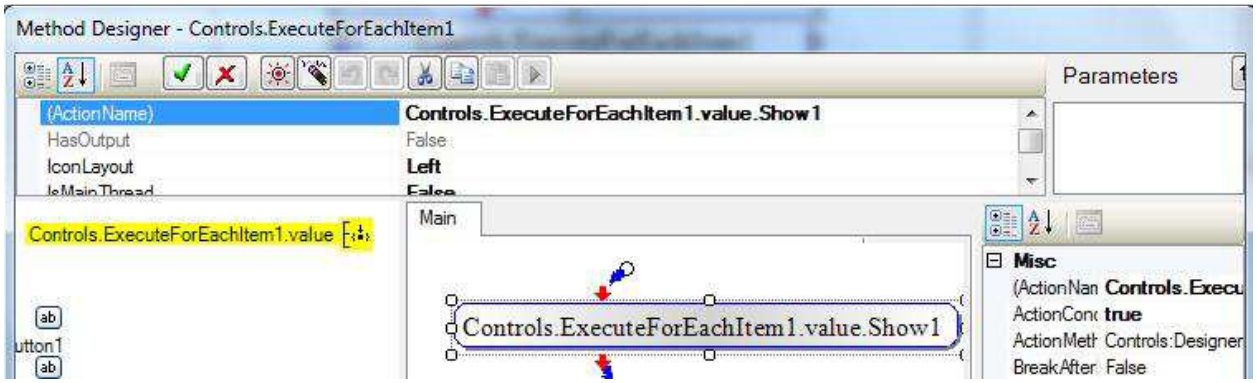
Suppose we want to make all controls visible. It can be done by a Show action. Right-click the "Controls.ExecuteForEachItem1.value" icon; choose "Create Action"; choose "Show" method.







The action appears:



15.3 Process specific types of items

In the previous example we created an action using the "Controls.ExecuteForEachItem1.value" icon. It represents all types of controls.

Sometimes we want to execute actions for specific types of items.

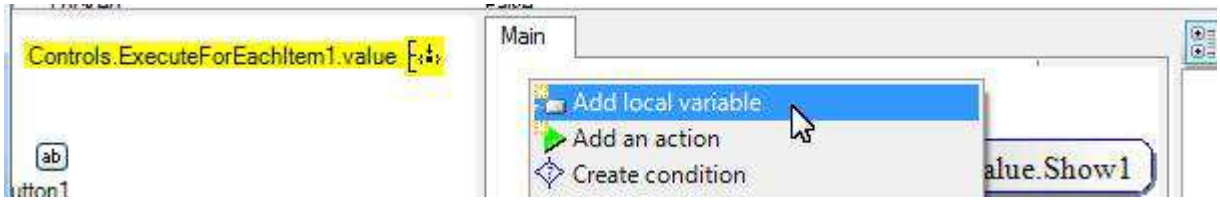
For example, suppose we want to make all text boxes read-only. This can be done by setting the ReadOnly property of all text boxes to True.

In the previous example, Show method is available for all kinds of controls. Therefore a Show action can be created using the "Controls.ExecuteForEachItem1.value" icon. But not all controls have ReadOnly property. From the "value" icon we will not be able to find a ReadOnly property.

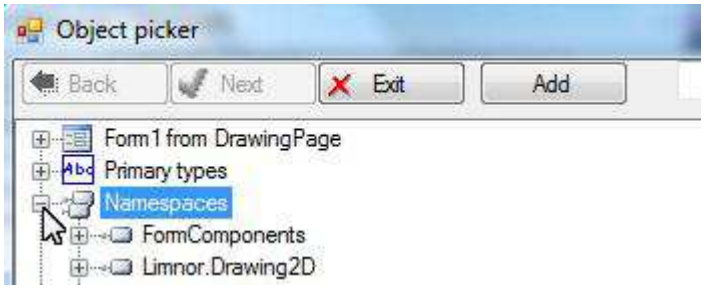
For another example, suppose we want to bring all buttons to front. This can be done by a BringToFront action. Even BringToFront method is available from the "value" icon we do not want to use it because we only want to execute BringToFront on buttons, not all controls.

To process specific types of items, create a local variable for each type to be processed.

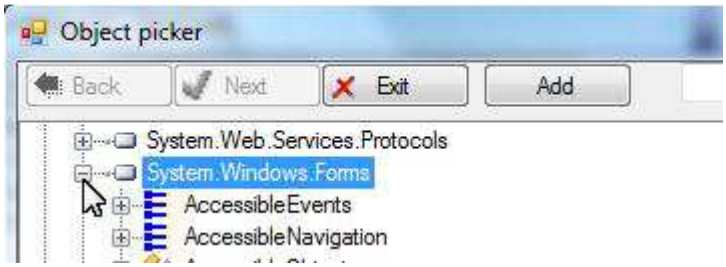
To create a text box variable, right-click the Icon Pane or the Action Pane; choose "Create local variable":



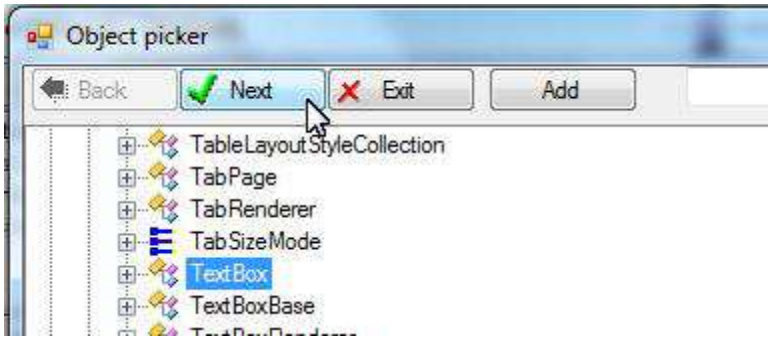
Expand Namespaces:



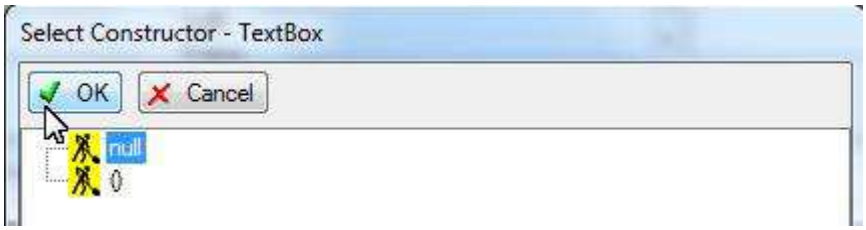
Expand System.Windows.Forms namespace:



Select the TextBox and click Next:

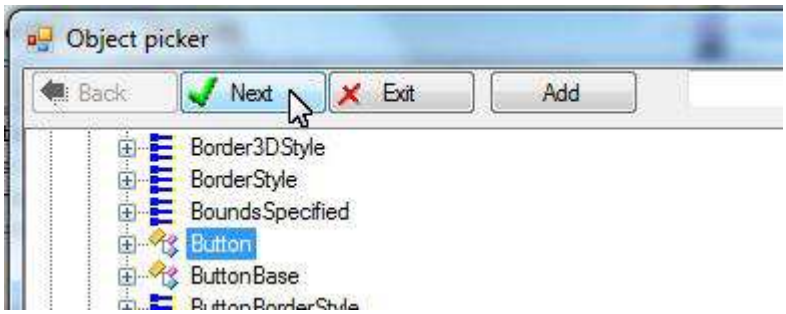


Select null because we do not create a new instance of TextBox:

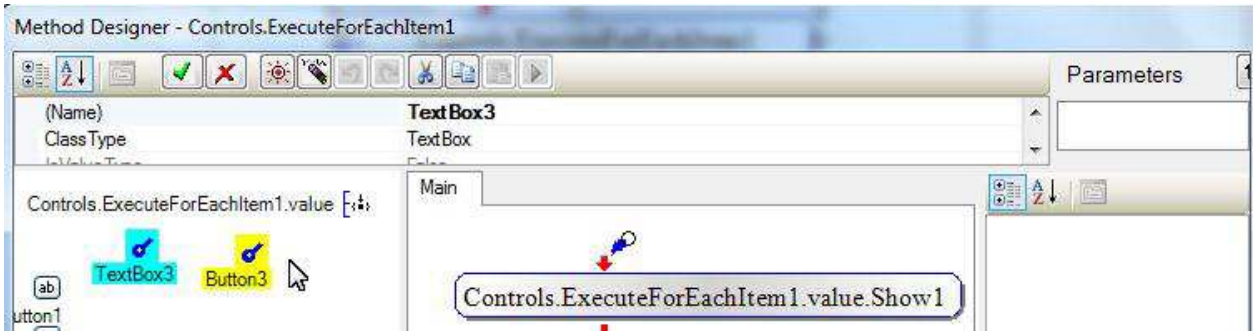


A TextBox variable is created.

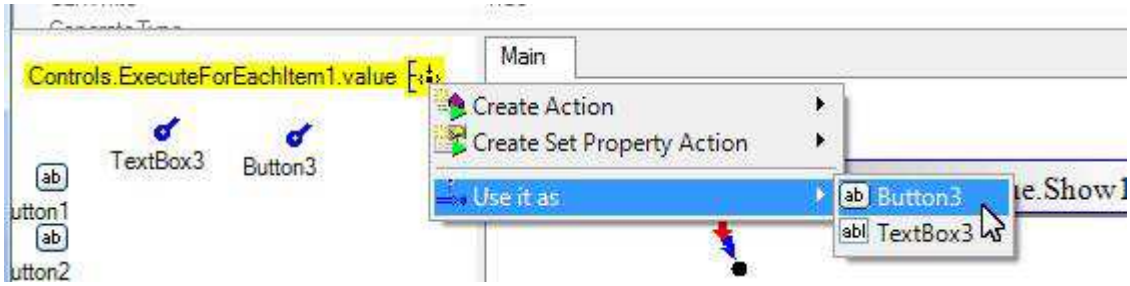
In the sample way, we may select the Button type



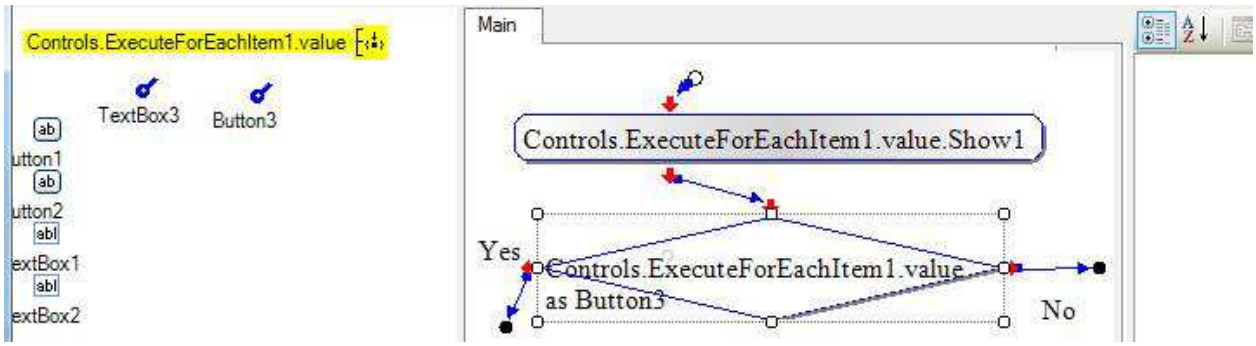
And create a button variable:



Once the variables are created, right-click the “Controls.ExecuteForEachItem1.value” icon; choose “Use it as” menu. The variables are listed under the “Use it as” menu. Select Button3:

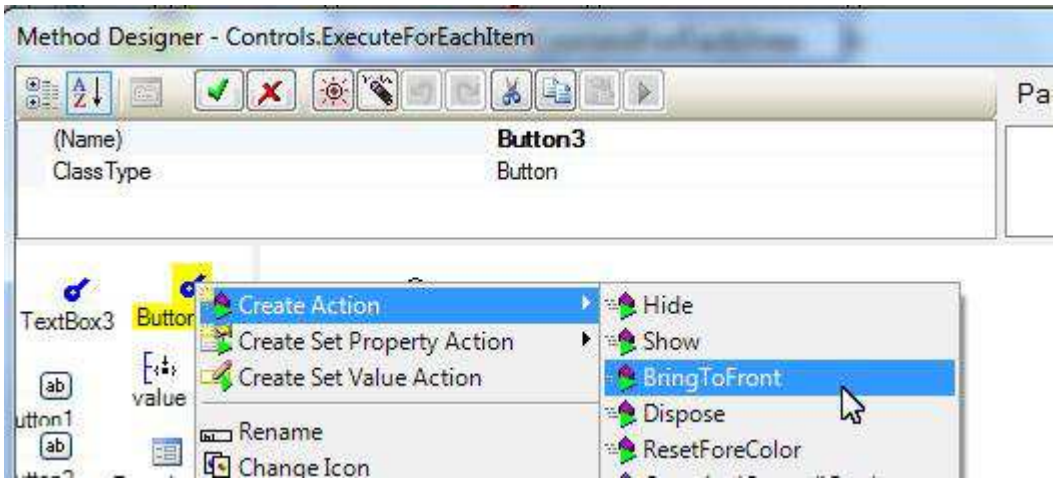


A “value as Button3” action appears:

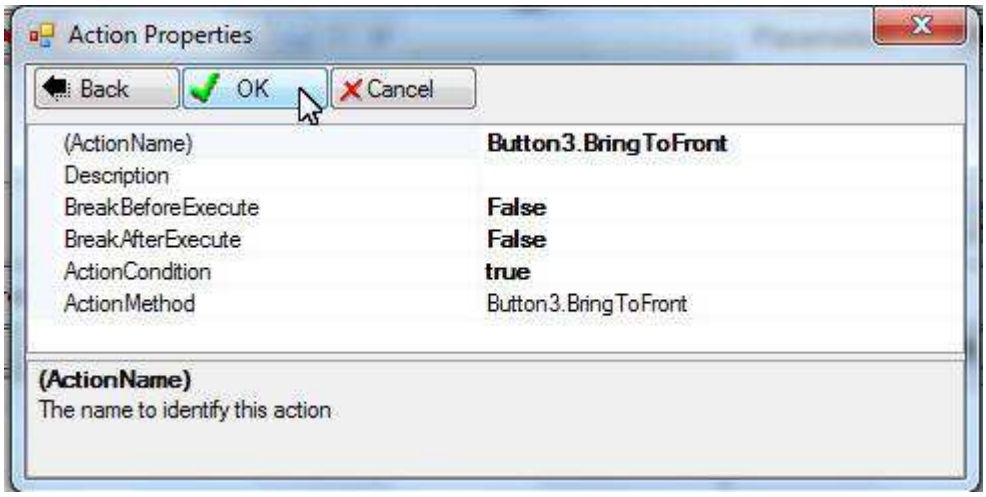


This action has two outputs. “Yes” port indicates that the value is a Button and Button3 can be used. “No” port indicates that the value is not a Button and Button3 should not be used.

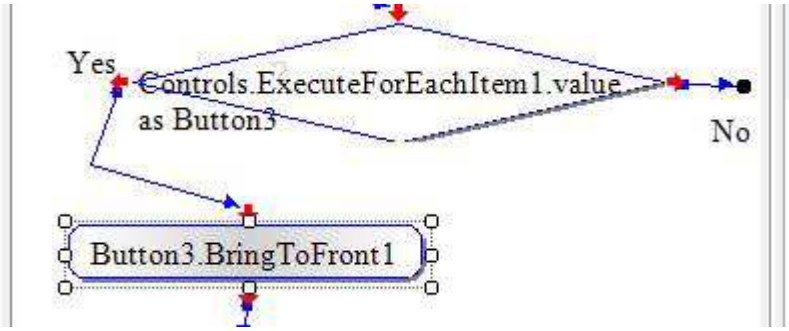
Right-click Button3; choose “Create Action”; choose “BringtoFront”:



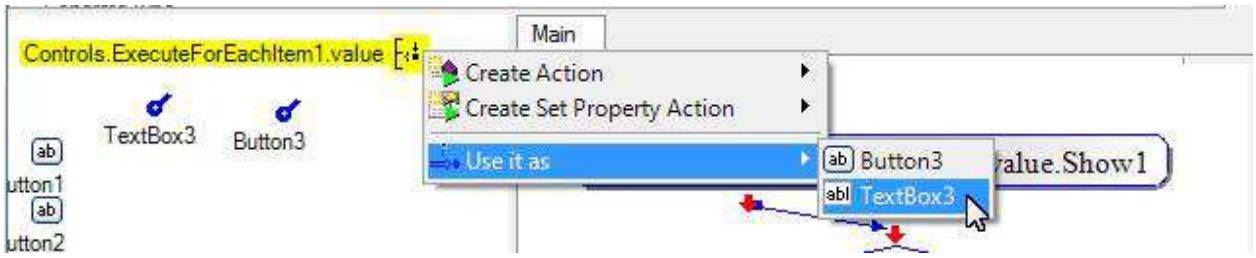
Click OK:



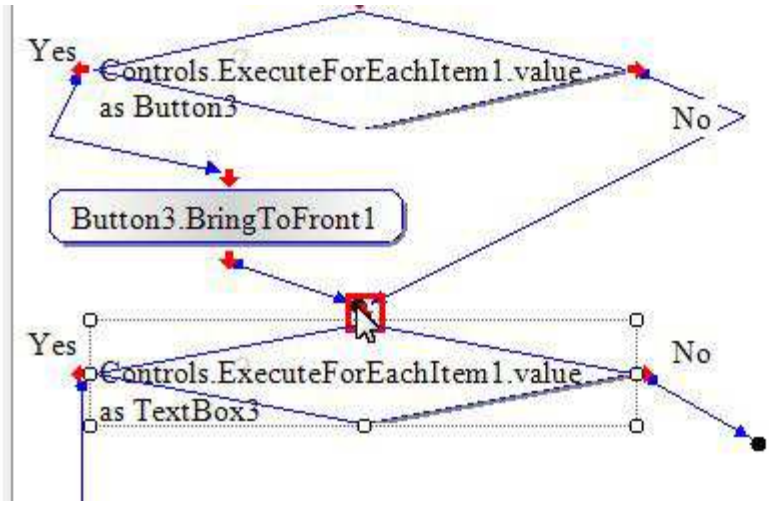
The action appears. Link the new action to the "Yes" port:



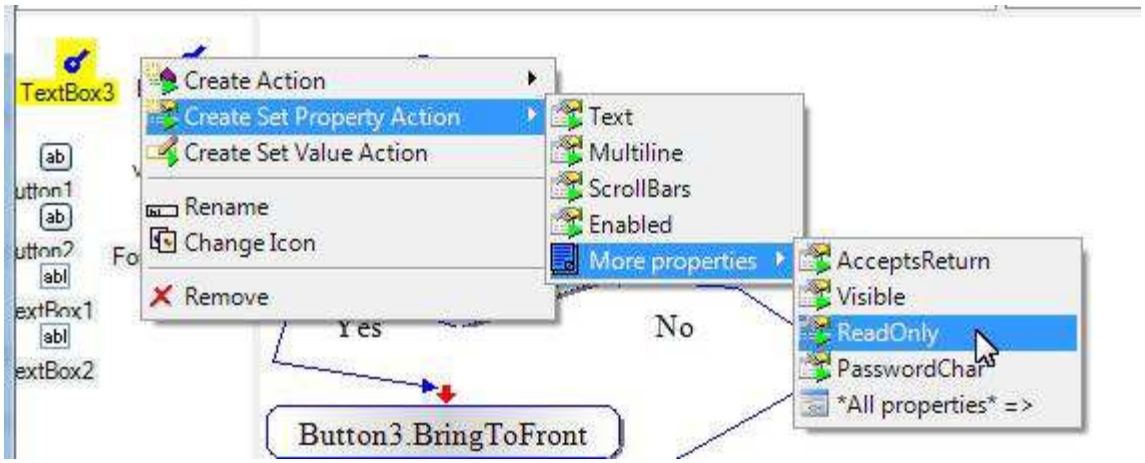
Right-click "Controls.ExecuteForEachItem1.value"; choose "Use it as"; choose "TextBox3":



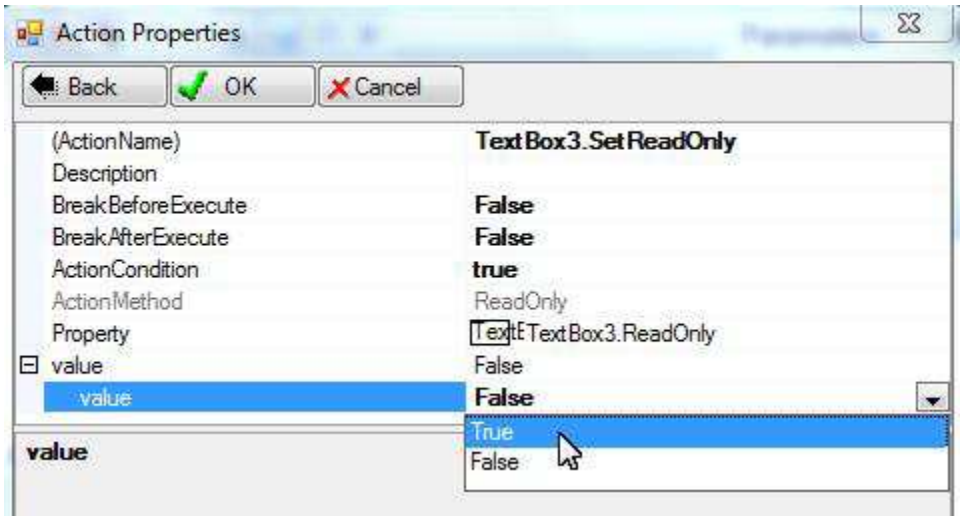
A new "value as TextBox3" action appears. Link it to existing actions:



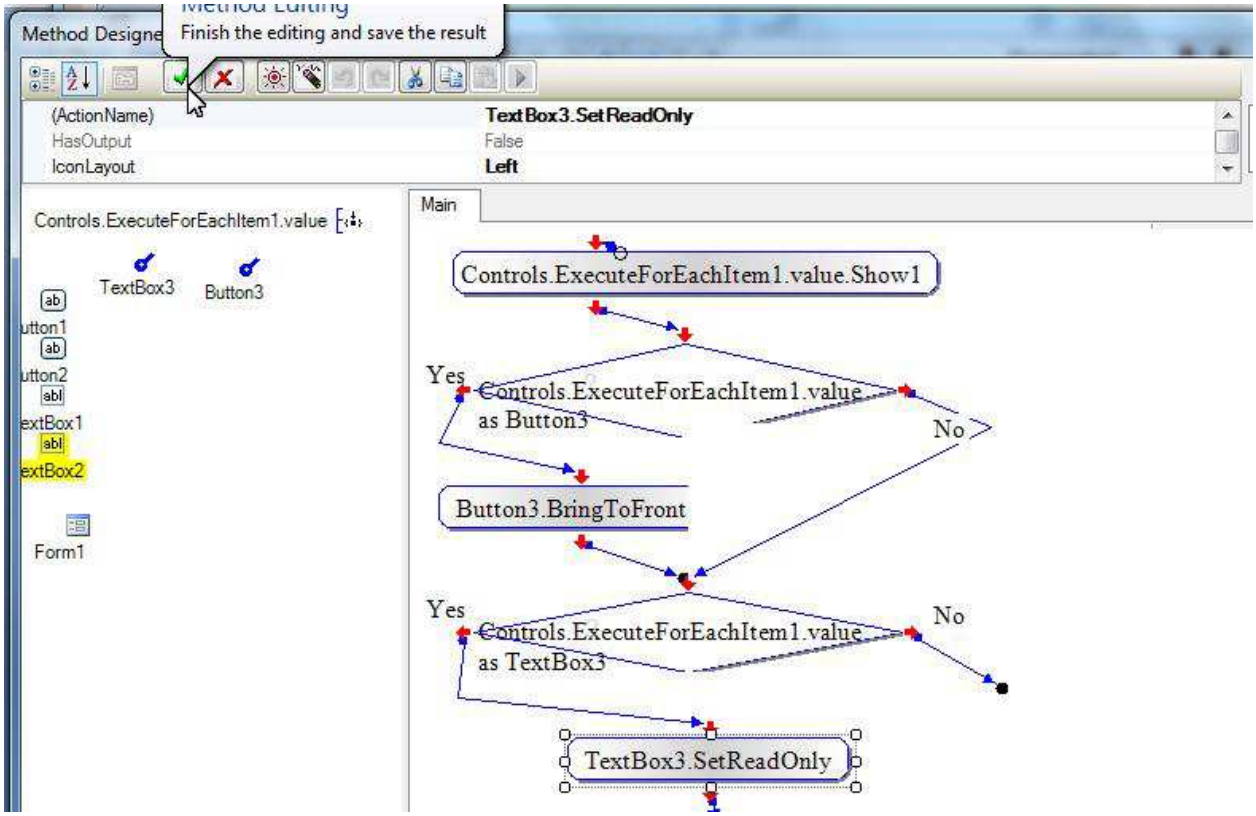
Right-click variable TextBox3; choose “Create set property action”; choose “ReadOnly”:



Set the property value to True:



The action appears. Link it to the “yes” port of the “value as TextBox3” action:

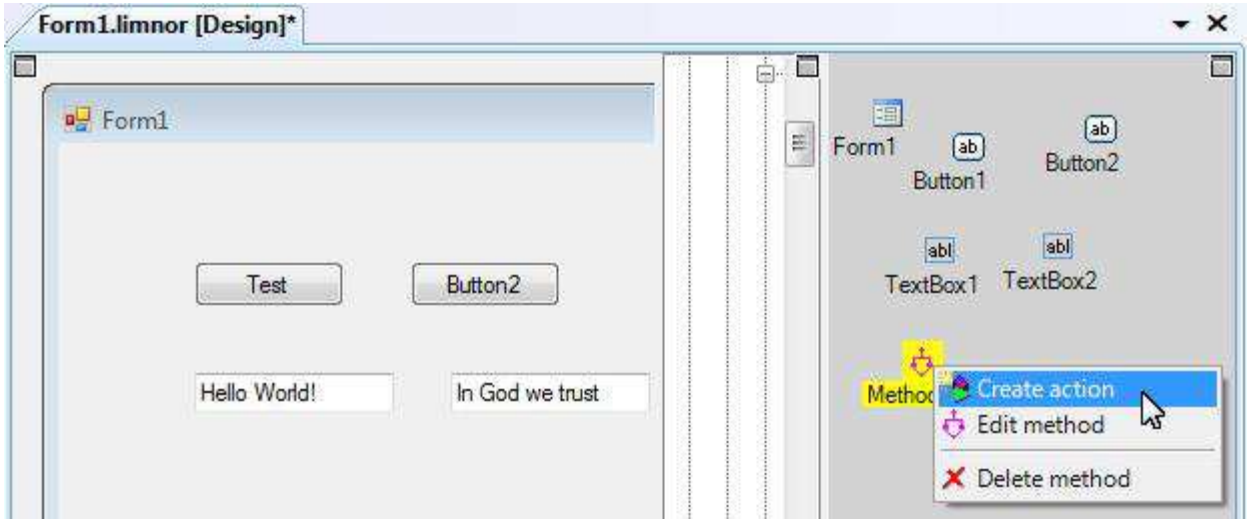


We may close all the Method Editors now.

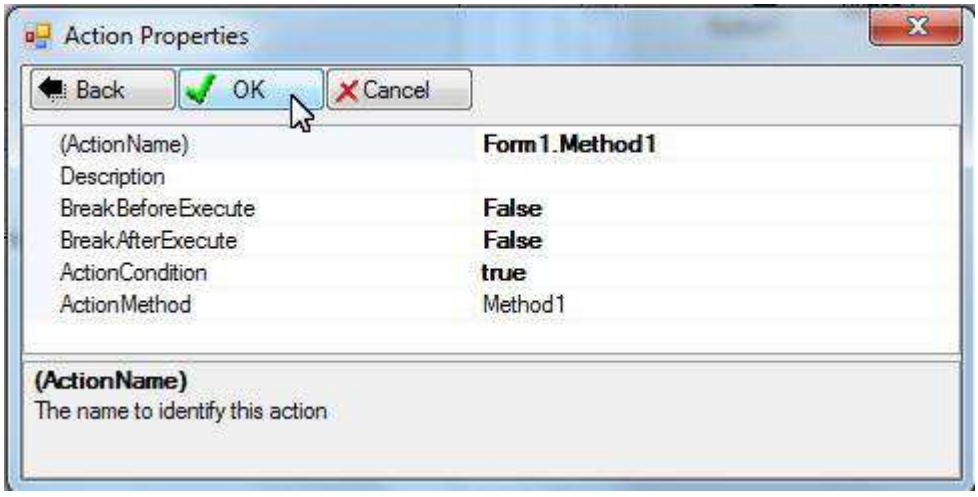
We created this method, Method1. When this method is executed it makes all controls visible by executing Show method of each control; it brings all buttons to front by executing BringToFront method of each button; it makes all text boxes read-only by setting ReadOnly property of each textbox to True.

This method is a demonstration of processing array, list and collection.

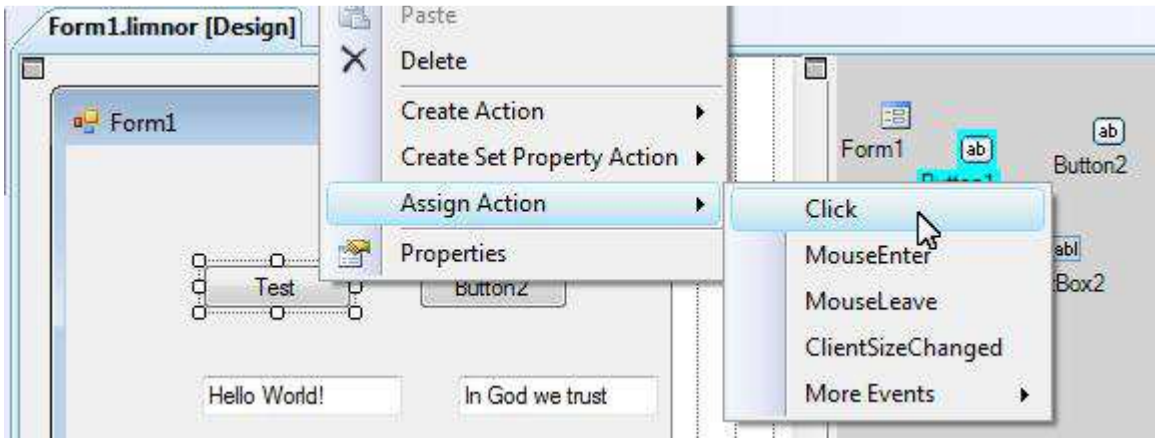
To execute Method1, right-click Method1, choose “Create action”:



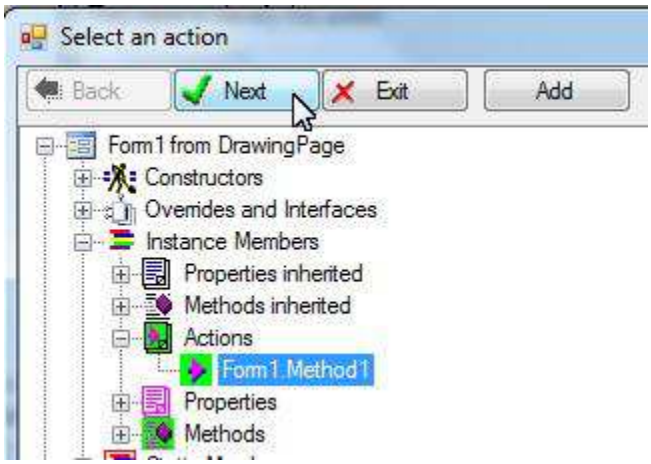
Click OK:



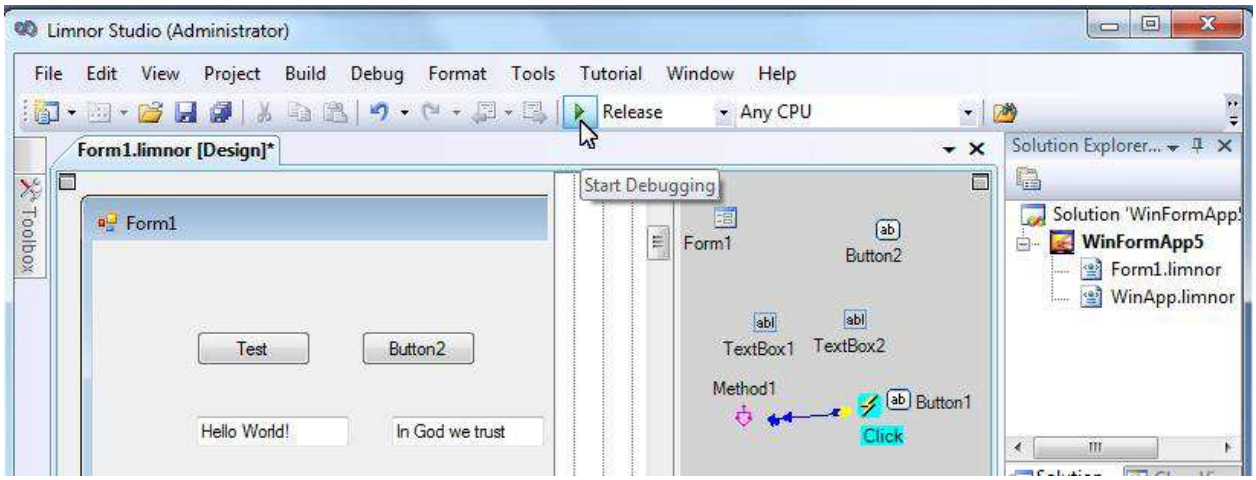
Assign the action to a button:



Select the action and click Next:



We may test the application now:



The form appears. Click Test button:

We can see that all text boxes become read-only:

16 Create Action Groups

When more and more actions are added to a method, the method editor becomes messy. We demonstrated creating action lists to make the control diagram looks clean. An action list does not allow

control branching. Several linked actions can be made into an action group. An action group allows including condition actions for branching. The following samples show the uses of action groups:

http://www.limnor.com/support/UseGenericTypes.pdf

