Limnor Studio – User's Guide

Part – IV

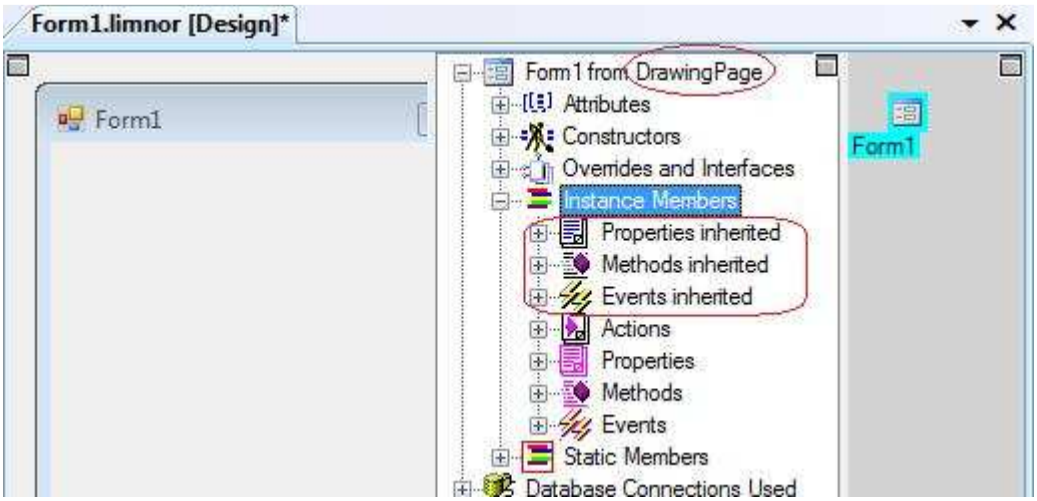
Create Properties, Methods and Events

Contents

1 Introduction

In previous samples, we have been using Properties, Methods and Events provided by various classes. The functionality of a class is mostly defined by its Properties, Methods and Events.

When we are developing a new class, we must derive the new class from an existing class. The Properties, Methods and Events of the existing class automatically become the Properties, Methods and Events of the new class. We call them inherited Properties, Methods and Events. They are displayed in the Object Explorer under “Properties inherited”, “Methods inherited” and “Events inherited”:

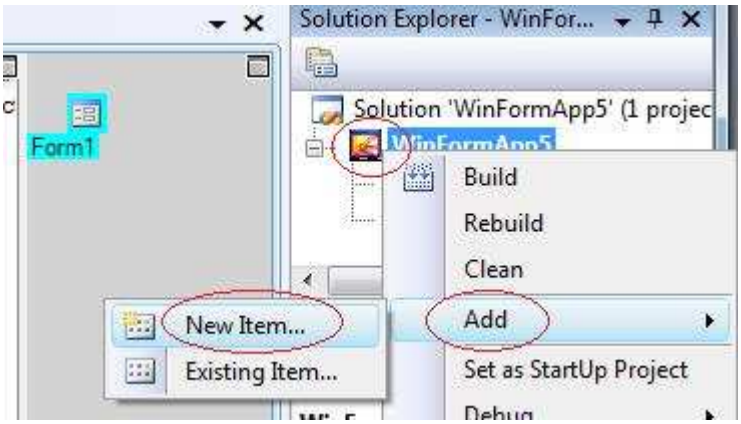


In this example, Form1 is the new class being developed. DrawingPage is the existing class. The existing class is also called the base class of the new class.

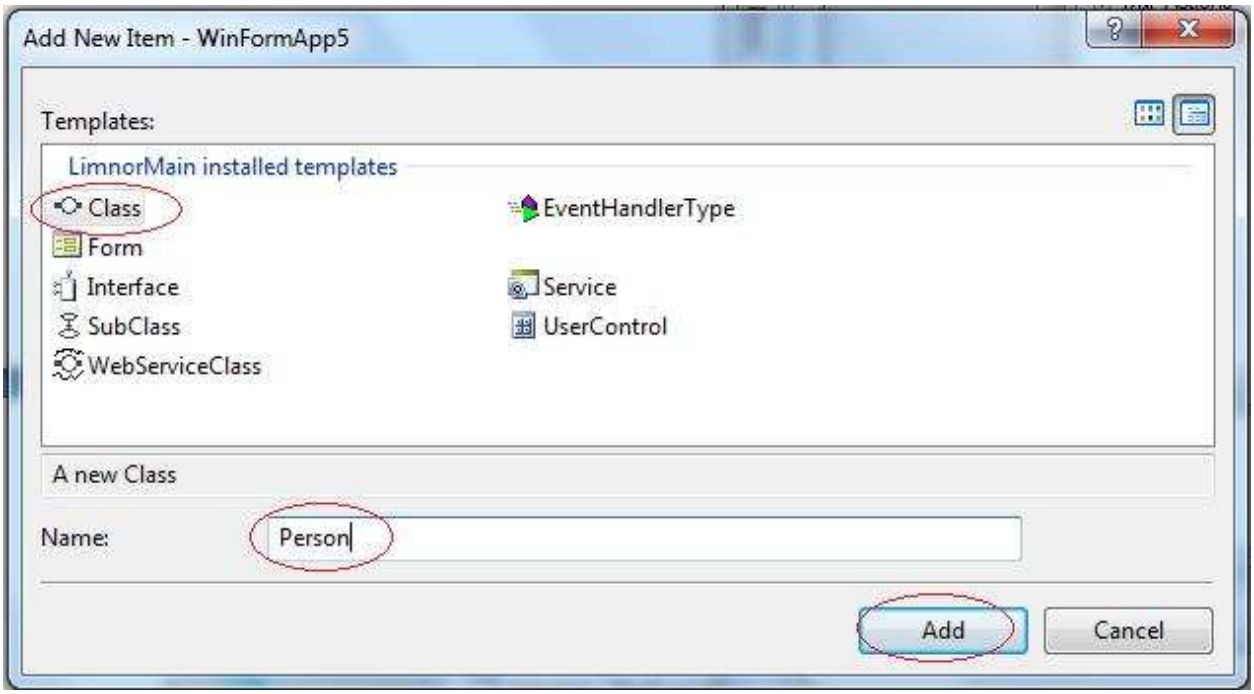
We may add new Properties, Methods, and Events to the new class being developed to provide required functionality not provided by the base class.

In this User’s Guide, we will create a new class named Person and add properties, Methods, and Events to this class to form its functionality.

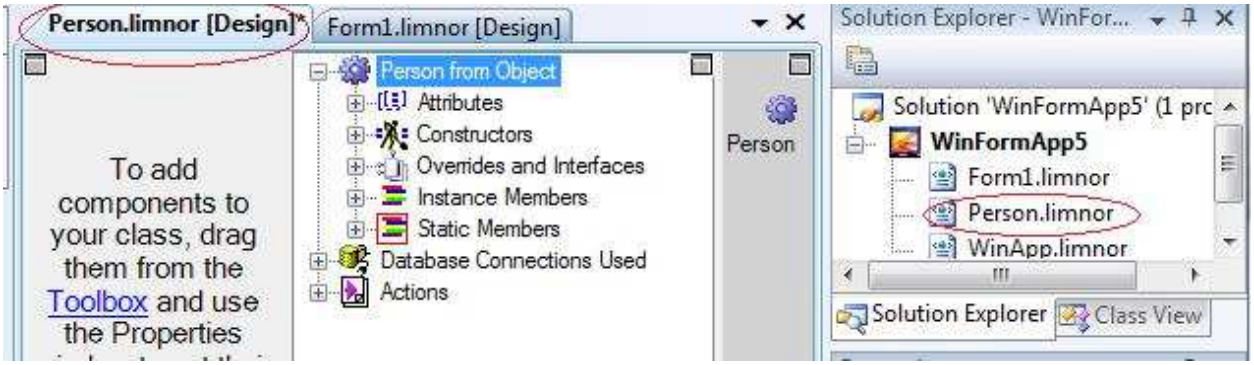
Suppose we have created a Windows Form Application project. To create a new class in this project, right-click the project, choose “Add” and “New Item…”:



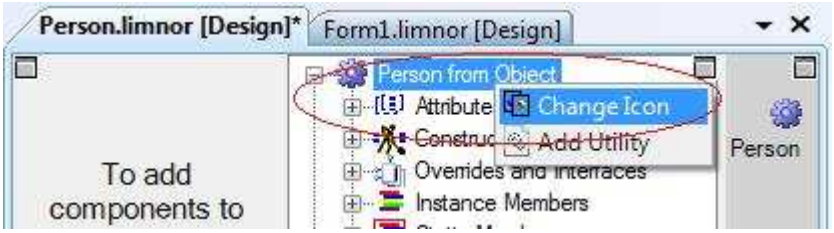
Choose an existing class to derive the new class from. We want to choose a class closest in functionality to the new class to be developed. Select “SubClass” will allow us to select a base class available in the current computer. If there is not an existing class close to the new class then we may select “Class”:



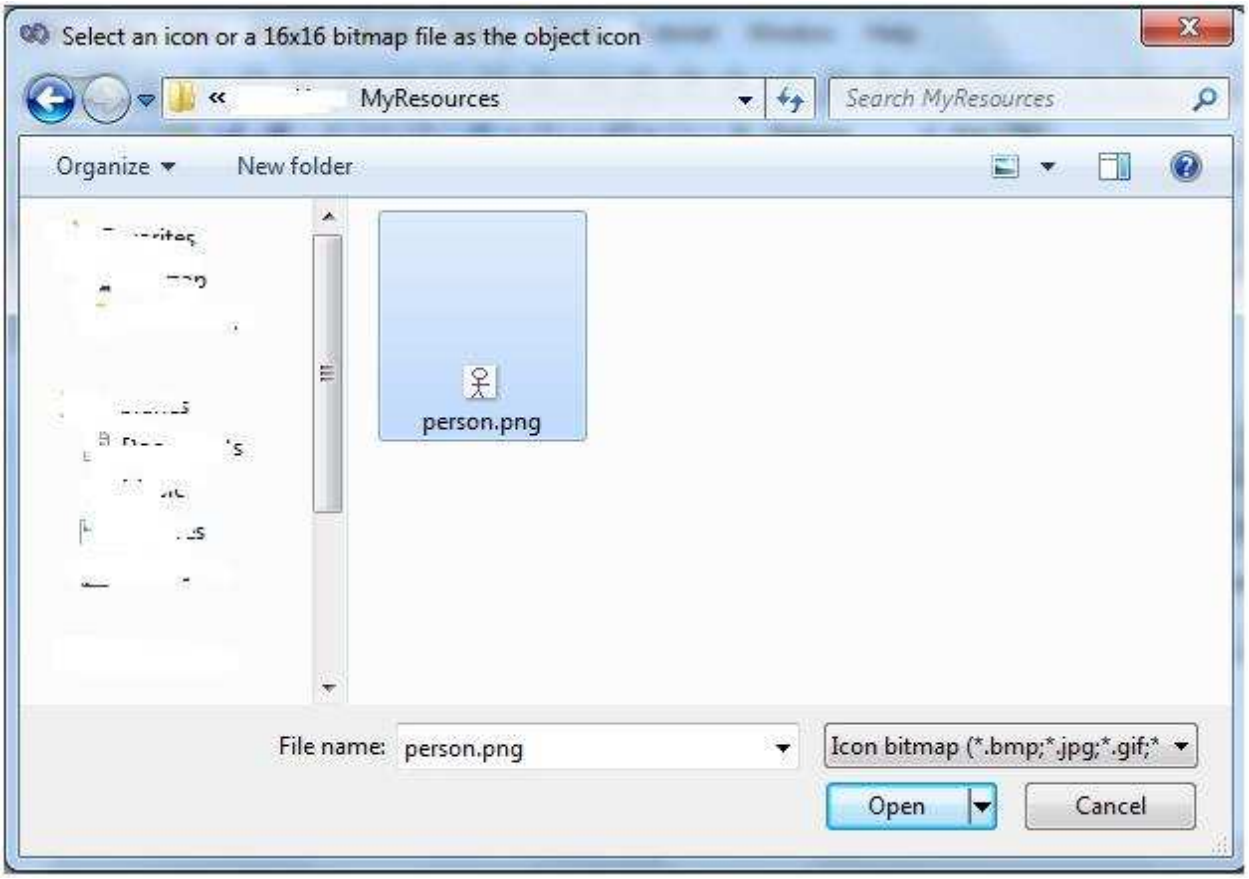
Give the new class a name. Click Add. The new class is created and loaded into the designers:



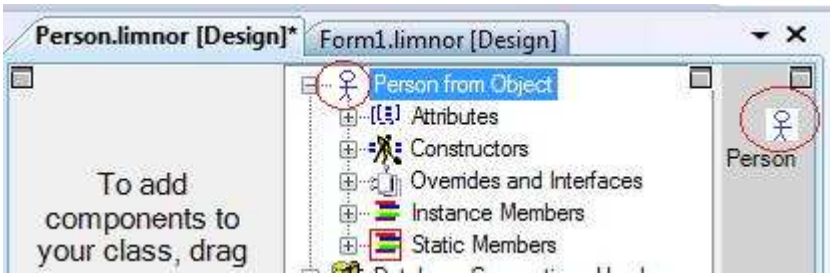
We may change the icon of the new class via the Object Explorer. Right-click the class, choose “Change icon”:



Locate the image file we want to use:



The icon for the class is changed:



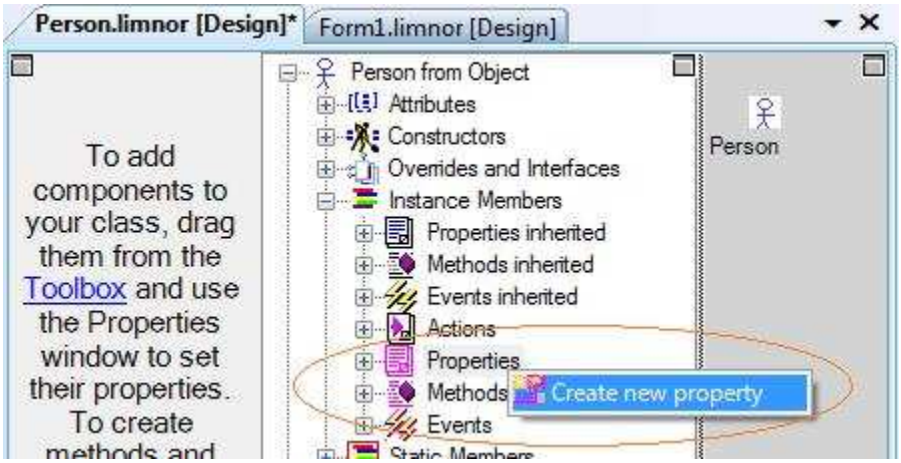
2 Create Properties

2.1 Add new properties

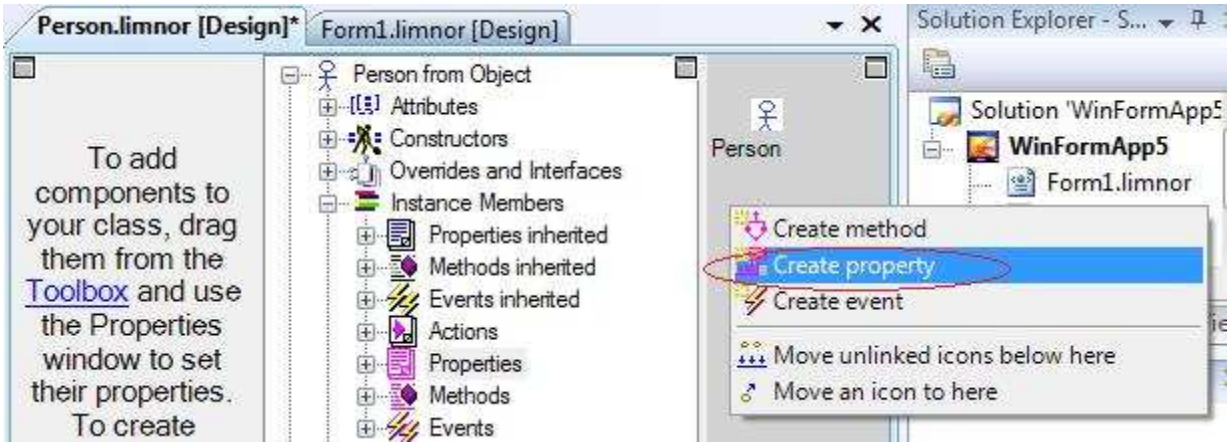
We want to let the Person class have properties such as FirstName and LastName.

The Object Explorer and Event Path both can be used to add new properties to a class.

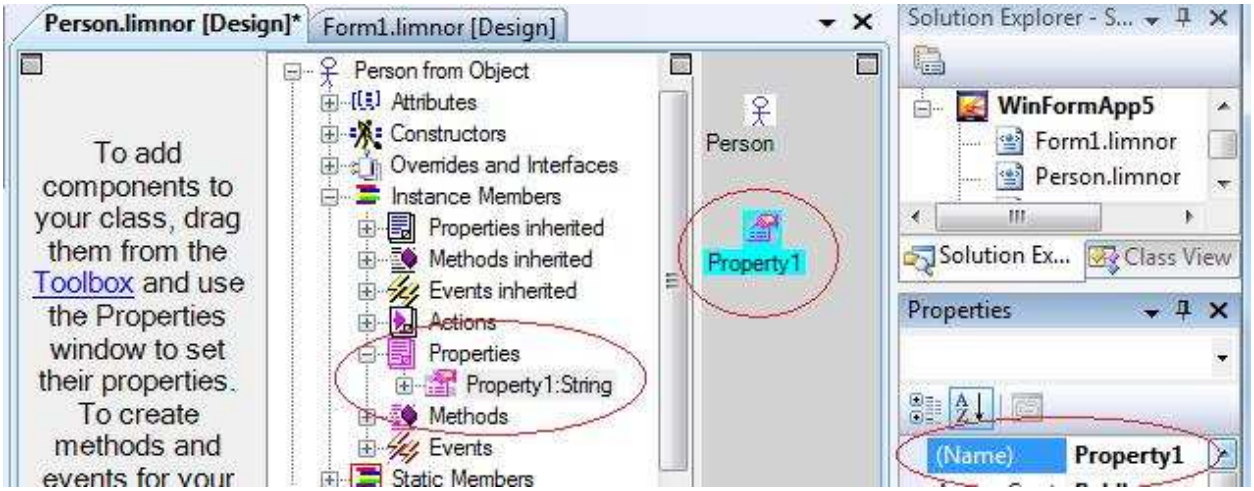
To use the Object Explorer to add a new property, right-click Properties, choose “Create new property”:



To use the Event Path to add a new property, right-click the Event Path, choose “Create property”:

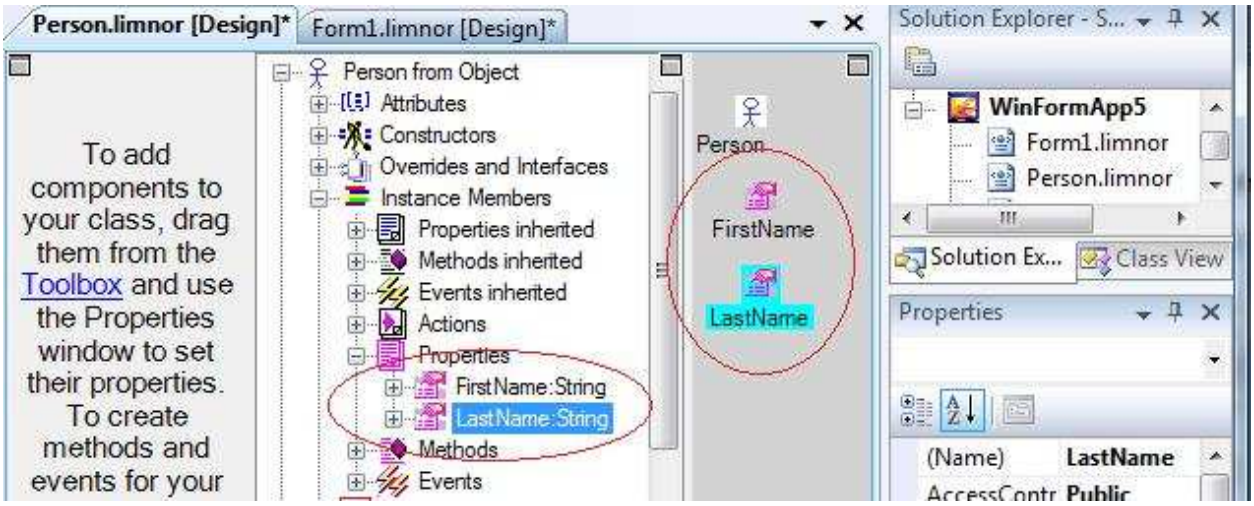


A new property appears under Properties in the Object Explorer, and shown as an icon in the Event Path:



Change the name for the new property from Property1 to FirstName:

In the similar way, we add a LastName property.

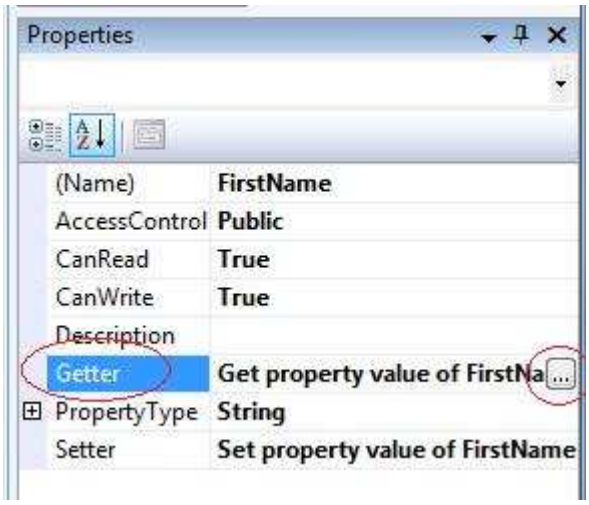


2.2 Property Getter and Setter

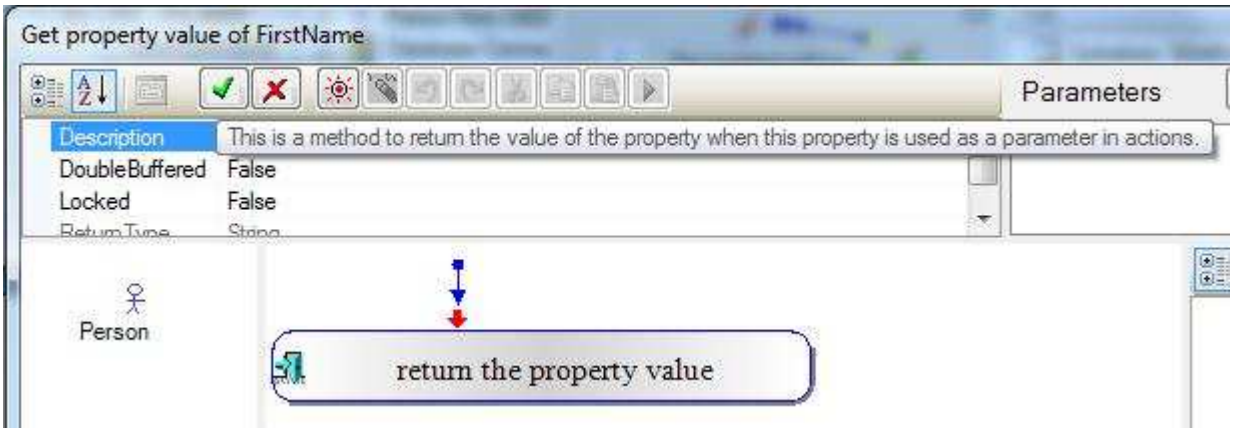
When a new property is created, two special methods are created for it: Getter and Setter.

2.2.1 Getter

The Getter method is executed whenever the property is accessed for its value. We may edit this method by clicking [...]:



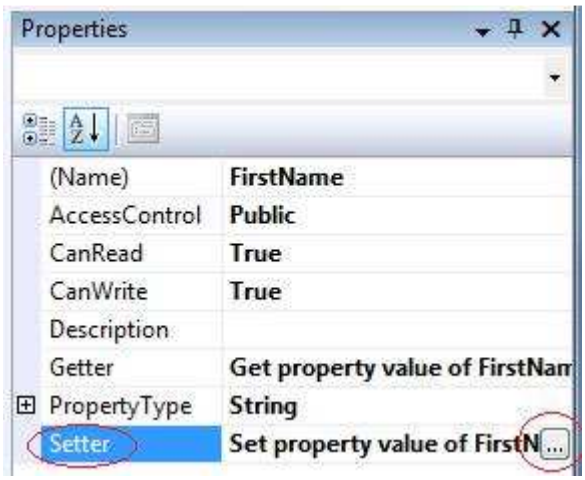
By default this method has one action which returns the internal value stored for this property:



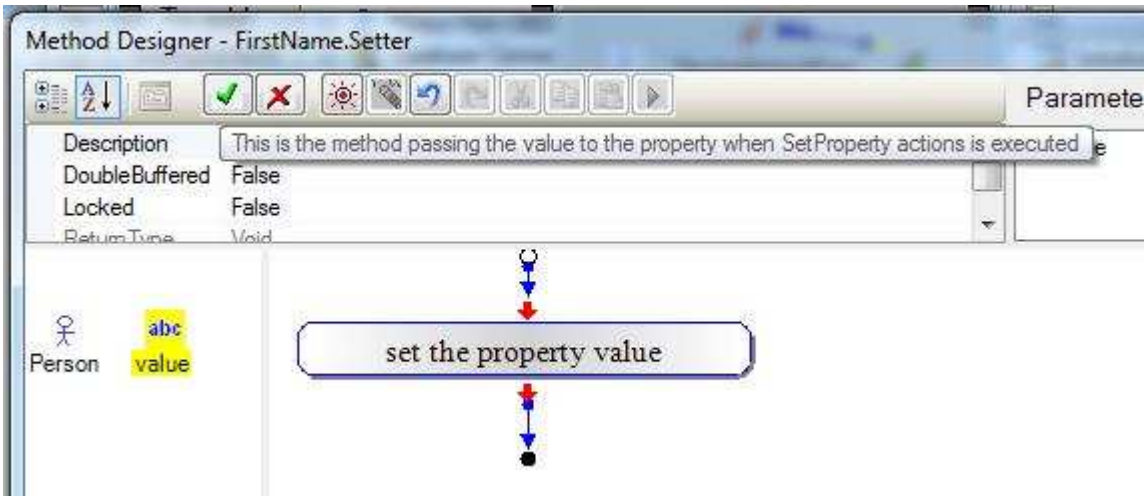
We may add other actions if needed.

2.2.2 Setter

The Setter method is executed whenever a set-property action is executed for this property. We may edit this method by clicking ...:



By default this method has one action which sets the internal value stored for this property to the value provided by the set-property action:



The value icon represents the value parameter of the set-property action. The action "set the property value" assigns the value to the internal storage for this property.

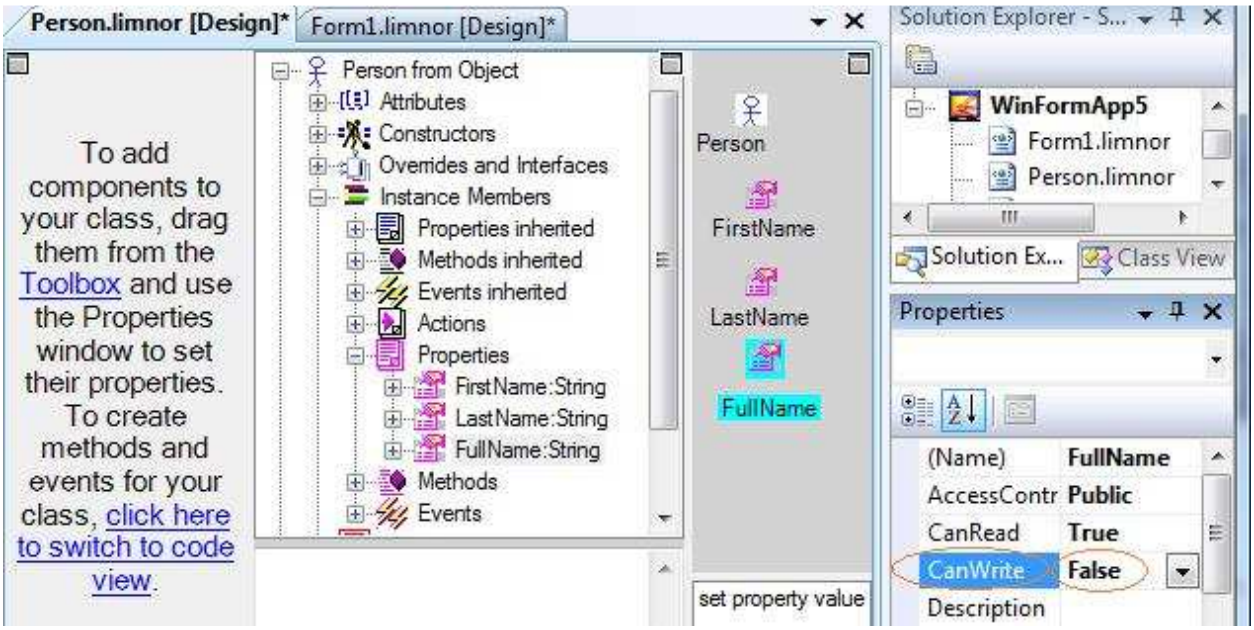
We may add more actions in this method if needed.

2.3 Create read-only property

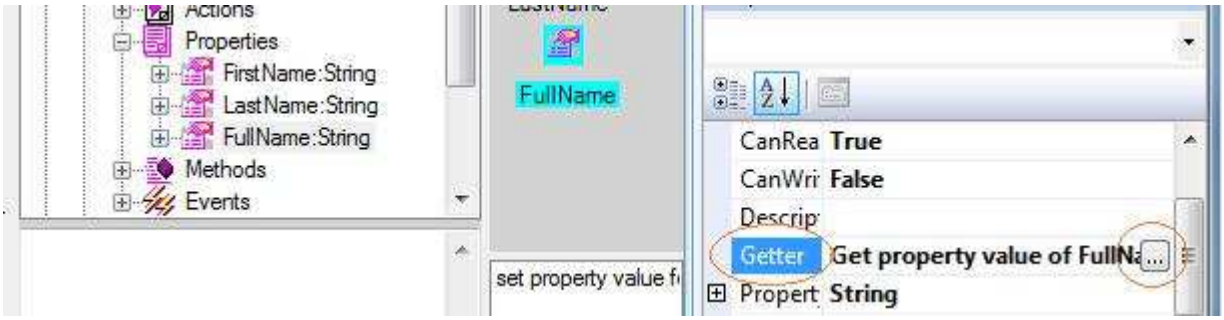
We may add a FullName property.

The value of FullName is formed by FirstName and LastName. We do not want the value of FullName to be changeable because if the value of FullName can be changed then it will mismatch with the FirstName and LastName properties.

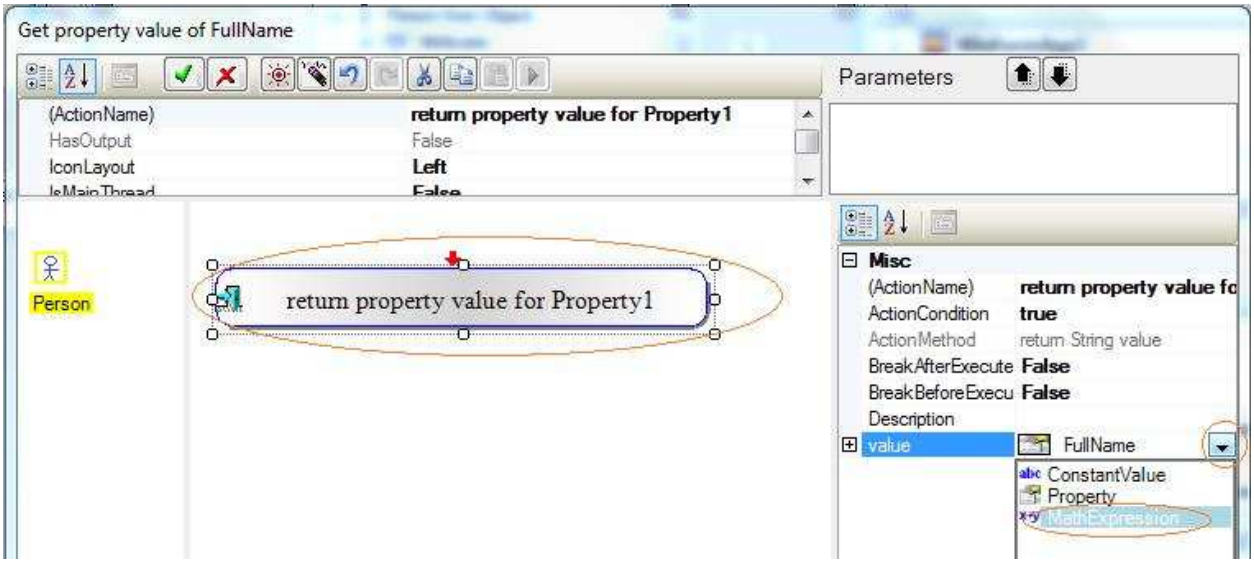
To prevent FullName from being changed, set its CanWrite property to False:



Since the value of FullName cannot be set by an action, its value must be generated by itself. This is done by editing the Getter. Click … to edit Getter:



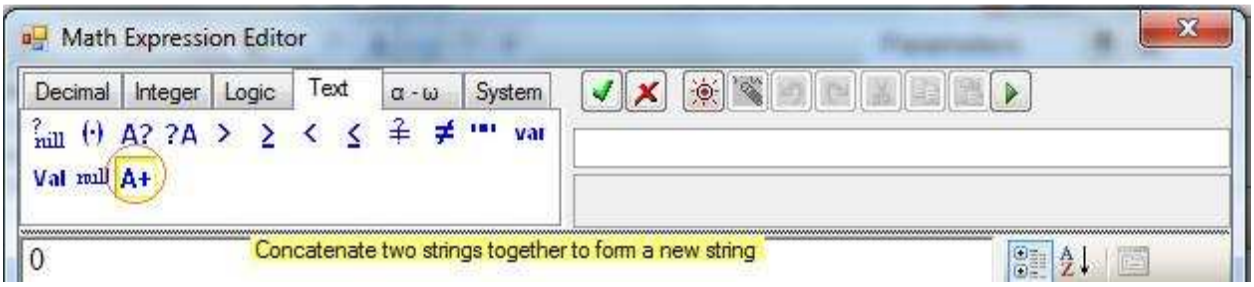
The method editor appears for editing Getter. By default the Getter has just one action to return the property value. This action has a value parameter which is the value for the property. We need to form the value parameter using the FirstName and LastName. Click ▼ and choose "Math Expression" to create an expression for forming the full name:



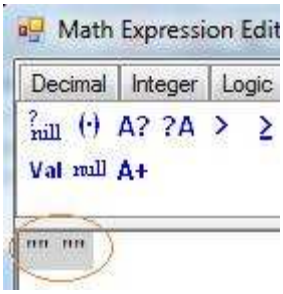
The Expression Editor appears. We want to concatenate FirstName and LastName to form the full name:

{FirstName}{a space}{LastName}

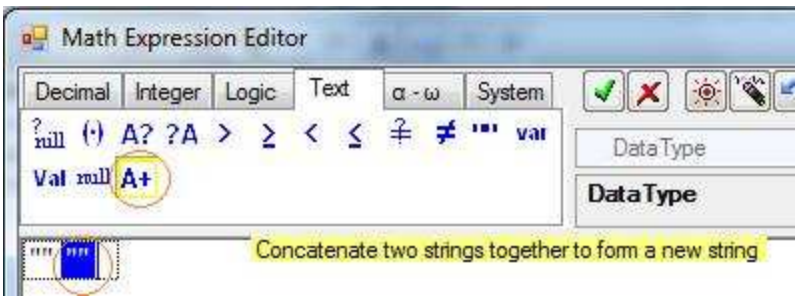
It is formed by 3 strings concatenated. So, click string concatenate operator A+:



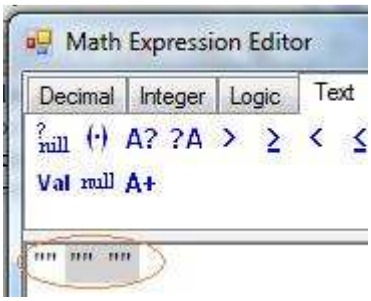
On clicking A+ the expression is formed by two empty strings:



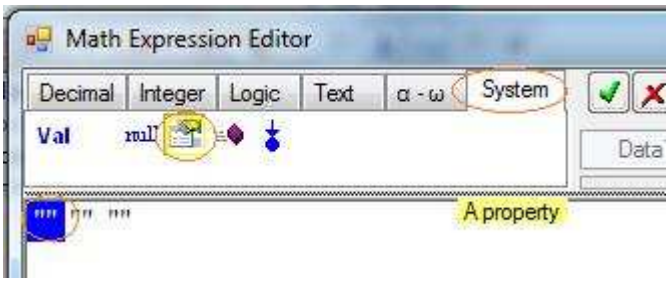
Select the second empty string and click A+ again:



Now the expression is formed by 3 empty strings:



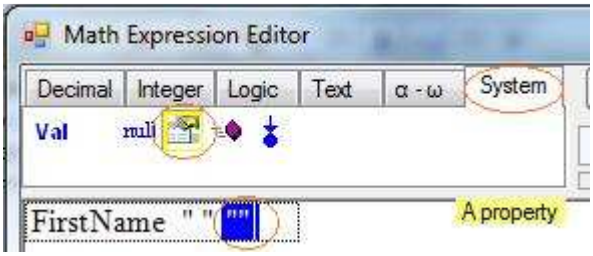
Select the first empty string, click Property icon to use the FirstName property:



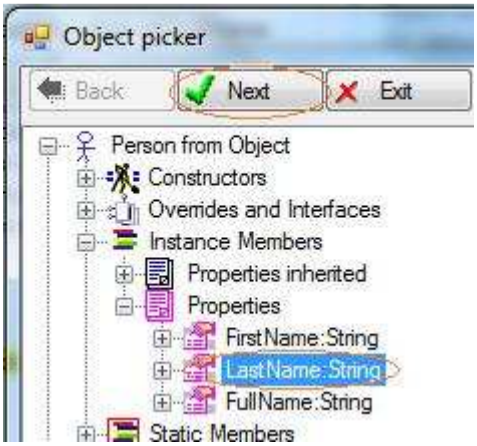
Select property FirstName and click Next:

The FirstName becomes the first string. Select the second empty string, type a space:

Select the last empty string; click the property icon for using the LastName property:

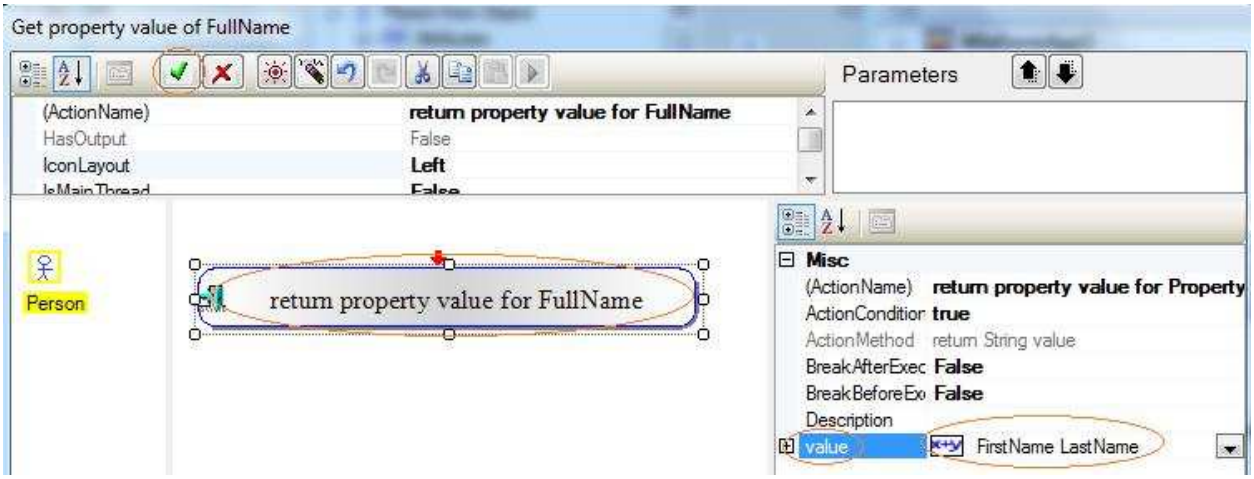


Select property LastName and click Next:



Click  to finish creating this expression:

Click  to finish modifying Getter:

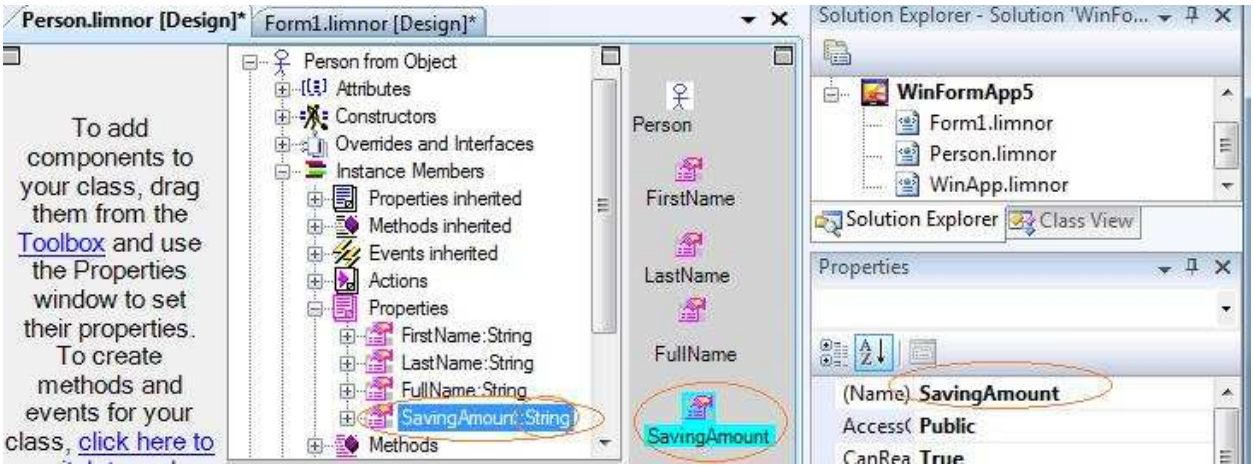


The Getter is thus changed.

A property also has a Setter for setting the value when a "Set property" action is executed if the property is not read-only. It can also be modified to add functionality beyond the default behavior. We will do it later.

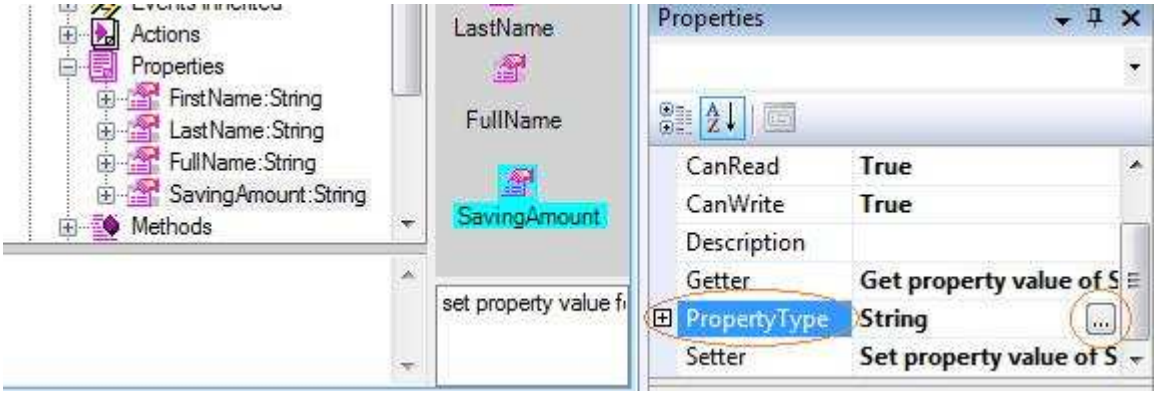
2.4 Change property type

Let's add another new property named SavingAmount:

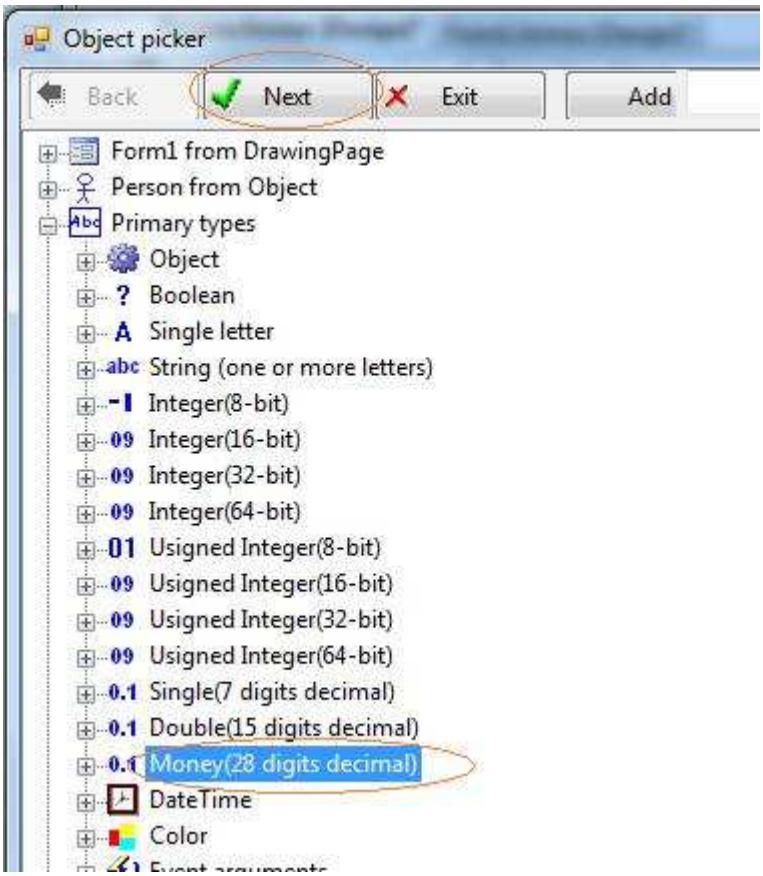


By default, a new property is a string for text data. SavingAmount should be a decimal number instead of a string.

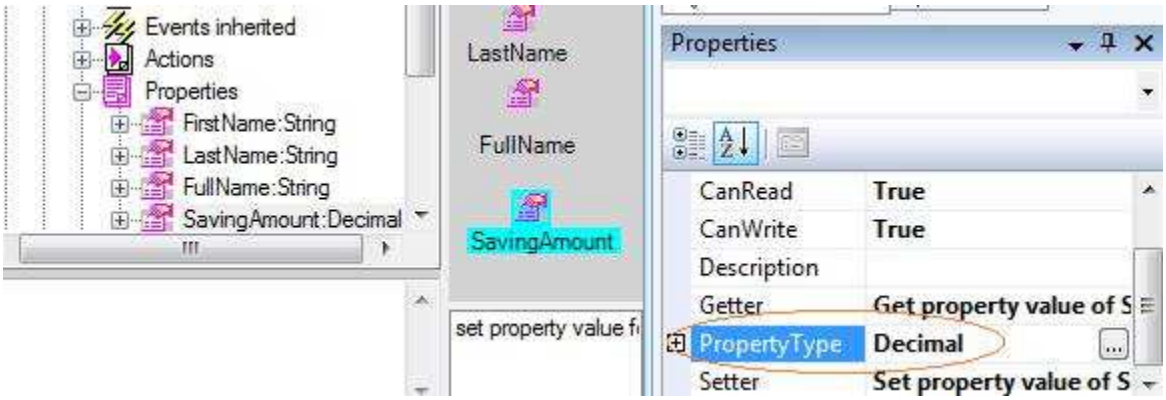
To change property type, click ... for selecting PropertyType:



Select Money as the property type, click Next:



The property type is changed:



3 Create Methods

Suppose we want use a text file to save the Person data in a comma delimited format:

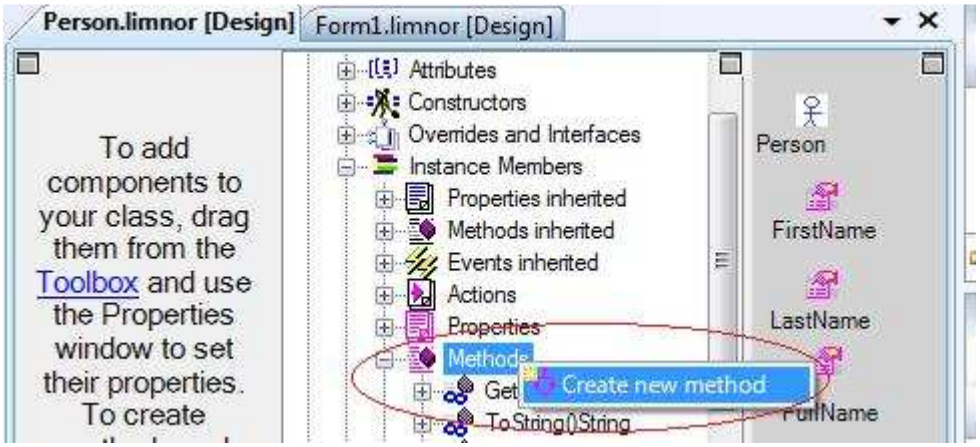
First name,Last name,Saving amount

We want to create a method to load Person data from a text file, and create another method to save Person data to a text file.

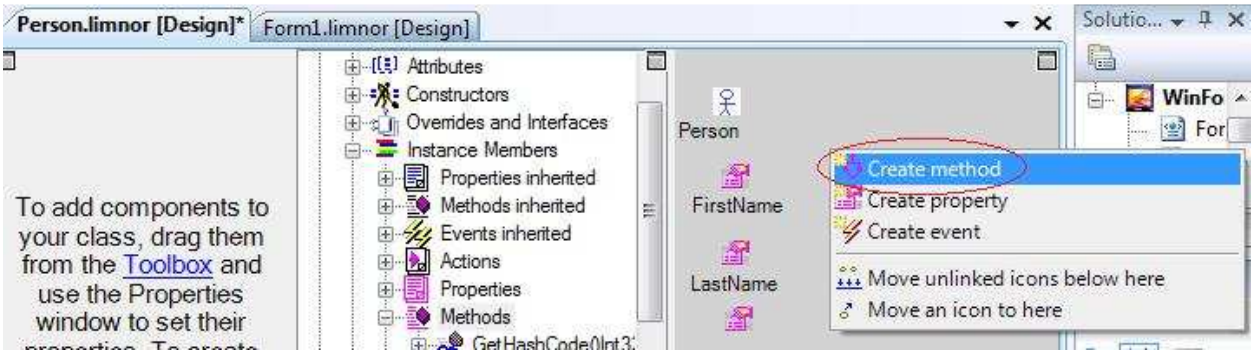
3.1 Add new methods

The Object Explorer and Event Path both can be used to add new methods to a class.

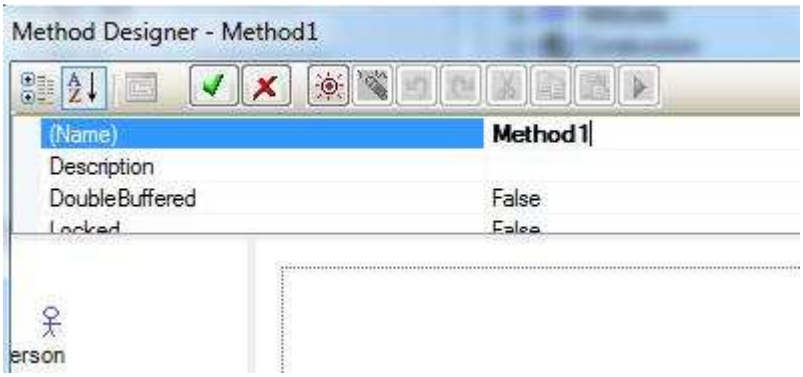
To use the Object Explorer to add a new property, right-click Methods, choose “Create new method”:



To use the Event Path to add a new method, right-click the Event Path, choose “Create method”:

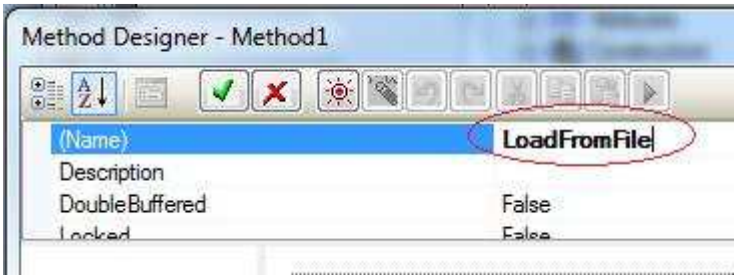


The Method Editor appears:



We already used the Method Editor when we were creating event-handler methods and modifying Getter of property FullName.

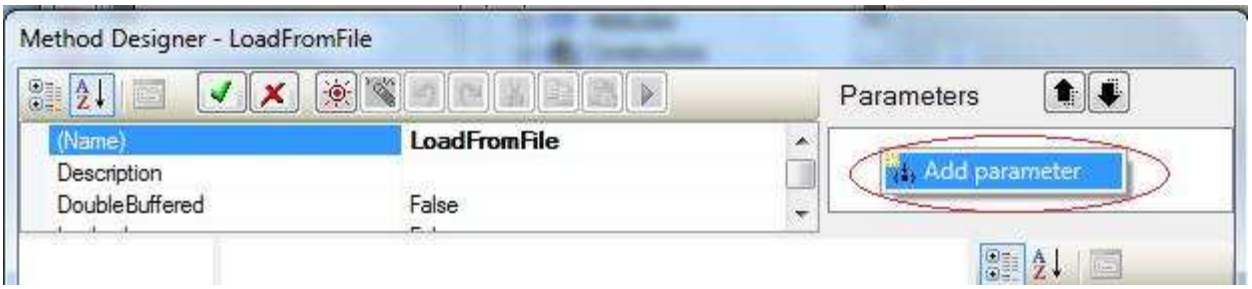
We change the method name to LoadFromFile:



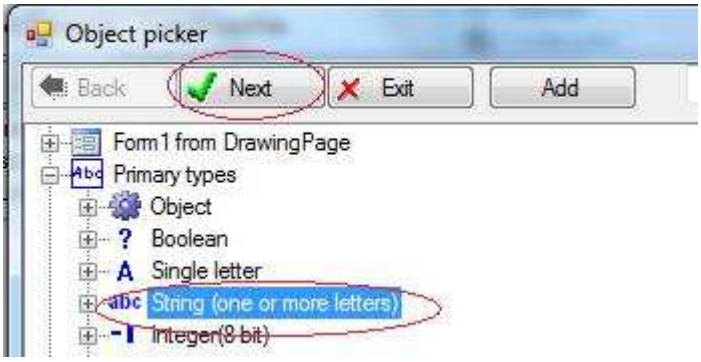
3.2 Add method parameter

We do not want to hard code the text file name to be a constant file name. We may add a method parameter to represent the file name. Later when actions are created to execute this method, the actual file name is used for each action.

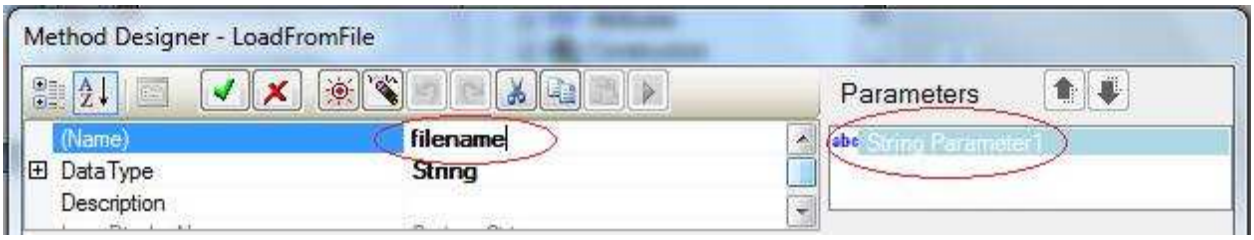
To add a parameter, right-click the parameter list, choose “Add parameter”



It asks for data type for the new parameter. We select String because a file name is represented by text:



The new parameter appears in the list as parameter1. Select it and change its name to filename:

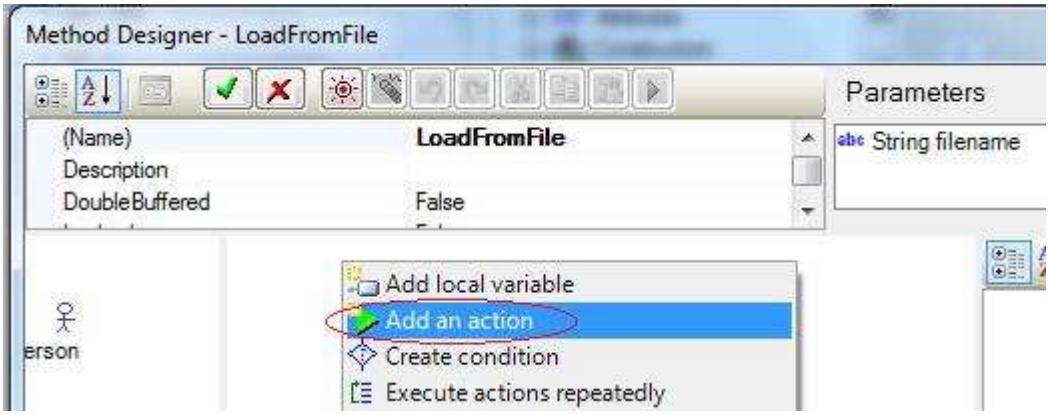


3.3 Add actions to method

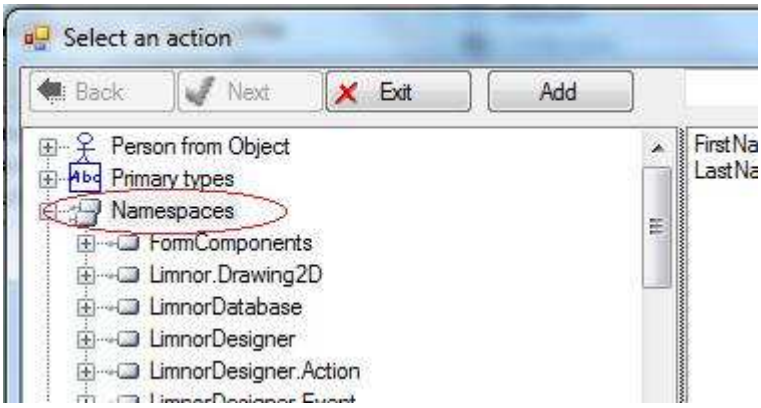
Existing actions can be added to the method. We may also create new actions within the method. Actions created within the method are only visible within the method.

3.3.1 Create action using static method

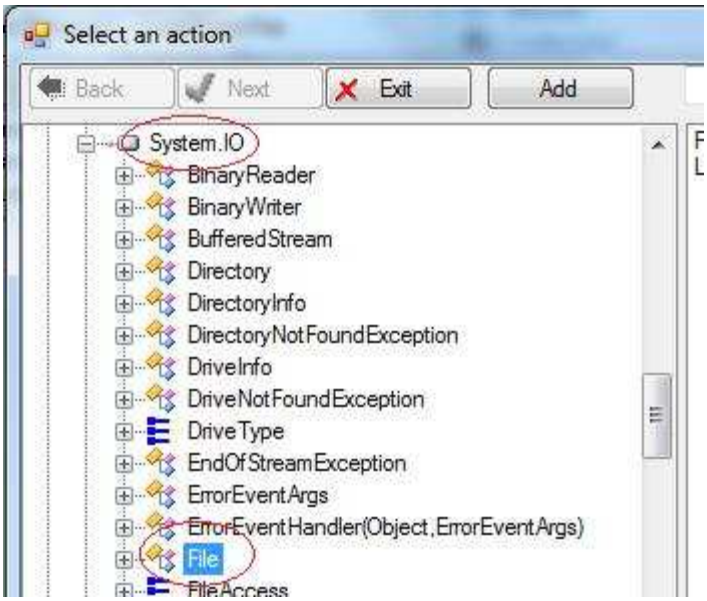
In the Microsoft .Net Framework library, there is a File class which has a ReadAllText method. It is a static method. A static method is executed by the class, not by an instance of the class. Therefore we do not have to create a File variable in order to call the method. So, we right-click in the Method Editor and choose "Add an action" to create an action to execute the ReadAllText method:



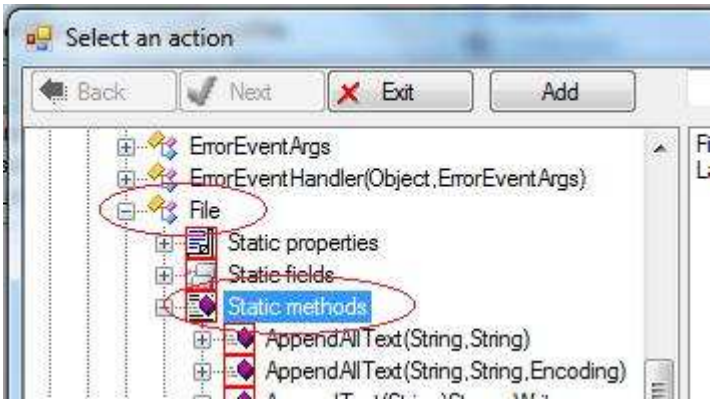
A dialogue box appears for selecting an action. We may use the File class to create a ReadAllText action. We'll find the File class under Namespaces. Most classes can be found under "Namespaces".



File class is under "System.IO" namespace:

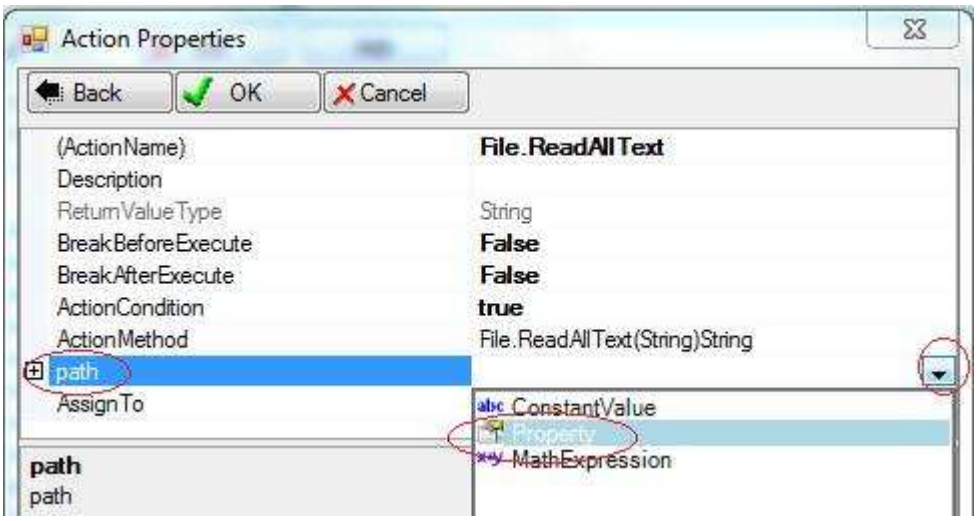


Expand “Static methods” and scroll down to find the ReadAllText method:

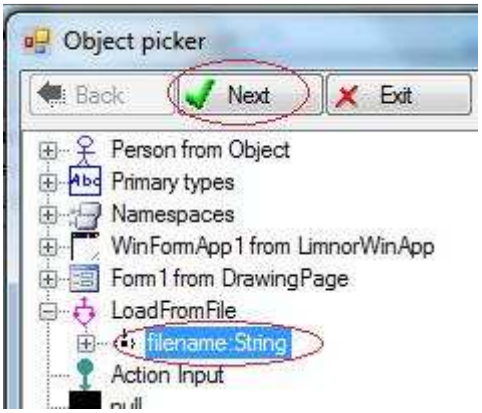


Select ReadAllText method; click “Next”:

This action requires a “path” parameter for specifying the file to read. We need to give it the filename parameter. Click  and select Property to choose the filename parameter:

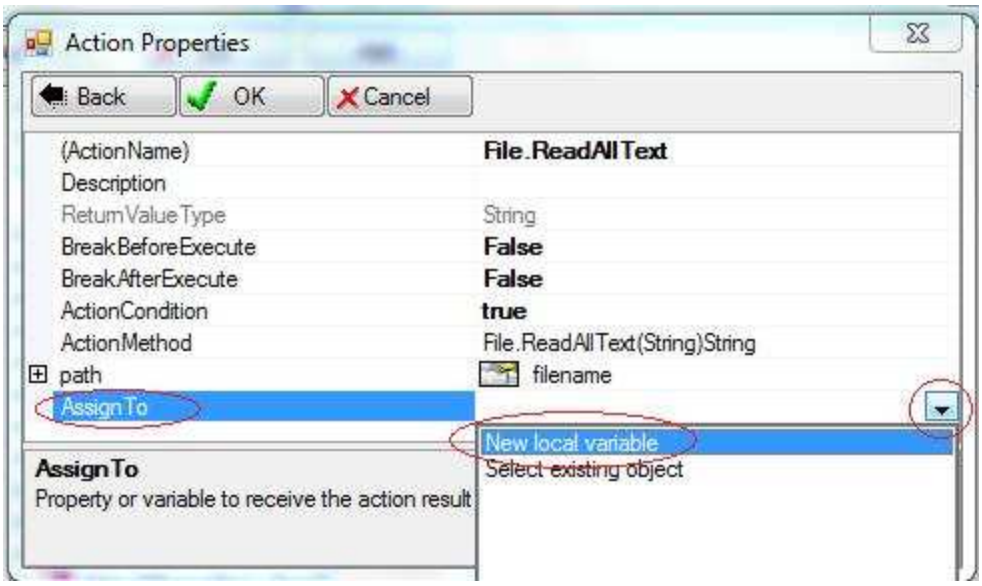


Select filename parameter of this method, click Next:



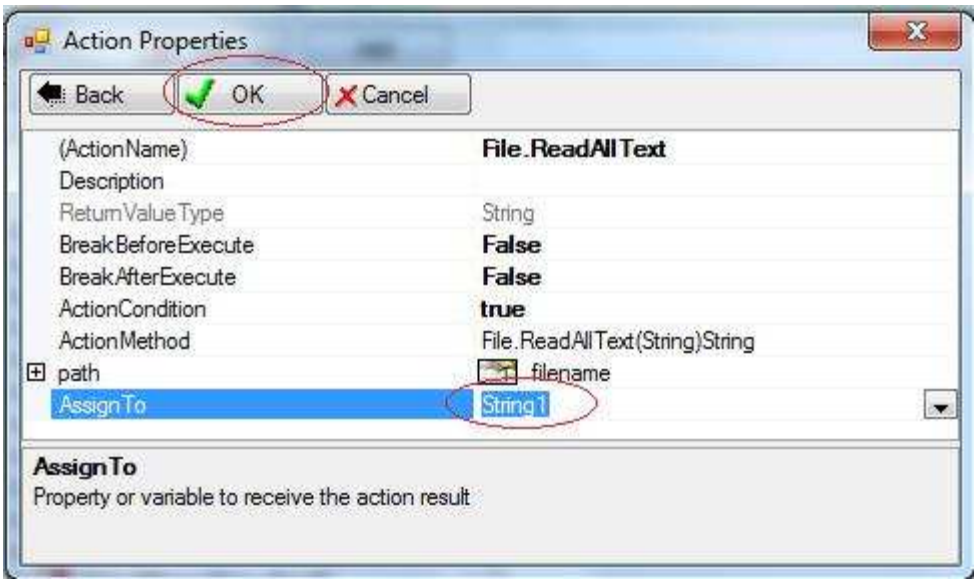
3.3.2 Create local variable to access action result

The "AssignTo" property of this action represents the destination to receive the file contents read from the file. Click  and choose "New local variable" to save file contents into a local variable:

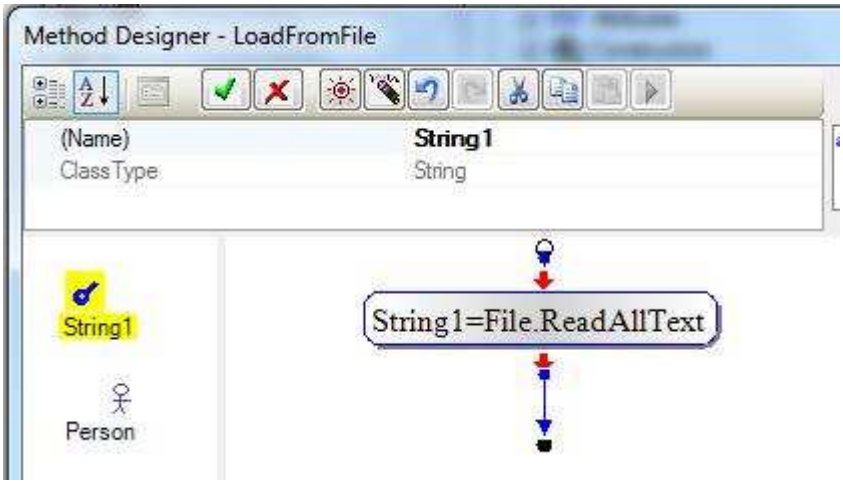


Give a name for the new variable:

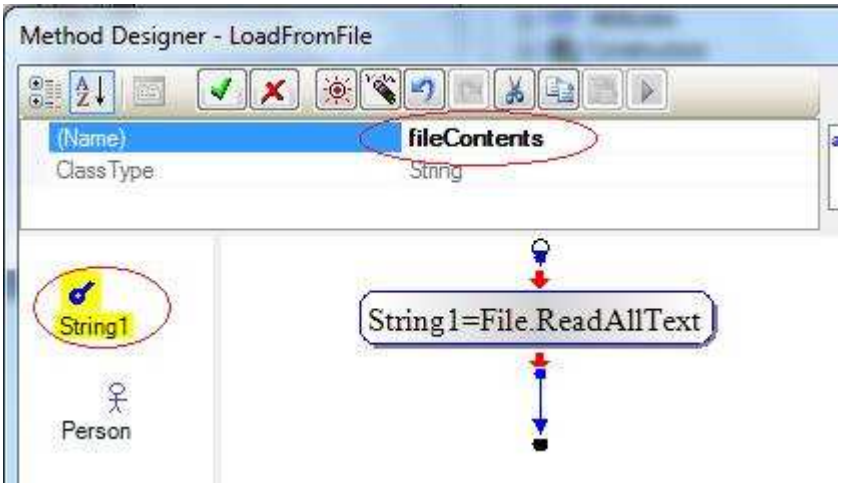
A new local variable named String1 is created:



Click OK. The new action appears in the method editor:



Variable can be renamed. Rename String1 to fileContents to make the variable name more meaningful:



After executing this action, the local variable fileContents contains the contents read from the file. We assume the contents are in following format:

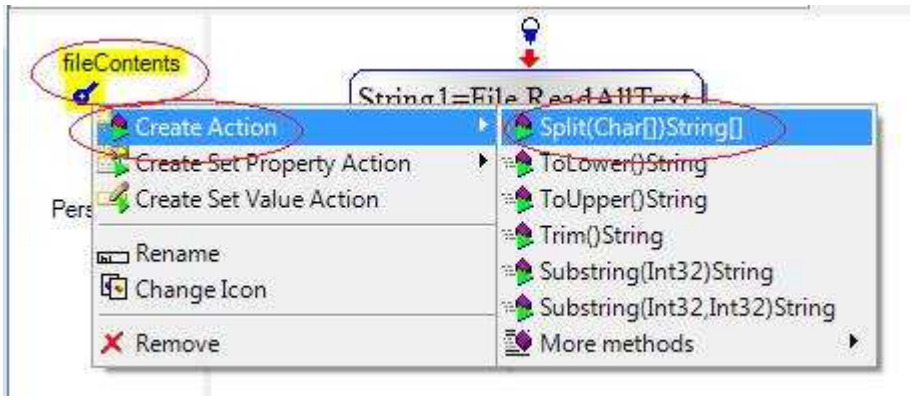
{first name},{last name},{saving amount}

We need to parse the above contents into properties of the Person class: FirstName, LastName, and SavingAmount

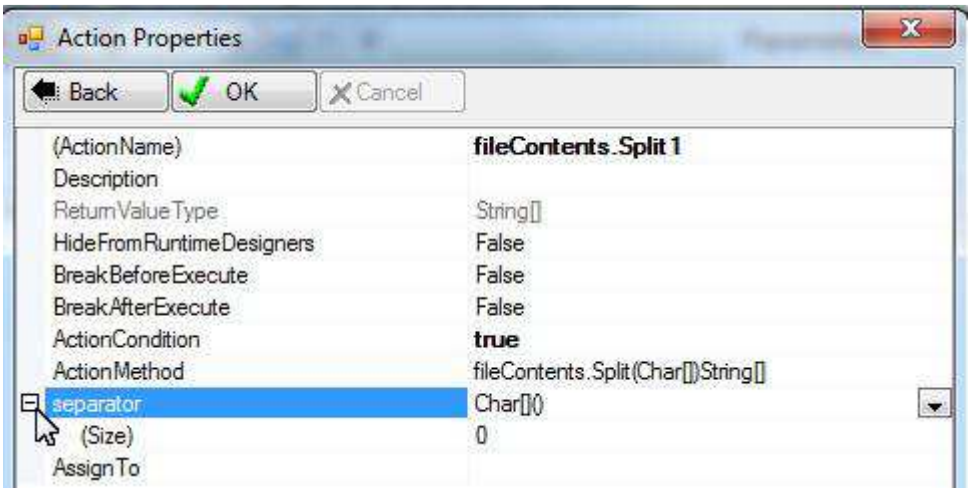
The local variable fileContents is a String. The String class has a Split method which may split string contents into an array. We will use Split method to create an action and add it to the method.

3.3.3 Execute multiple actions sequentially

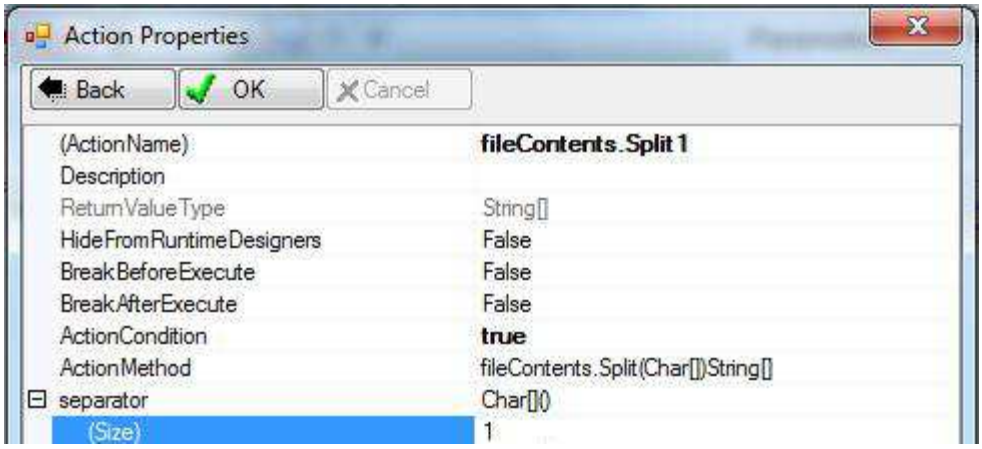
Right-click fileContents, choose “create Action”. Choose “Split” method:



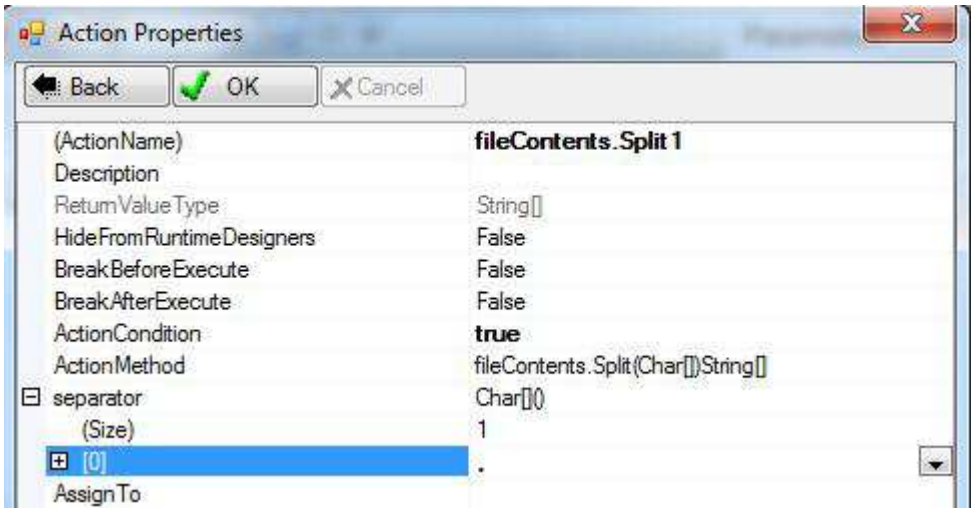
Expand parameter “separator” to enter separators:



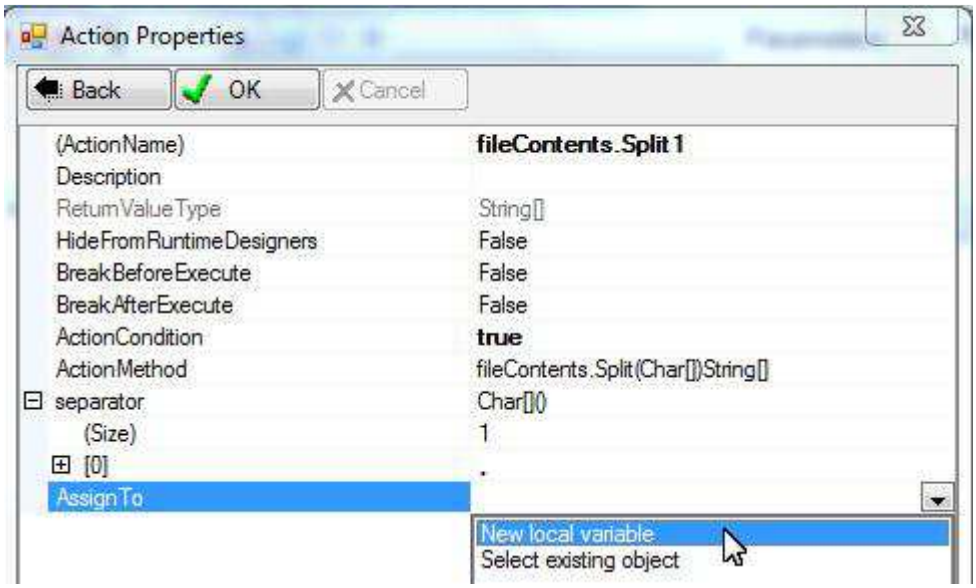
Set (Size) to 1 because we just want to enter one separator:



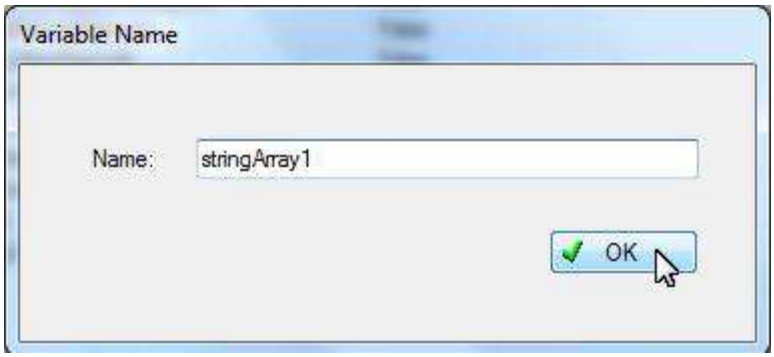
On leaving “(Size)” property, an array item appears. Enter comma for this array item as the separator, because we assume the Person data are separated by commas:



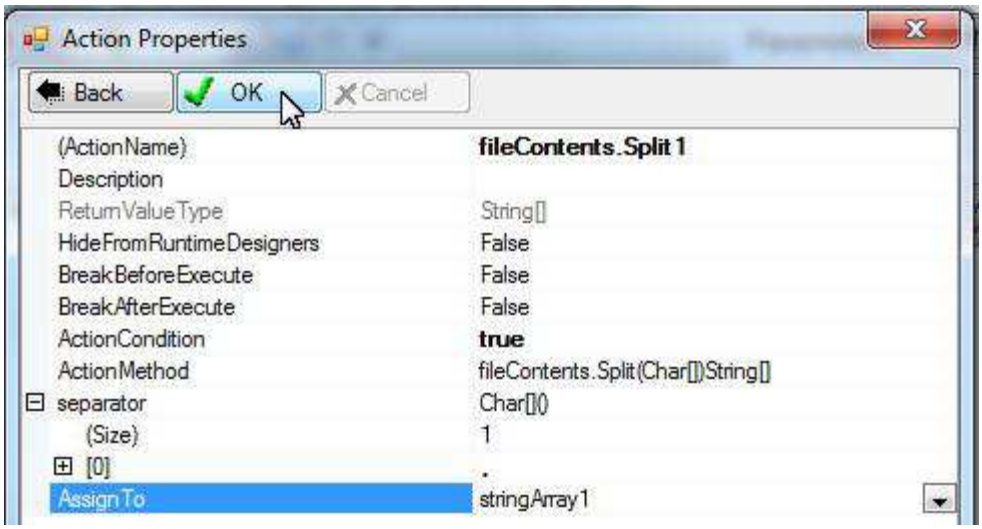
Use a new local variable to receive the action result:



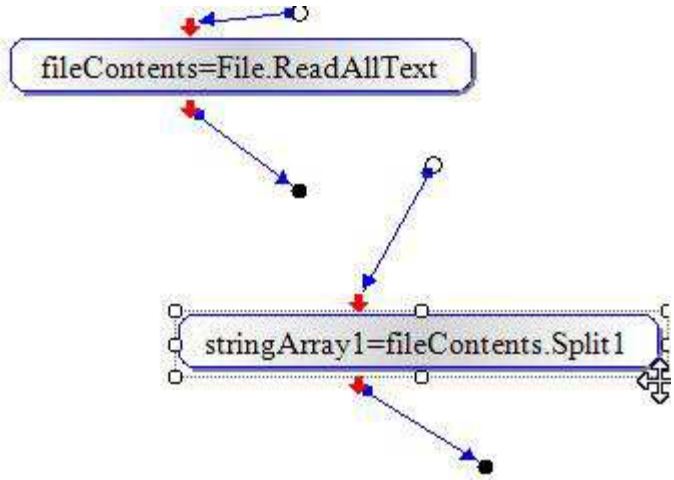
Give a name for the local variable:



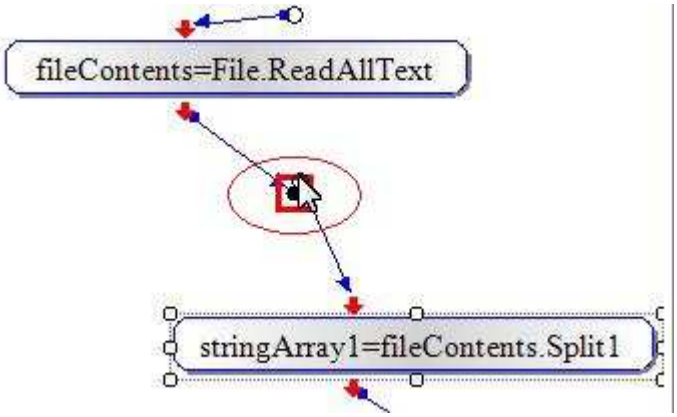
A new local variable named stringArray1 is created to receive the result of Split method:



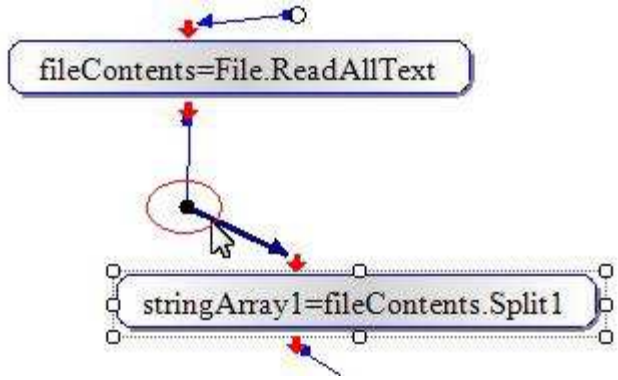
Click OK. The new action appears in the Method Editor:



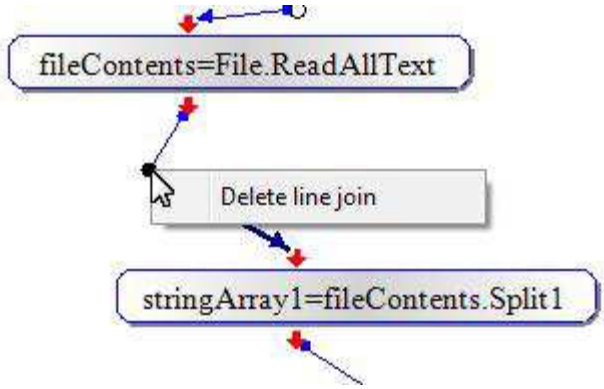
Link the two actions together to form a sequential execution order:



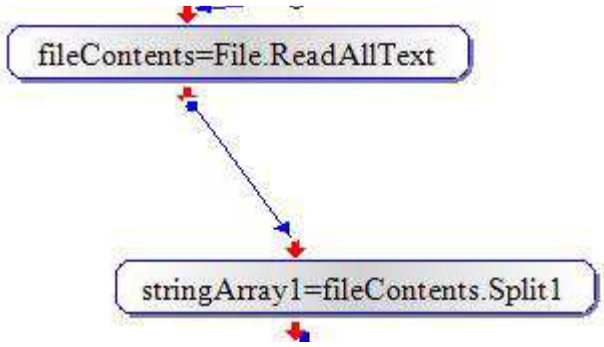
Two actions may be linked by one-way lines. Moving mouse over a line may highlight the line and the lines join:



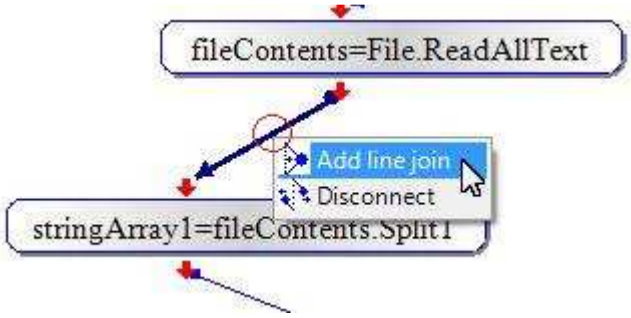
A lines join can be removed by right-clicking it and select “Delete line join”:



The lines join is removed:



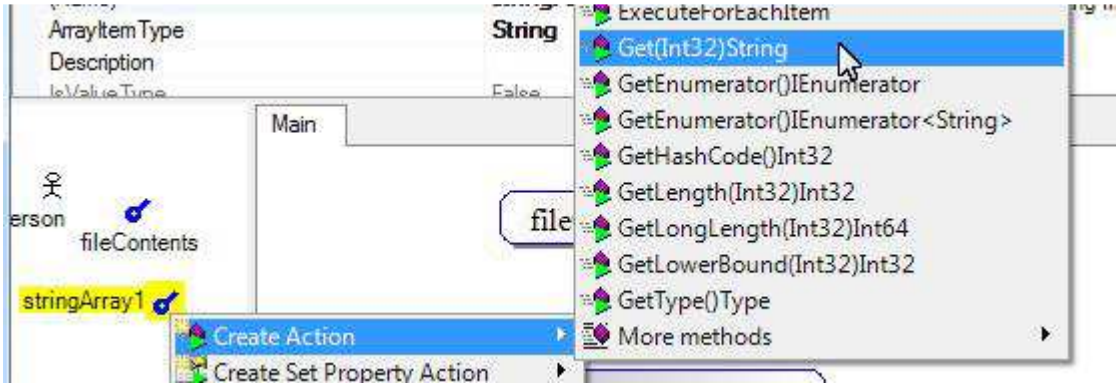
Right-click a line, it allows us to create lines join or break the line:



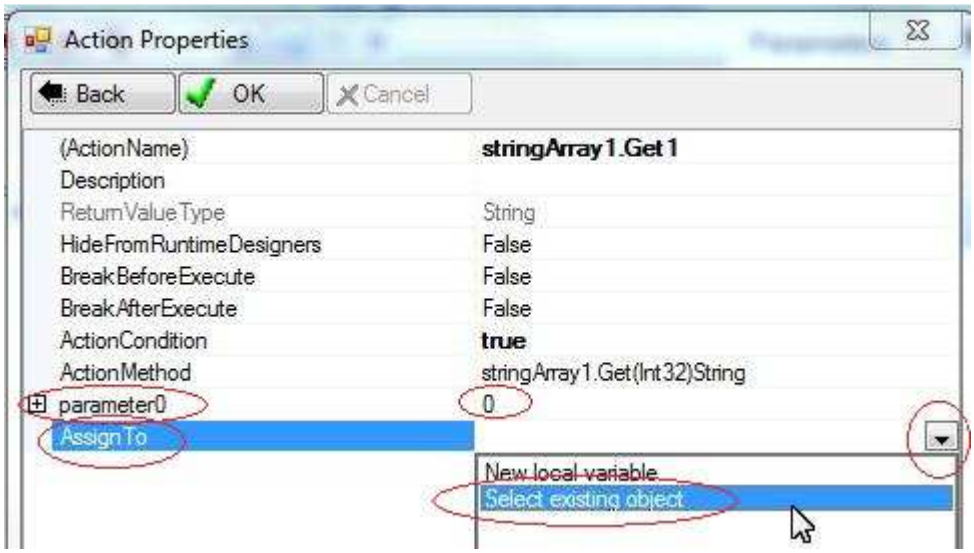
3.3.4 Getting individual array items

The second action splits a string in such a format "first name,last name,saving amount" into an array of 3 items. The first item is first name; the second item is last name; the 3rd item is saving amount. The array is named stringArray1. Now we want to assign these array items to their corresponding properties of Person class: FirstName, LastName, and SavingAmount. This task can be done by the Get method of the array. The Get method uses a parameter to indicate which array item to get. 0 indicates the first item; 1 indicates the second item; and 2 indicates the 3rd item.

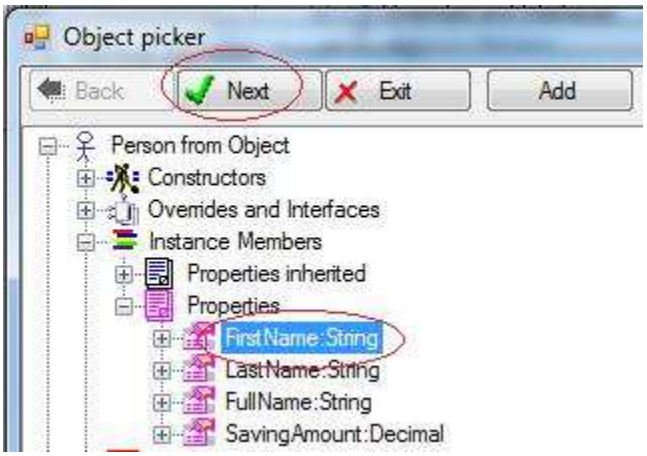
Let's create a Get action to get the first array item and assign it to property FirstName. Right-click the array stringArray1, choose "Create Action". Choose "Get":



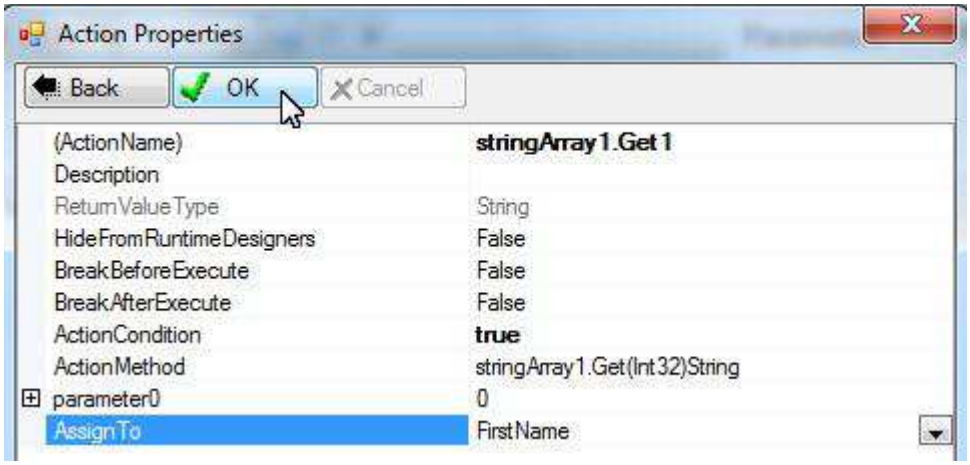
Use 0 for parameter0 for indicating the first array item. For "AssignTo", choose "Select existing object" to select property FirstName:



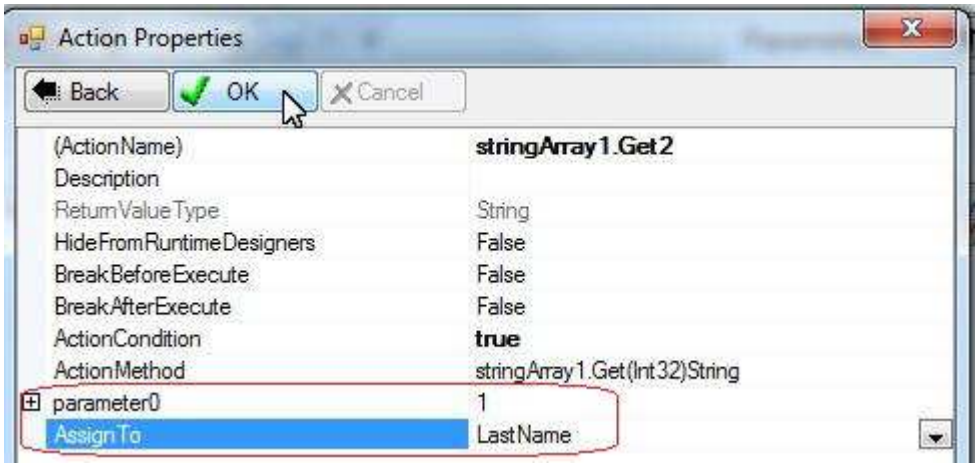
Select property FirstName:



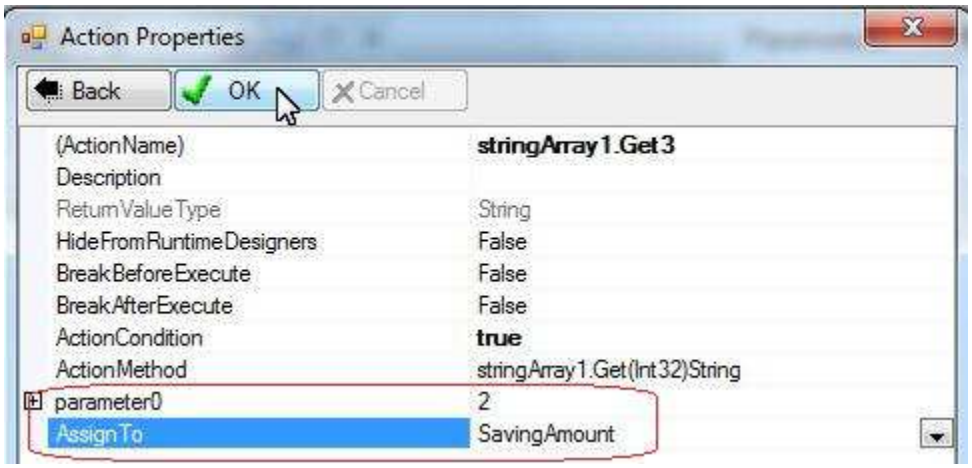
Click OK to finish creating this action. This action gets the first array item and assigns its value to the FirstName property.



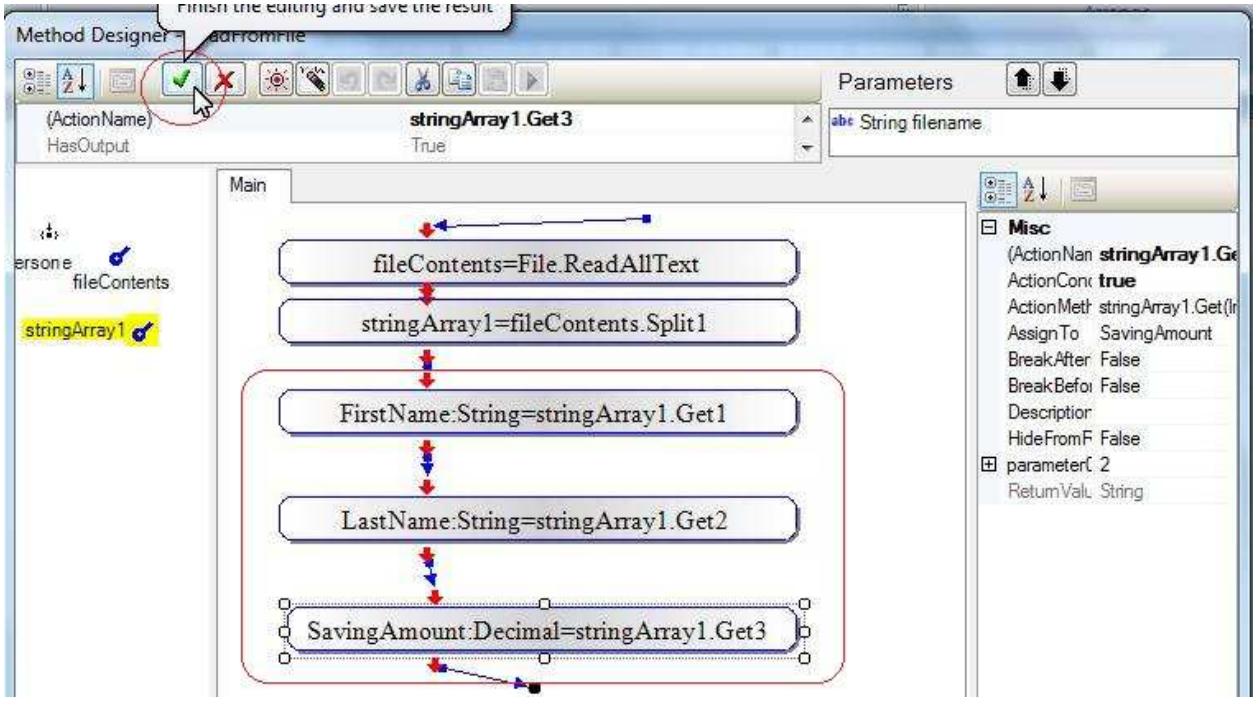
In the similar way we may create another 2 actions. One action is to get the second array item and assign it to the LastName property:



One action is to get the 3rd array item and assign it to the SavingAmount property:



Link the 3 action together and with the two other actions created previously. We are done creating this method:

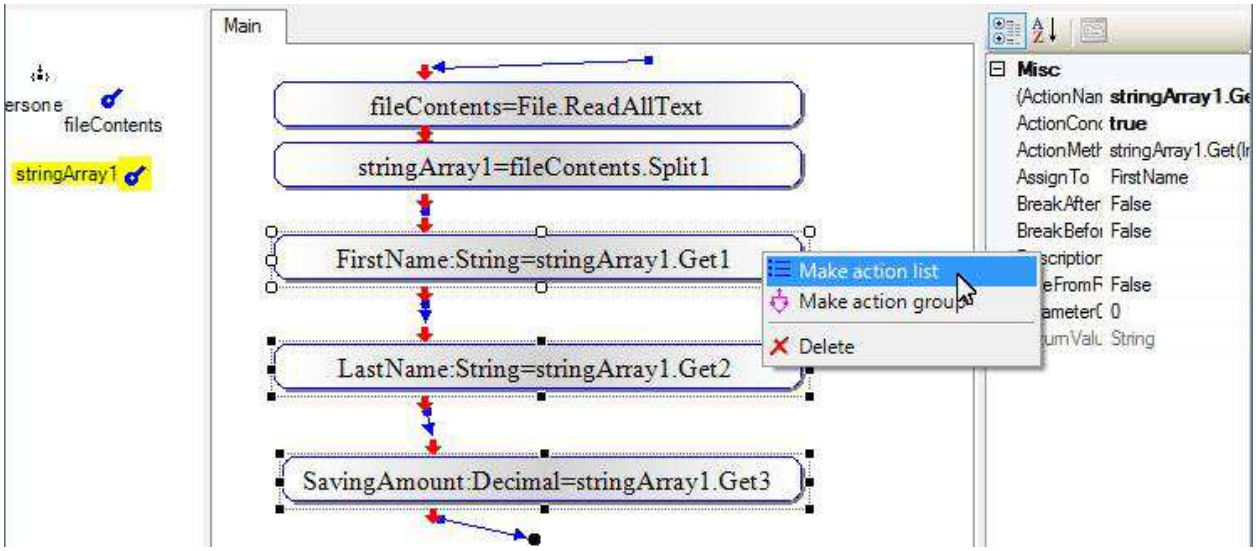


Click  to finish creating method LoadFromFile.

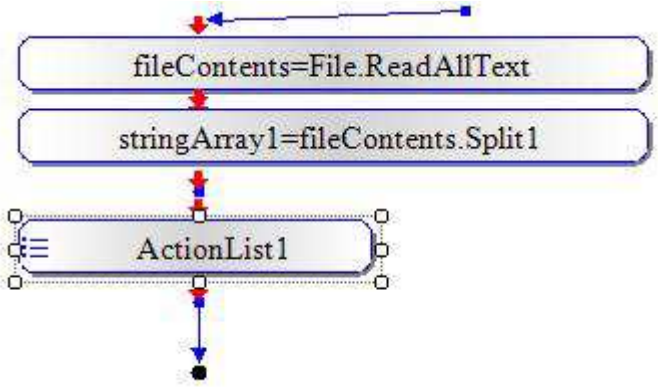
3.4 Make action list

For a sequential list of actions, we may make them into one single action so that the screen looks cleaner and the programming logic is expressed clearer.

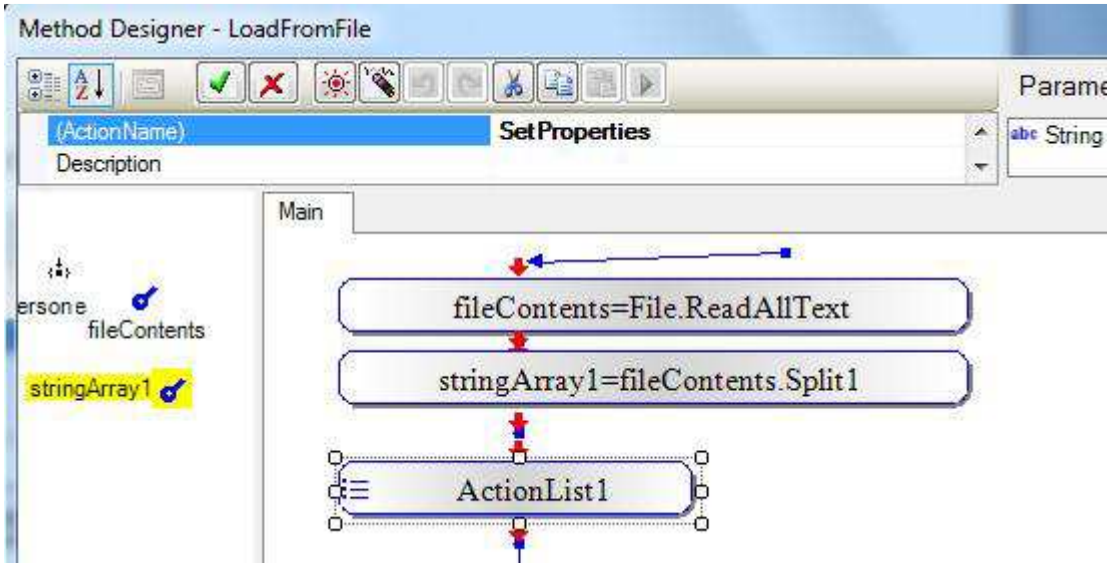
Select all the actions to be included, right-click on the selection, choose “Make action list”:



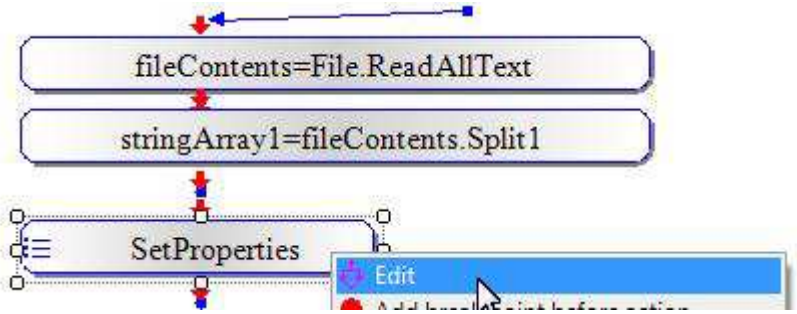
A new action list is created to represent all the actions selected:



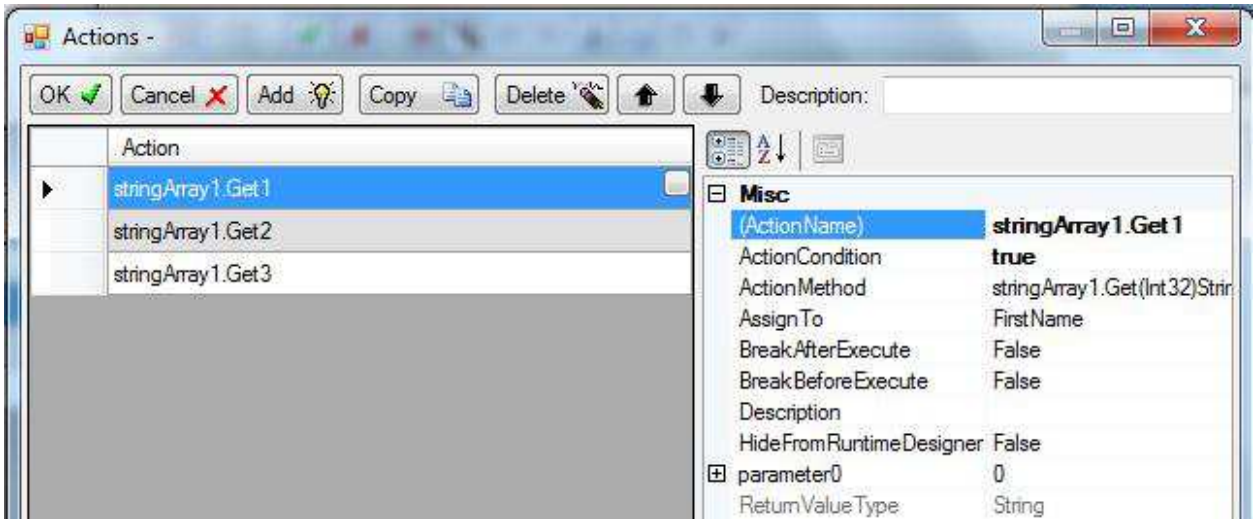
You may change the action name and other properties:



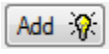
To modify the action list, right-click it, choose “Edit”:



Each action can be edited by select an action and modify its properties:



An action can be replaced by clicking  and select another action.



-- Add a new action to the list

-- Remove the selected action from the list

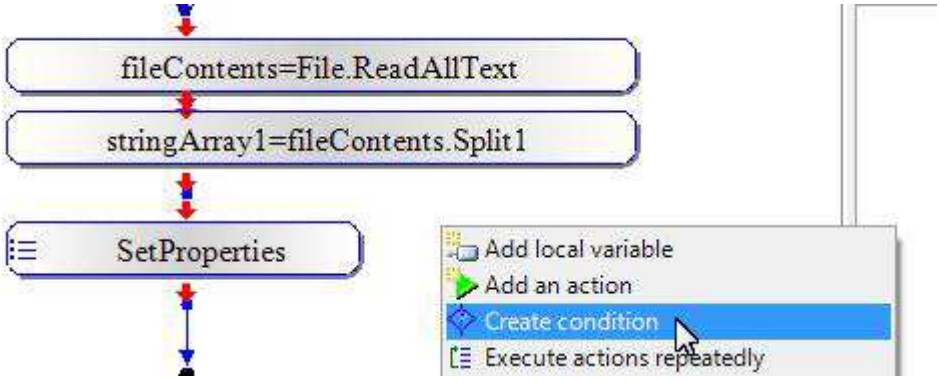
 -- Move the selected action up in the list

 -- Move the selected action down in the list

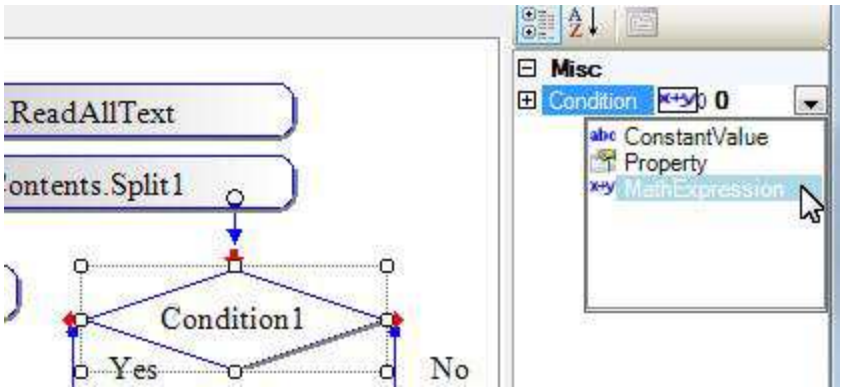
3.5 Action flow branching

The execution of actions one by one forms an “action flow”. Sometimes, under a condition different actions should be executed. It is like forming such a logic: “IF {condition} … ELSE …”. For example, in the above example, we read the contents of a file into an string array and use 3 “set property” actions to assign the first 3 array items to properties FirstName, LastName and SavingAmount. But if the array does not have 3 items then the program will crash. So, we want to execute the 3 actions of setting properties if the length of the array is 3. If it is not 3 then we want to display a message box to let the user know.

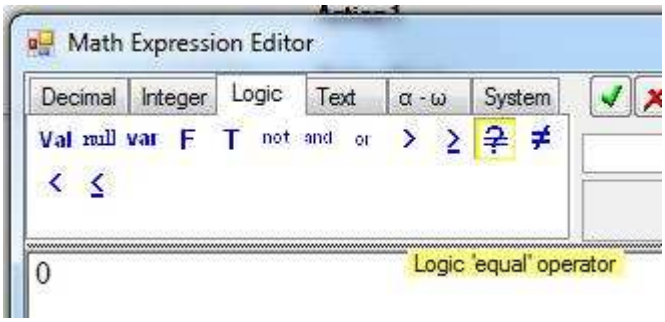
Right-click the middle pane, choose “Add condition”:



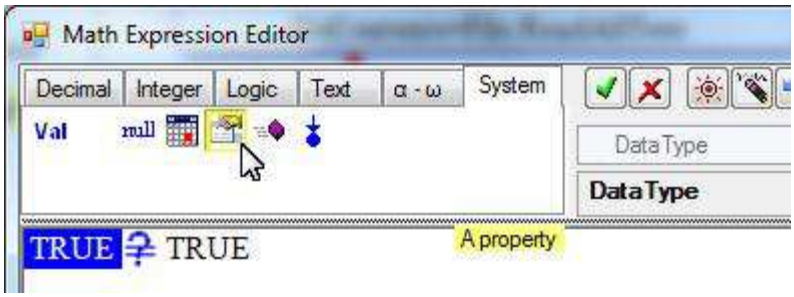
The Condition appears. Set its Condition property to “Math Expression” for checking the number of array items:



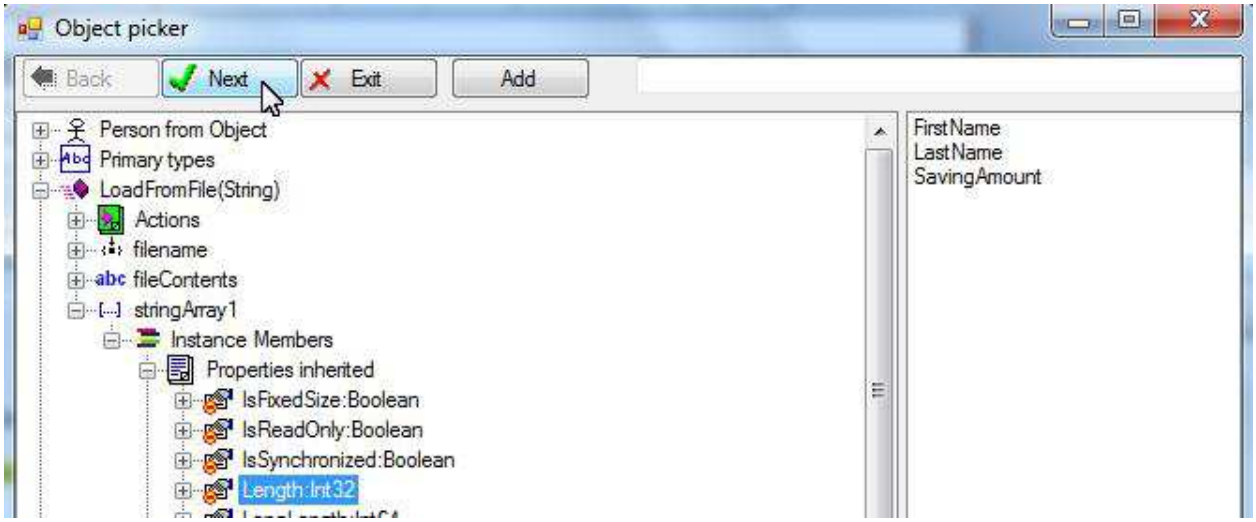
Click  for checking equality:



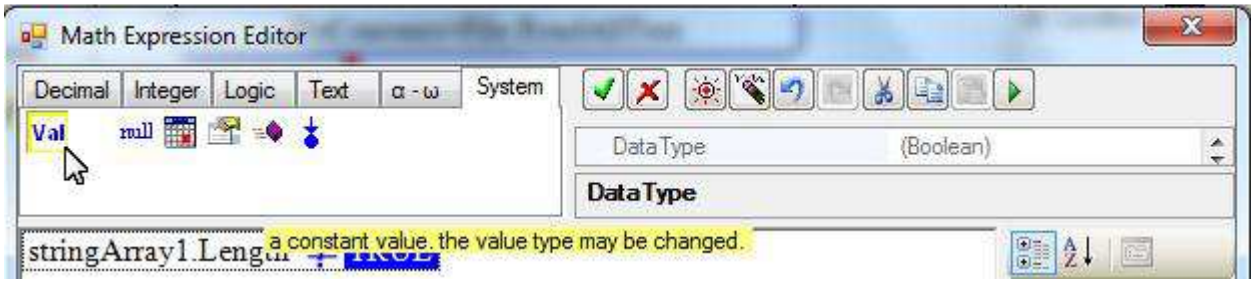
Select the first element, click the property icon  :



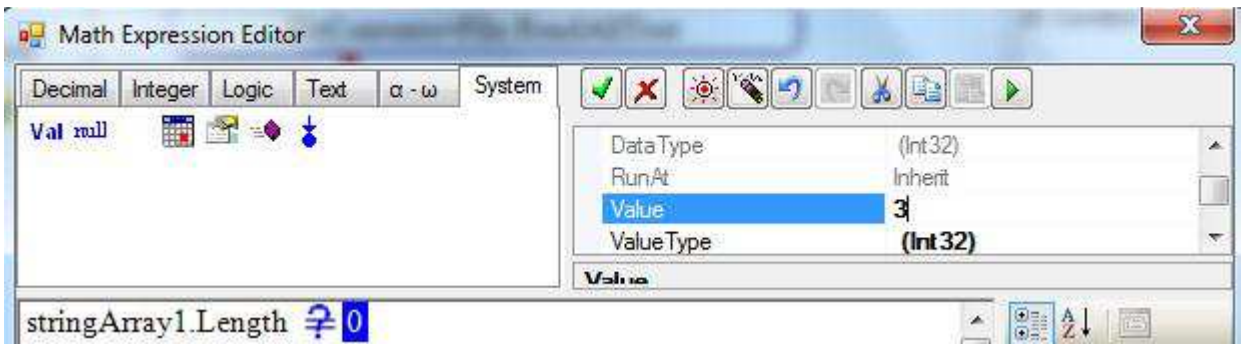
Select the Length of the array, click Next:



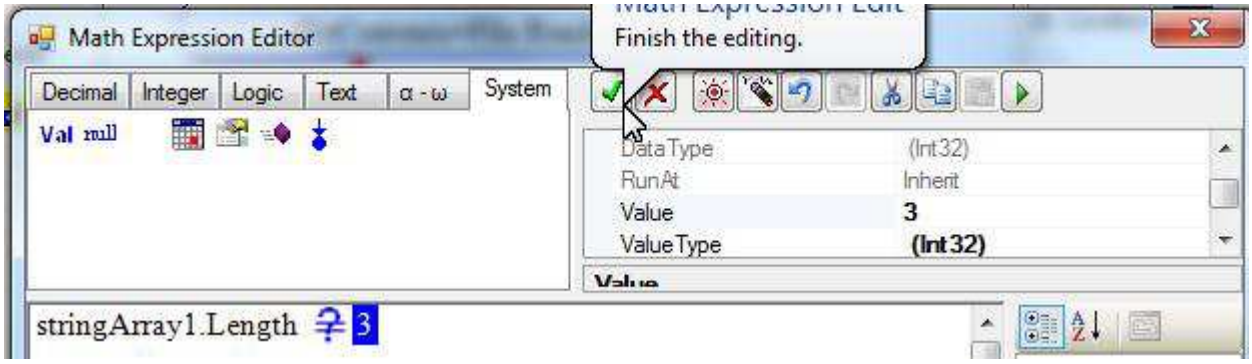
Select the second element and click Val:



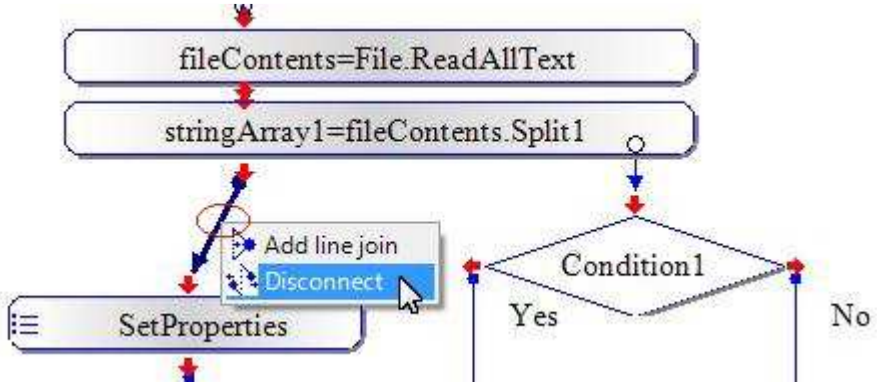
Enter 3 for the value:



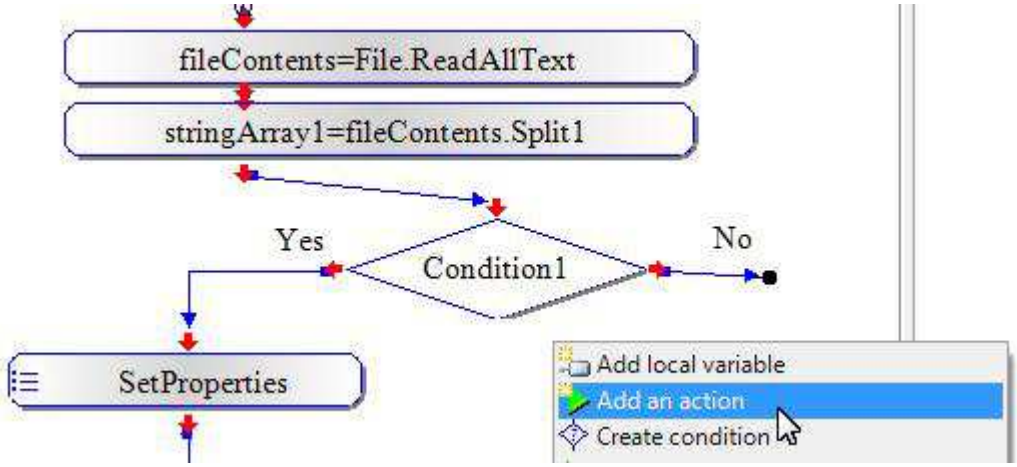
Click  :

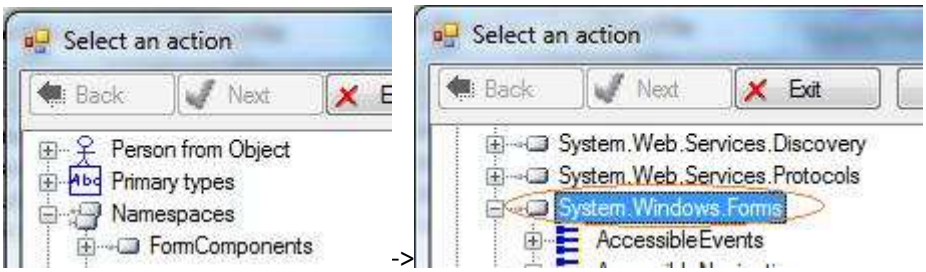


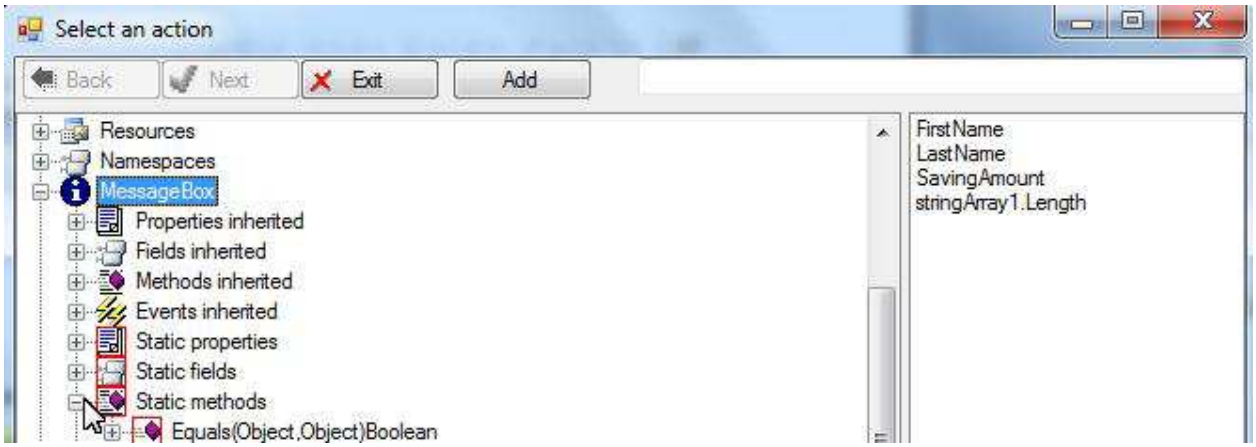
Break the link line to insert the condition:

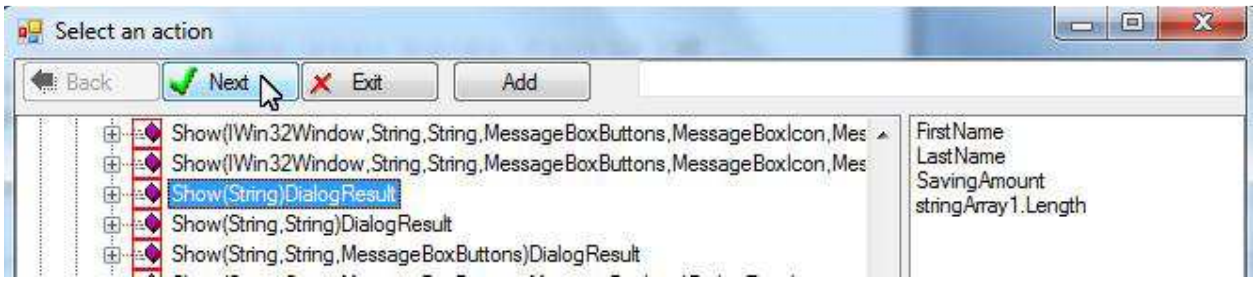


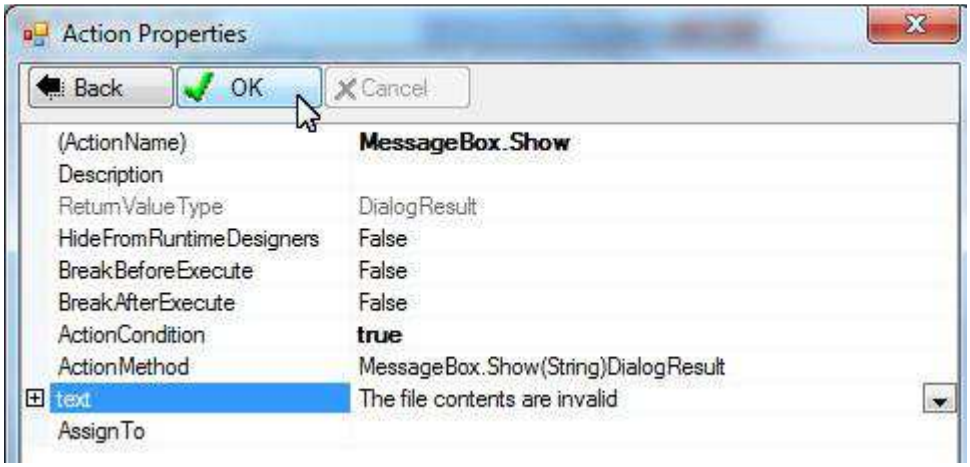
Add a message box action:

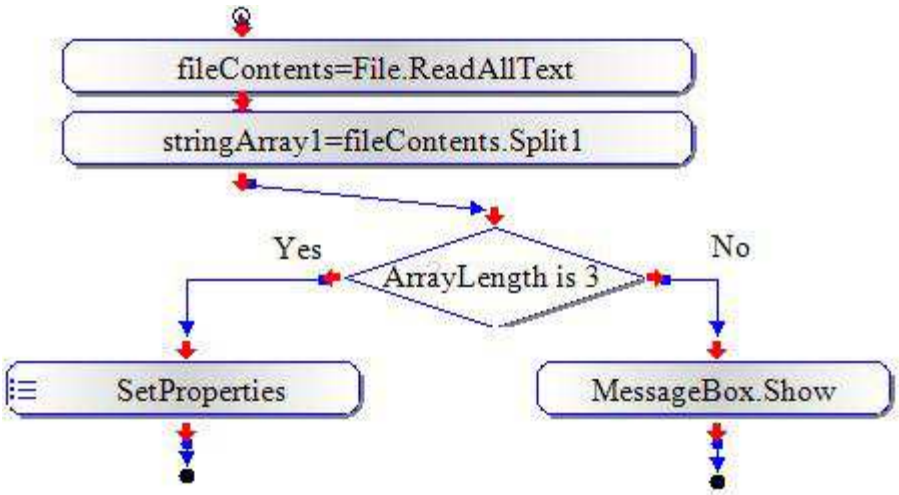










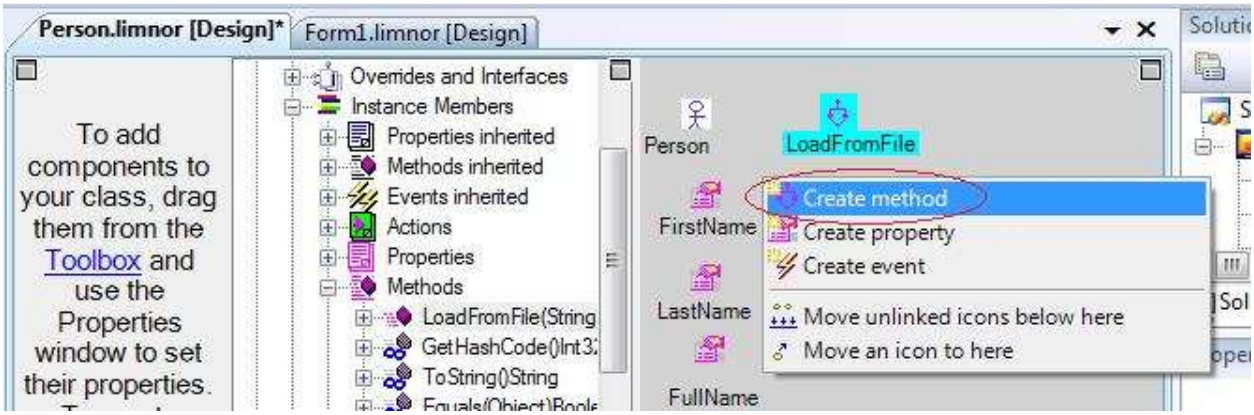


3.6 Create SaveToFile method

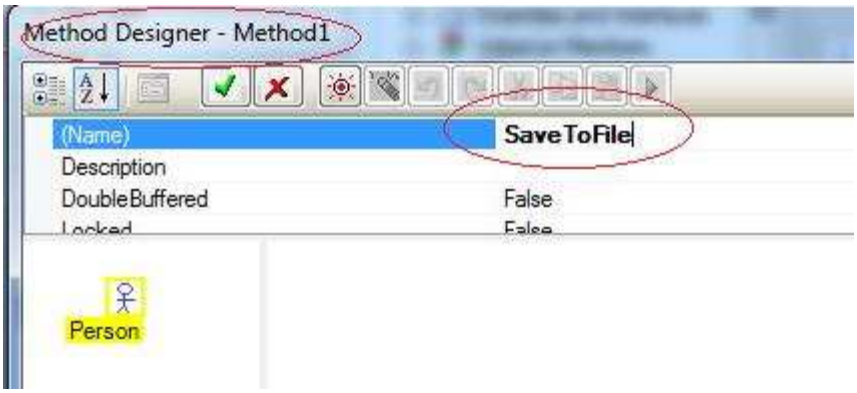
All techniques used in this section are already used previously. If any portion of this section is not detailed enough then refer to the previous section for details.

Since we have created a LoadFromFile method for class Person, we want to create a SaveToFile method too.

Right-click Event Path, choose “Create method”:



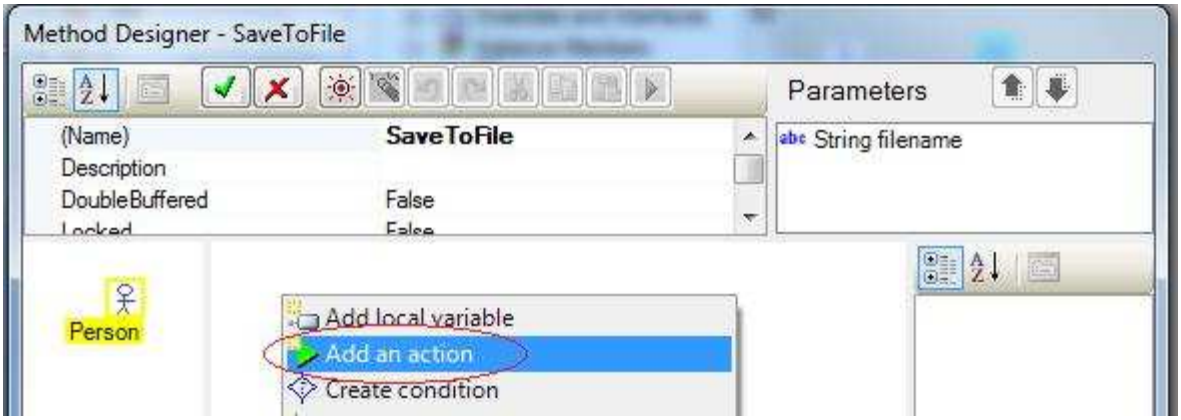
The Method Editor appears. Change the default method name from method1 to SaveToFile:



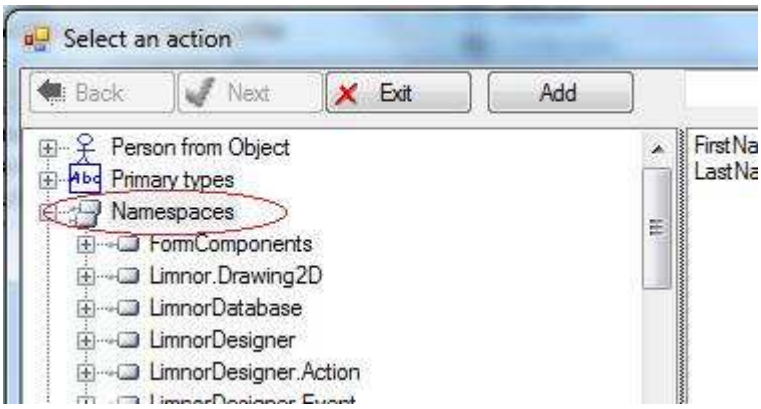
Add a string parameter and name it filename:

In method LoadFromFile, we use a ReadAllText action to read file. Here we may use a WriteAllText action to write data of Person class to file.

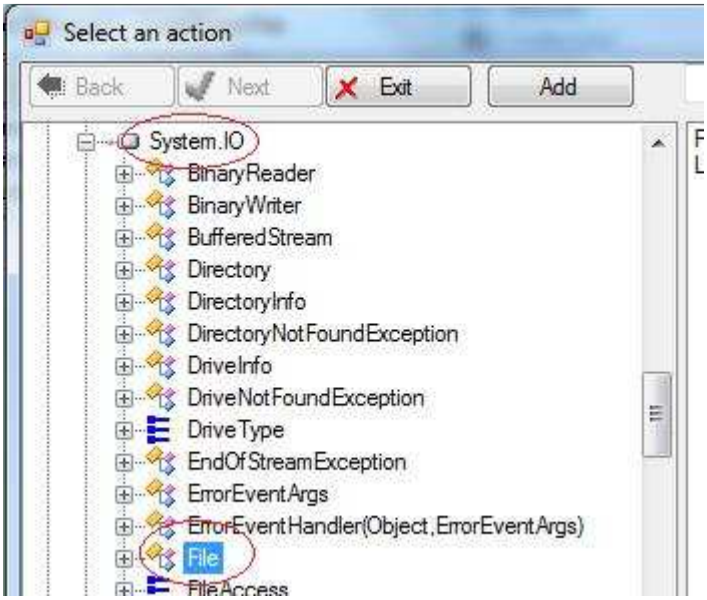
Right-click in the Method Editor and choose "Add an action" to create an action to execute the WriteAllText method:



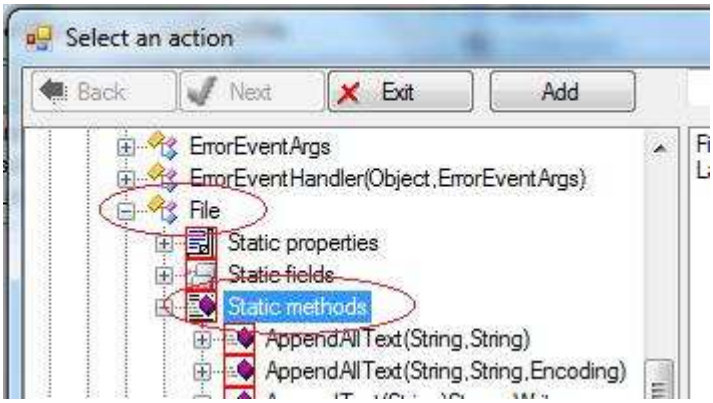
A dialogue box appears for selecting an action. We may use the File class to create a WriteAllText action. We'll find the File class under Namespaces. Most classes can be found under "Namespaces".



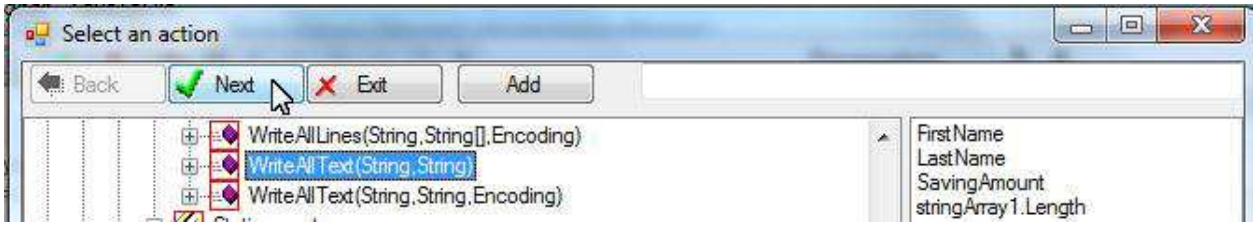
File class is under "System.IO" namespace:



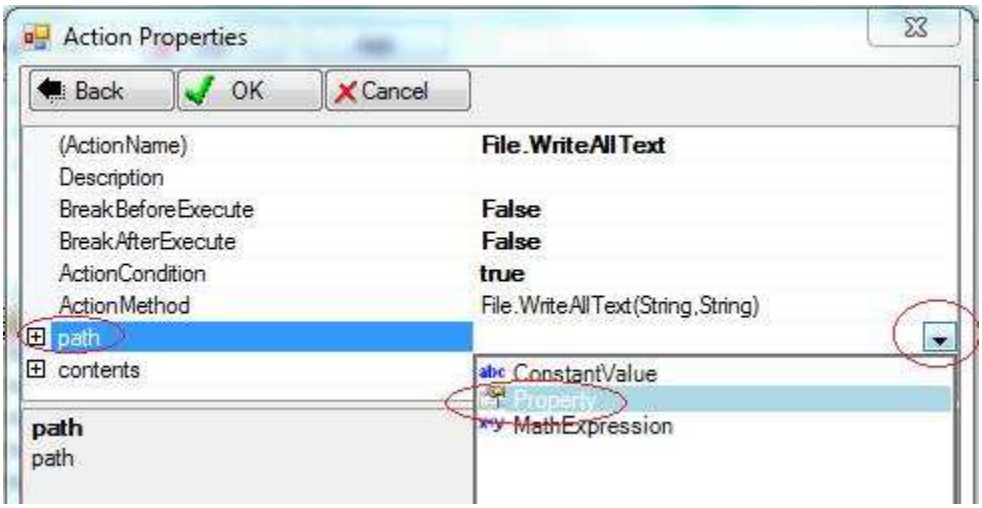
Expand “Static methods” and scroll down to find the WriteAllText method:



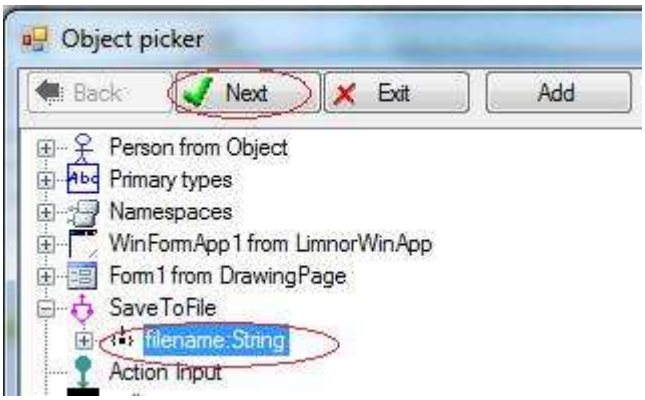
Select WriteAllText method, click “Next”:



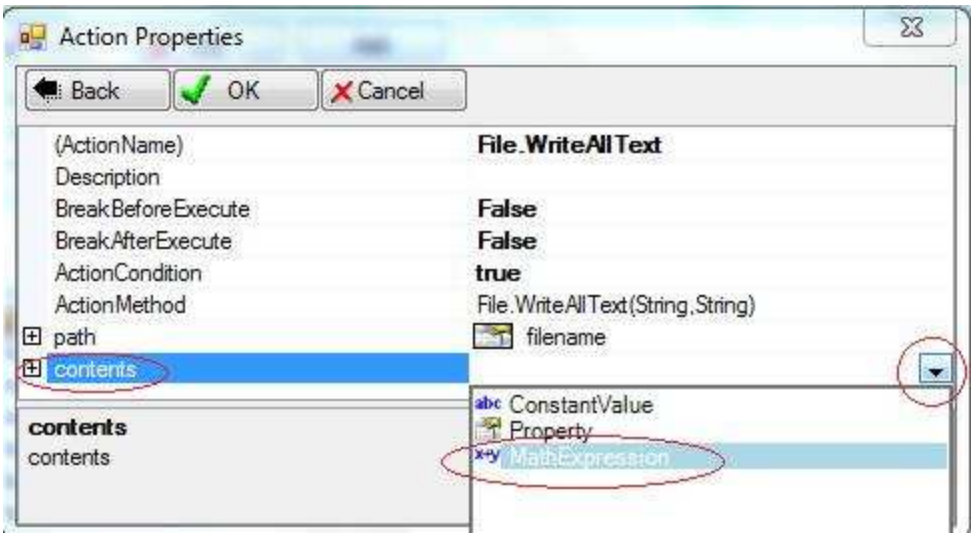
"path" parameter of this action represents the file name to write the file. We need to use the filename parameter of this method for it. Choose "Property" for it to select filename parameter:



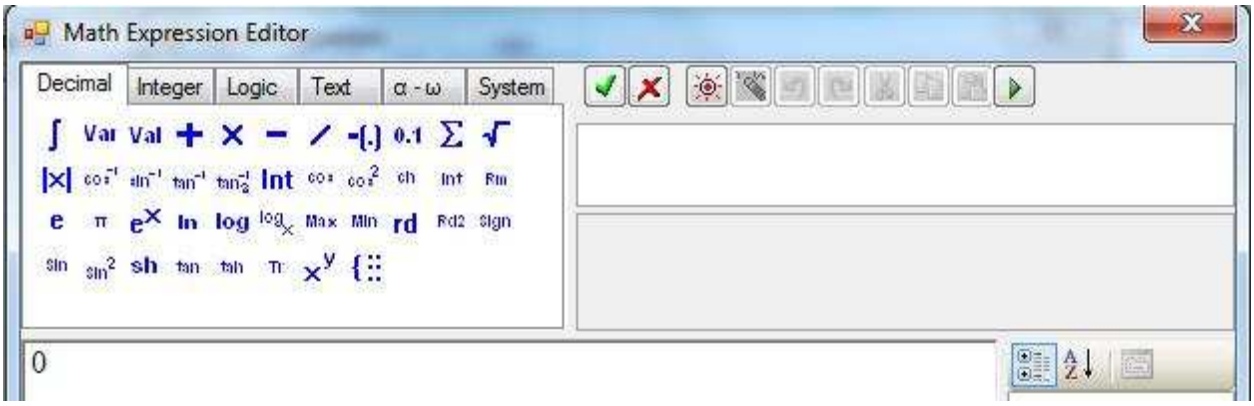
Select filename, click Next:



"contents" parameter of this action represents the contents to be written to the file. Select MathExpression for it to form an expression as the file contents:



The Expression Editor appears.

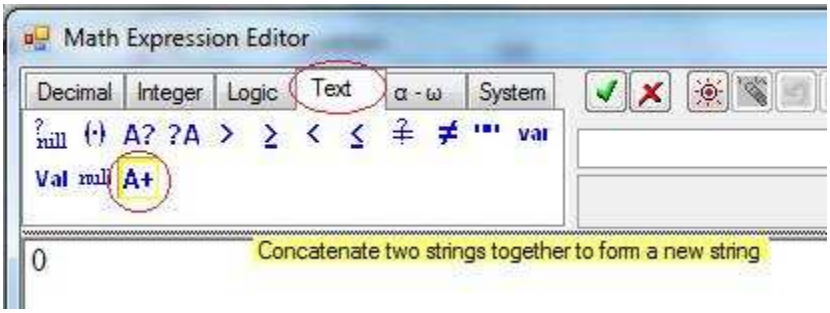


We want to form the file contents with properties FirstName, LastName and SavingAmount, using commas as separators:

{first name}{comma}{last name}{comma}{saving amount}

So, the expression is formed by 5 strings concatenated together.

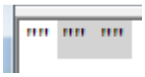
Click string-concatenate operator A+ to add strings to the expression:



Two empty strings appear in the expression:

We need 5 strings. So, we will add more strings.

Select the second empty string: 

Click A+. One more empty string is added: 

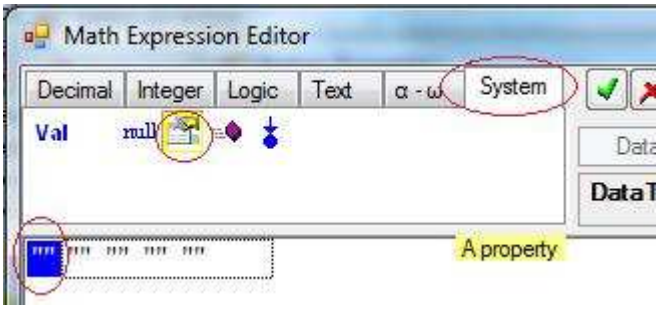
Select the last empty string: 

Click A+. One more empty string is added: 

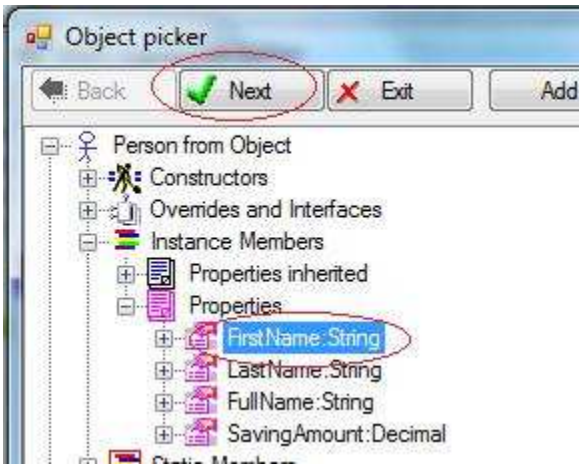
Select the last empty string: 

Click A+. We have 5 strings we needed: 

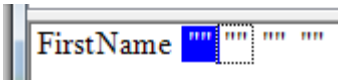
The first string should be property FirstName. Select the first string, click property icon  to make it a property:

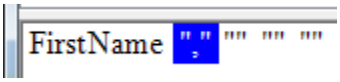


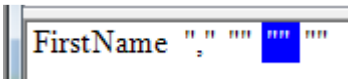
Select property FirstName, click Next:

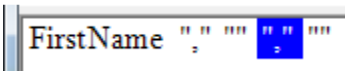


Property FirstName becomes the first string.

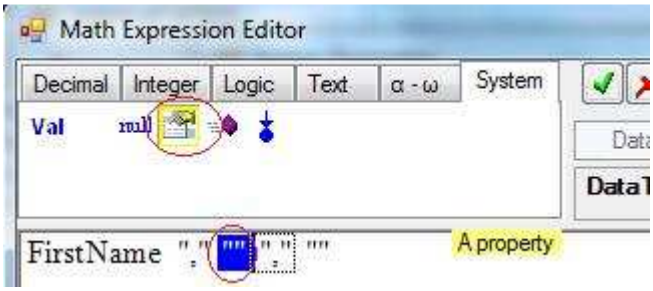
Select the second string: 

Type comma: 

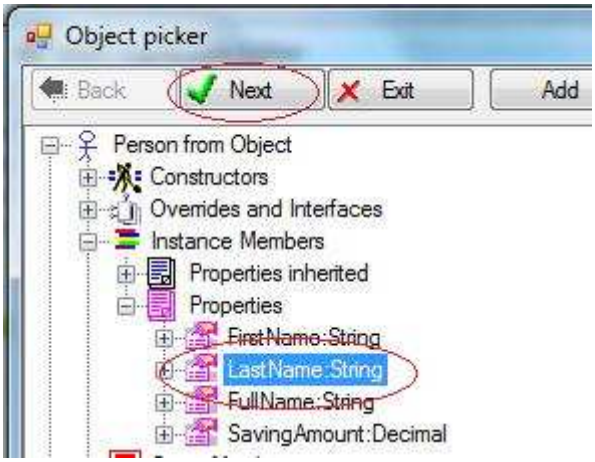
Select the 4th string: 

Type comma: 

Select the 3rd string. Click property icon 

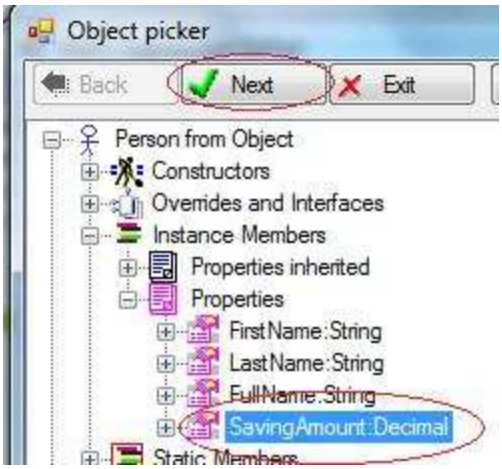


Select LastName, click Next:

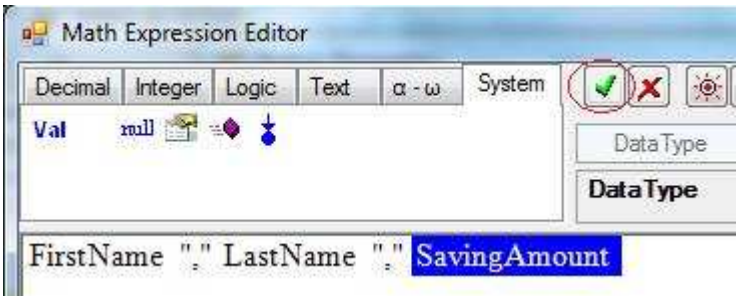


Select the last string and click  :

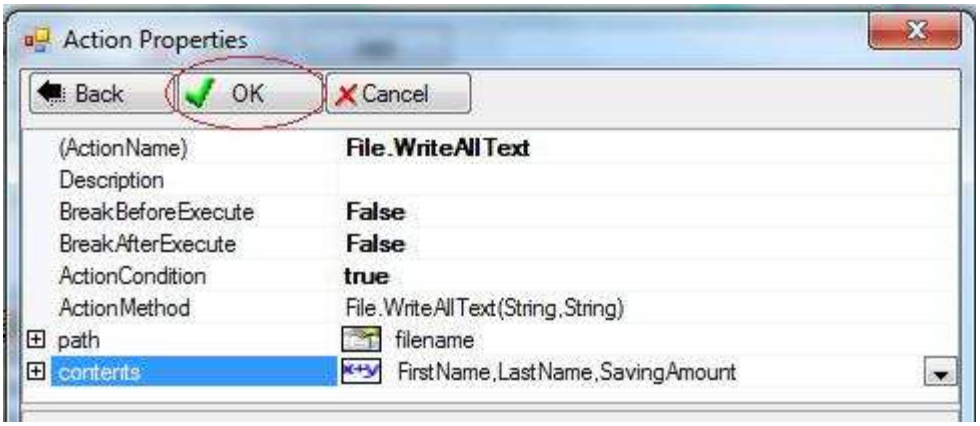
Select SavingAmount, click Next:



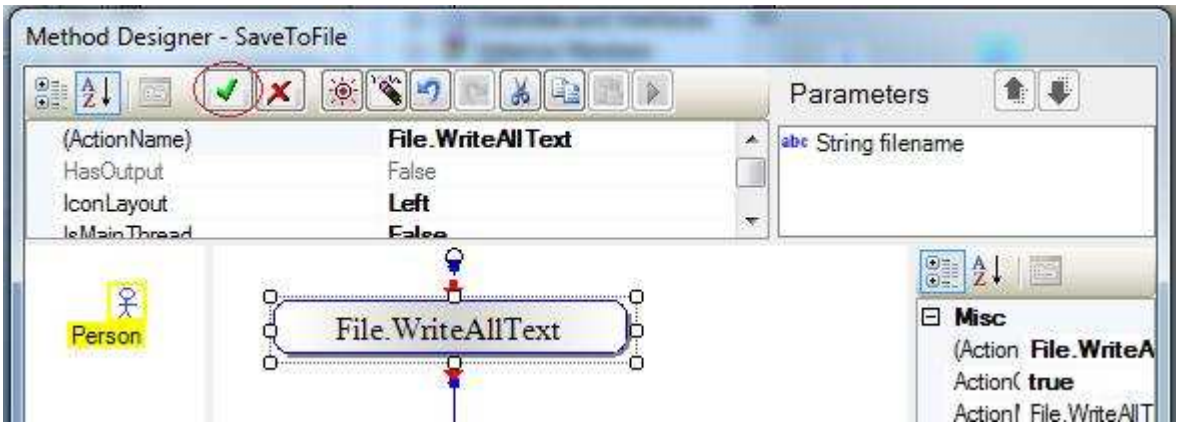
Click  to finish creating this expression:



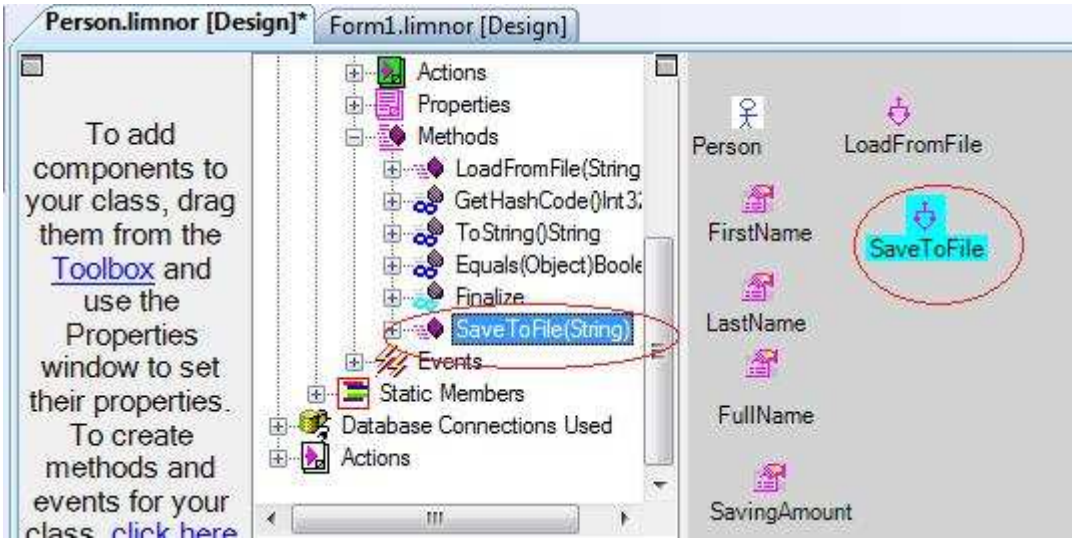
Click OK to finish creating this action:



The action appears in the method. Click  to finish creating this method because we do not need other actions for this method:



The new method appears in the Object Explorer and the Event Path:



3.7 Decision Table

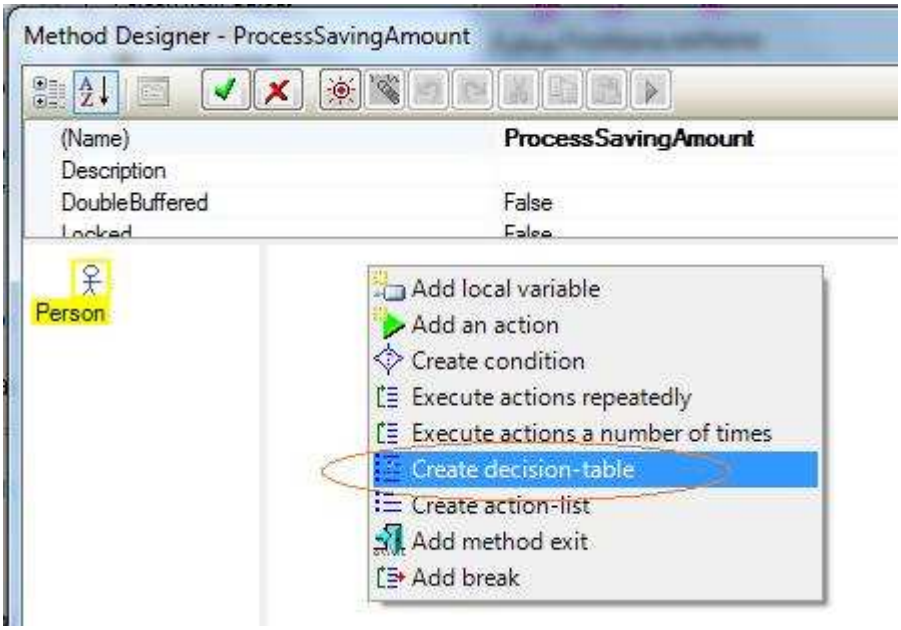
A Decision Table contains a list of decision items. Each decision item has conditions and corresponding actions. When a Decision Table is executed, conditions for each decision item are evaluated one by one to find the first decision item whose conditions are evaluated to True. The actions of the found decision item are then executed. Once a decision item is found and executed the execution of the Decision Table finishes.

Because ActionCondition can be assigned to an action, "Action List" looks similar to "Decision Table". Comparing to "Action List" all actions are executed in an action list, but only one decision item could be executed for a Decision Table. If no conditions are met then no actions are executed for a Decision Table.

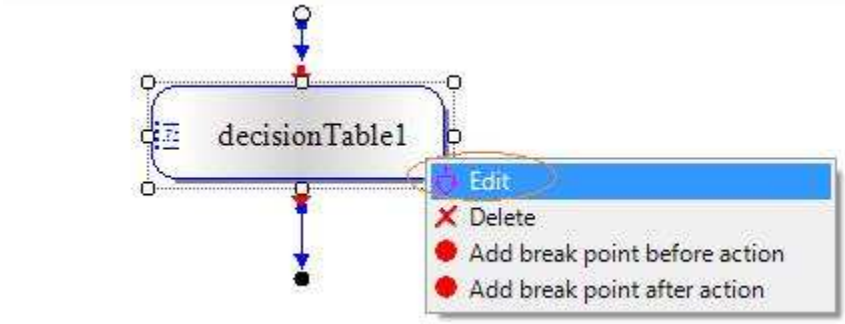
As a sample, suppose we want to do some processing of the Person class based on the value range of the SavingAmount property. We may have such a "Decision Table":

SavingAmount <= 0 → do some things
SavingAmount >0 and < 30 → do some things
SavingAmount >= 30 and < 80 → do some things
SavingAmount >= 80 → do some things

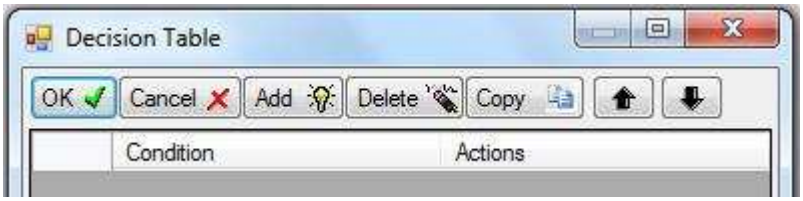
Each "do some things" is a list of actions to be executed.
Let's create a new method named ProcessSavingAmount. Right-click the middle pane of the method and choose "Create decision-table":

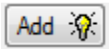


Right-click the decision table, choose “Edit”:



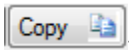
A dialogue box appears for editing the decision table:





-- Add a new decision item to the list

-- Remove the selected decision item from the list

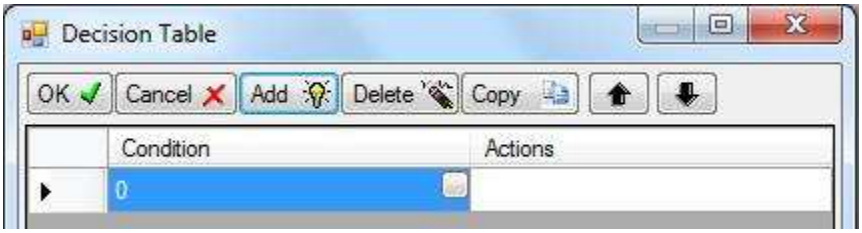


--Create a new decision item by copying the selected decision item

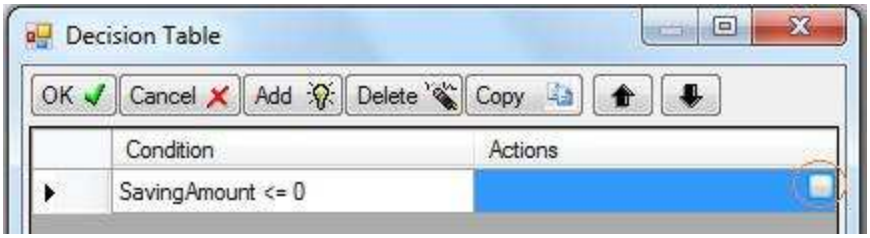

-- Move the selected decision item up in the list


-- Move the selected decision item down in the list

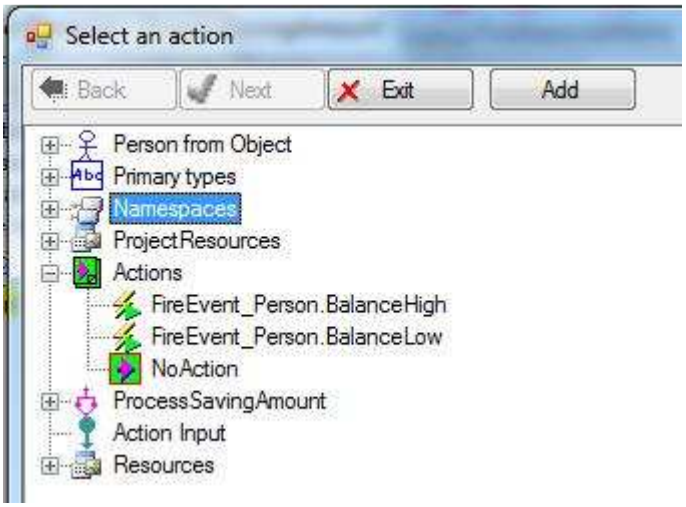
Click Add to add a new decision item. Click to set conditions for the decision item.



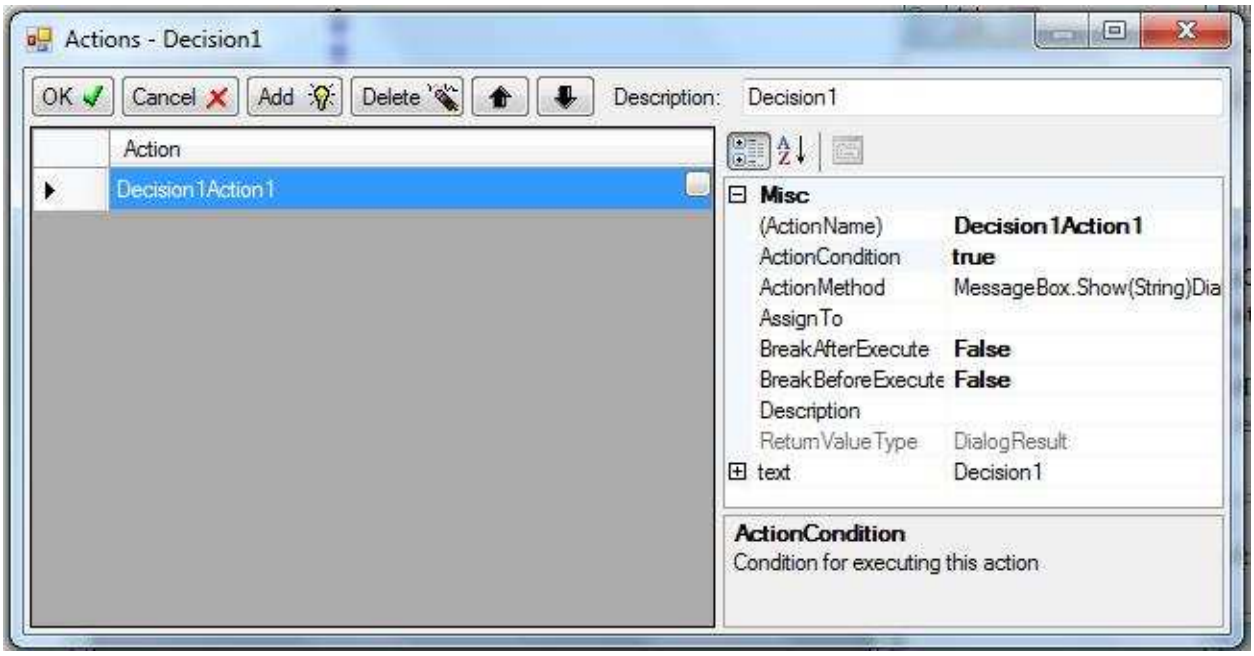
Click to select actions for the decision item:



Select the actions for this decision item:

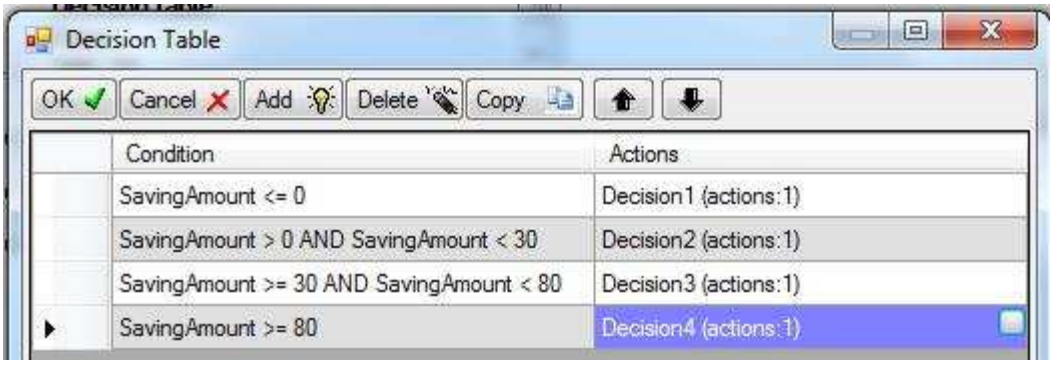


For this sample, we may simply create an action to show a message box. The selected actions are displayed in a dialogue box:



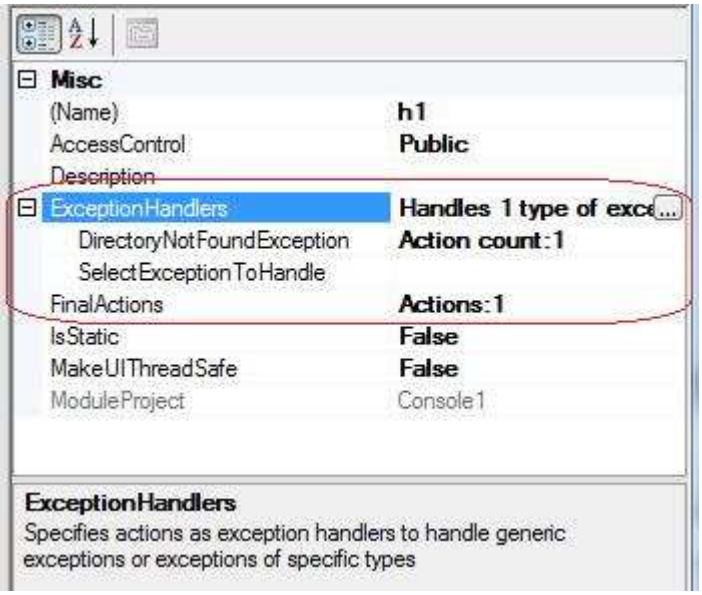
Many actions can be added for this decision item.

Add more decision items:



3.8 Handle Exceptions

Some actions in the method may cause unexpected errors. Such errors are called Exceptions. An unhandled exception will bring the application to a crash. Every method has an ExceptionHandlers property and a FinalActions property for handling exceptions.



For details of handling exceptions, see http://www.limnor.com/support/handleExceptions.pdf

4 Create Events

An event is a special moment for executing actions assigned to it. When we are developing a new class, we may think about which moments are of interests to the users (programmers) of the class. That is, the programmers using the class will want to have some actions executed at those moments. Those moments should be the events to be created for the new class.

For our Person class, it has a SavingAmount property. When the SavingAmount decreases to a certain level, we want to **make it possible** to execute some actions. When the SavingAmount increases over certain amount, we want to **make it possible** to execute some actions.

We do not know what actions to be executed at this time. The programmers using the Person class will know what actions to be executed at these moments. For different instances of Person class, the actions can be different.

Our task is just to **make it possible**. How to make it possible? The answer is to create events for the Person class.

We define two events for the Person class.

- Event **SavingAmountLow**. This event is defined for the moment when SavingAmount property decreases lower than a certain amount. Let's call this certain amount SavingAmountLowThreshold. To be exact, this event is defined for the moment when SavingAmount decreases from a value larger than or equal to SavingAmountLowThreshold to a value lower than SavingAmountLowThreshold.

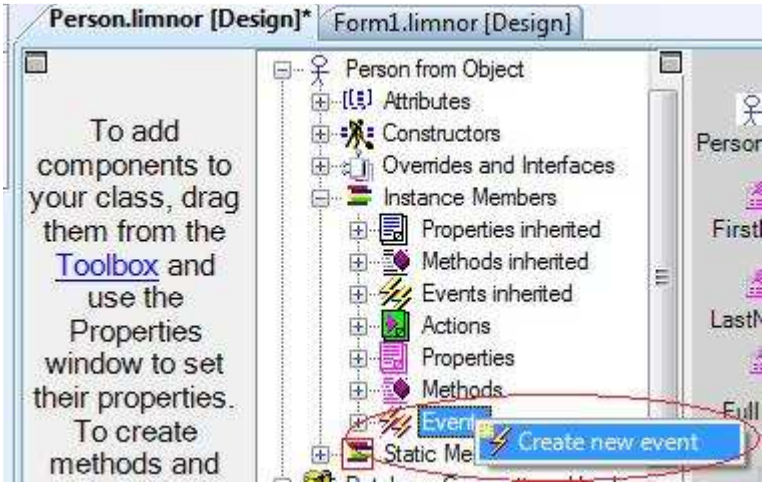
- Event **SavingAmountHigh**. This event is defined for the moment when SavingAmount property increases higher than a certain amount. Let's call this certain amount SavingAmountHighThreshold. To be exact, this event is defined for the moment when SavingAmount increases from a value lower than or equal to SavingAmountHighThreshold to a value higher than SavingAmountHighThreshold.

Let's see how to add these two events to our Person class.

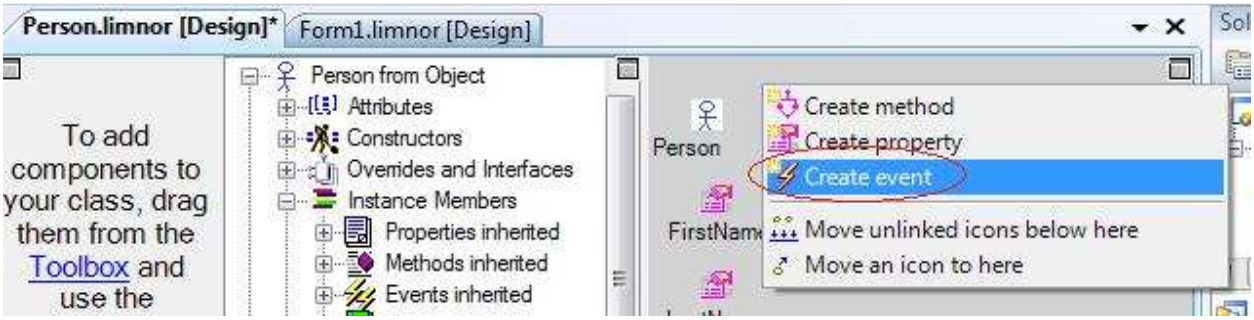
4.1 Add new events

The Object Explorer and Event Path both can be used to add new events to a class.

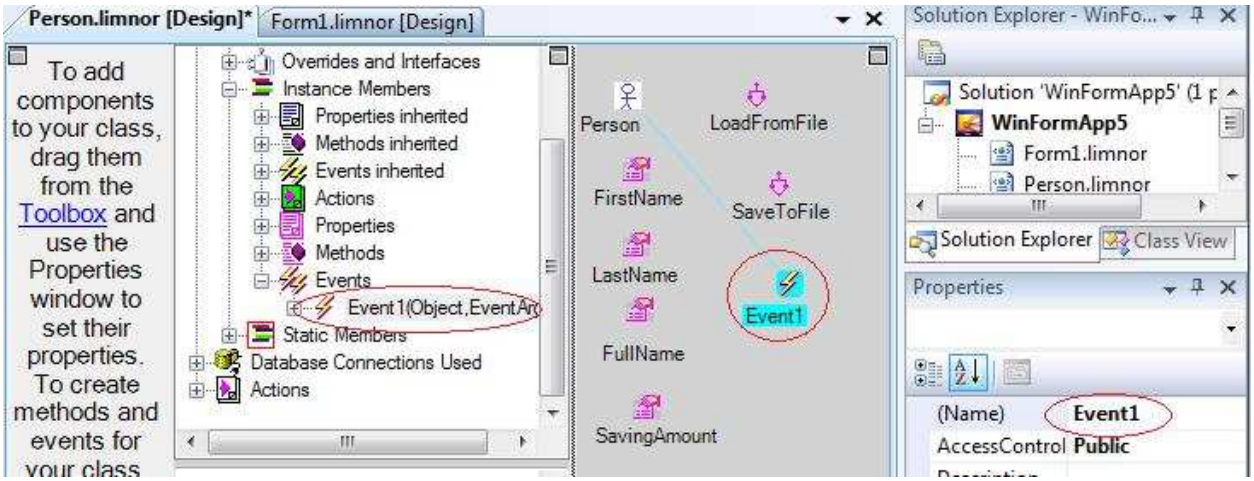
To use the Object Explorer to add a new event, right-click Events, choose "Create new event":



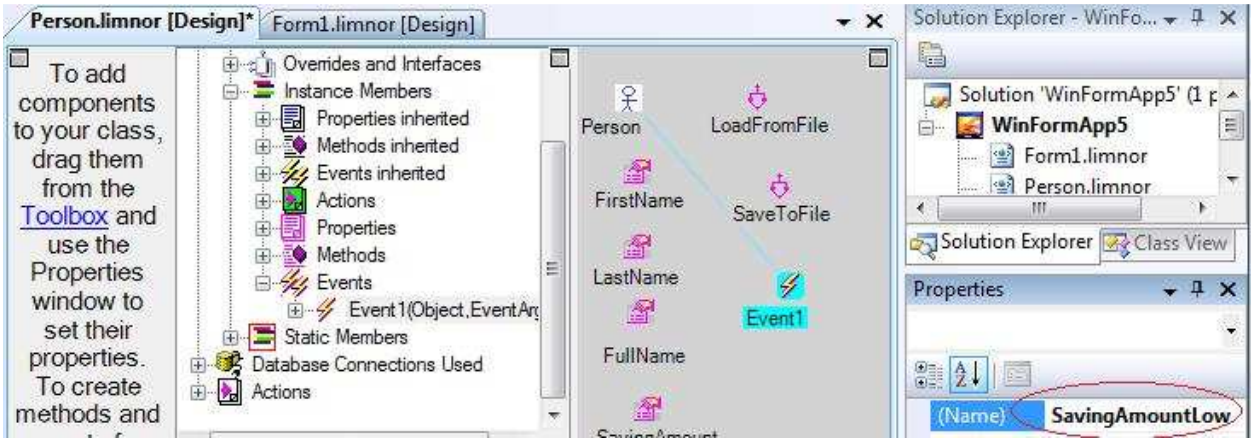
To use the Event Path to add a new event, right-click Event Path, choose "Create event":



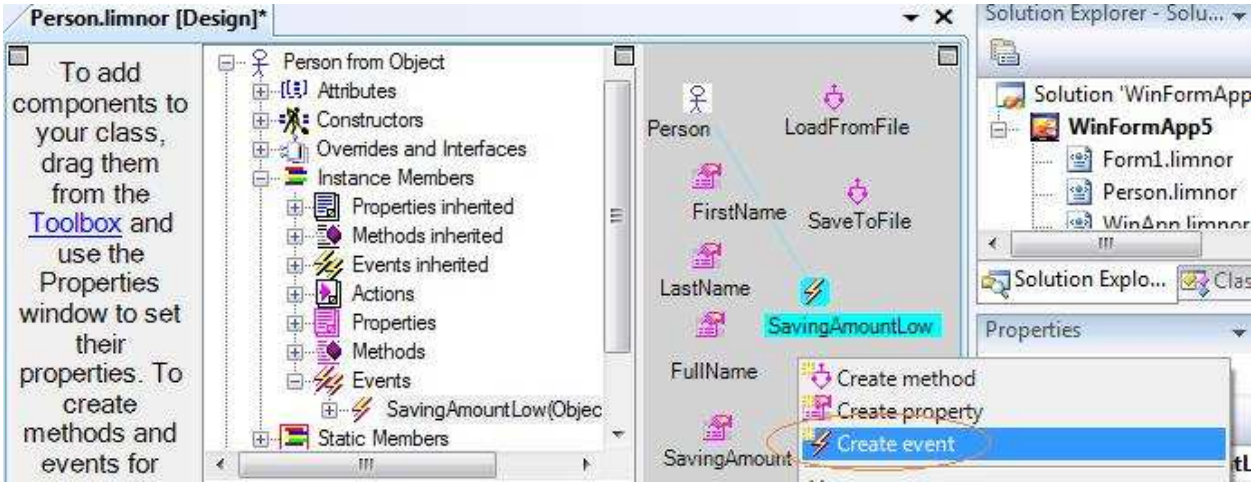
A new event is created:

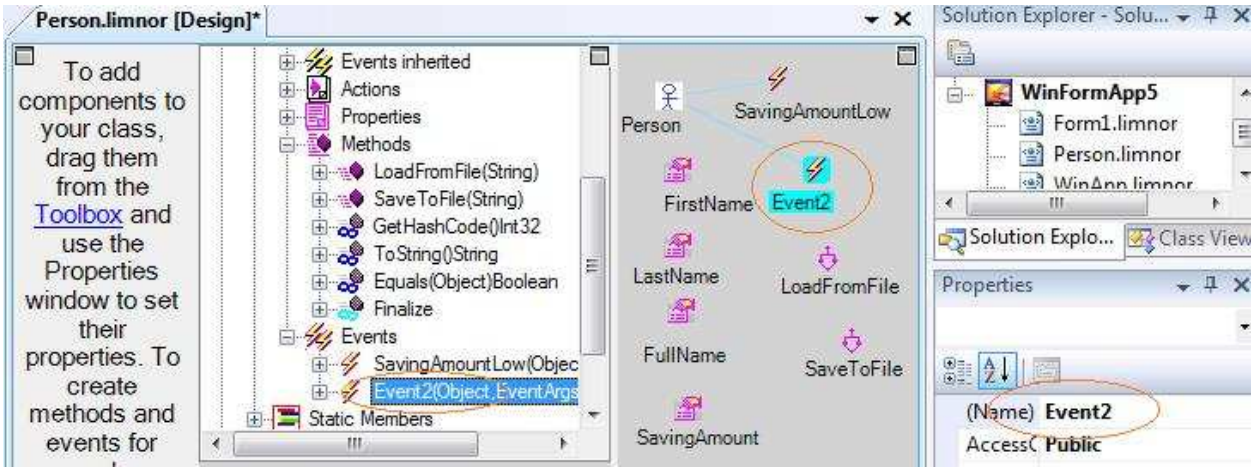


Rename the new event to SavingAmountLow:

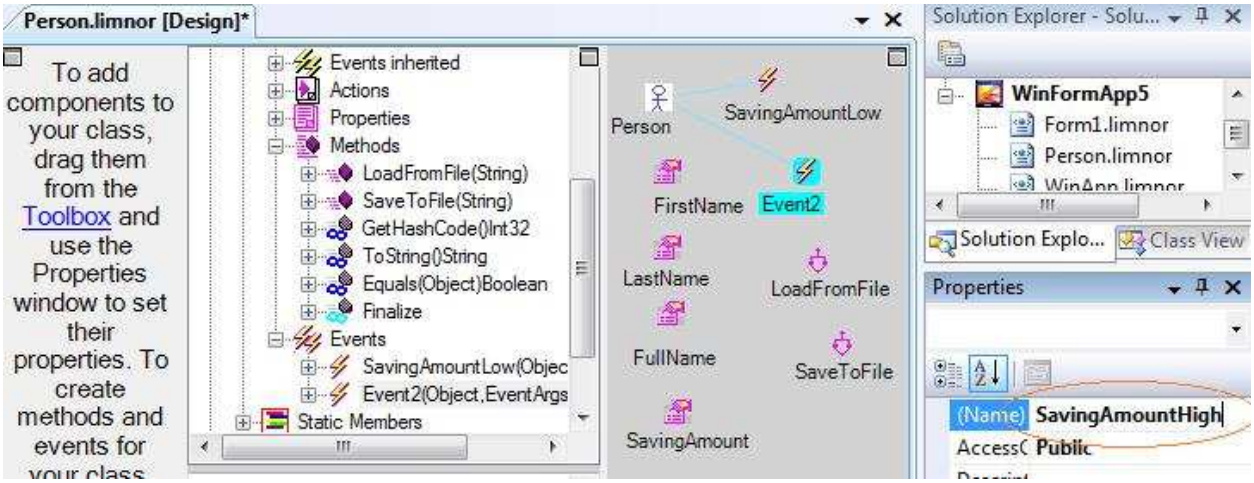


Add another event:

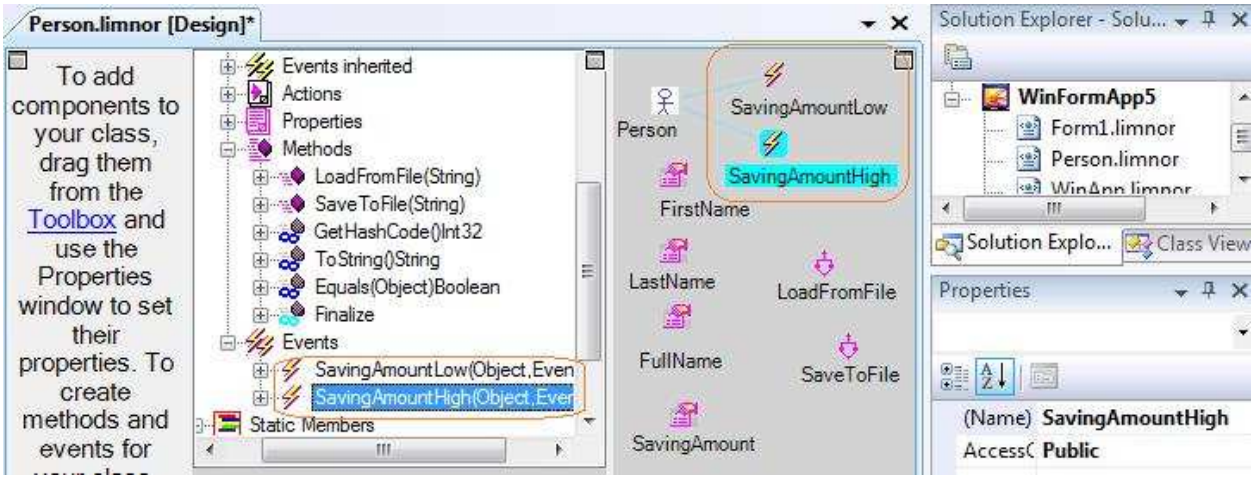




Rename the new event to SavingAmountHigh:



We just defined two events in the Person class: SavingAmountLow and SavingAmountHigh.



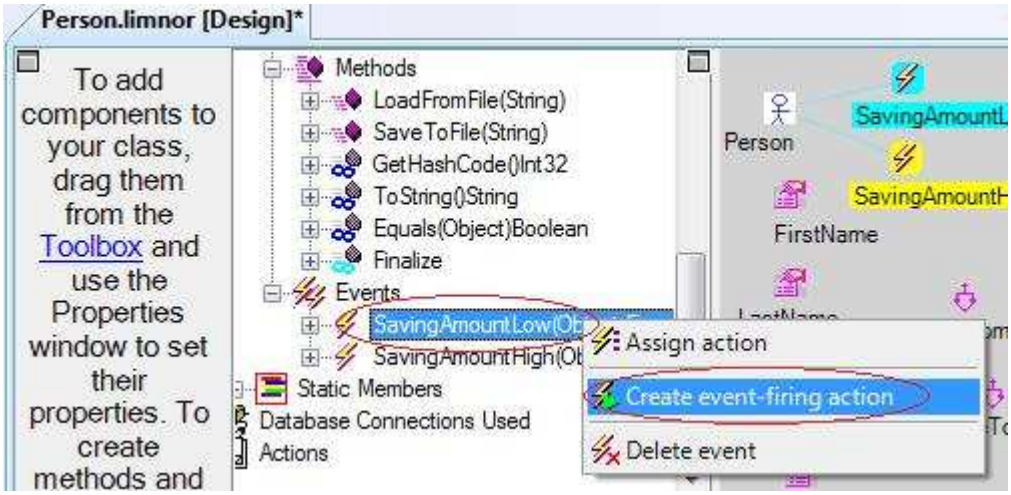
4.2 Create event-firing actions

We mentioned that there 3 types of actions: Set Property Action; Method Execution Action; Event-Firing Action. We have created and used many set-property actions and method-execution actions.

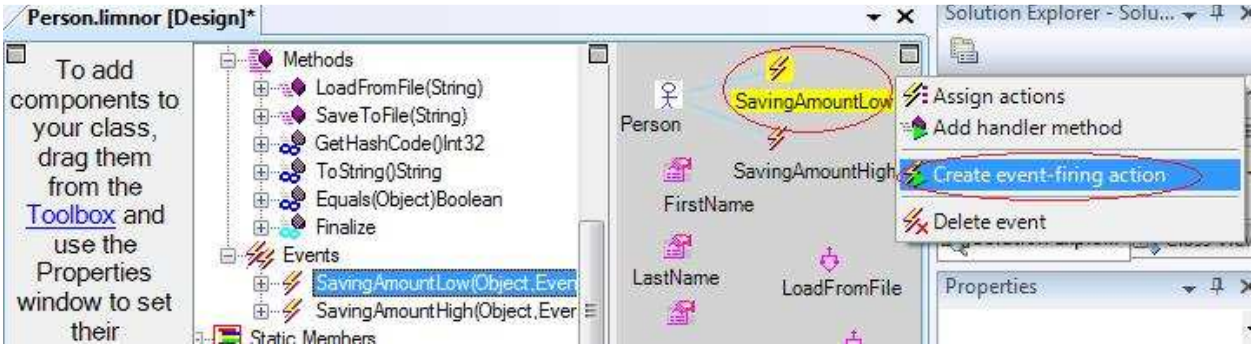
In this section, we will create event-firing actions and then explain what event-firing actions are and how to use them.

The Object Explorer and Event Path both can be used to create event-firing actions.

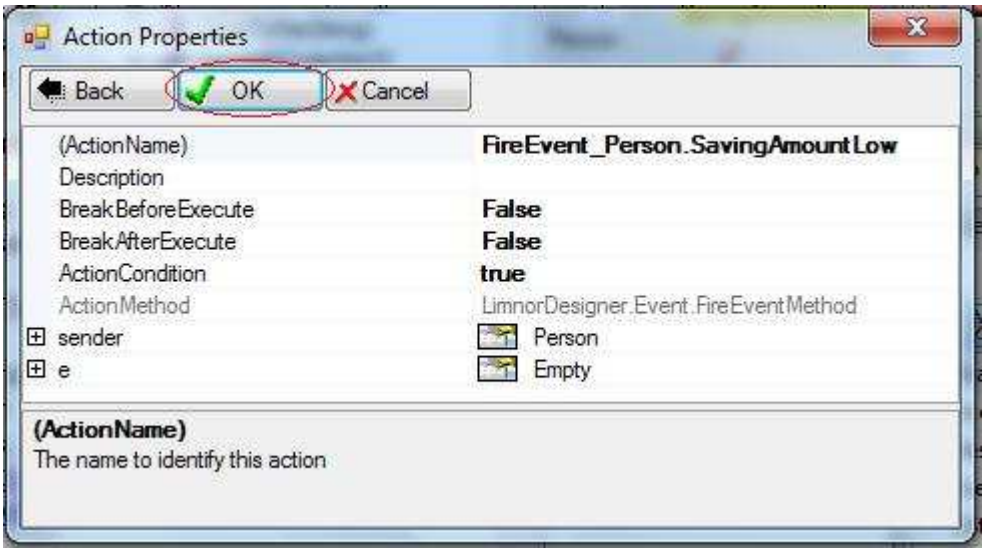
To use the Object Explorer to create an event-firing action, right-click the event, choose “Create event-firing action”:



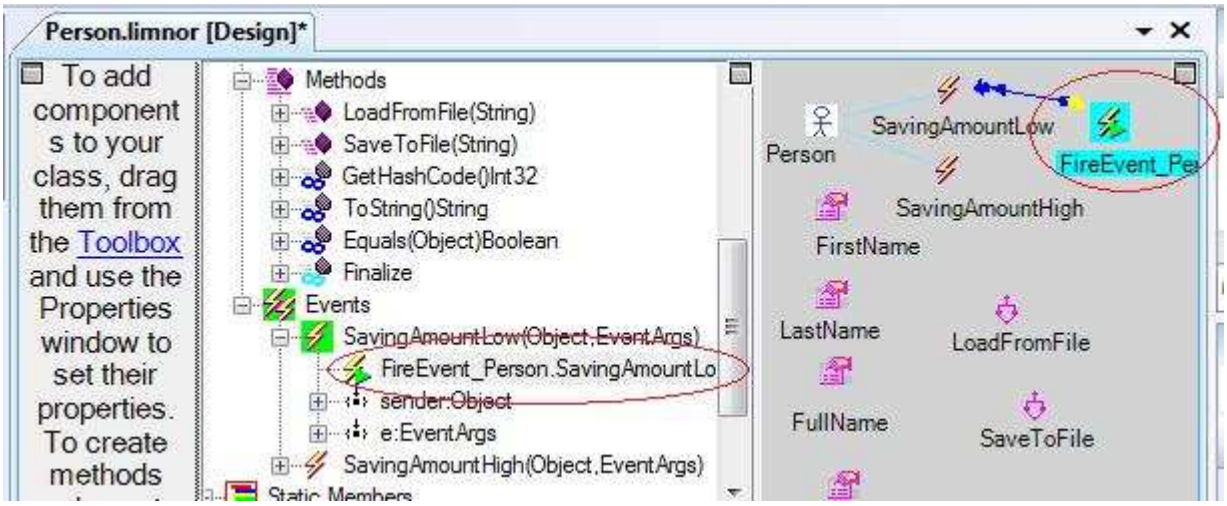
To use the Event Path to create an event-firing action, right-click the event, choose “Create event-firing action”:



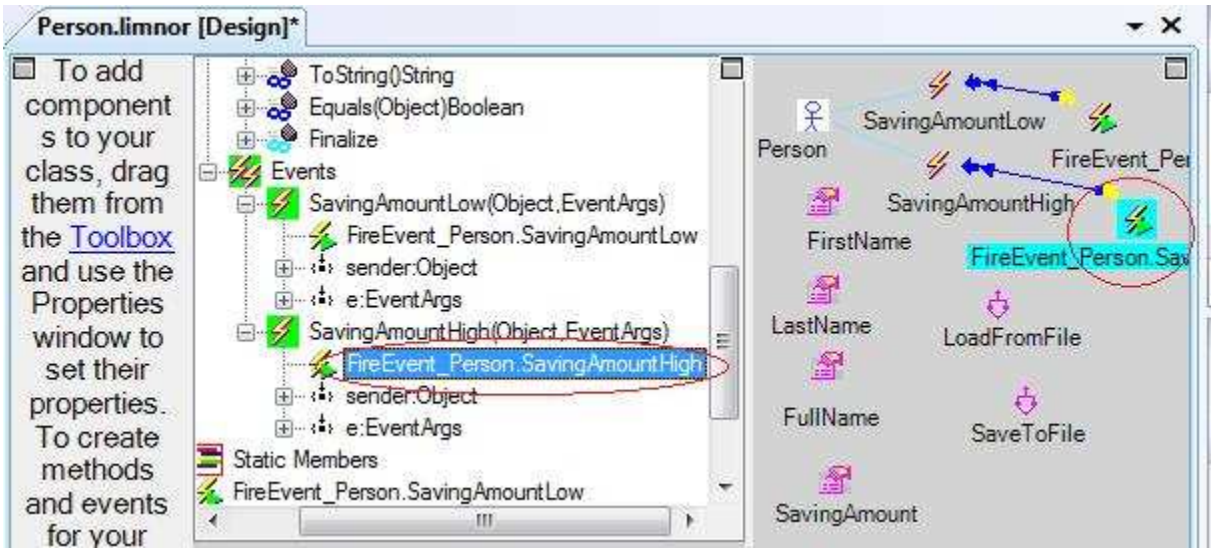
We may accept the default parameters for the new event-firing action and click OK:



In the Object Explorer this new event-firing action appears under the event. In the Event Path this new action appears as an icon and linked to the event icon:



We can do the same for event SavingAmountHigh:



What is an event-firing action? It is for firing an event. An event occurs because an event-firing action is executed.

When an event-firing action is executed, the corresponding event occurs. It is our responsibility to execute the right event-firing action at the right moment.

4.3 Firing Events

When we define a new event, it is our responsibility to execute event-firing actions at the right moments for the event.

We defined two events SavingAmountLow and SavingAmountHigh. We must figure out when to execute event-firing actions for them.

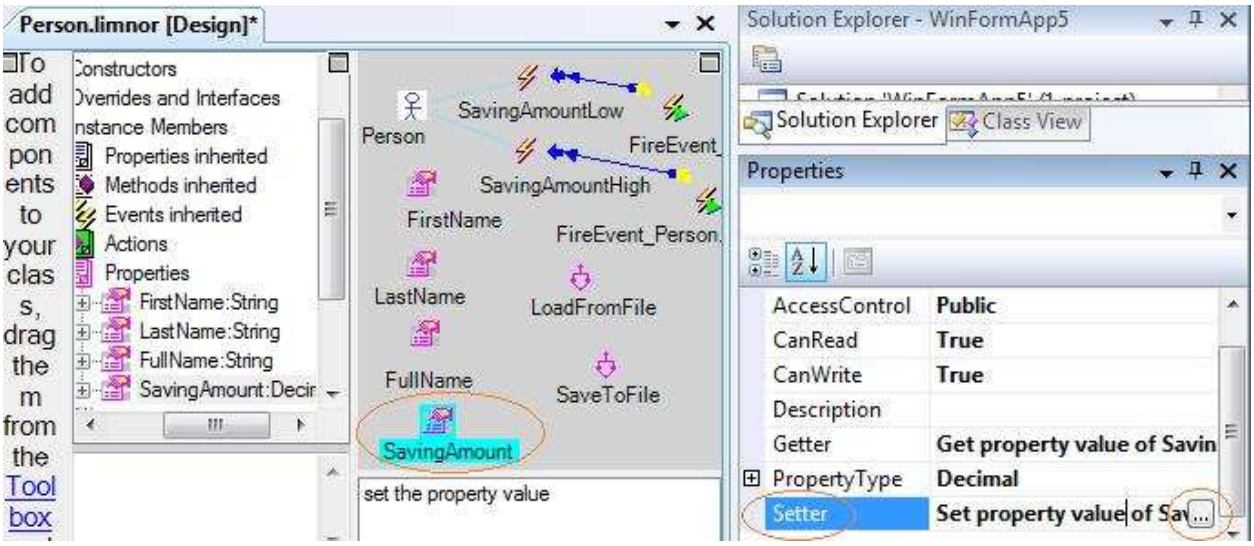
4.3.1 Find out the moments for events

For events SavingAmountLow and SavingAmountHigh, they may occur only when property SavingAmount is changed.

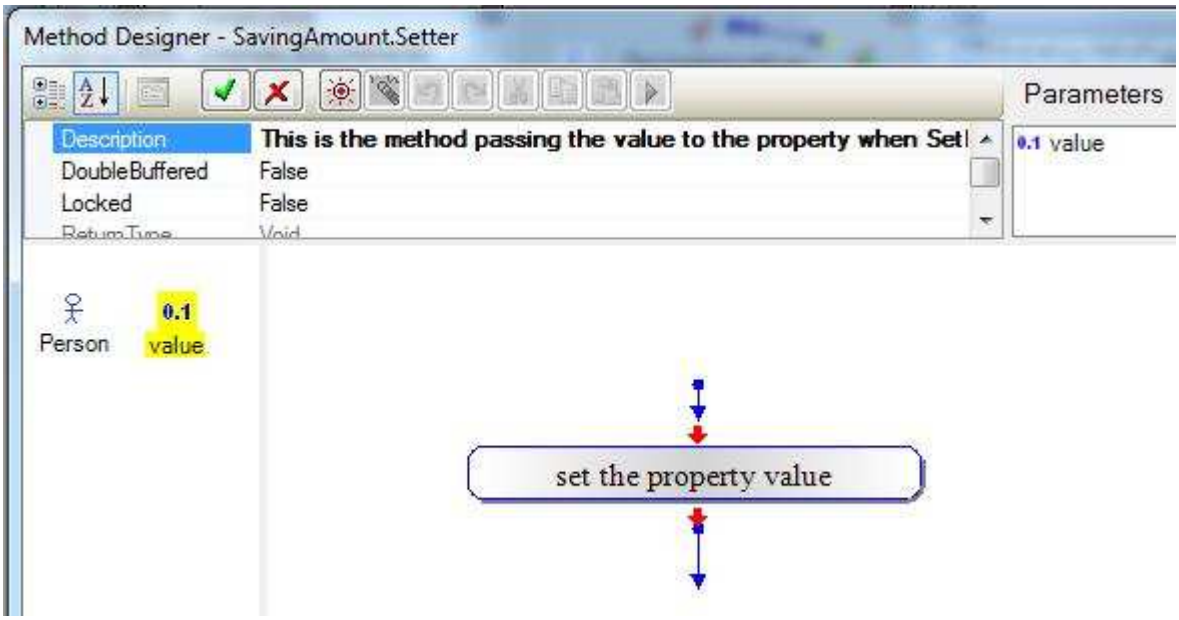
A property is changed when a set-property action for it is executed. When a set-property action is executed, the Setter method of the property is executed.

The Setter method can be edited. This is the place where we may execute event-firing actions for these two events.

Click ... to edit the Setter method:



The method editor appears:



The value parameter is the new property value. It is the SavingAmount value to be changed to.

The “set the property value” action assigns value to the internal storage for SavingAmount.

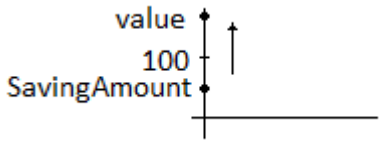
In this method, “value” parameter represents the new value of the SavingAmount. SavingAmount represents the current value.

We may compare the “value” parameter with SavingAmount to determine if we need to fire events and fire which events.

4.3.2 Moment for event SavingAmountHigh

If "value" is larger than SavingAmount then SavingAmount is increasing. SavingAmountHigh event might occur. Suppose we use 100$ as the threshold for firing SavingAmountHigh. Then we should fire SavingAmountHigh event if value is larger than 100 and SavingAmount is smaller than or equal to 100:

- ✓ "value" > 100
- ✓ SavingAmount <= 100



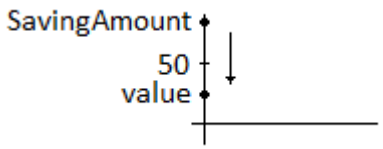
These conditions can be expressed by an expression:

```
value > 100 and SavingAmount <= 100
```

4.3.3 Moment for event SavingAmountLow

If "value" is smaller than SavingAmount then SavingAmount is decreasing. SavingAmountLow event might occur. Suppose we use 50$ as the threshold for firing SavingAmountLow. Then we should fire SavingAmountLow event if value is smaller than 50 and SavingAmount is larger than equal to 50:

- ✓ "value" < 50
- ✓ SavingAmount >= 50



These conditions can be expressed by an expression:

```
value < 50 and SavingAmount >= 50
```

4.3.4 Execute event-firing actions

We have identified the conditions for executing event-firing actions. Note that those conditions are good only before the action "set the property value" is executed. After executing the action, SavingAmount is changed to be equal to the "value" parameter.

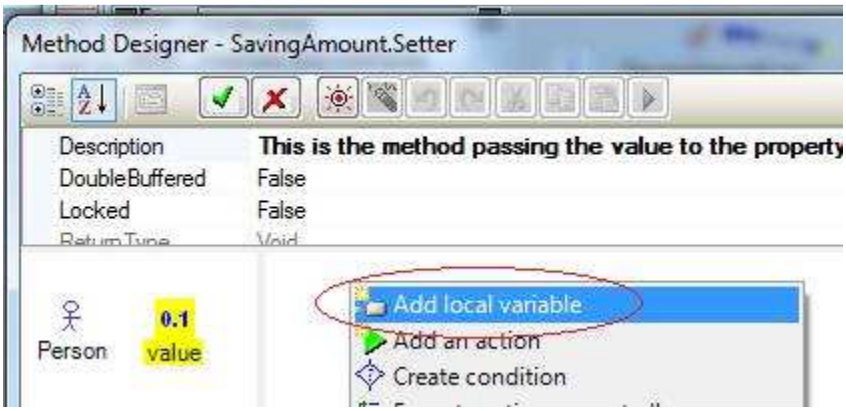
So, we may check the conditions before executing action "set the property value" and execute event-firing actions. But at this time, the SavingAmount is still in old value.

Another design can be to execute event-firing actions after executing action "set the property value" so that the property SavingAmount is already in new value when the events occur.

But we must preserve the condition checking results before executing action “set the property value” so that the conditions will not be damaged by action “set the property value”.

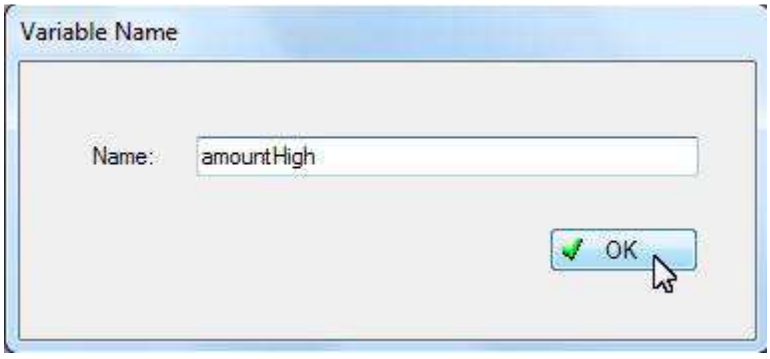
We may use variables to preserve conditions checking results.

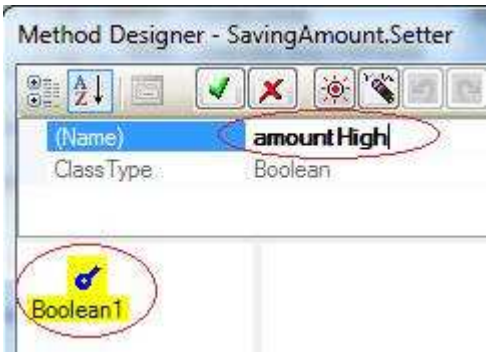
Right-click the method editor, choose “Create local variable”:



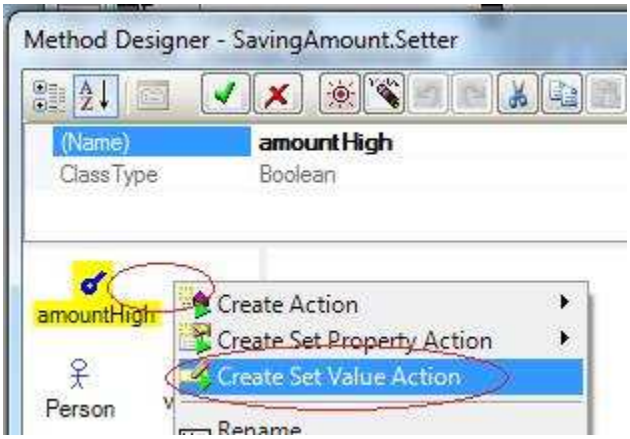
Choose Boolean as the variable type:

Change the variable name to “amountHigh”:

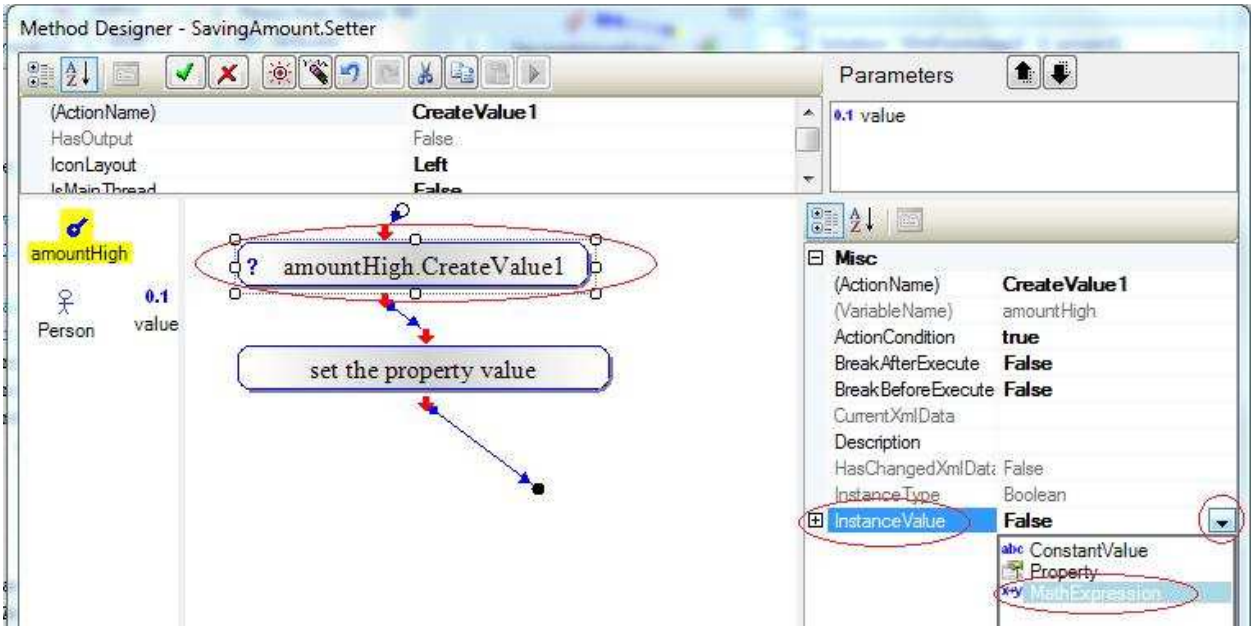




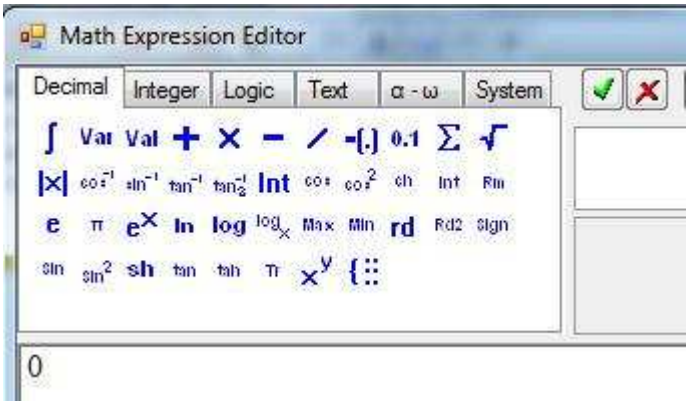
Right-click the variable, choose “Create Set Value Action” to assign the condition checking result to the variable:



The new action appears. Link the action to the existing action. The “InstanceValue” parameter of the new action is the value for the variable. Choose MathExpression to create condition checking:



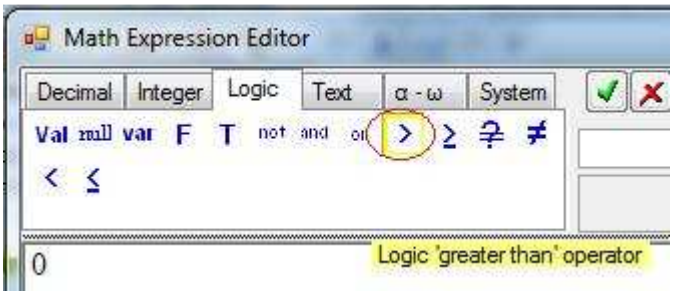
The Expression Editor appears:



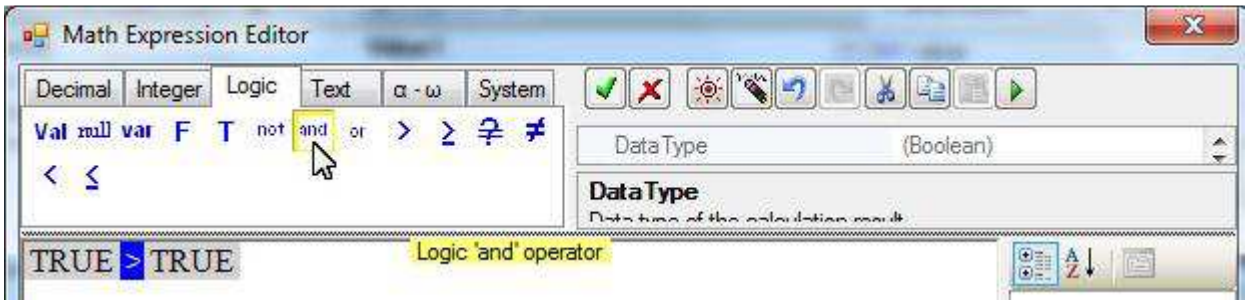
We already know that the expression for checking conditions for SavingAmountHigh is

value > 100 and SavingAmount <= 100

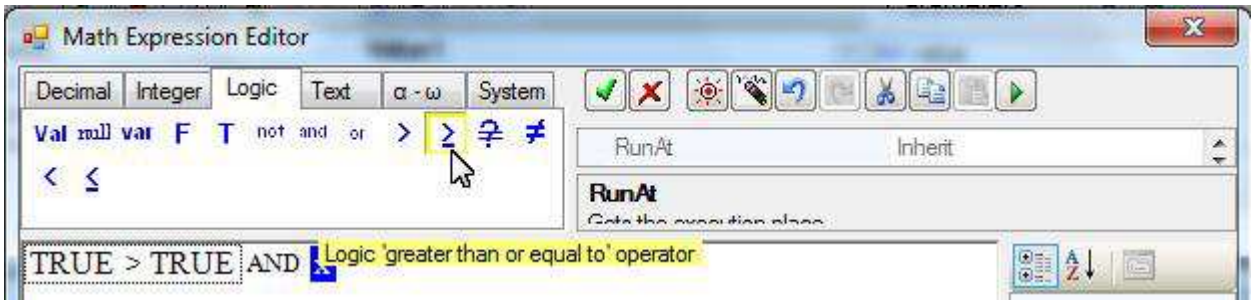
Click > :



Click and:

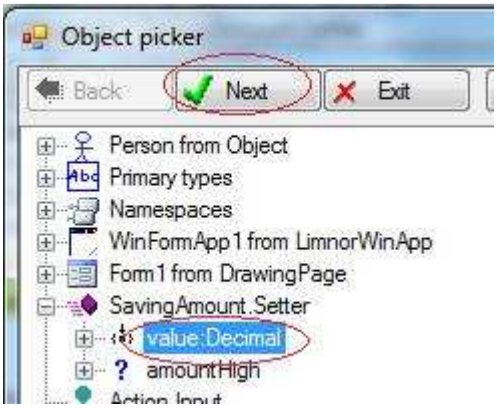


Select the last element, click ≤

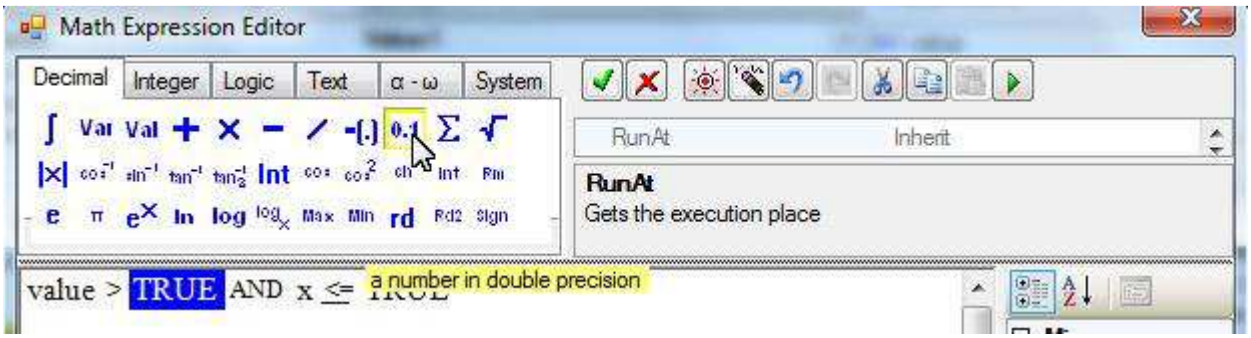


Select the first element, click  :

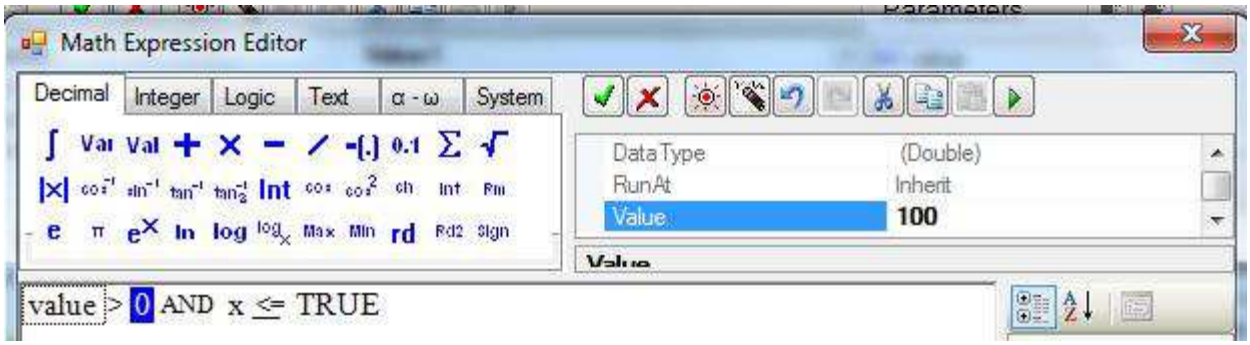
Select value, click Next:



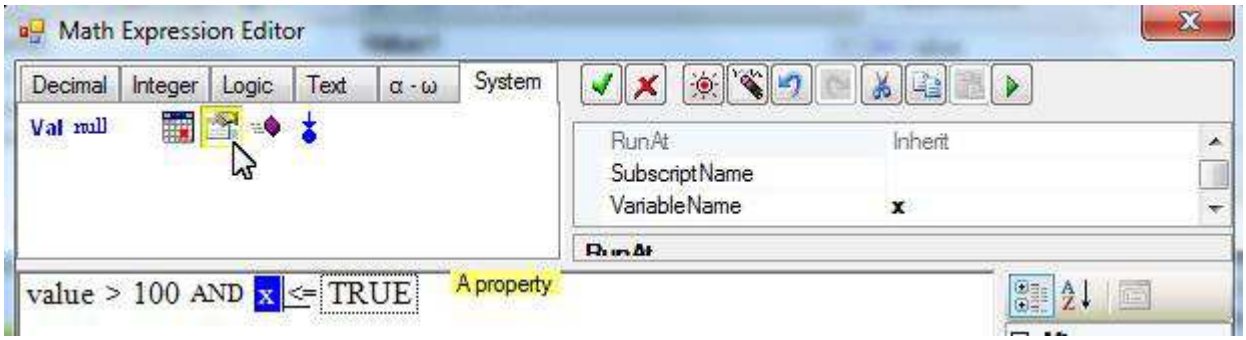
Select the second element, click 



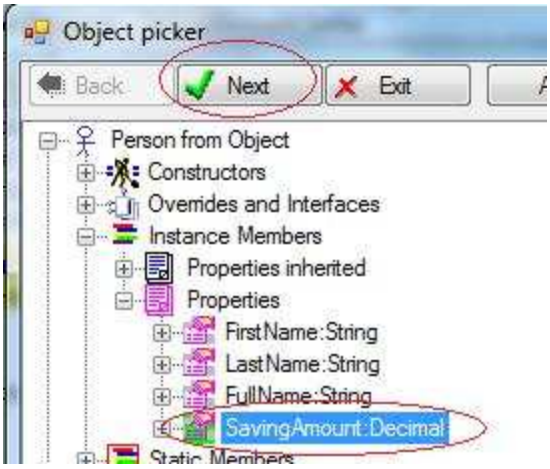
Type 100:



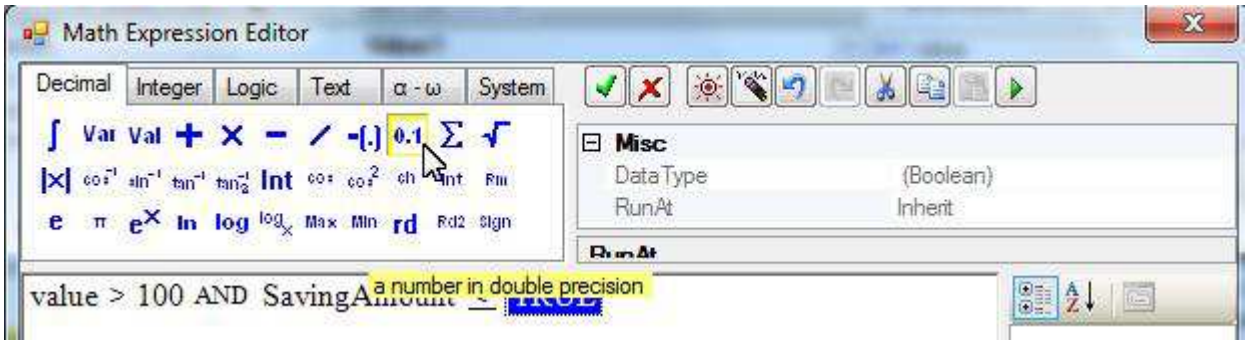
Select x, click  :



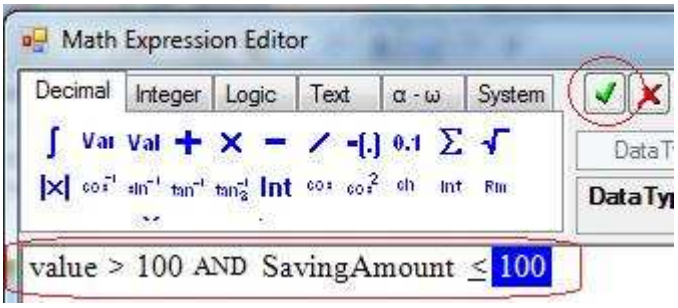
Select SavingAmount, click Next:



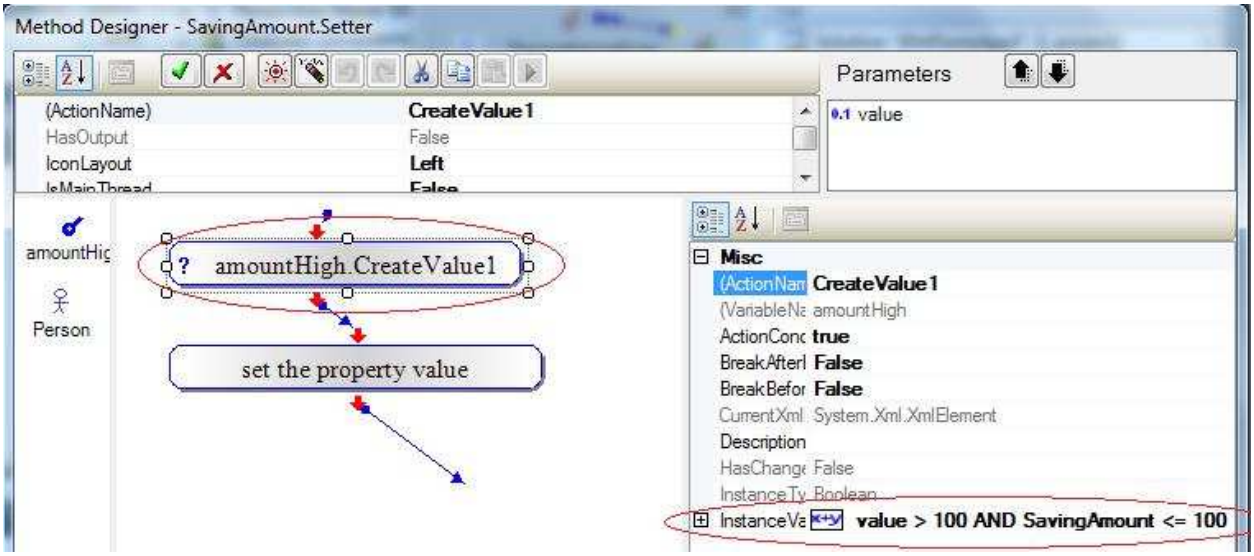
Select the last element, click 0.1



Type 100. Click  to finish creating this expression:

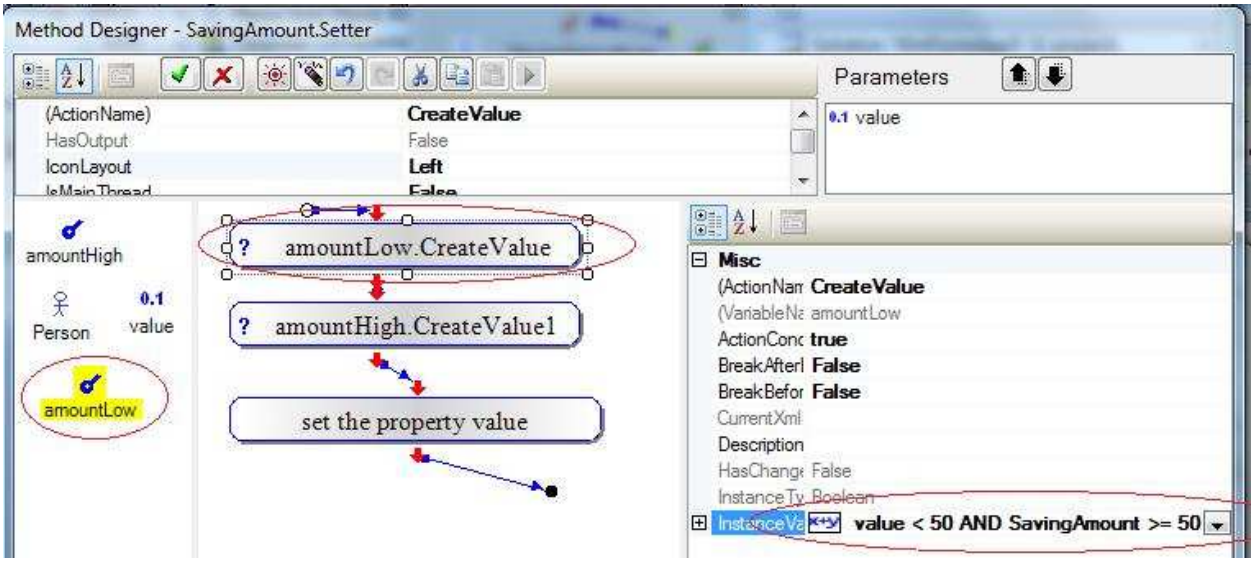


The conditions are now saved to variable amountHigh:

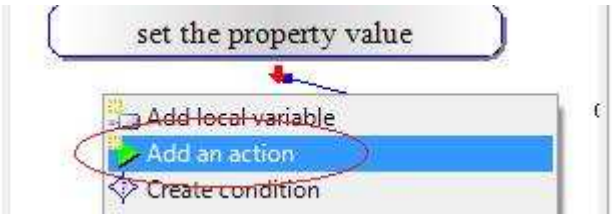


As an excise, you may try to create a local variable named amountLow to save the conditions for event SavingAmountLow:

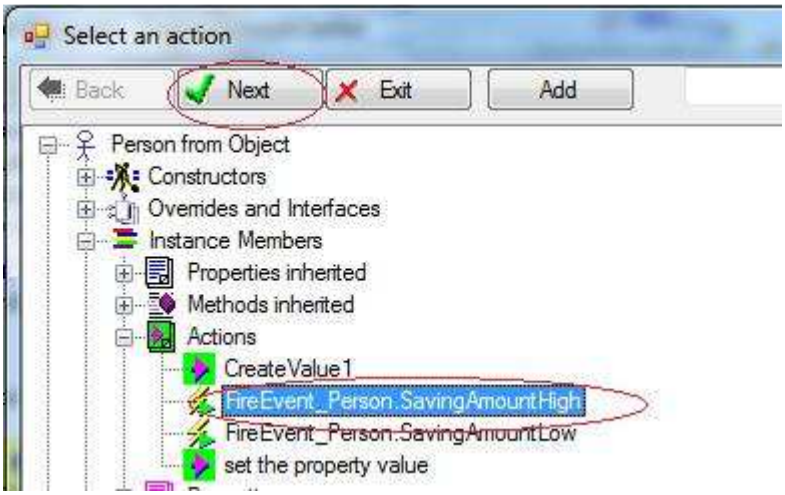
value < 50 and SavingAmount >= 50



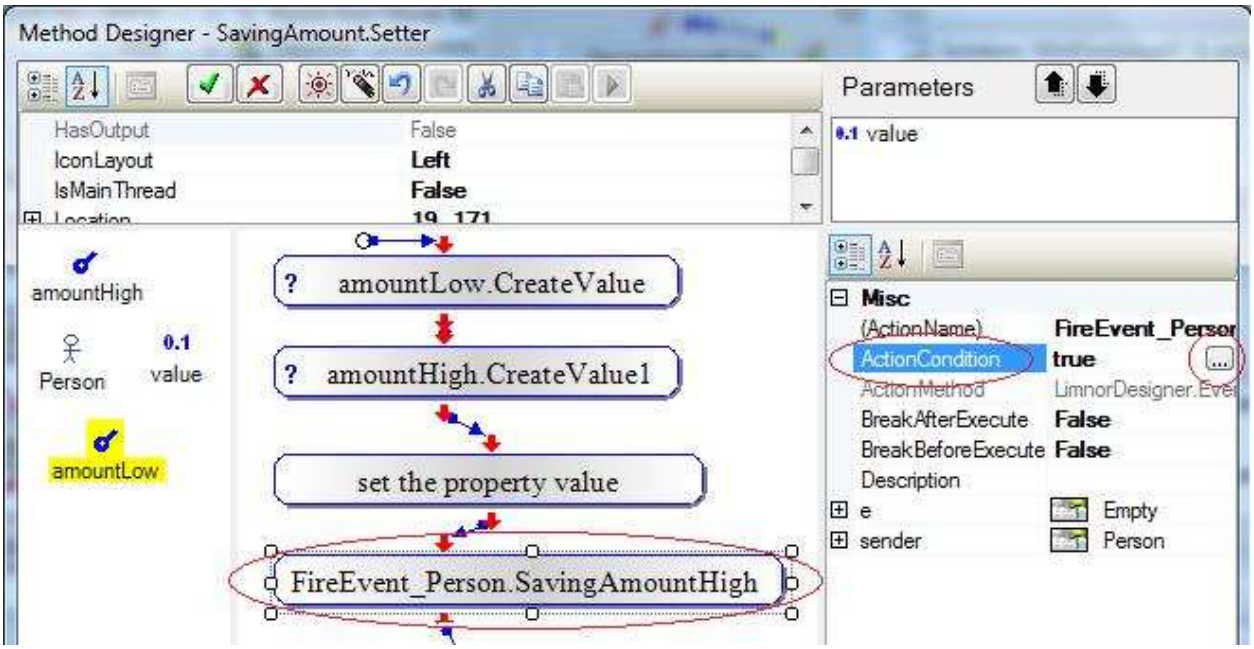
Now we may add event-firing actions into the method. Right-click the Method Editor, choose "Add an action":



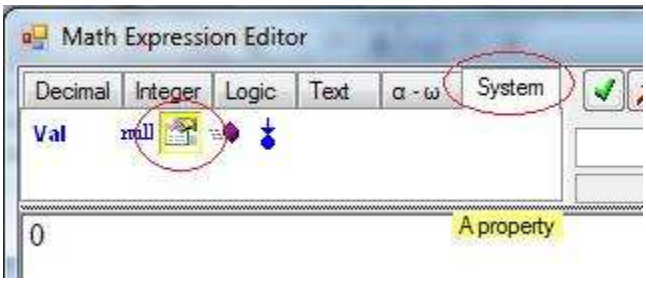
Select the event-firing action and click Next:



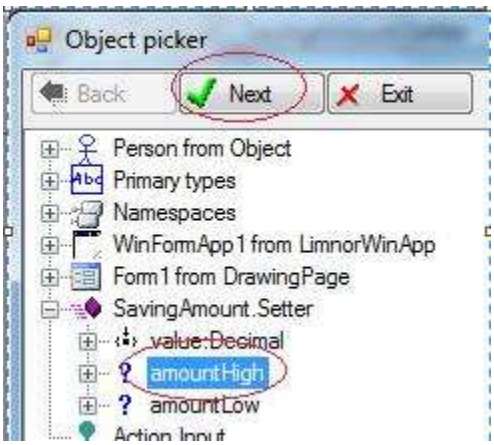
Click ... to set the action condition:



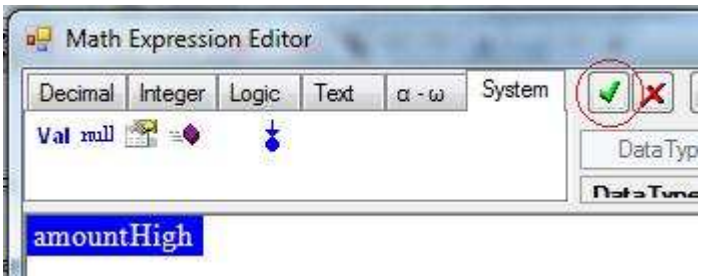
Click  to use the local variable:



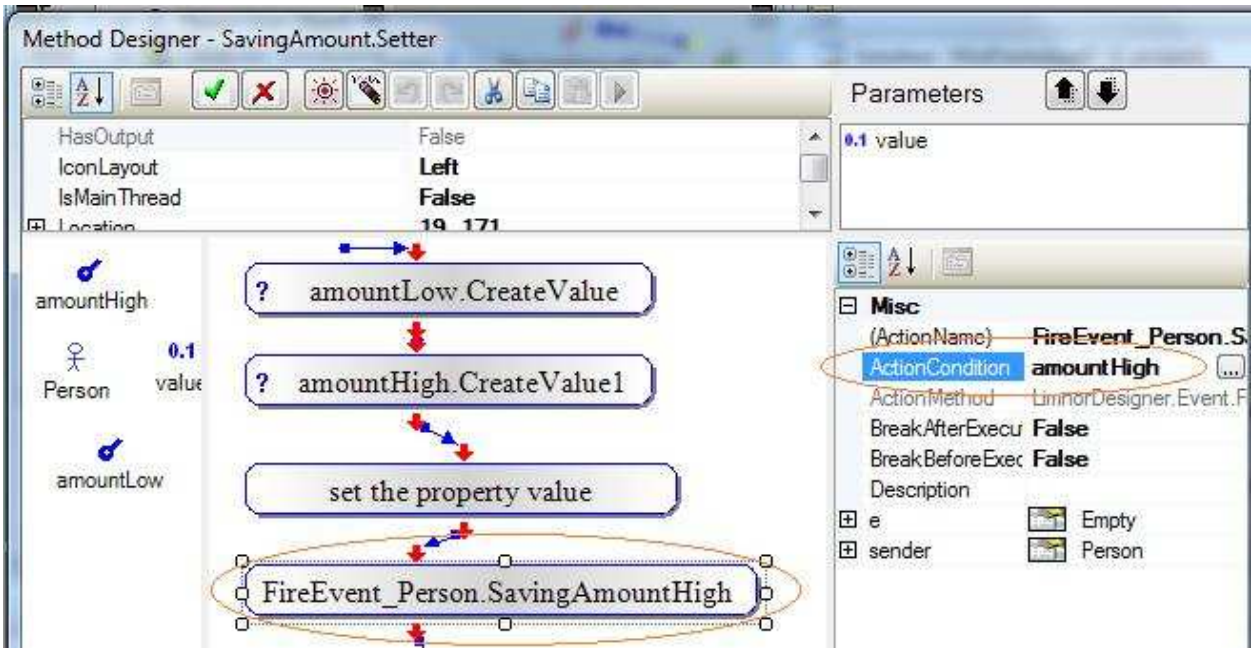
Select “amountHigh”, click Next:



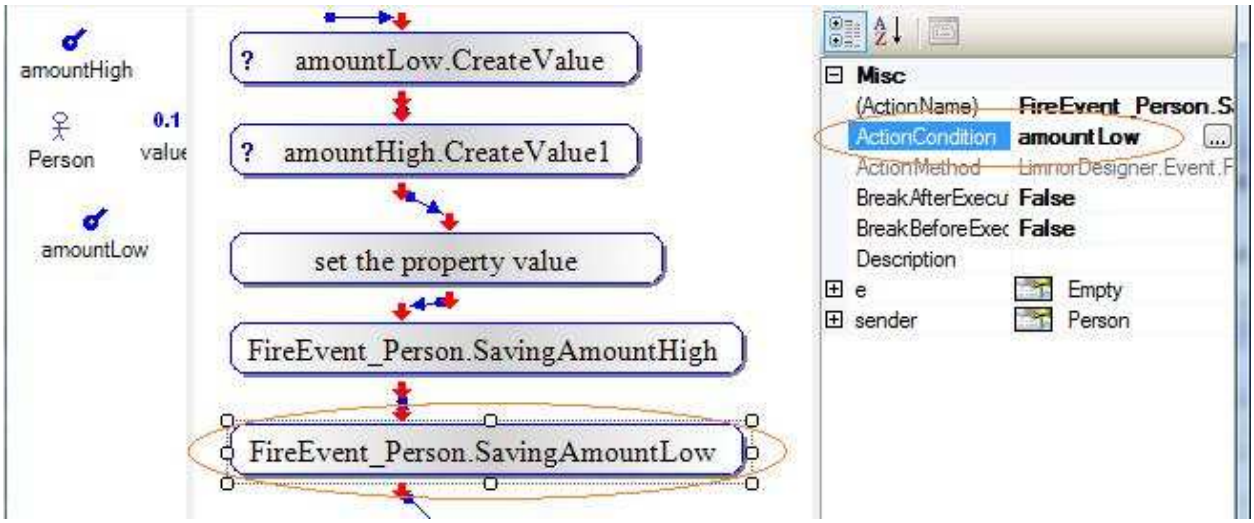
Click  to finish creating this expression:



Now we have the event-firing action, Fire_Event.SavingAmountHigh, added to the end of the method. Its ActionCondition is the variable amountHigh. So, if amountHigh is True, this event-firing action is executed, and thus the event SavingAmountHigh occurs.



We may also add the event-firing action, FireEvent_Person.SavingAmountLow, to the end of the method and set its ActionCondition to variable amountLow:



We are done. Click  to finish editing this Setter method for property SavingAmountHigh.

Now we finished implementing two events, SavingAmountLow and SavingAmountHigh. We defined them and then make them occur at the right moments.

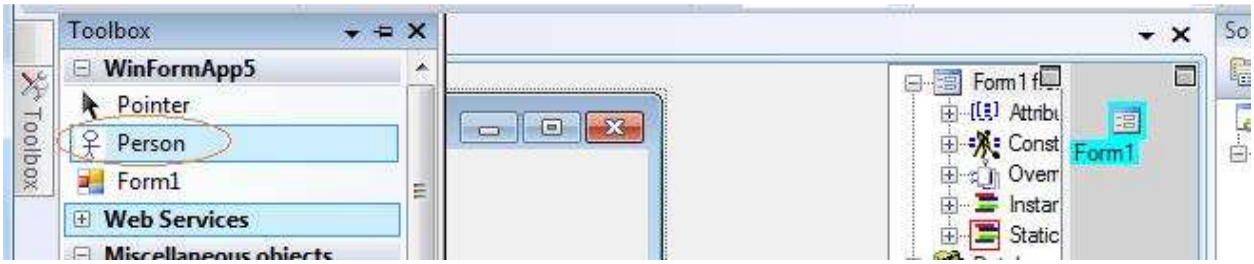
5 Test Usages of Properties, Methods, and Events

We have added properties, methods and events to the Person class. Using these properties, methods and events are in the same way as using properties, methods and events inherited from base classes.

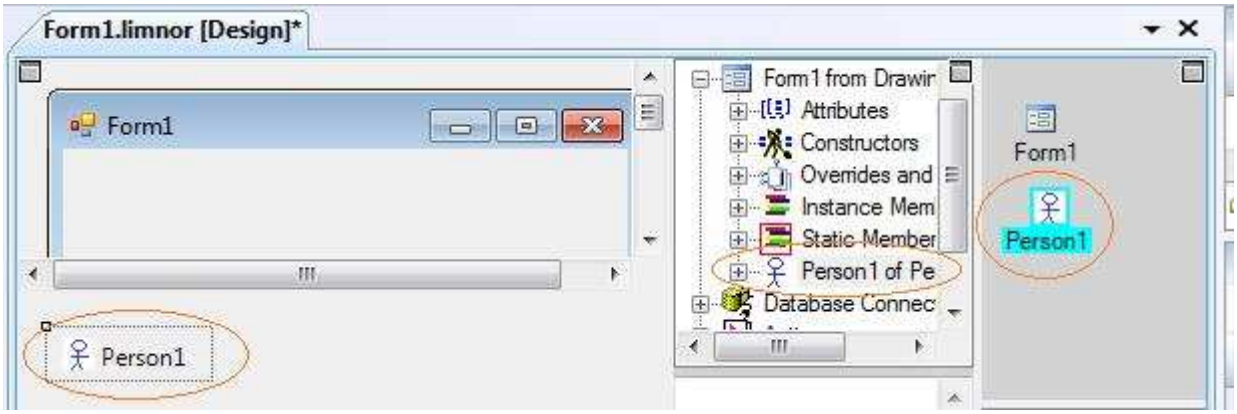
We create examples of using the properties, methods, and events of the Person class to show the functionality we added via adding these properties, methods, and events.

5.1 Create class instance

We use a Form to show the use of the Person class. Drop a Person to the Form:



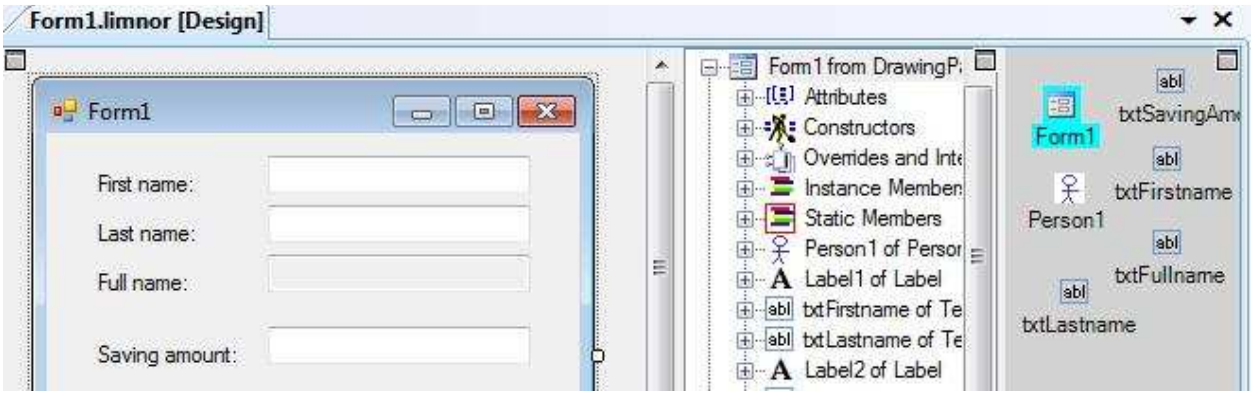
An instance of Person class appears in the designers:



5.2 Use properties

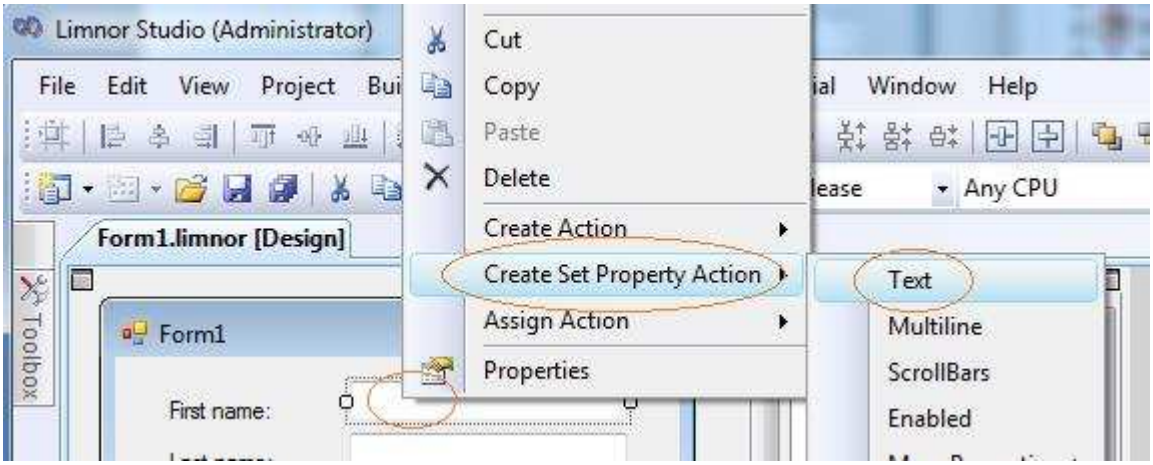
5.2.1 Get property values

We use Text Boxes to display properties of Person:

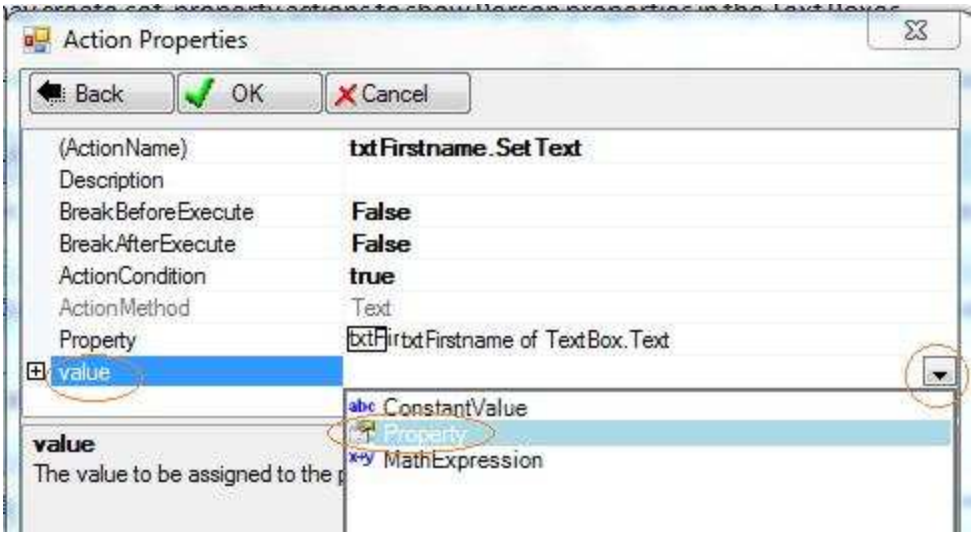


We may create set-property actions to show Person properties in the Text Boxes.

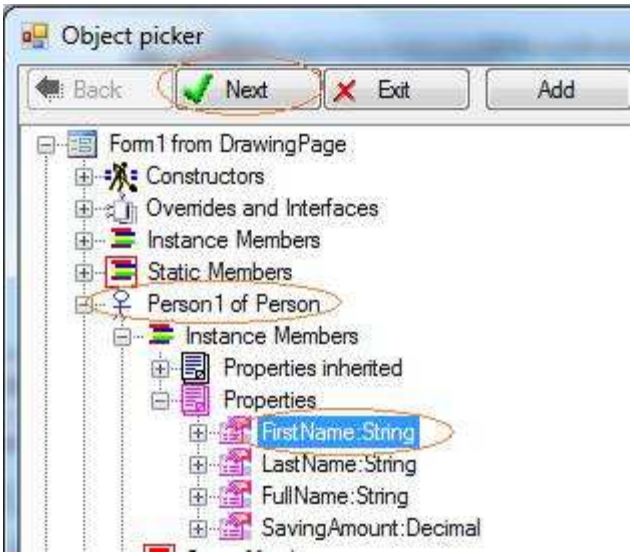
Right-click the Text Box for the first name, choose “Create Set Property Action”. Choose “Text” property:



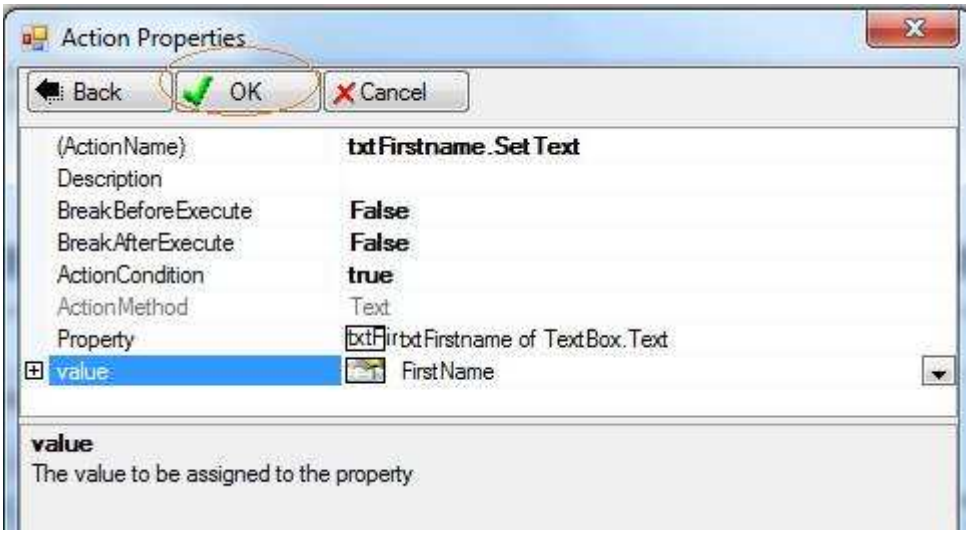
Choose Property for the value to select property from Person:



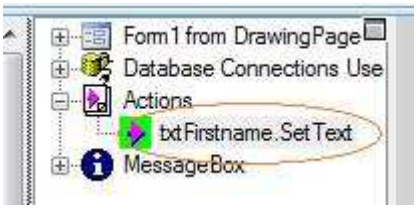
Select FirstName property, click Next:



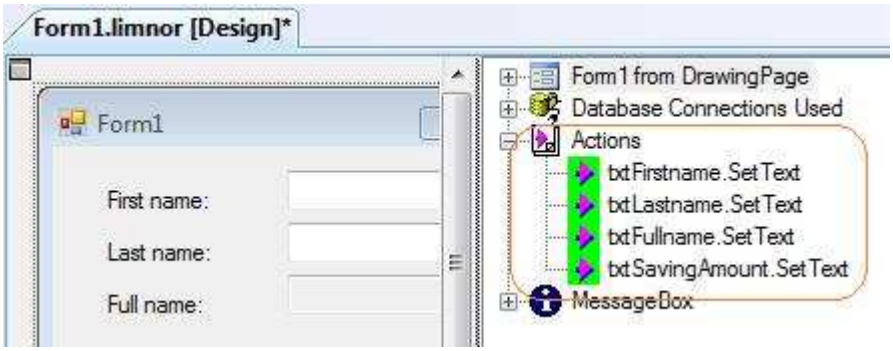
Click OK:



The new action is created:



Create set-property actions for getting other Person properties:



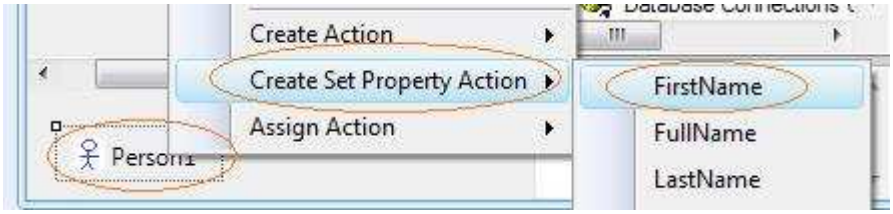
These actions get values from Person object and assign the values to text boxes.

5.2.2 Set property values

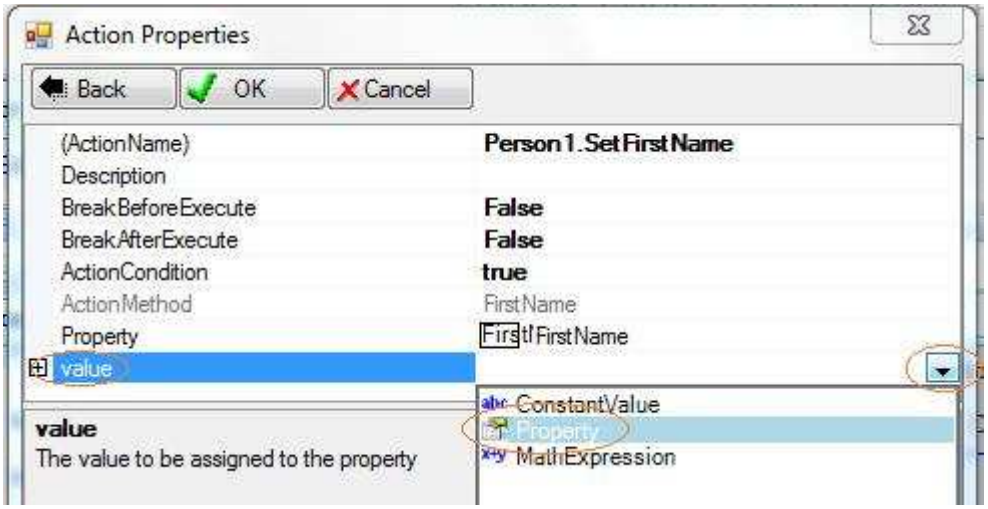
To change the property value of a Person object, a set-property action can be used.

We may use the value from the text boxes to change the property values of the corresponding Person properties.

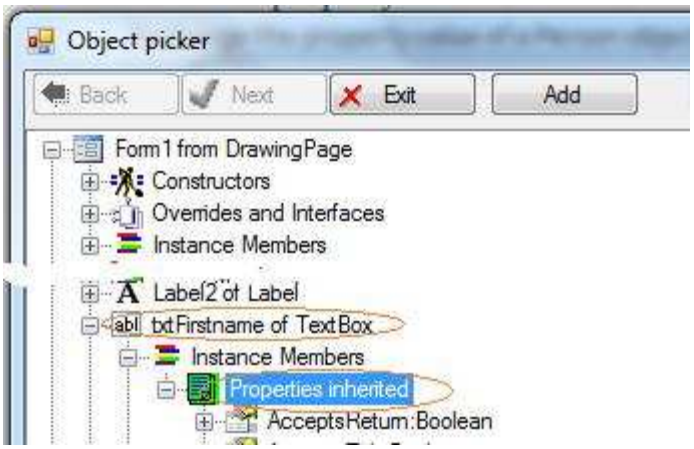
Right-click the Person object, choose “Create Set Property Action”. Choose “FirstName”:



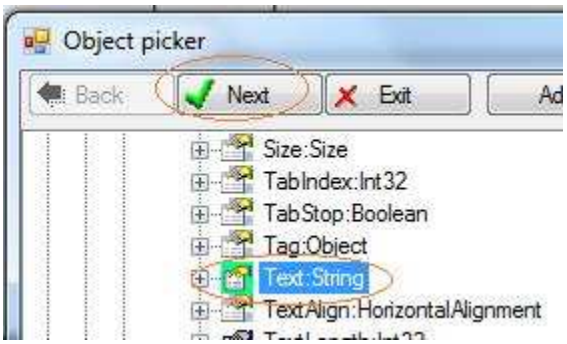
Select Property for the "value" parameter of the action for selecting value from a property:



Expand "Properties inherited" of text box txtFirstName:

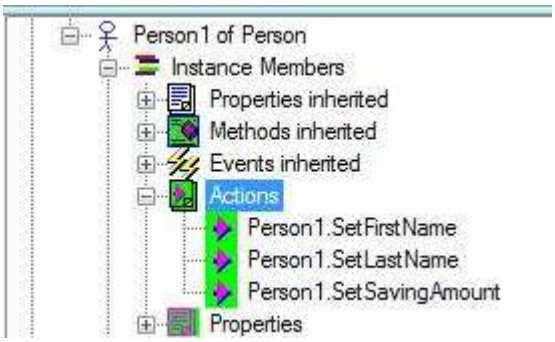


Scroll down. Select Text property. Click Next:



Click OK:

In the same way we may create other actions for setting Person properties using Text property from text boxes.



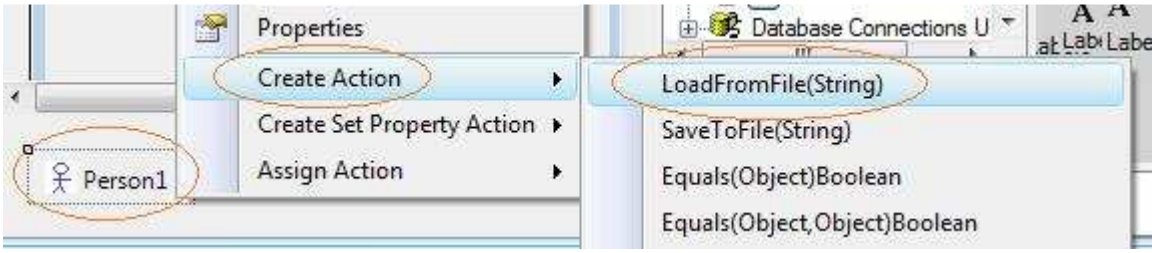
5.3 Use methods

Person class has a LoadFromFile method and a SaveToFile method. We create an action using each method.

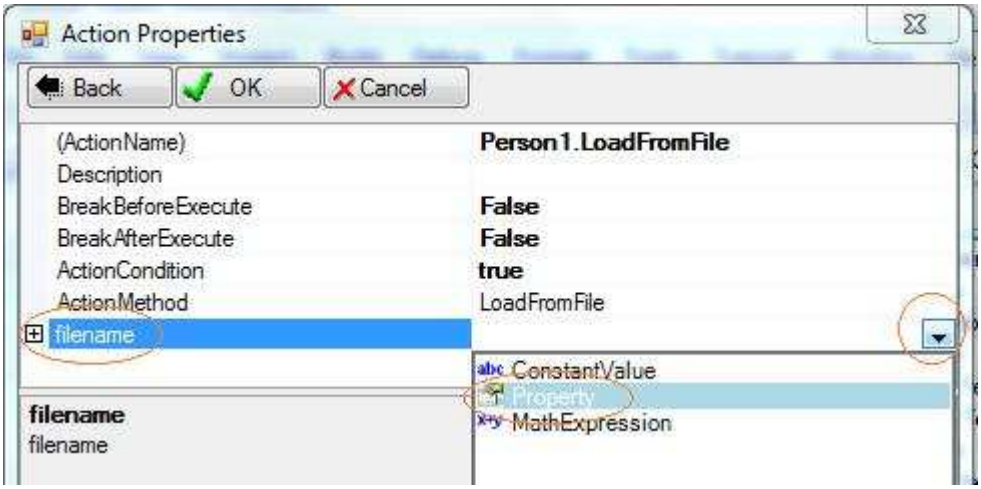
Both methods require a filename parameter. We use a text box for entering file name.

5.3.1 Create and use LoadFromFile action

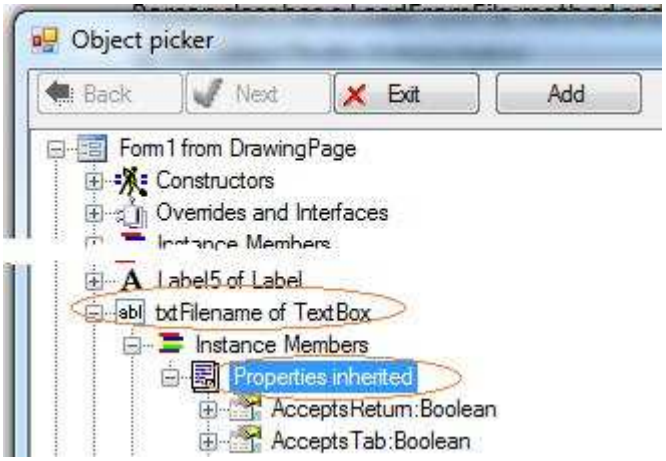
Right-click Person1, choose “Create Action”. Choose “LoadFromFile”:



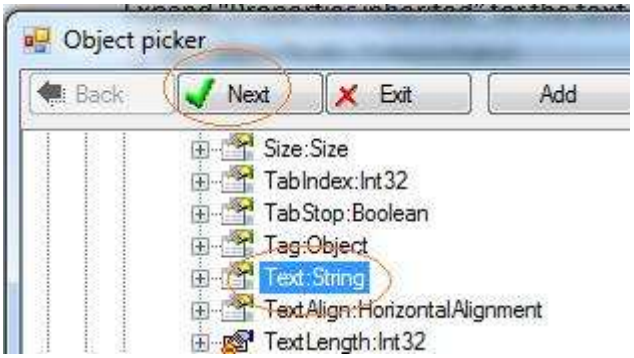
Choose Property for the filename parameter for selecting the Text property of the text box:



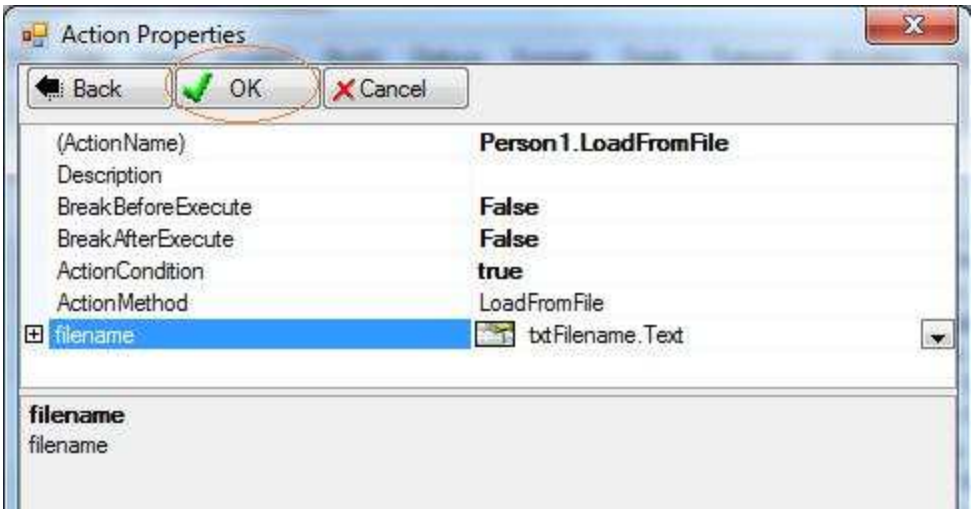
Expand “Properties inherited” for the text box for the file name:



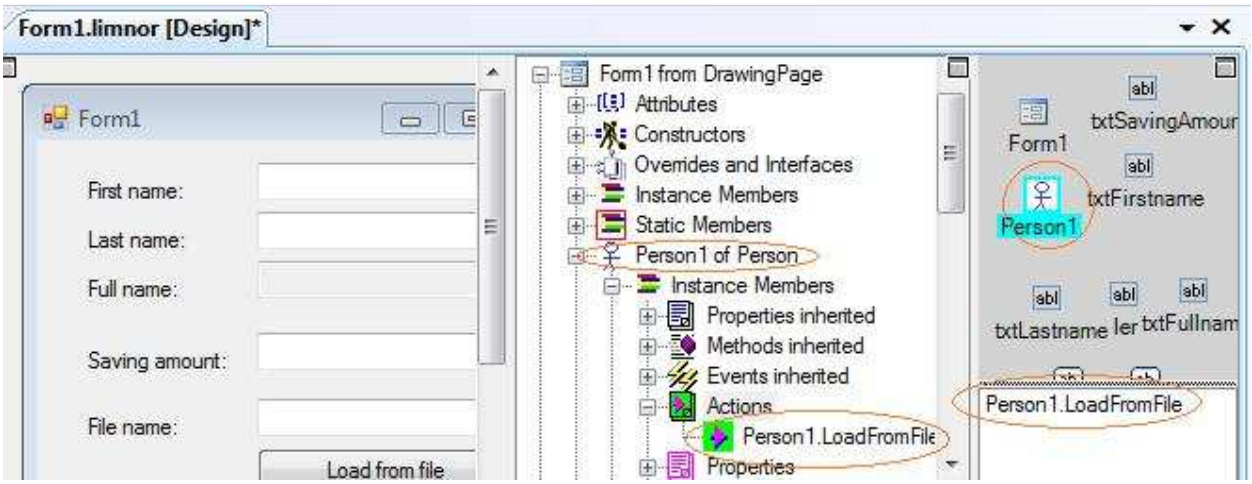
Scroll down to select Text property and click Next:



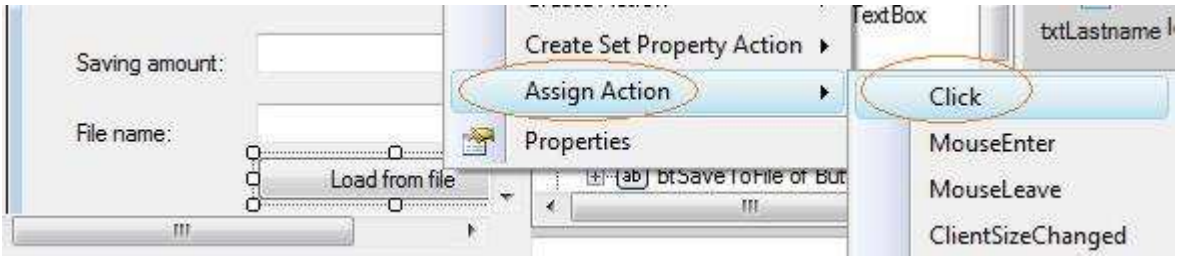
Click OK:



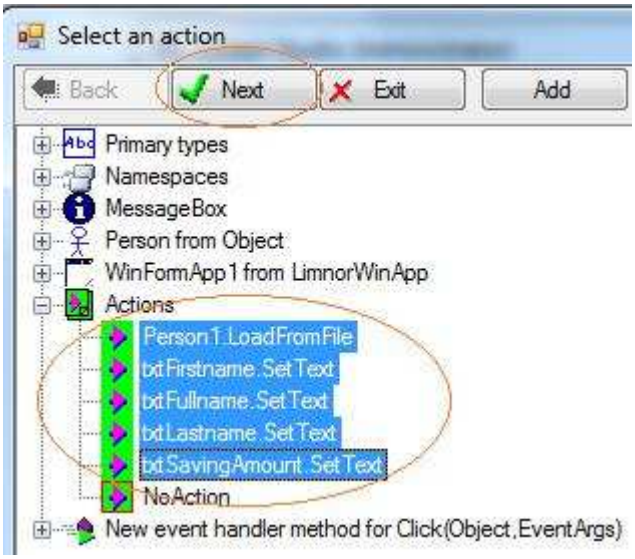
The action is created:



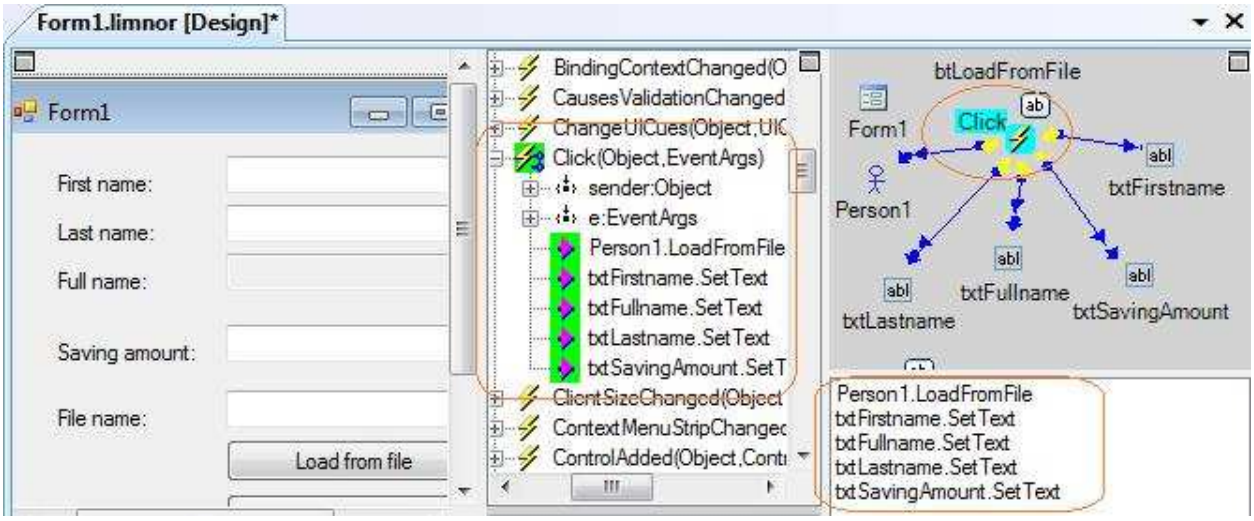
Assign actions to button “Load from file”. Right-click the button, choose “Assign Action”. Choose “Click”:



Select actions:

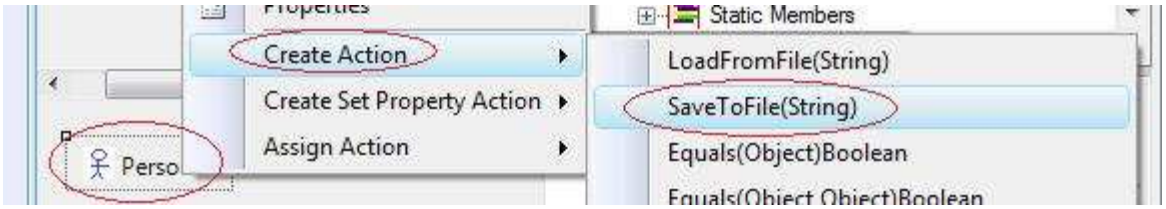


The actions assigned to the event:

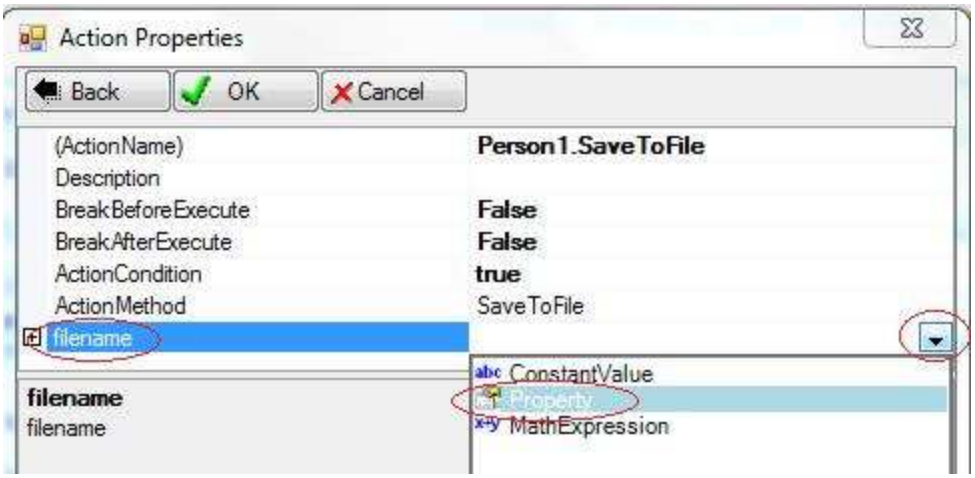


5.3.2 Create and use SaveToFile action

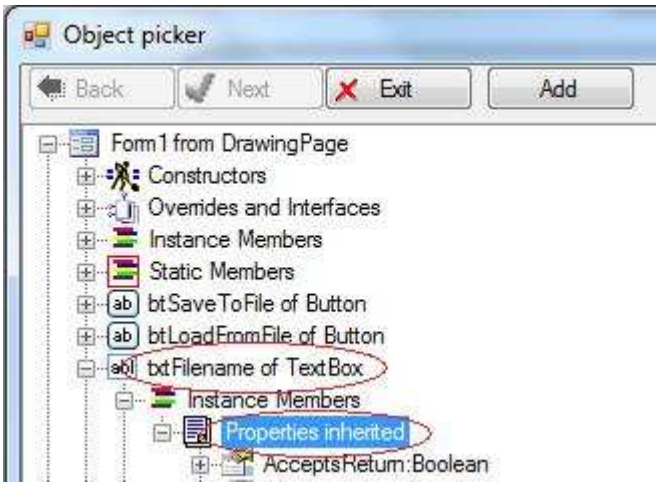
Right-click Person1, choose “Create Action”. Choose “SaveToFile”:



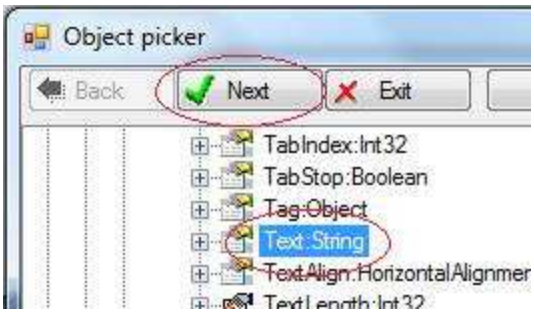
For filename parameter, choose Property to select value from the text box:



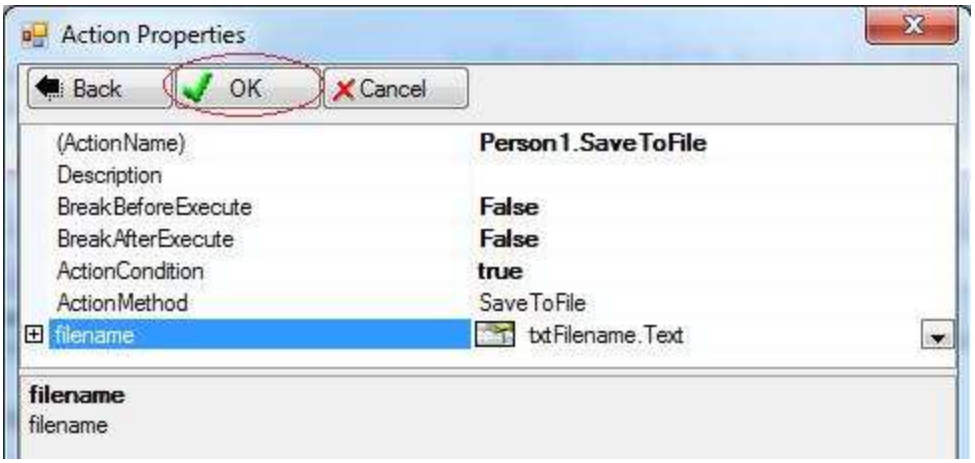
Expand “Properties inherited” node of the text box txtFilename:



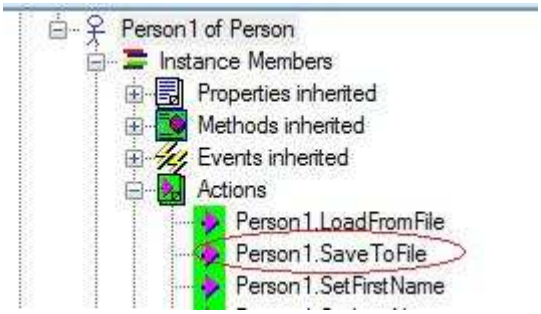
Scroll down. Select Text. Click Next:



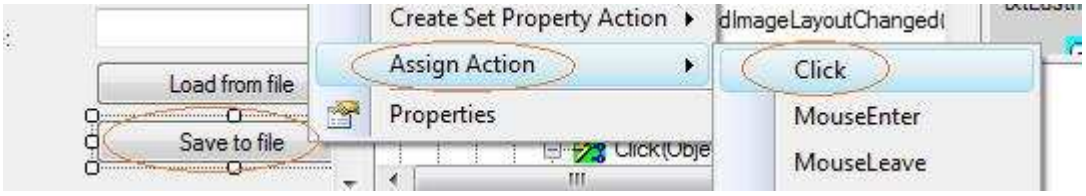
Click OK:



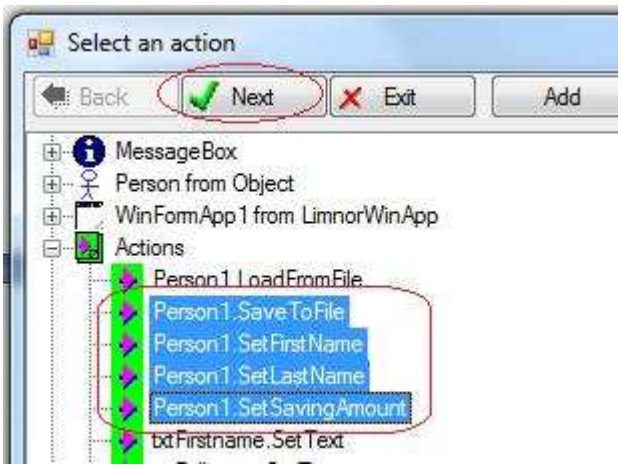
The new action is created:



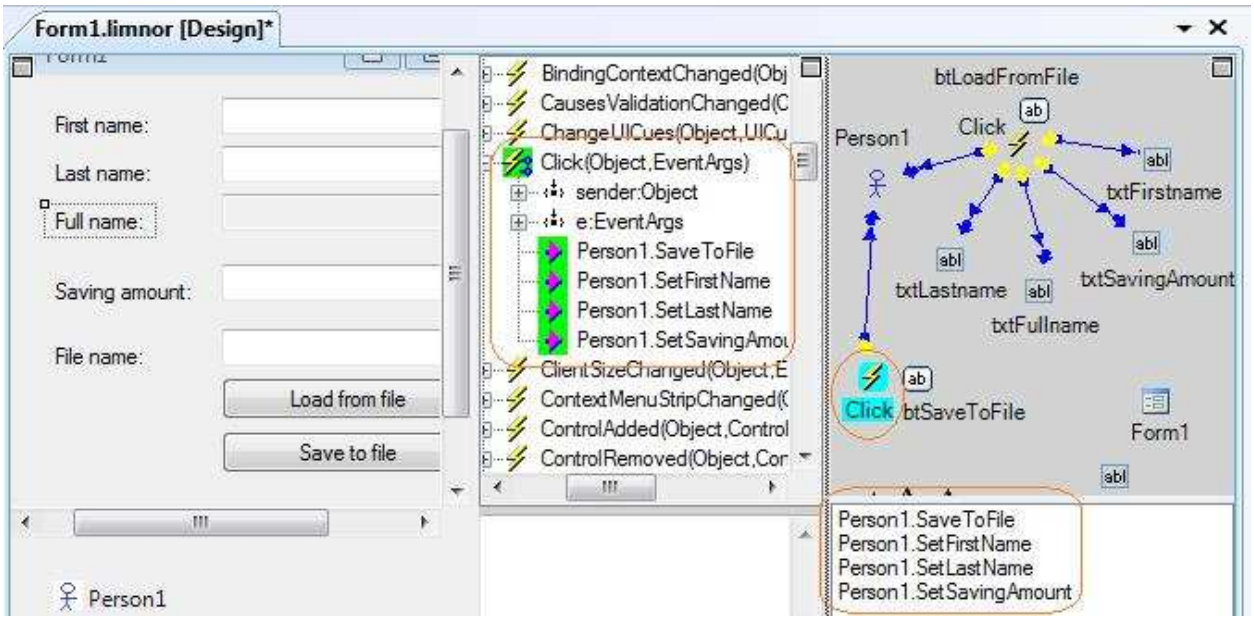
Assign actions to button “Save to file”. Right-click the button. Choose “Assign Action”. Choose “Click”:



Select the actions for this button. Click Next:

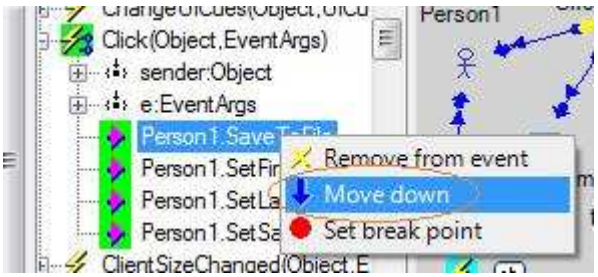


The actions are assigned to the button:

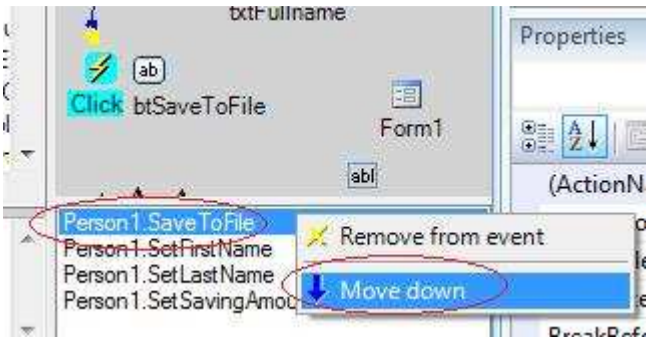


The action SaveToFile is the first action. We want to move it to be the last action because we want to set the properties of the Person object first before saving the values to file.

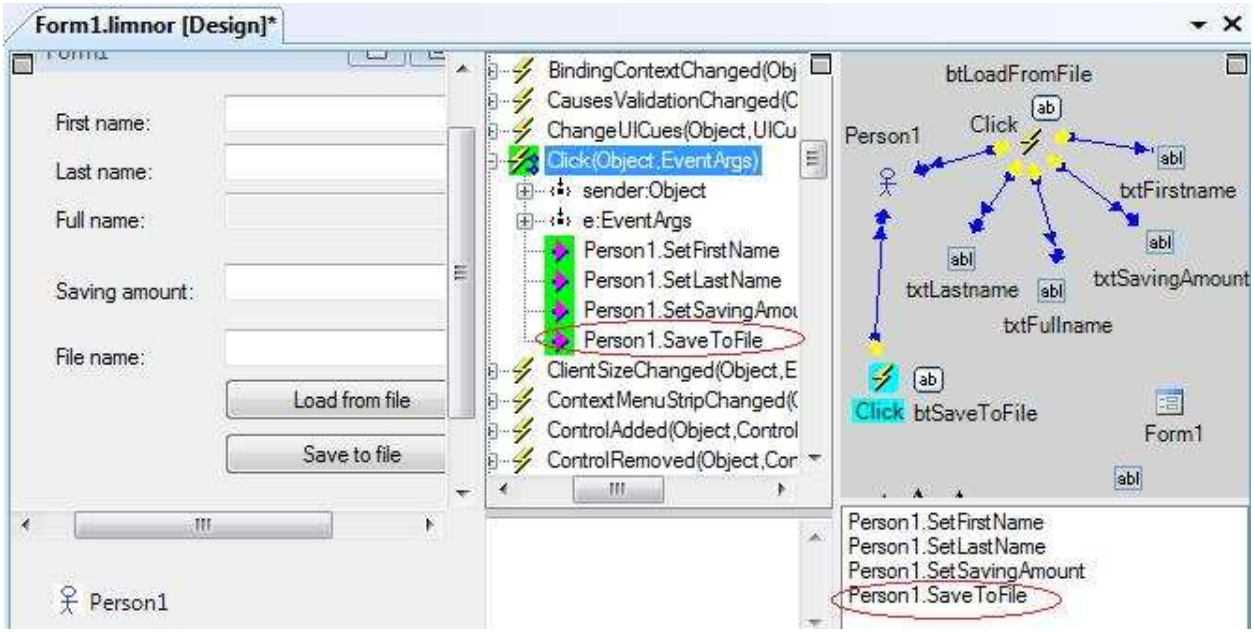
To use the Object Explorer to move an action assigned to an event, right-click the action, choose “Move down” or “Move up”:



To use the Event Path to move an action assigned to an event, right-click the action, choose “Move down” or “Move up”:



Keep moving SaveToFile down until it is the last action:



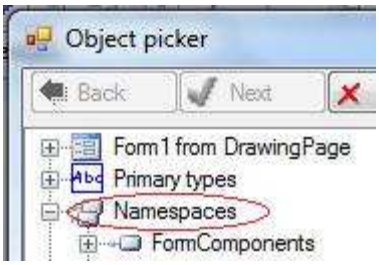
5.4 Use events

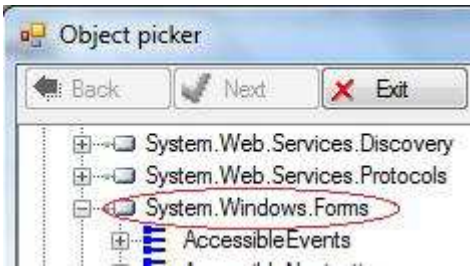
We developed two events for Person class: SavingAmountHigh and SavingAmountLow. Now we are going to show examples of using these events.

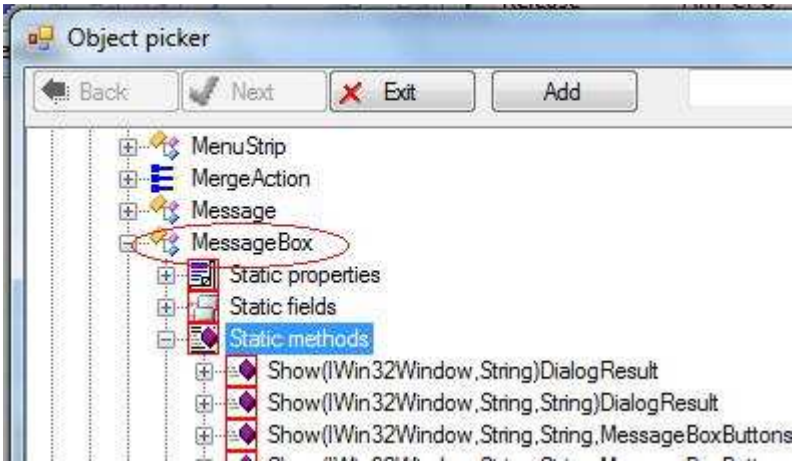
Our use of these events is very simple: just show a message box saying “Saving amount is low” when event SavingAmountLow occurs; and show a message box saying “Saving amount is high” when event SavingAmountHigh occurs.

5.4.1 Create actions to be executed at the events

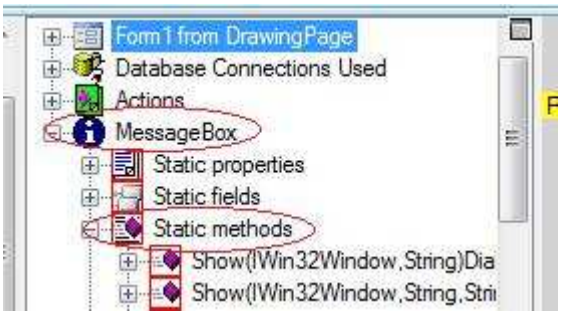
Microsoft .Net Framework provides a class MessageBox to show message box. It can be found under System.Windows.Forms:



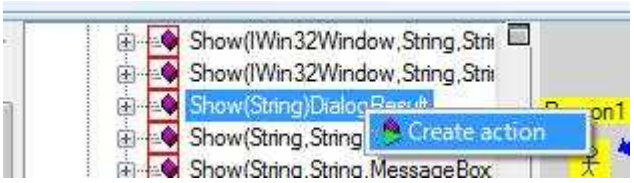




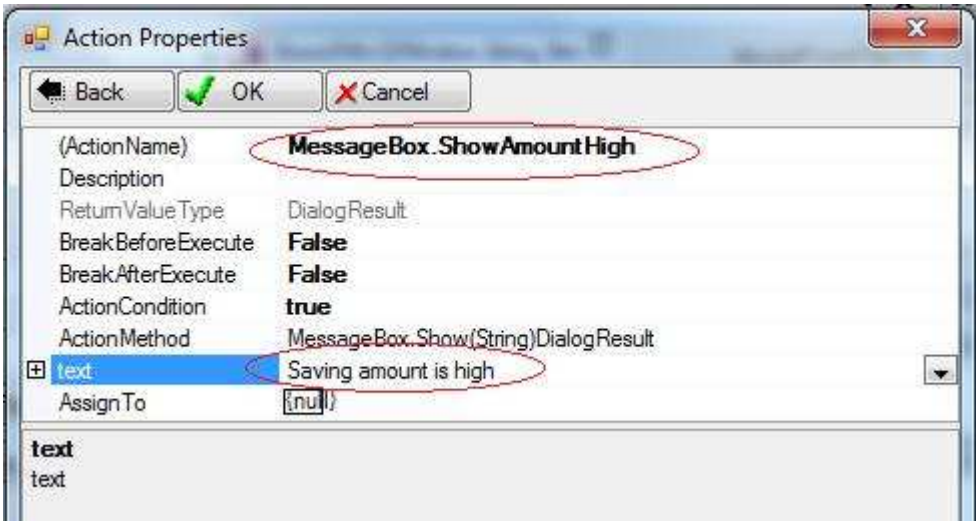
For your convenience, class MessageBox is listed in the Object Explorer:



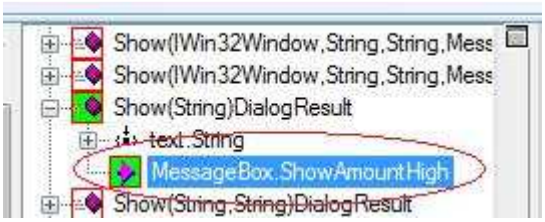
Scroll down to choose a method we want to use. Right-click the method, choose “Create action”:



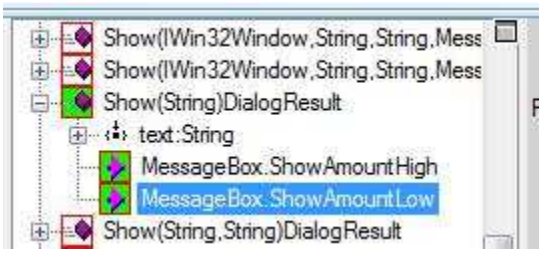
Change the action name and type in message text to be displayed:



The new action appears:

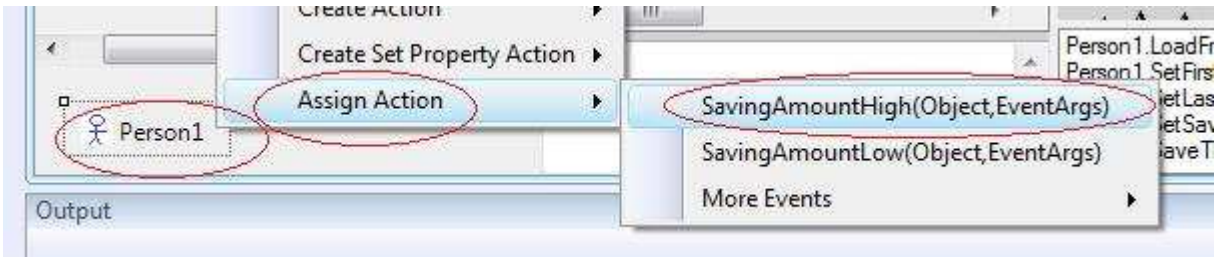


We may create another action to show text “Saving amount is low”:

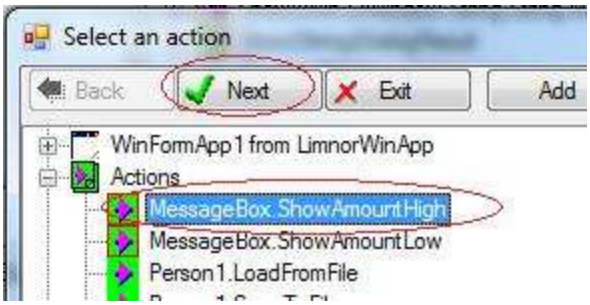


5.4.2 Assign actions to events

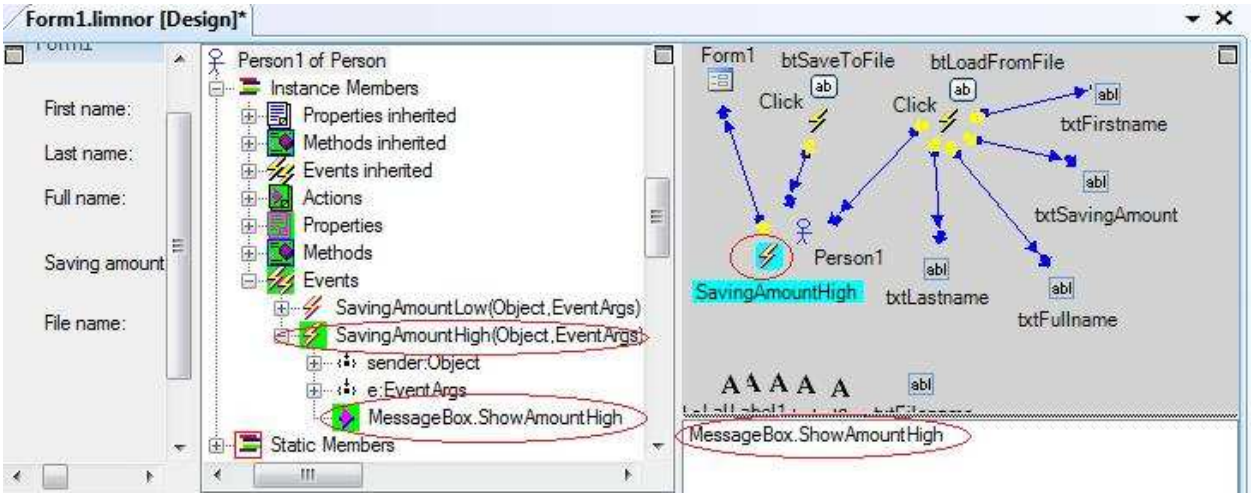
Right-click Person1, choose “Assign Action”. Choose “SavingAmountHigh”:



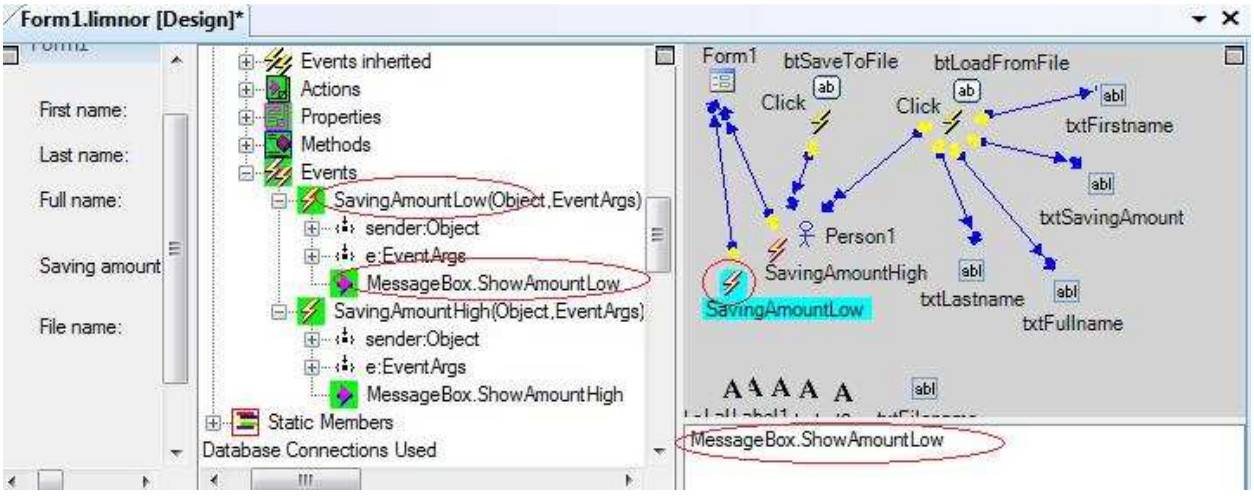
Choose action MessageBox.ShowAmountHigh, click Next:



The action is assigned to the event:



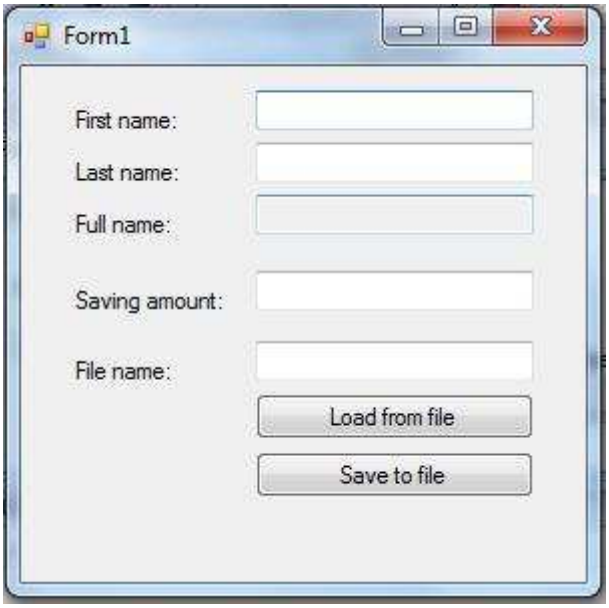
We need assign action MessageBox.ShowAmountLow to event SavingAmountLow:



5.5 Test run

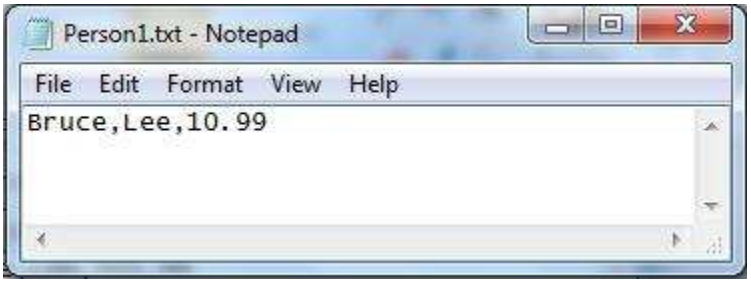
Click  to compile and run the application.

The form appears:



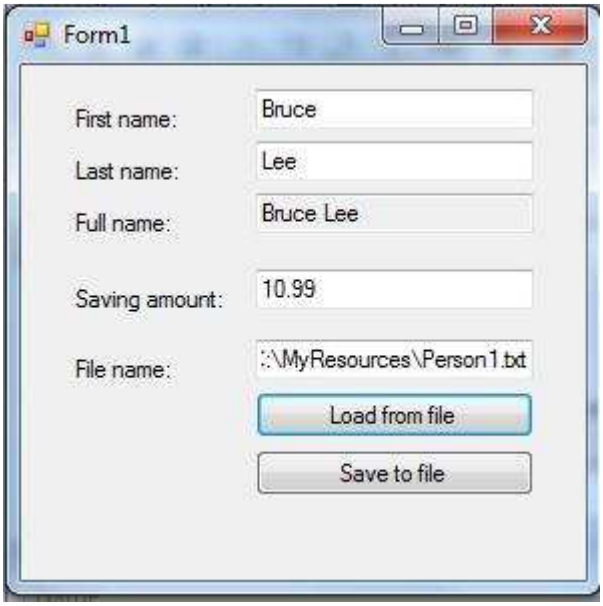
5.5.1 Load data from file

Suppose we have a text file storing a Person data:



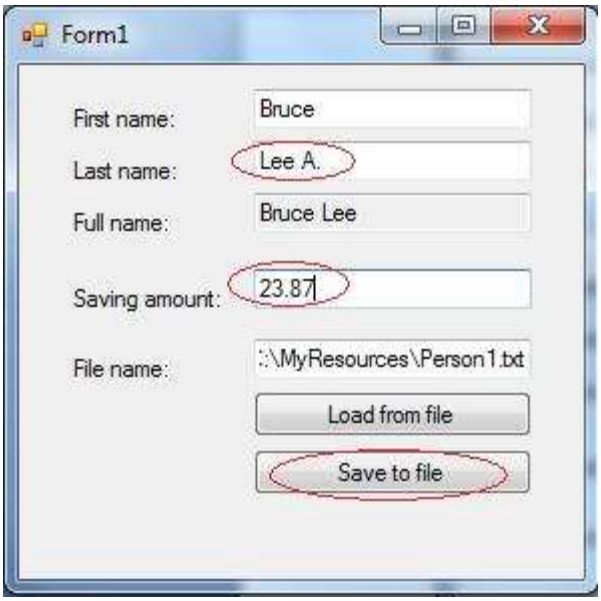
We enter the file path and click button “Load from file”:

The data from the text file is loaded into object Person1 and displayed in the text boxes:

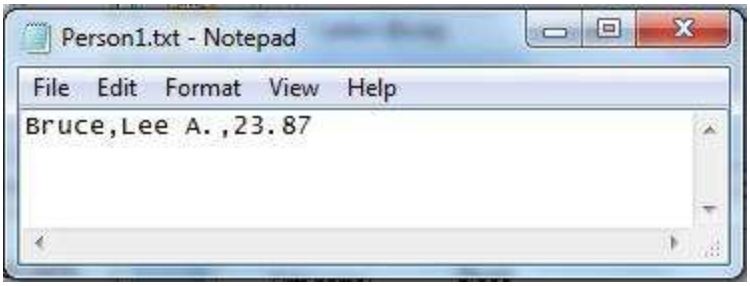


5.5.2 Save data to file

Modify some data in the text boxes and click button “Save to file”:



The data are modified in the text file:



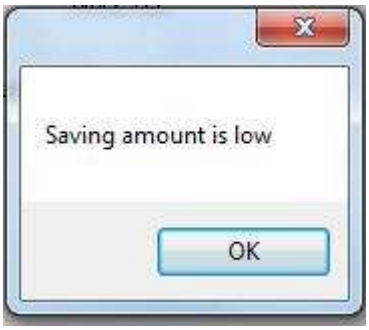
5.5.3 Trig events

Set "Saving amount" to 112, which is larger than 100. Click "Save to file".

The value 112 is assigned to object Person1. Since 112 is larger than 100 and the original saving amount, 23.87, is smaller than 100, the SavingAmountHigh event occurs. The message box display action, MessageBox.ShowAmountHigh, is executed:

Set “Saving amount” to 32, which is smaller than 50. Click “Save to file”.

The value 32 is assigned to object Person1. Since 32 is smaller than 50 and the original saving amount, 112, is larger than 50, the SavingAmountLow event occurs. The message box display action, MessageBox.ShowAmountLow, is executed:



6 Summary

This part of User's Guide shows how to develop a new class by adding new properties, methods and events to the class.

It shows that the use of the new properties, methods and events is the same as the use of properties, methods and events inherited from the base classes.

