Use Default Form Instances

Created: 2011-01-03   Modified:2012-07-05

Contents

Introduction

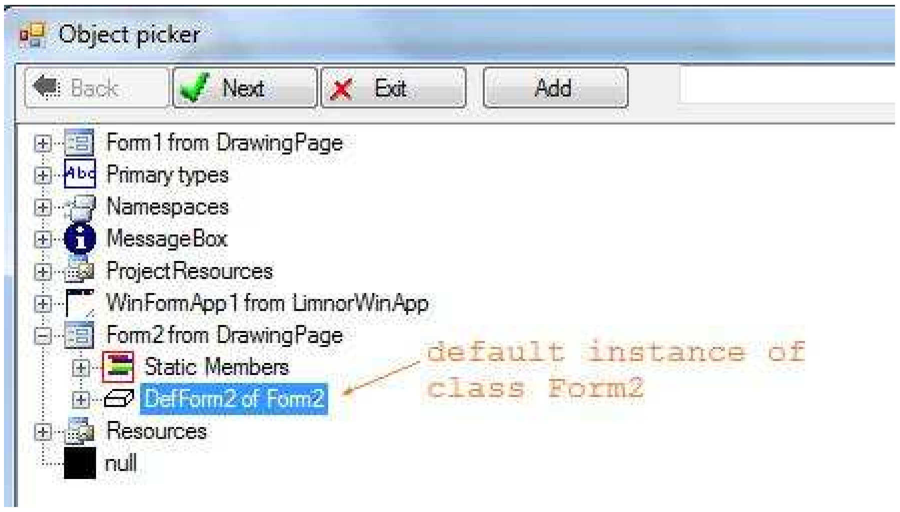
To simplify the development of Windows Form applications, since build {1.2.1.279}, Limnor Studio invented a new concept of “default instance” for Forms. For developers familiar with object-oriented programming, it saves the instance creation operations. For beginners, it allows them to develop programs before fully understanding related programming concepts such as class/type, instance, static, etc.

Form is a special class comparing to other classes. For example, when a form variable is out of scope it may still valid.

In providing support for Limnor Studio users, form-instance-creation and instance-accessing proved to be a difficult issue for many users.

On the other hand, when a developer is creating a Form class, in most cases the developer does not have the intention of creating multiple instances out of the form class. A form class looks so concrete in the Form Designer. Most developers regard a form class being developed as an instance, not a template for creating many instances. ”default instance” is thus invented to make a form class concrete.

Limnor Studio automatically adds a public static instance to every Form class a developer creates, as the “default instance” for the class.



Supporting of "default instance" is an added feature. The other documentations on Form developments are still valid. You may still create multiple instances from a form class.

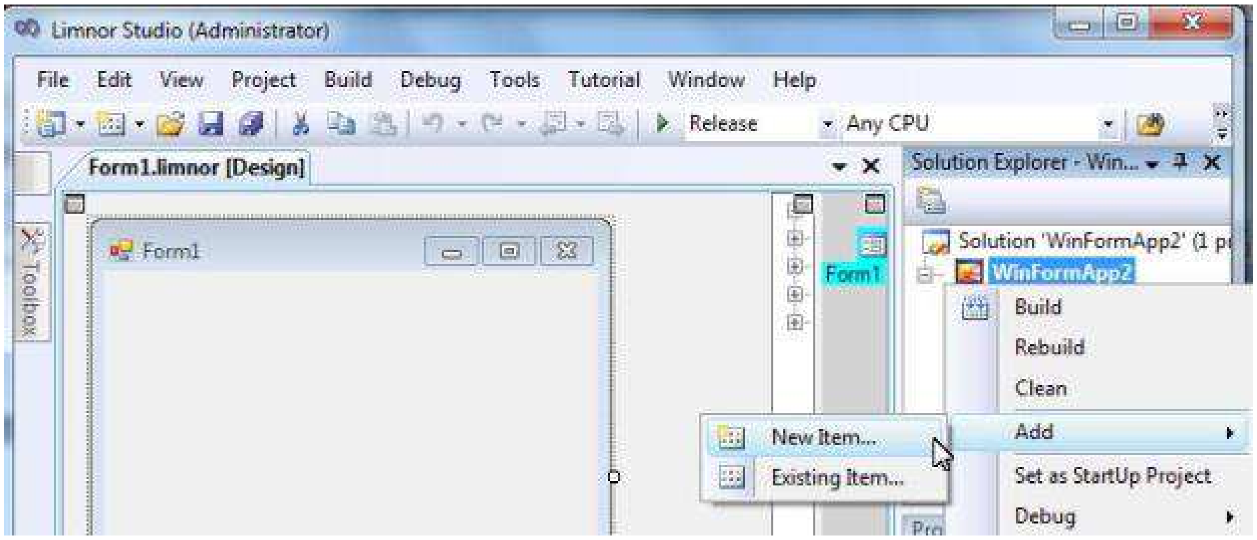
This article describes the use of "default instance". You can see that in most aspects the programming is much simplified.

In this article, the name of a Form class is used for referring both the Form class and the default instance of the class.

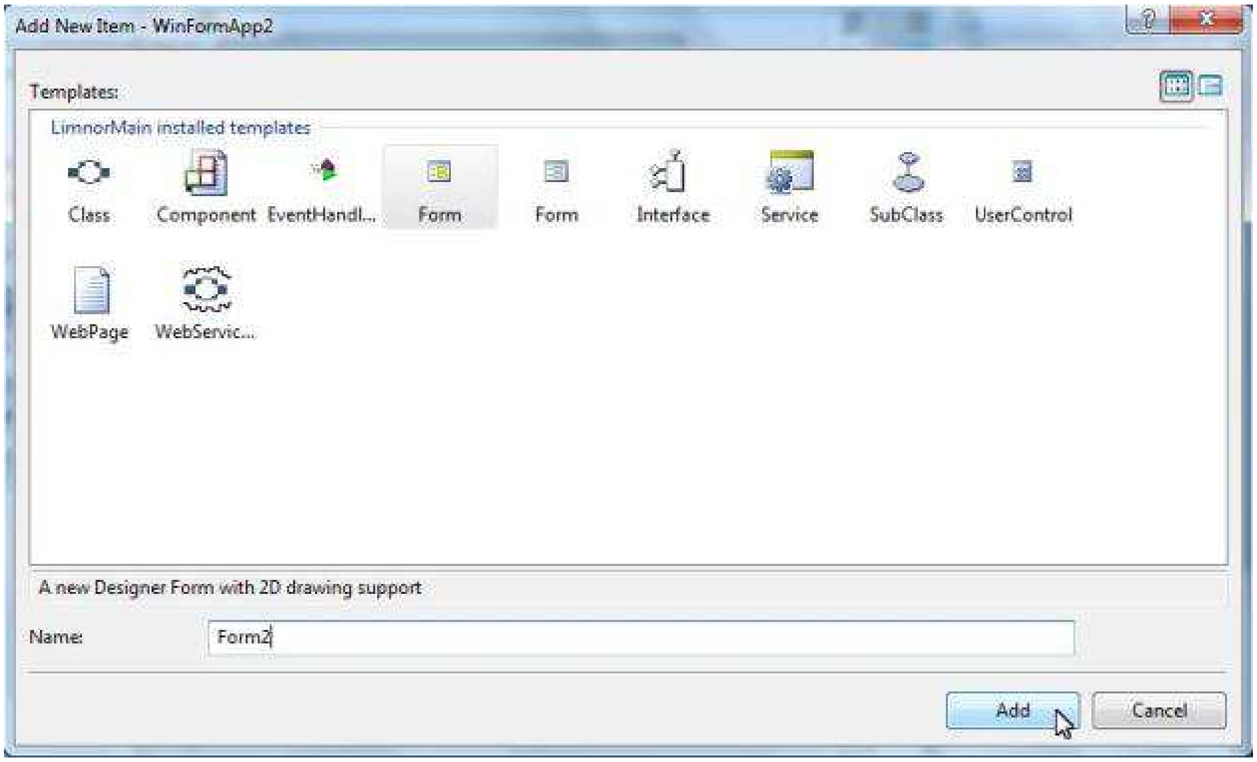
Add Form Classes

The contents of this article make sense only for programs using more than one form.

To add a new Form class, right-click the project; choose "Add"; choose "New Item…":


Limnor Studio (Administrator)
File Edit View Project Build Debug Tools Tutorial Window Help
Release
Any CPU
Form1.limnor [Design]
Solution Explorer - Win...
Form1
Solution 'WinFormApp2' (1 p
WinFormApp2
Build
Rebuild
Clean
Add
Set as StartUp Project
Debug
New Item...
Existing Item...
Toolbox

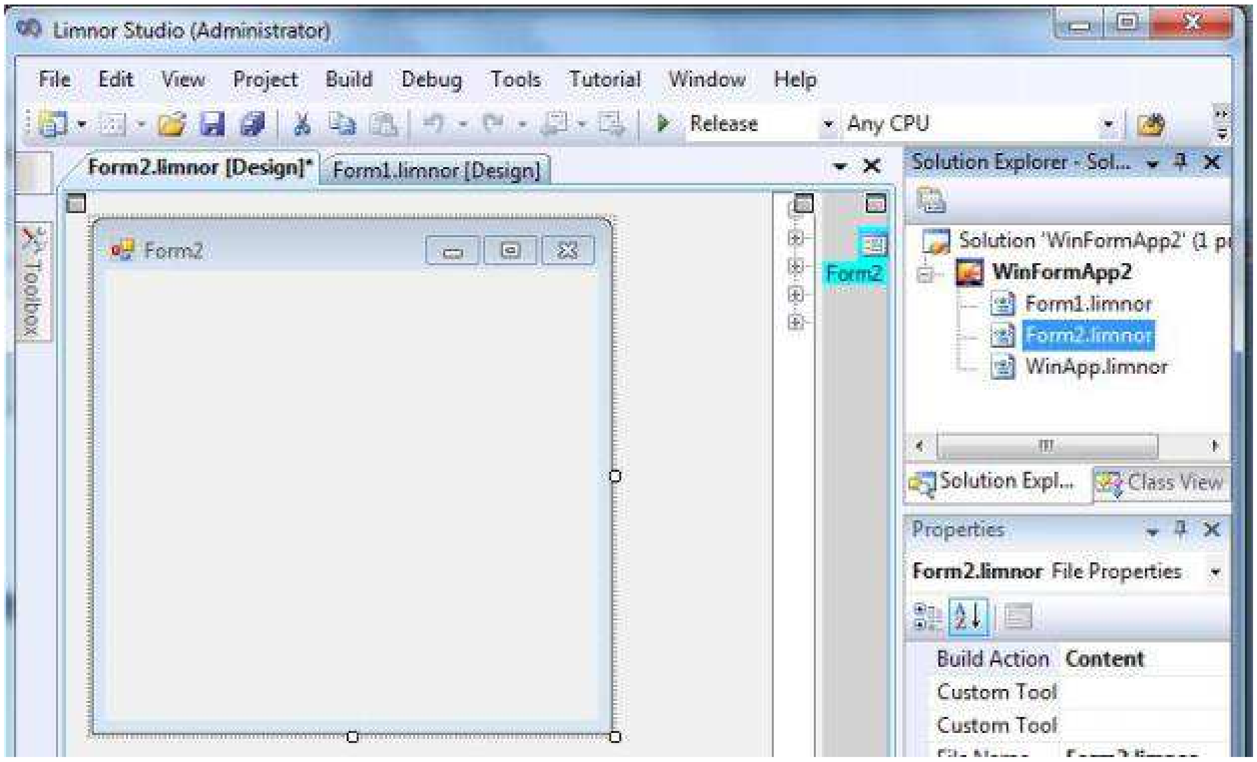
Choose "Form" and give the new class a name, for example, Form2, click Add:



A new Designer Form with 2D drawing support

Name: Form2

A new Form class is added to the project:

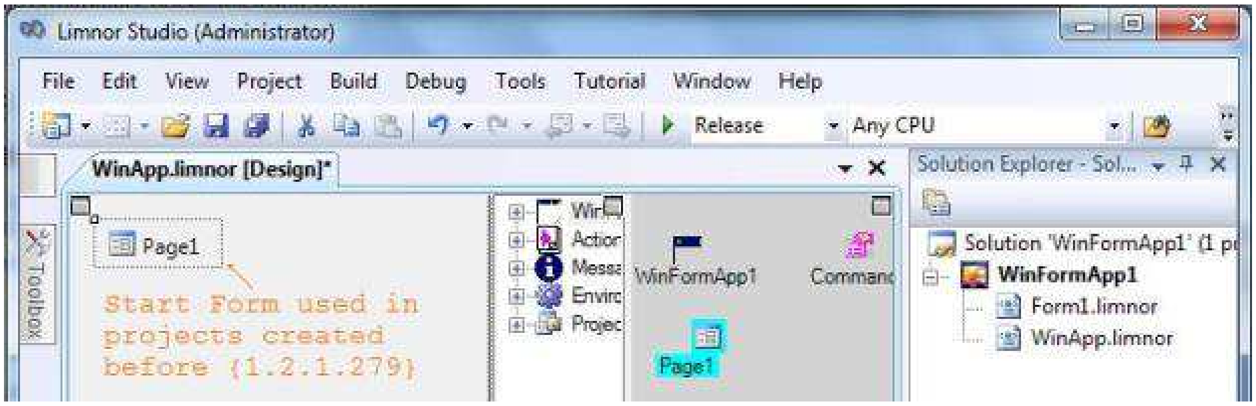


Build Action	Content
Custom Tool	
Custom Tool	

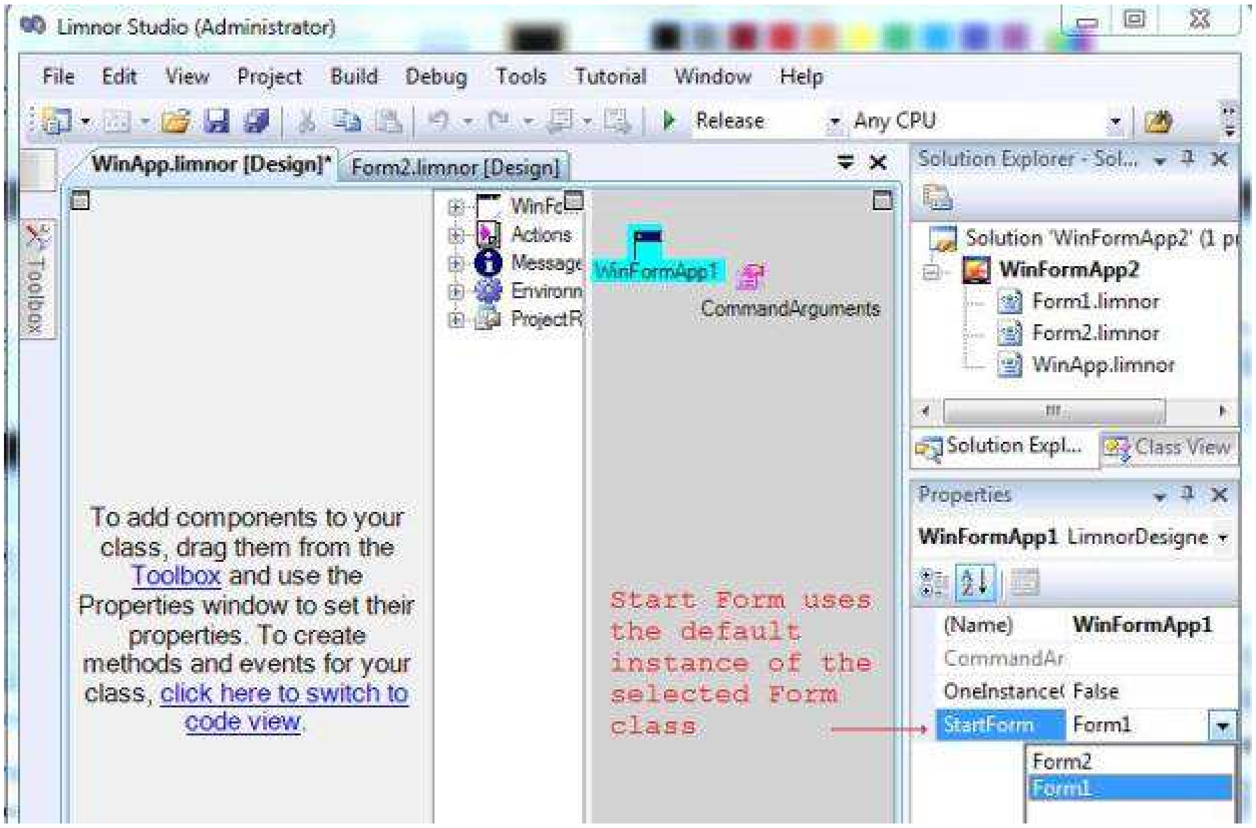
Starting Form (Home Page)

Every Windows Form application has a “Starting Form”. It is the first form appears on the screen when the application starts. If this “Starting Form” is closed then the application shuts down.

Open a project created before build {1.2.1.279}, we can see that an instance of Form1 named Page1 is created and used as the “Starting Form”:


Limnor Studio (Administrator)
File Edit View Project Build Debug Tools Tutorial Window Help
Release
Any CPU
WinApp.limnor [Design]*
Solution Explorer - Sol...
Page1
Start Form used in projects created before {1.2.1.279}
WinFormApp1
Command
Page1
Solution 'WinFormApp1'
WinFormApp1
Form1.limnor
WinApp.limnor

For a project created since build {1.2.1.279}, we can see that the instance Page1 is not used. The default instance of a selected form class is used as the Starting Form:

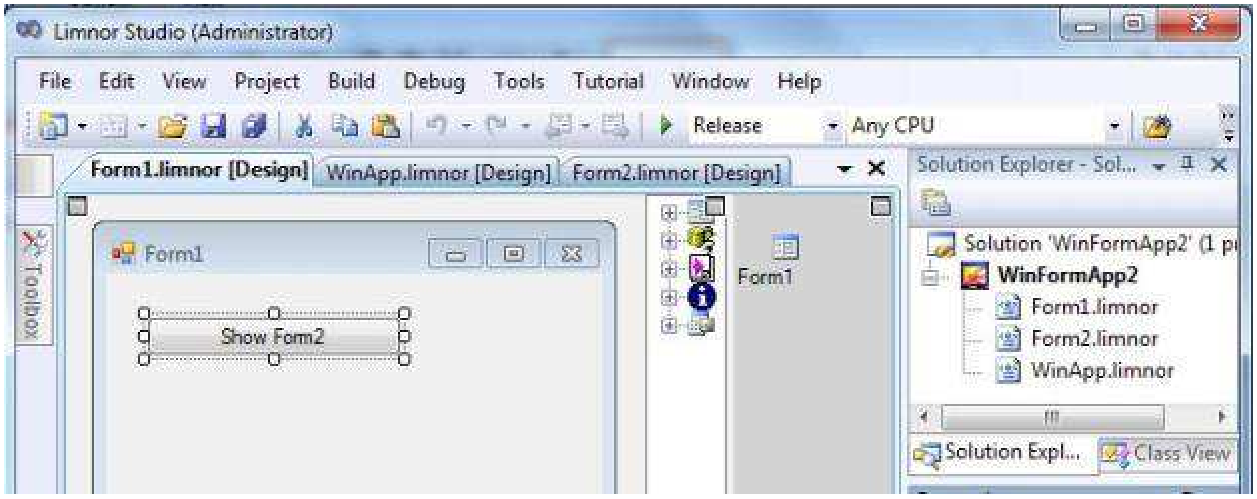

Limnor Studio (Administrator)
File Edit View Project Build Debug Tools Tutorial Window Help
Release
Any CPU
WinApp.limnor [Design]*
Form2.limnor [Design]
Solution Explorer - Sol...
Solution 'WinFormApp2'
WinFormApp2
Form1.limnor
Form2.limnor
WinApp.limnor
Solution Expl...
Class View
Properties
WinFormApp1 LimnorDesigne
WinFormApp1
CommandArguments
Start Form uses the default instance of the selected Form class
Form2
Form1

To add components to your class, drag them from the Toolbox and use the Properties window to set their properties. To create methods and events for your class, click here to switch to code view.

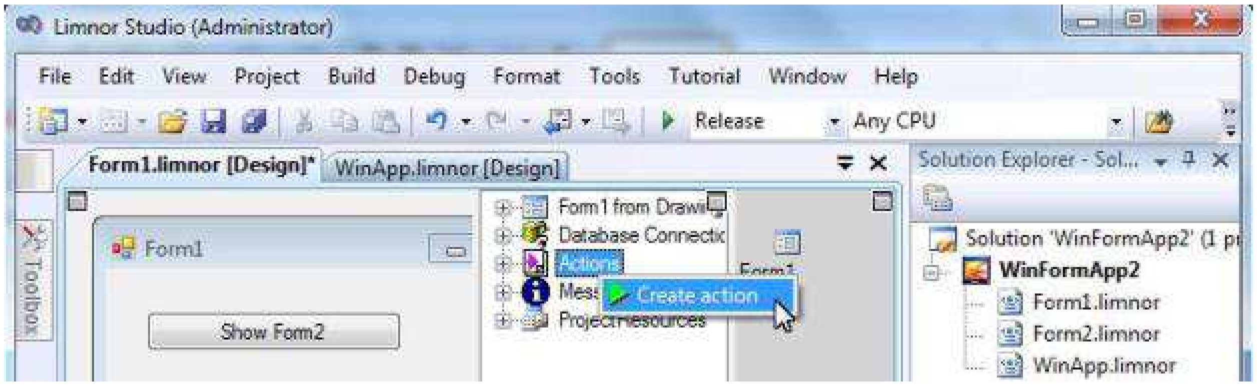
(Name)	WinFormApp1
CommandAr	
OneInstance(	False
StartForm	Form1

Use Multiple Forms

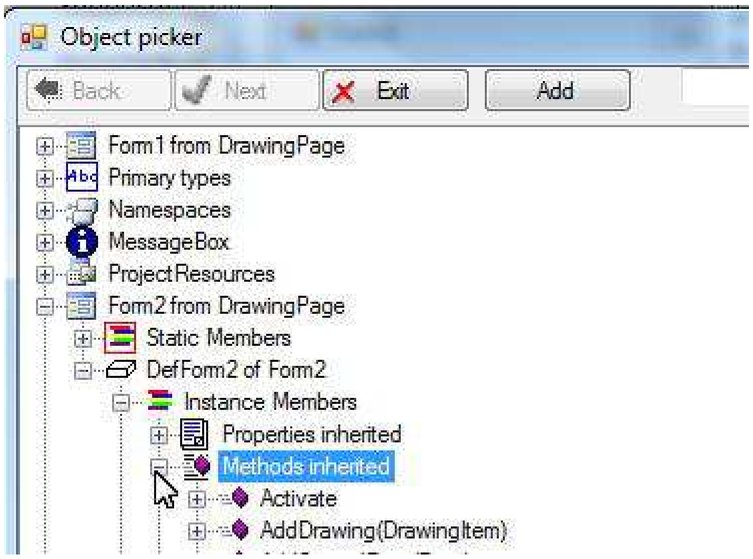
Suppose from Form1, we want to show Form2:

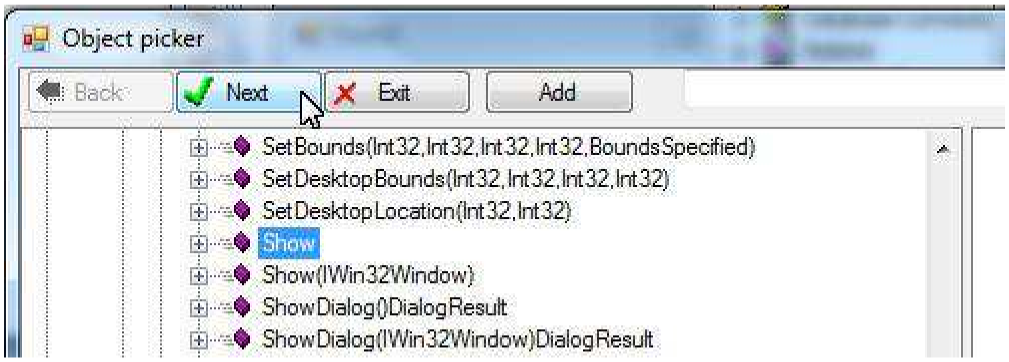


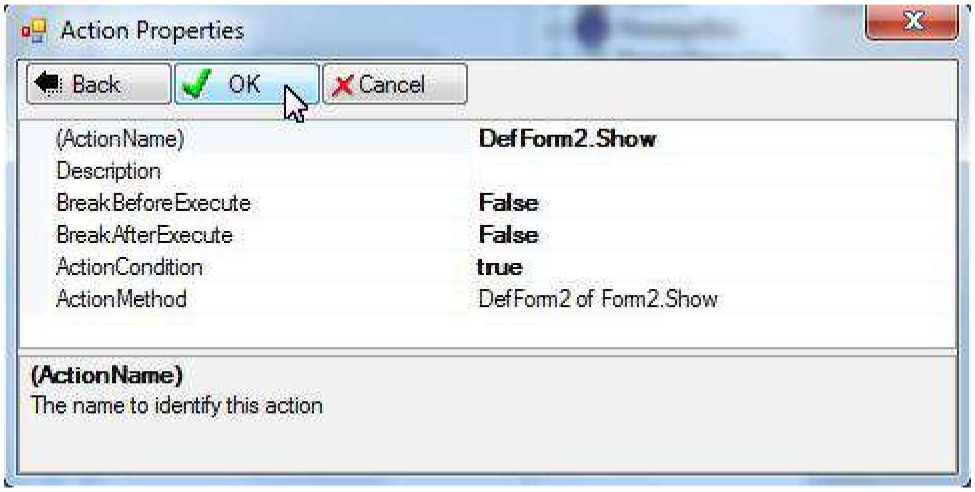
We need to create an action to show Form2. Since the action is to be executed from Form1 when a button on Form1 is clicked, the action has to be created from Form1. Right-click Actions of Form1; choose "Create action":



We will use Form2's Show method to create the action:



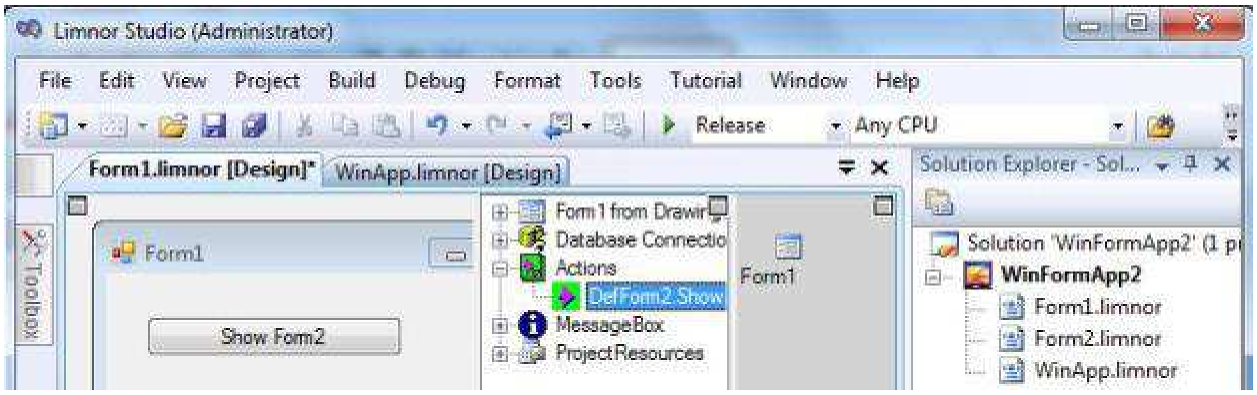

Object picker
Back
Next
Exit
Add
SetBounds(Int32,Int32,Int32,Int32,BoundsSpecified)
SetDesktopBounds(Int32,Int32,Int32,Int32)
SetDesktopLocation(Int32,Int32)
Show
Show(IWin32Window)
ShowDialog()DialogResult
ShowDialog(IWin32Window)DialogResult



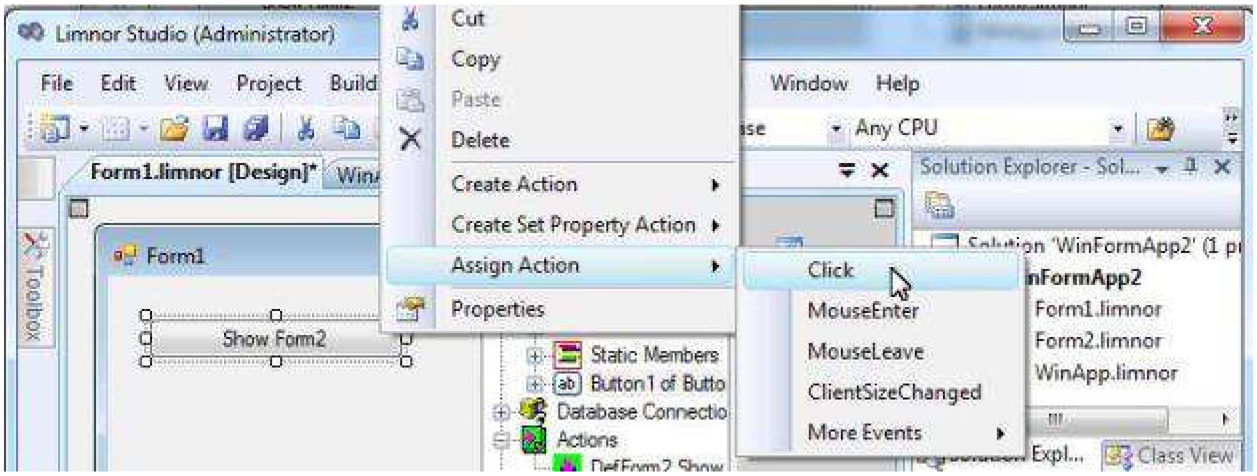
(ActionName)	**DefForm2.Show**
Description	
BreakBeforeExecute	**False**
BreakAfterExecute	**False**
ActionCondition	**true**
ActionMethod	DefForm2 of Form2.Show

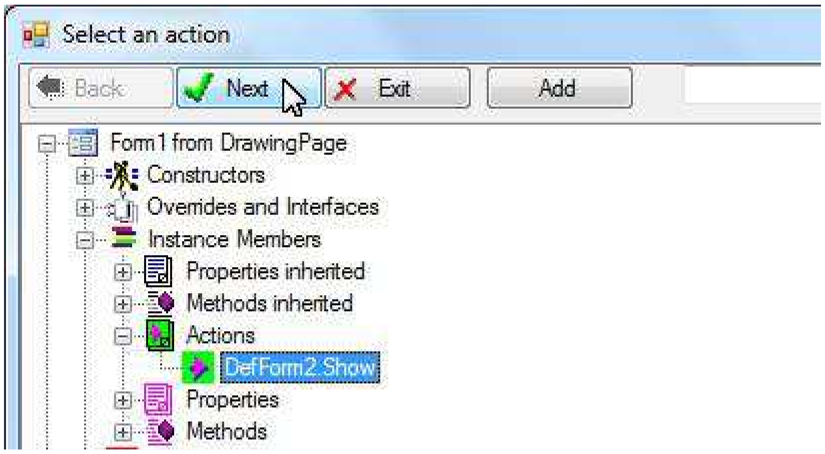
(ActionName)
The name to identify this action

The action appears under Actions:

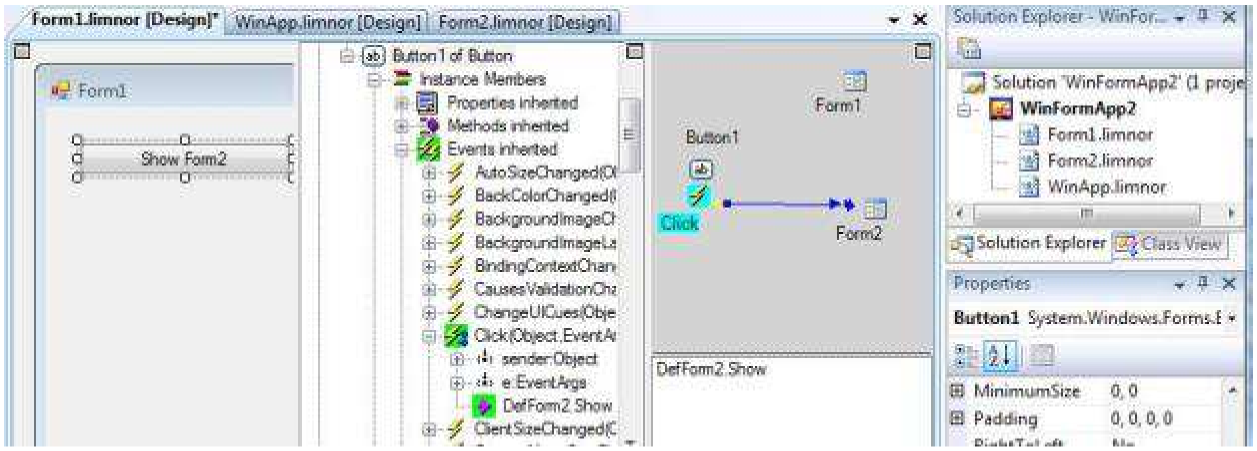


We may assign this action to the button:

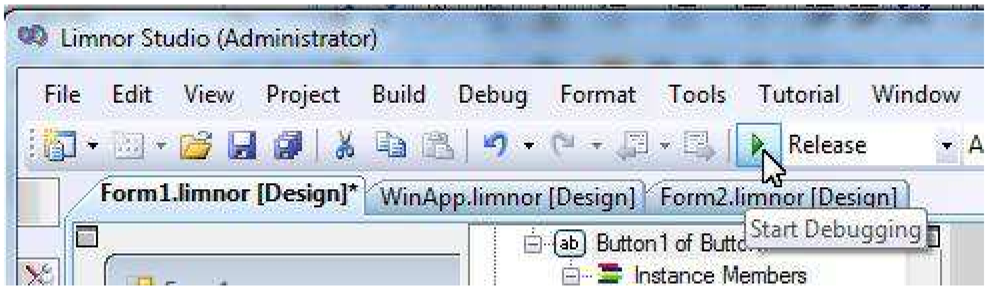




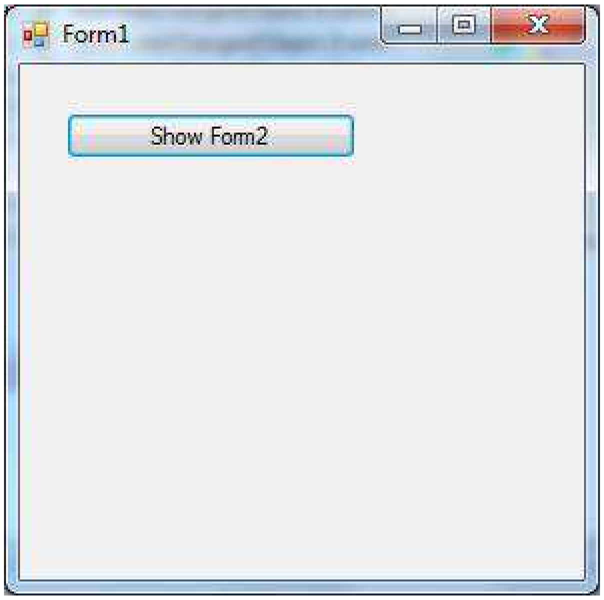
The action is assigned to the Click event of the button:



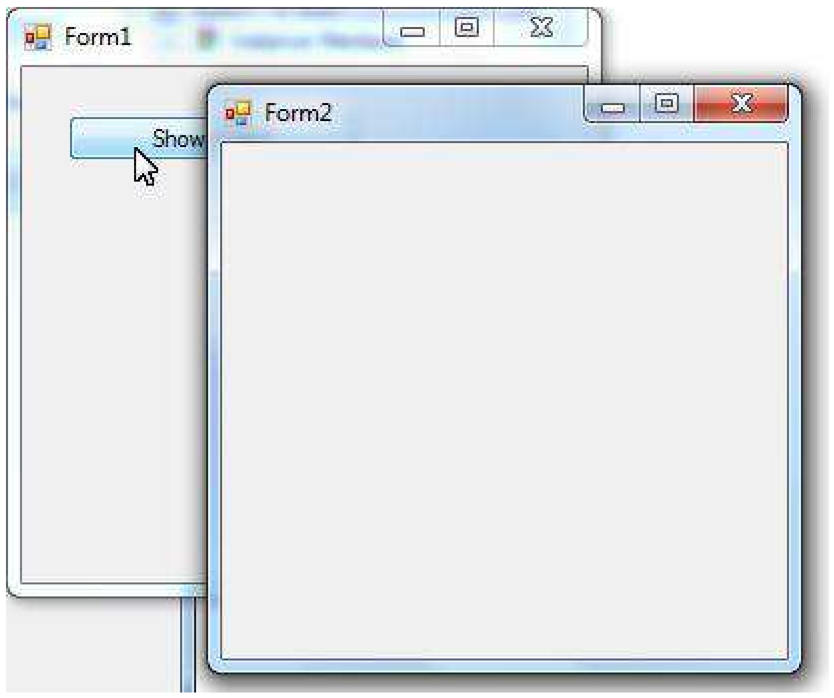
We may test the application now:



The “Starting Form” appears, which is the default instance of Form1:



Click the button, Form2 appears:



Different Ways of Showing Forms

A form can be displayed as a dialogue box or not as a dialogue box. A dialogue box blocks user access to other forms from the same application until the user closes the dialogue box.

When showing a form or closing a form, some transition effects can be used.

Show form not as a dialogue

The following methods show form not as a dialogue box.

- **Show**() – It shows the form not as a dialogue box so that the user may access other forms and bring other forms to front.
- **Show**(**owner**) – It works in a similar way as Show() but the form will stay on top of the owner. The owner is another form. If the form is already visible then calling this method will crash the program. To avoid such problems use ShowOwned (owner) method instead.
- **ShowOwned** (**owner**) – It works in the same way as Show(owner) but it will not crash if the form is already visible. It is recommended to use this method in place of Show(owner).

Show form as a dialogue

The following methods show form as a dialogue box.

- **ShowDialog**()– It shows the form as a dialogue box.
- **ShowDialog**(**owner**) – It shows the form as a dialogue box. The dialogue box is owned by the owner. The owner is another form.

Show/Hide form with fade-in and fade-out

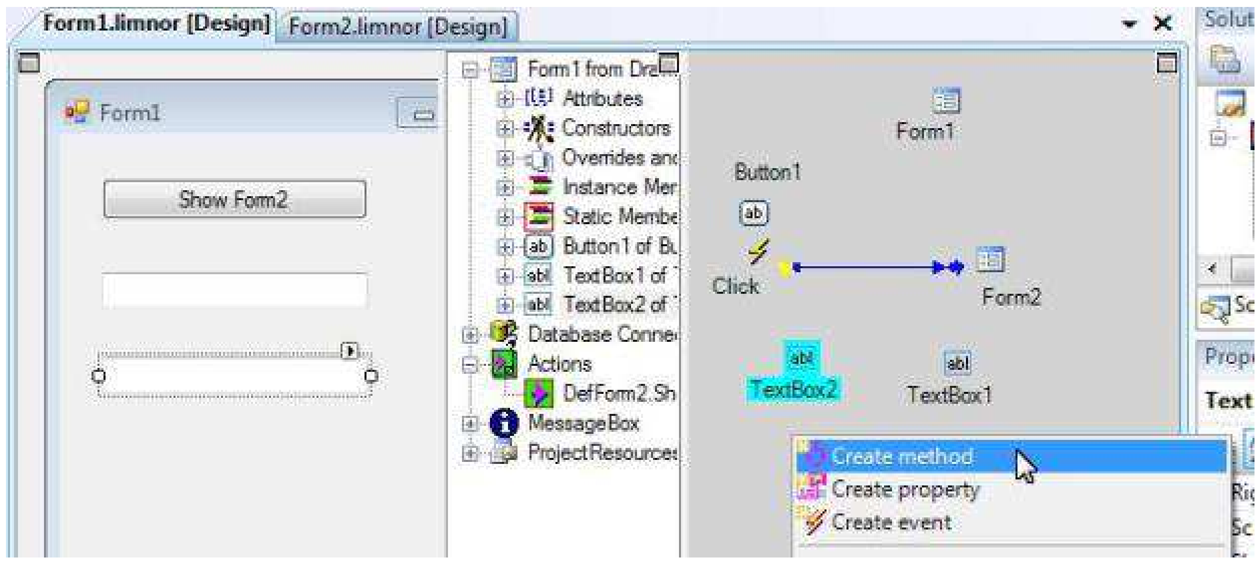
The following methods show form with fade-in and fade-out effects. Parameter showFadeTime indicates the fade-in time, in milliseconds. If you do not know what value to use then try use 500 to see if the fade time is OK with you. Parameter closeFadeTime indicates the fade-out time when the form is closing.

- **ShowWithFading(showFadeTime**, **closeFadeTime**) – It works in the same way as Show() except that it uses fade-in and fade-out.
- **ShowWithFading(owner**, **showFadeTime**, **closeFadeTime**) – It works in the same way as ShowOwned(owner) except that it uses fade-in and fade-out.
- **ShowDialogWithFading**(**showFadeTime**, **closeFadeTime**) – It works in the same way as ShowDialog() except that it uses fade-in and fade-out.
- **ShowDialogWithFading**(**owner**, **showFadeTime**, **closeFadeTime**) – It works in the same way as ShowDialog(owner) except that it uses fade-in and fade-out.
- **`HideWithFading()` –** It hides the form with a fade-out effect. The fading time is the parameter closeFadeTime in above methods. Hide() does not use fade-out. Close() method uses fade-out. Hiding a form makes the form invisible but still alive in memory and keeps all its data. Closing a form removes it from the memory and loses all its data.

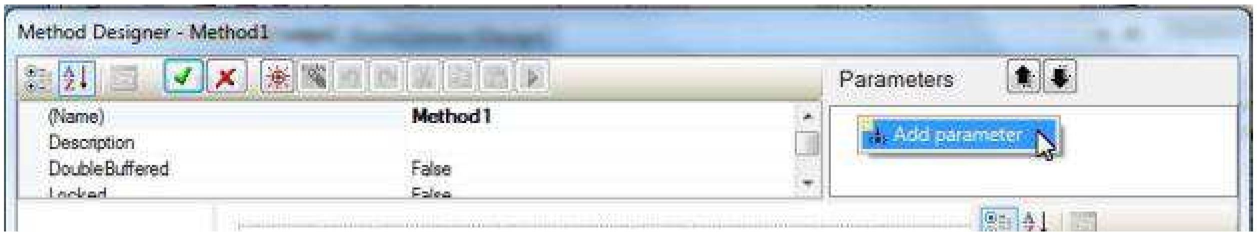
Accessing between forms – sample 1

We show that Form2 is loaded from Form1 when the user clicks a button on Form1. This is an example of accessing between forms. Here we give another example of accessing between forms. This time we create a method in Form1 and call this method from Form2.

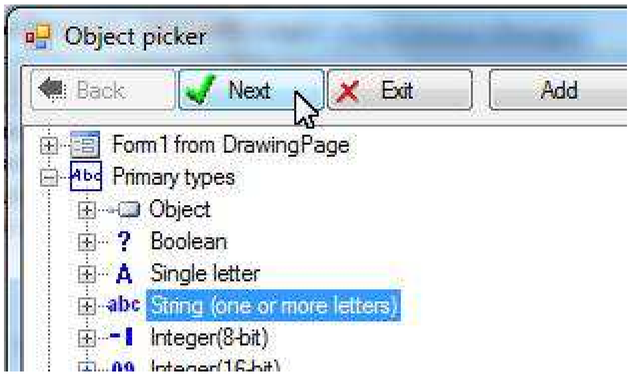
We add two text boxes on Form1. We create a method which has two parameters. The method displays the two parameters on the two text boxes.



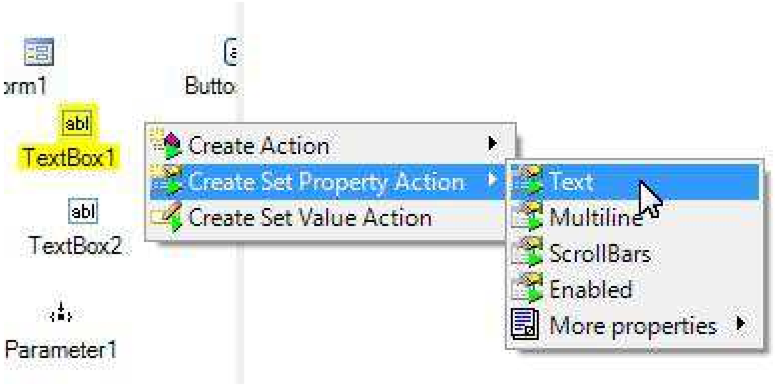
Add two parameters to the new method:



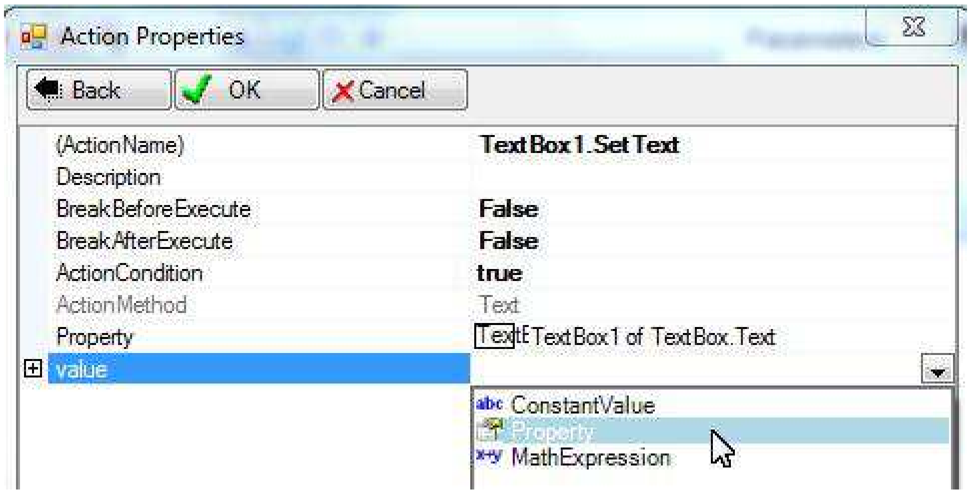
Use “string” as the data type for the parameter:



Create an action to display the first parameter on the first text box by right-clicking the text box icon and choosing “Create Set Property Action” and choosing “Text” property:

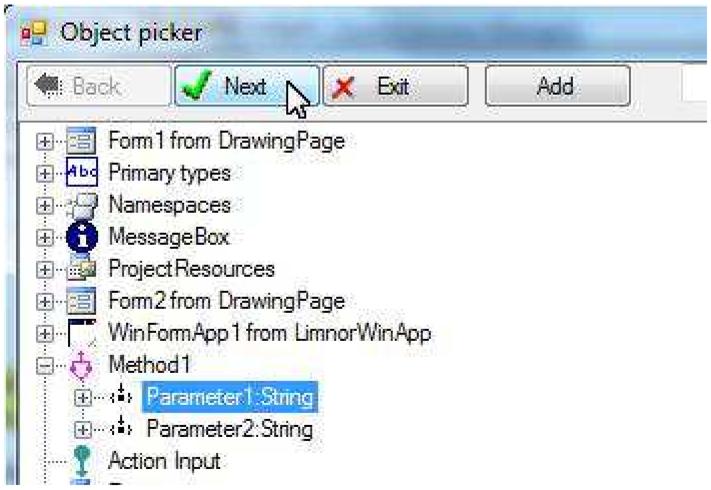


Select “Property” for the value of the action:

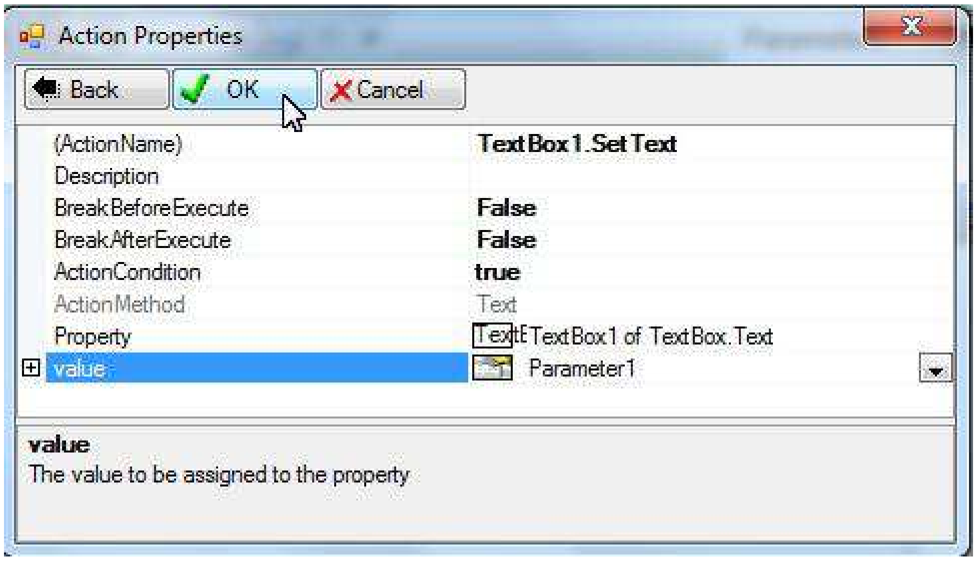


(ActionName)	TextBox1.SetText
Description	
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	Text
Property	TextBox1 of TextBox.Text
value	

Select the first parameter:



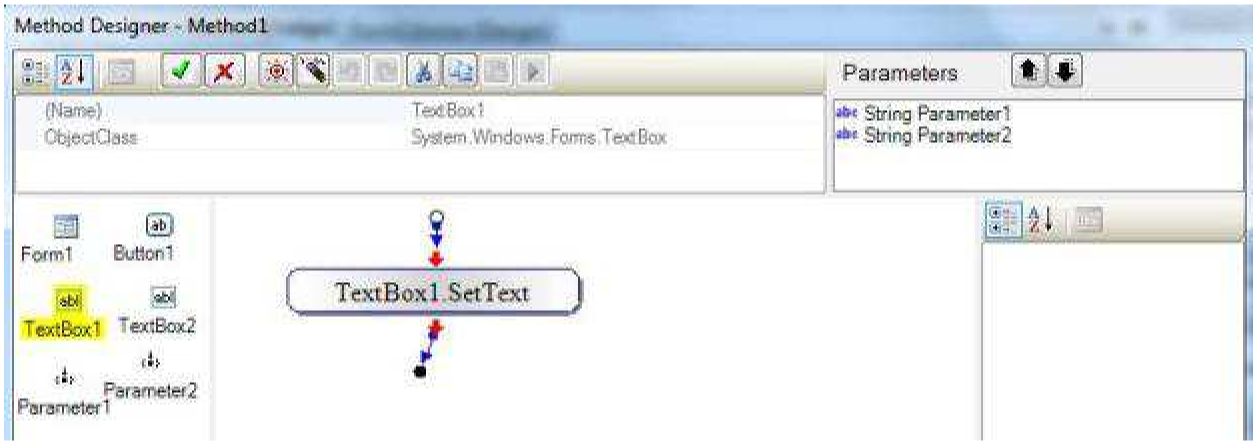
This action assigns the first parameter of the method to the Text property of the first text box:



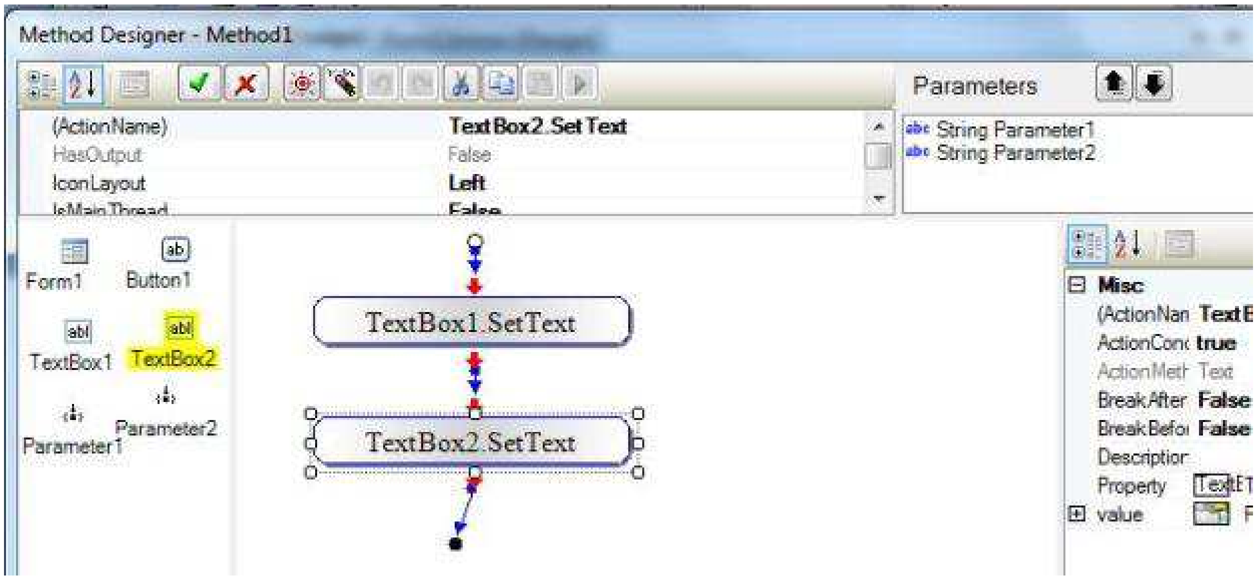
(ActionName)	TextBox1.SetText
Description	
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	Text
Property	TextBox1 of TextBox.Text
value	Parameter1

value
The value to be assigned to the property

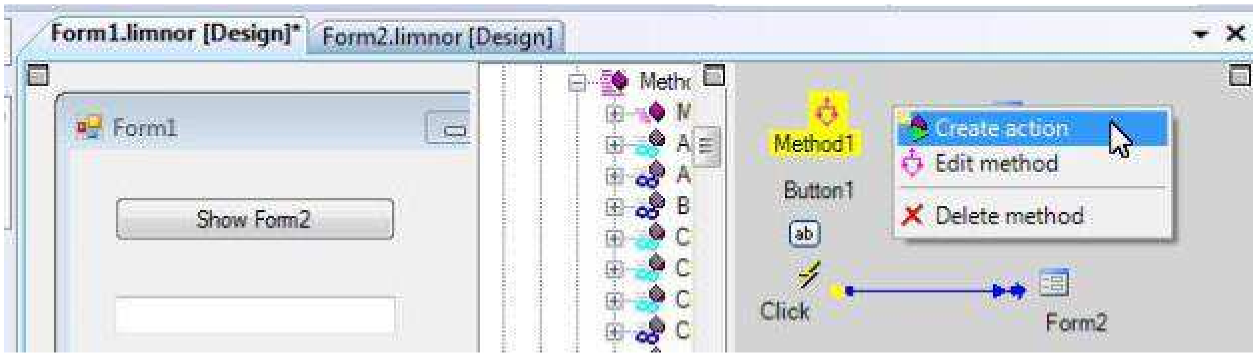
The action appears in the Action Pane of the method:



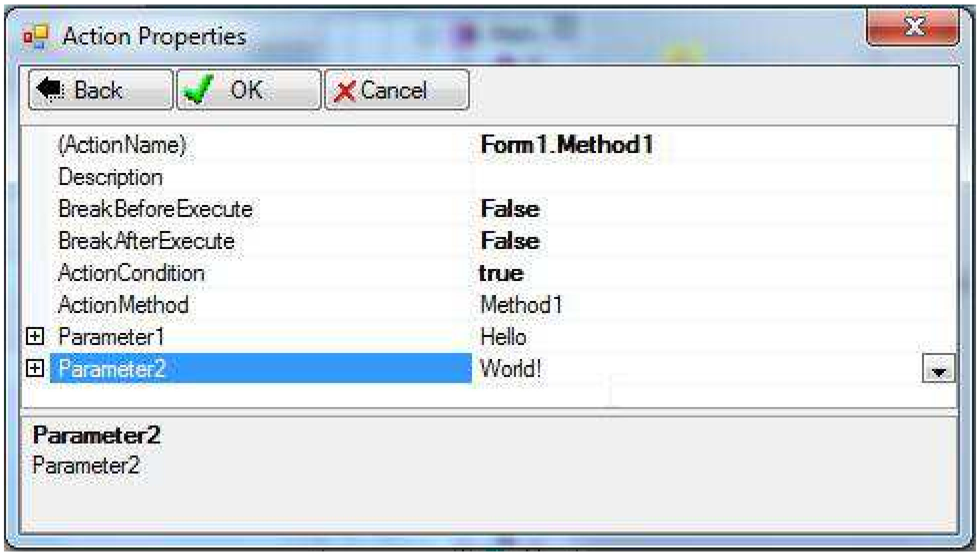
Create another action to assign Parameter2 to the Text property of TextBox2 and link the action to the first action:



That is all for this method. We may create an action of this method to show the effect of executing this method. Right-click the method, choose “Create action”:



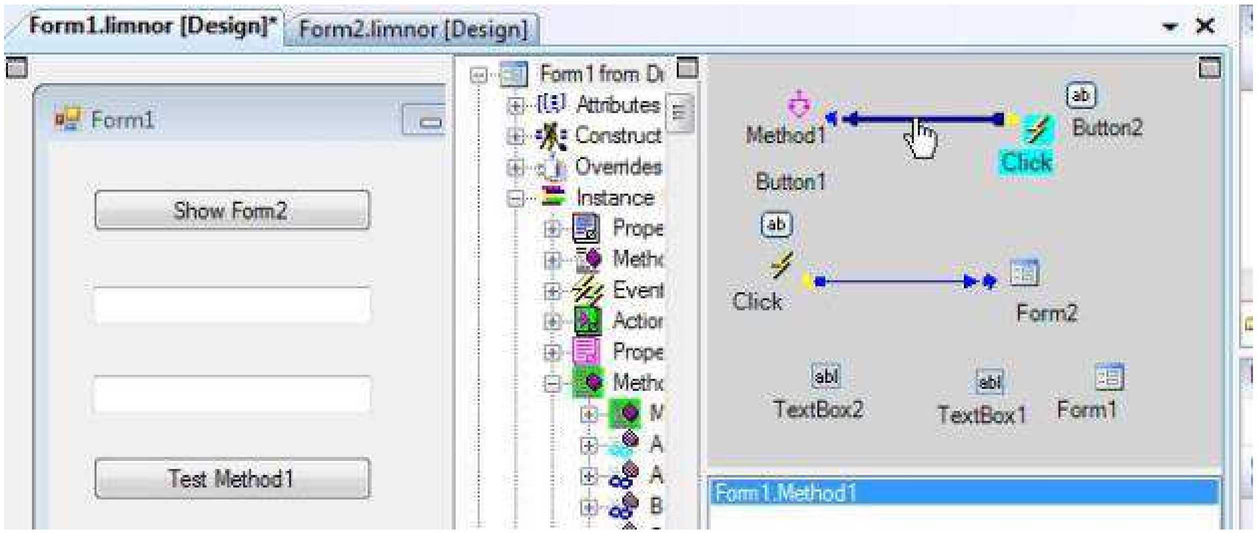
Give some testing data for the method parameters:



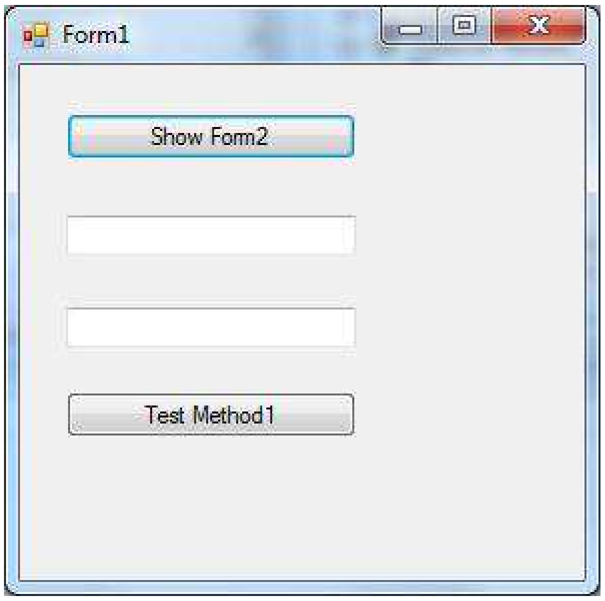
(ActionName)	**Form1.Method1**
Description	
BreakBeforeExecute	**False**
BreakAfterExecute	**False**
ActionCondition	**true**
ActionMethod	Method1
Parameter1	Hello
Parameter2	World!

Parameter2
Parameter2

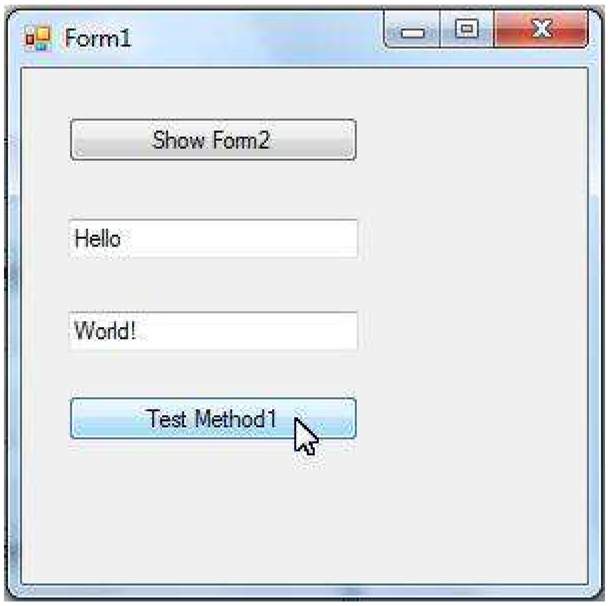
Assign the action to a button:



Let's test the application:

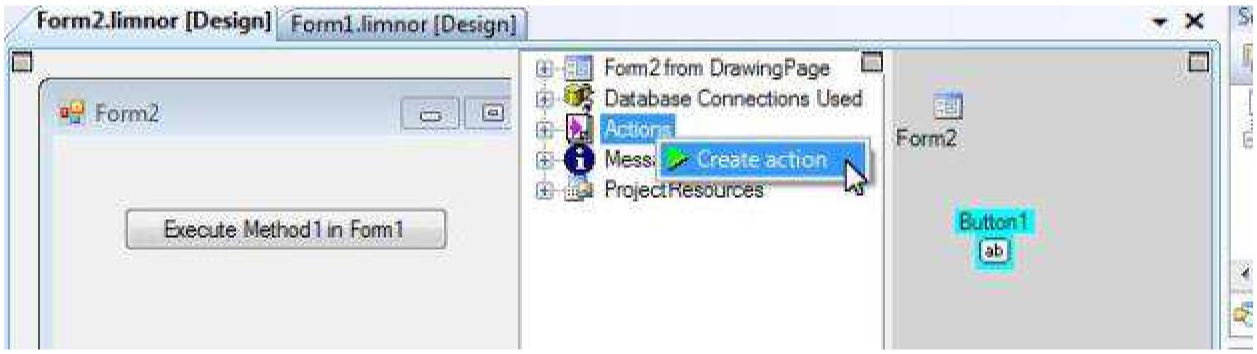


Click button “Test Method1”, we can see the result of executing Method1 with parameters “Hello” and “World!”:

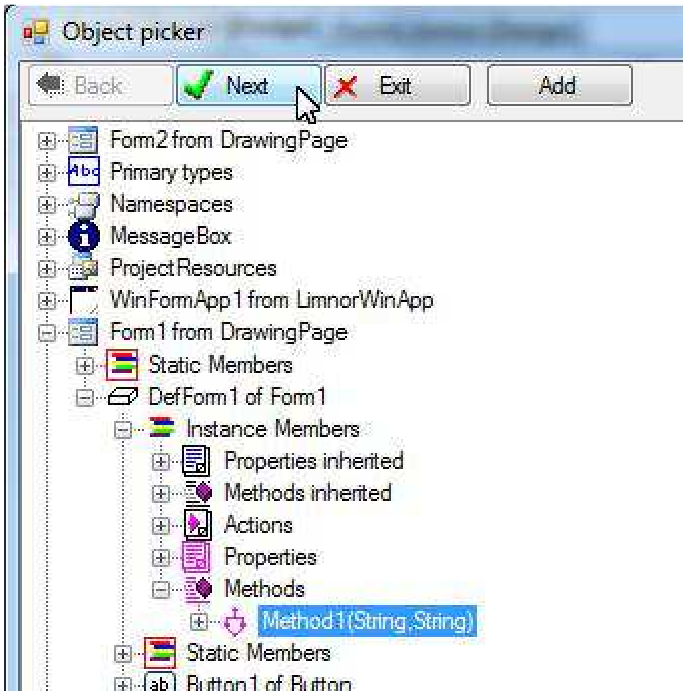


Now we will program it to execute Method1 from Form2. We need to create an action to do it. Since the action will be executed from Form2, the action has to be created from Form2.

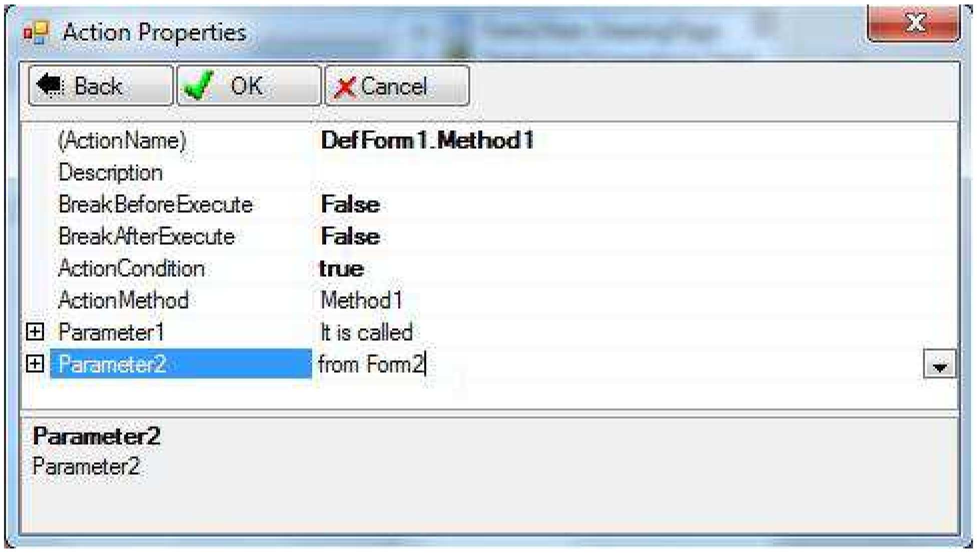
Right-click Actions in Form2, choose “Create action”:



Select Method1 under the default instance of Form1:



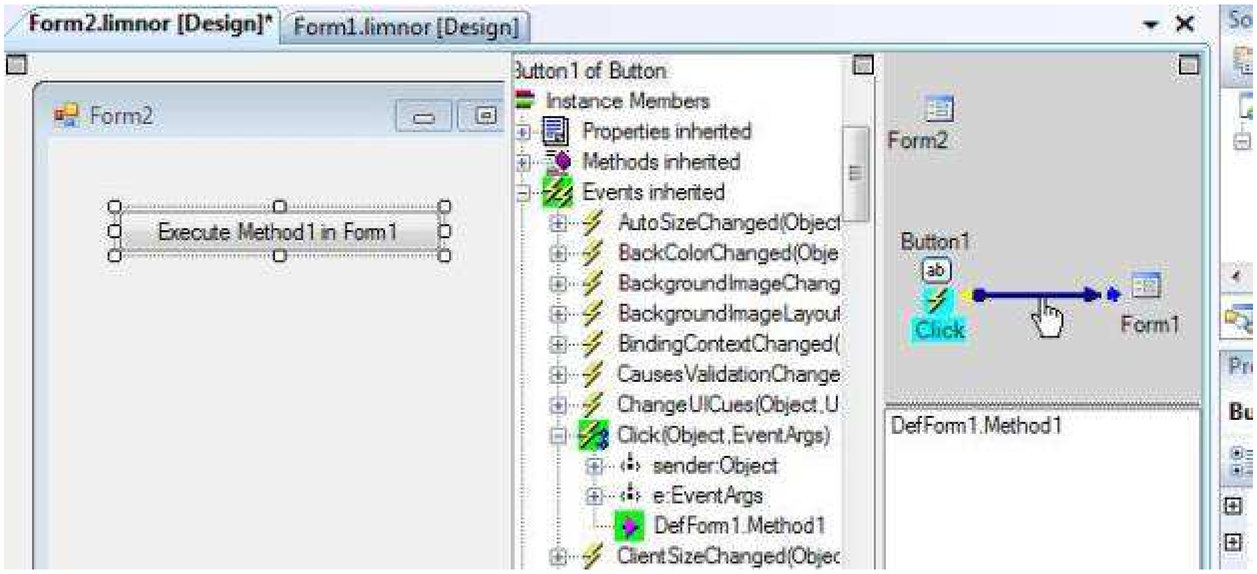
Give some data to the parameters for testing purpose:



(ActionName)	**DefForm1.Method1**
Description	
BreakBeforeExecute	**False**
BreakAfterExecute	**False**
ActionCondition	**true**
ActionMethod	Method1
Parameter1	It is called
Parameter2	from Form2

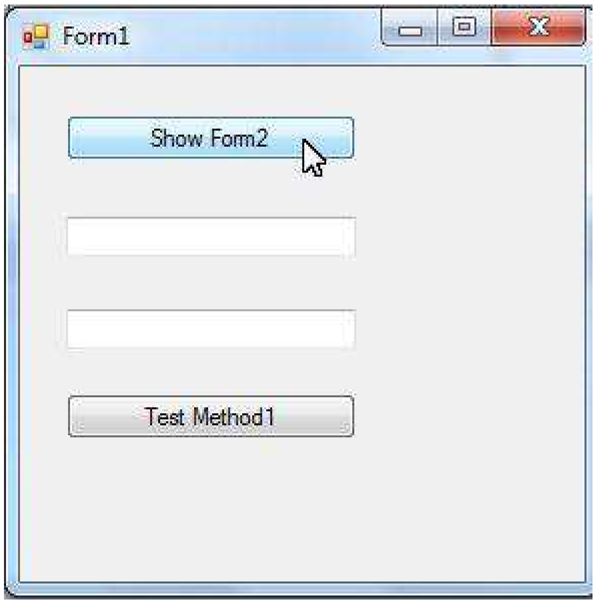
Parameter2
Parameter2

Assign the action a button on Form2:

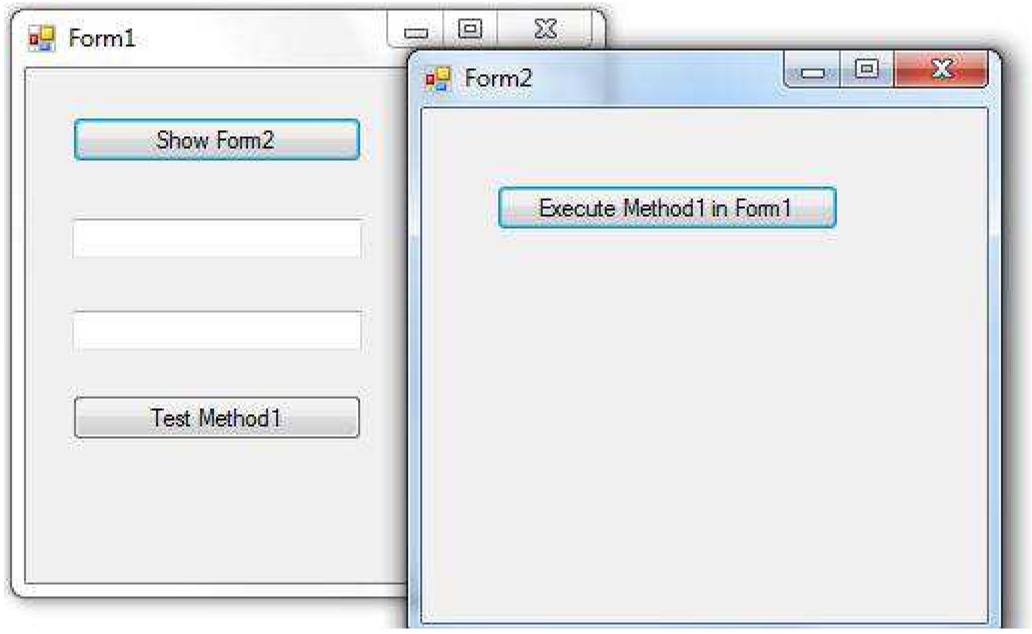


We may test the application now.

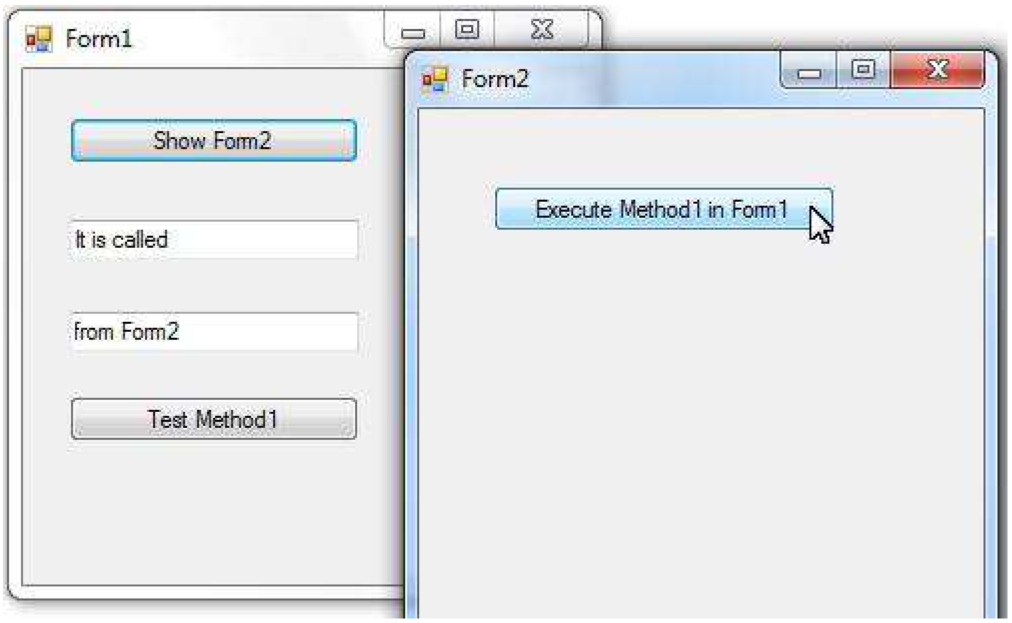
Click button “Show Form2” to display Form2:



Form2 appears:



Click button “Execute Method1 in Form1” on Form2, we can see the result on Form1:

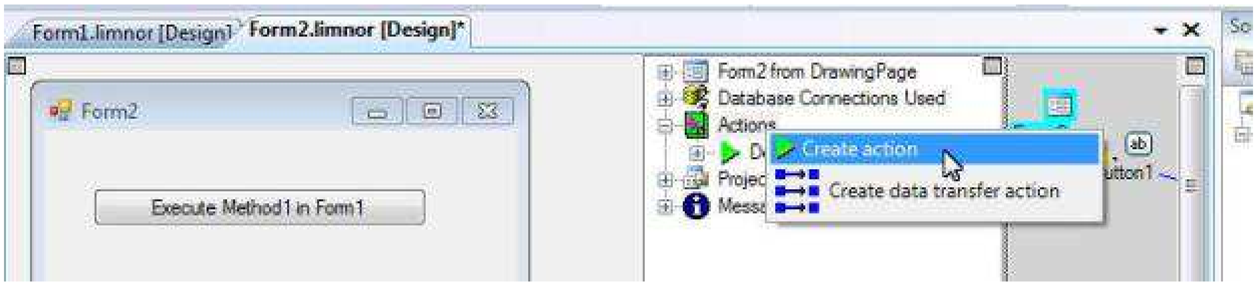


Accessing between forms – Sample 2

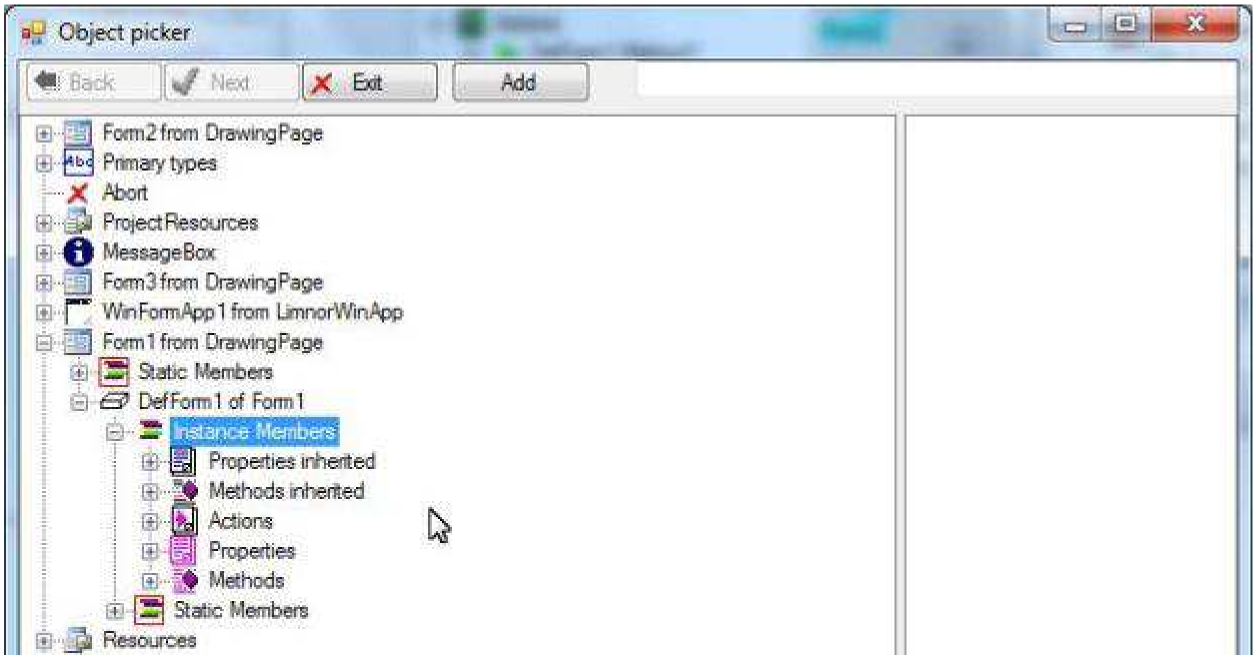
Modular Pattern

In the above sample, Method1 in Form1 allows other objects to modify the two text boxes. The sample shows that from Form2, the text boxes in Form1 are modified. You might ask that why we need to create Method1, why we cannot simply modify directly the text boxes in Form1.

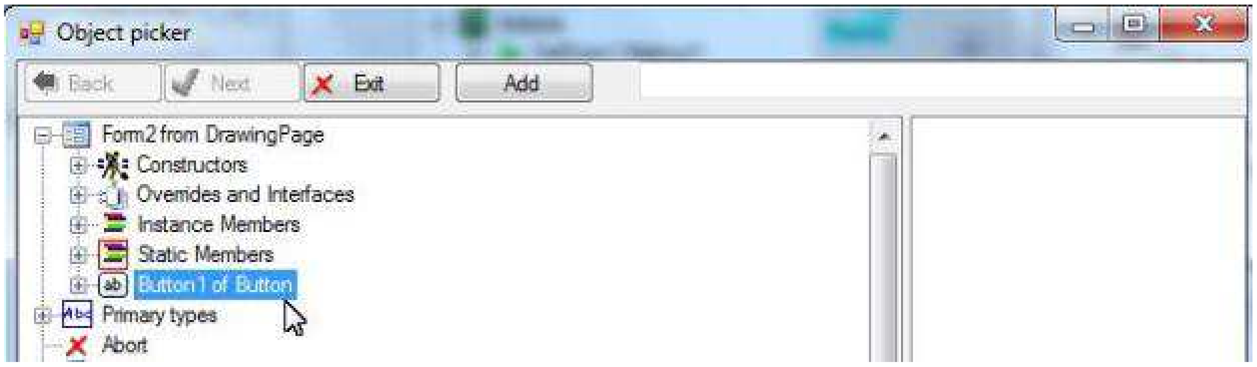
Actually direct modifications of components between forms are not allowed. For example, if we try to create an action from Form2:



We cannot see the components inside Form1:



We may only see the components of Form2:



Hiding components from other forms is for ensuring modular design pattern. Under such a pattern, each form may be designed independently. Method1 created in Form1 is an interface to be used by other forms.

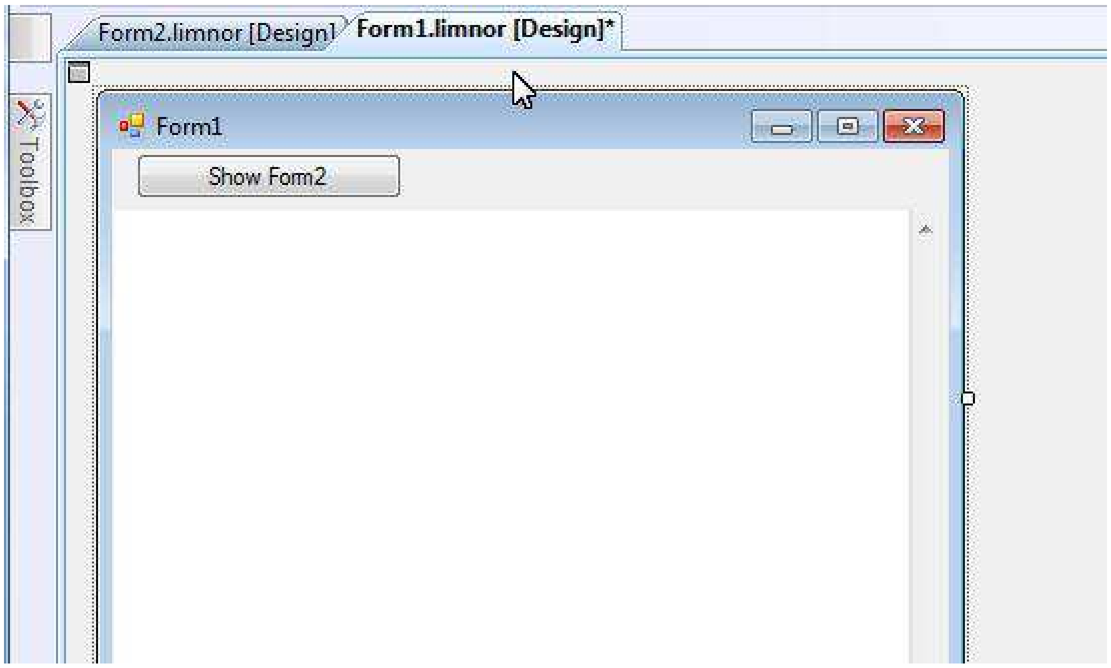
Method as interface

Here we provide another sample of creating a method as an interface for modifying component from outside.

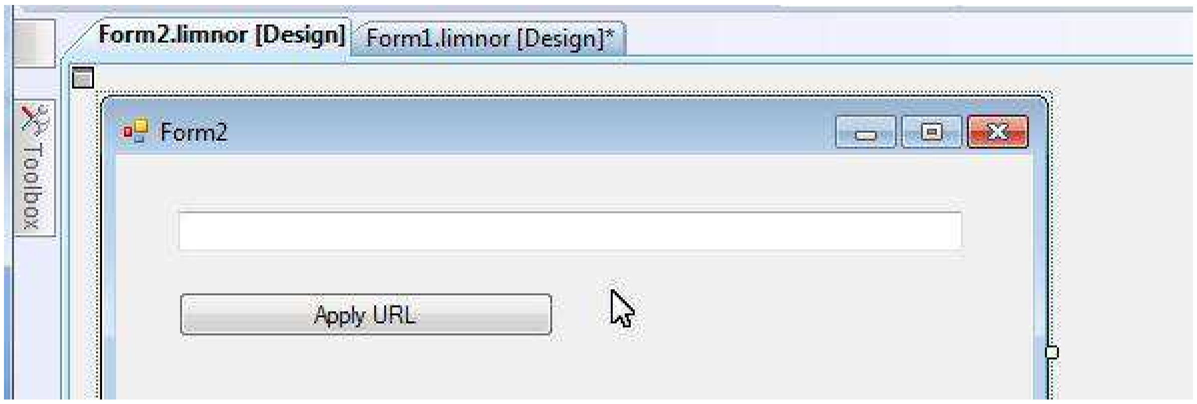
Suppose we have a form named Form1 containing a web browser control. Suppose we have a form named Form2. We want to change the URL of the web browser control on Form1. The web browser has a Navigate method for navigating to a web page. But from Form2, we cannot see the web browser control on Form1. We create a GotoURL method in Form1 to execute Navigate of the web browser control. We then execute GotoURL from Form2.

What are benefits of Modular design? Think about such a scenario: suppose you get a better web browser control and you switch to it in Form1. You need to modify GotoURL method in Form1. But you do not need to modify Form2.

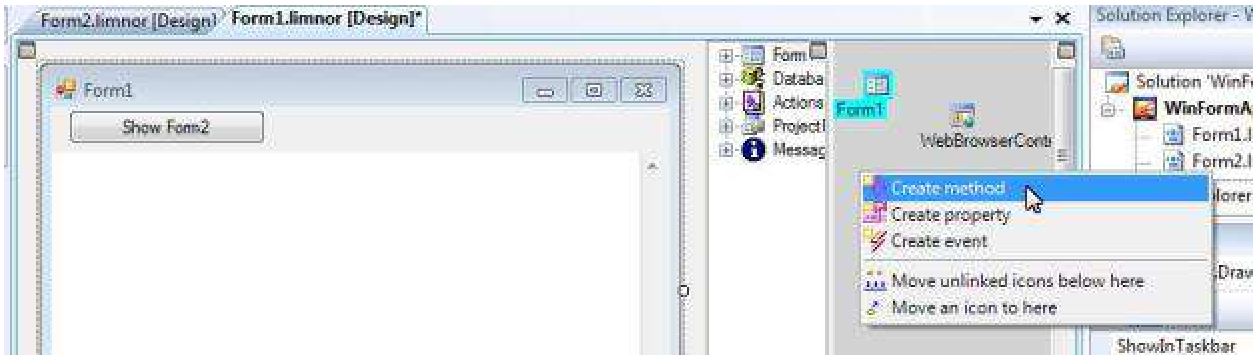
Form1:



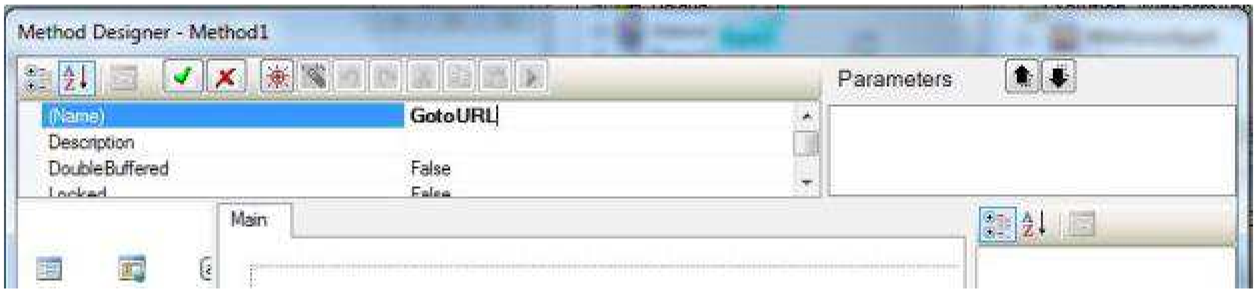
Form2:



Create GotoURL method on Form1

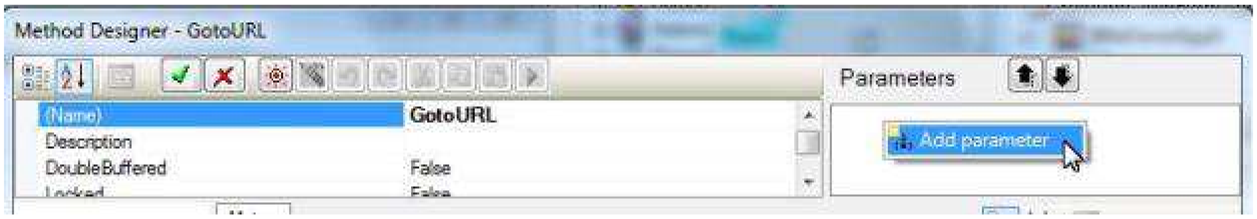


Rename it to GotoURL:

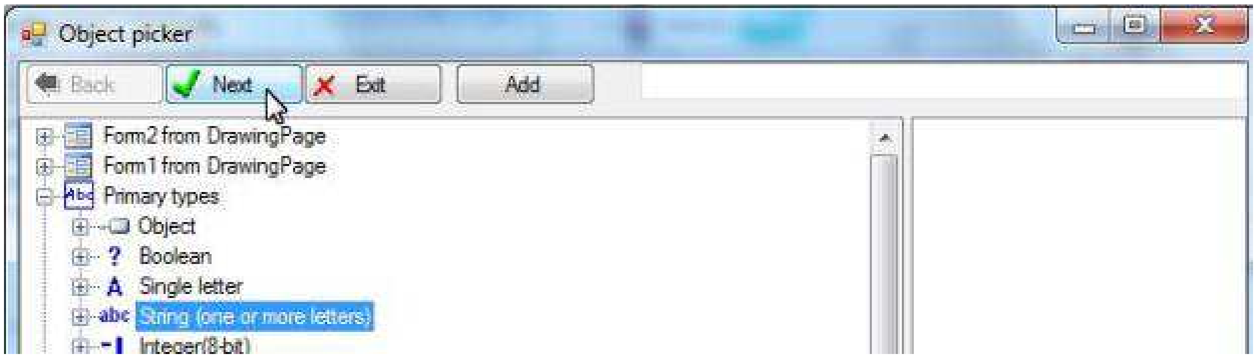


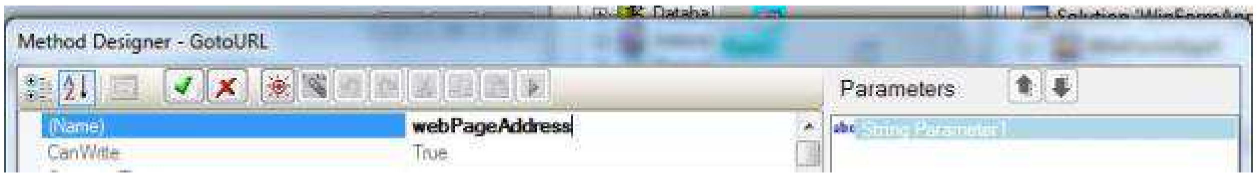
(Name)	GotoURL
Description	
DoubleBuffered	False
Locked	False

Add a parameter and rename the parameter to webPageAddress:



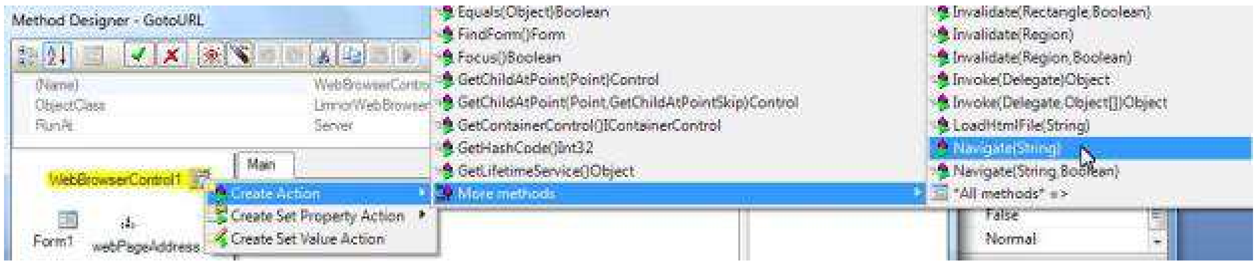
(Name)	GotoURL
Description	
DoubleBuffered	False
Locked	False





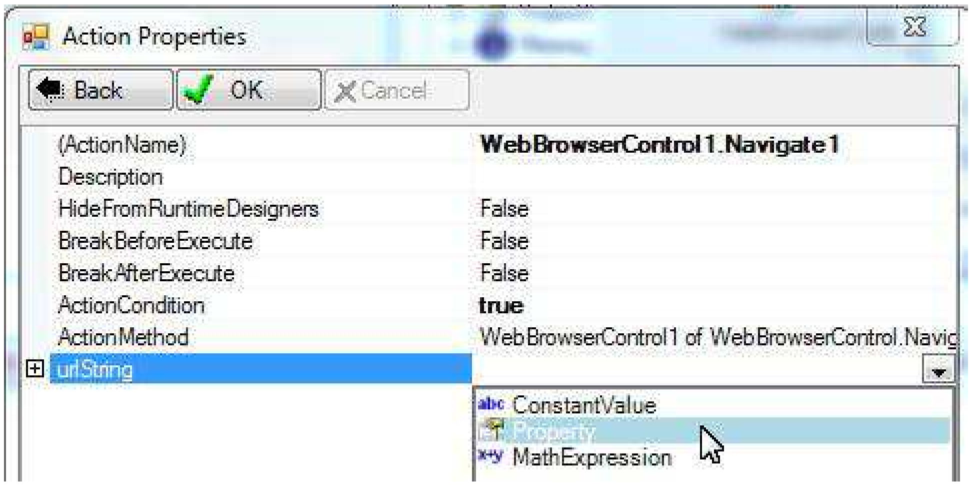
(Name)	webPageAddress
CanWrite	True

Add a Navigate action:

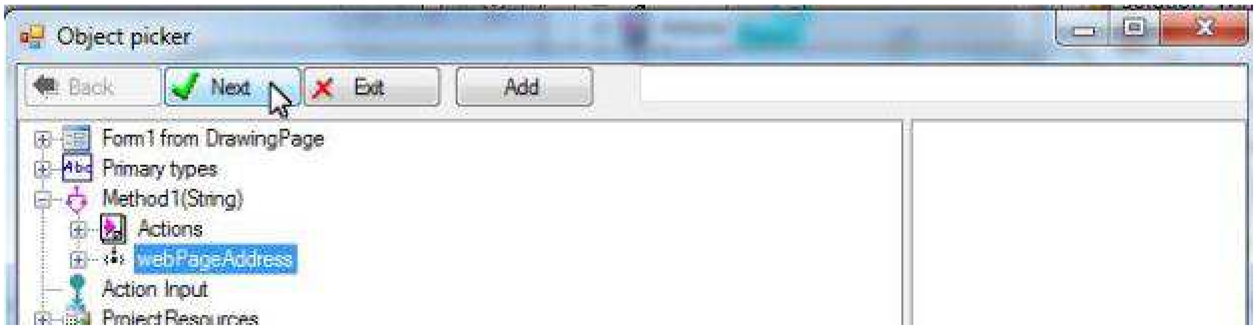


(Name)	WebBrowserContro
ObjectClass	LimnorWebBrowser
RunAt	Server

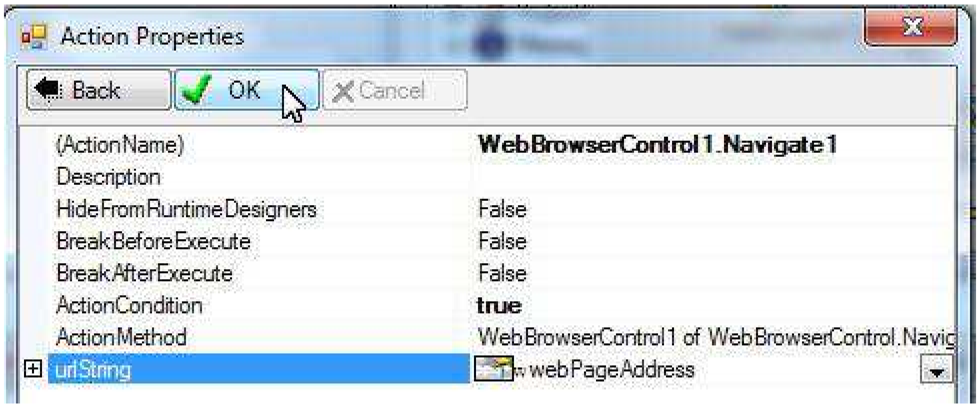
Pass webPageAddress to the action:



(ActionName)	**WebBrowserControl1.Navigate1**
Description	
HideFromRuntimeDesigners	False
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	**true**
ActionMethod	WebBrowserControl1 of WebBrowserControl.Navig
urlString	

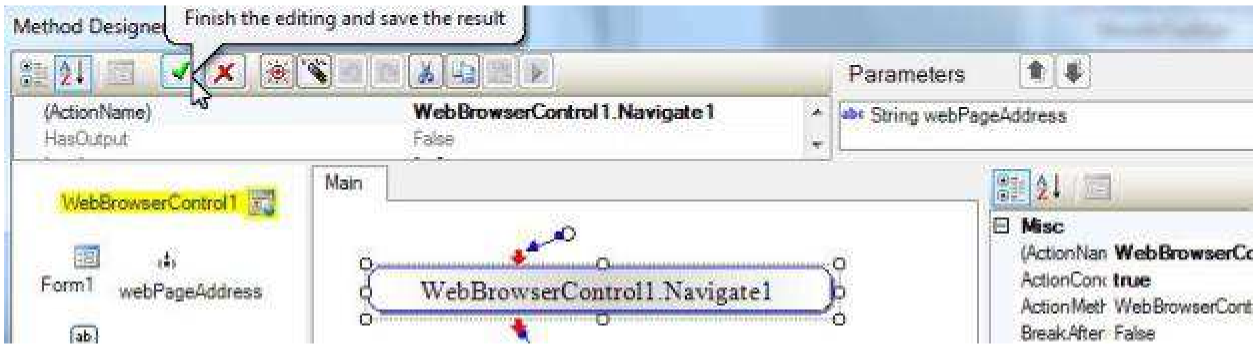


Click OK:


Action Properties

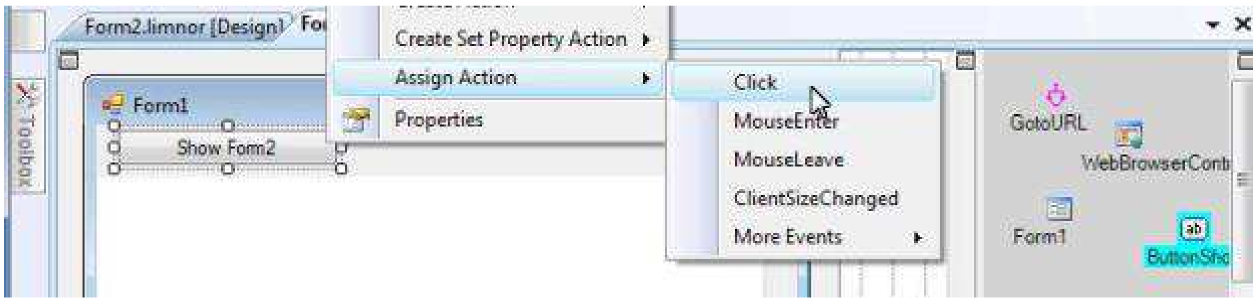
(ActionName)	WebBrowserControl1.Navigate1
Description	
HideFromRuntimeDesigners	False
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	WebBrowserControl1 of WebBrowserControl.Navig
urlString	webPageAddress

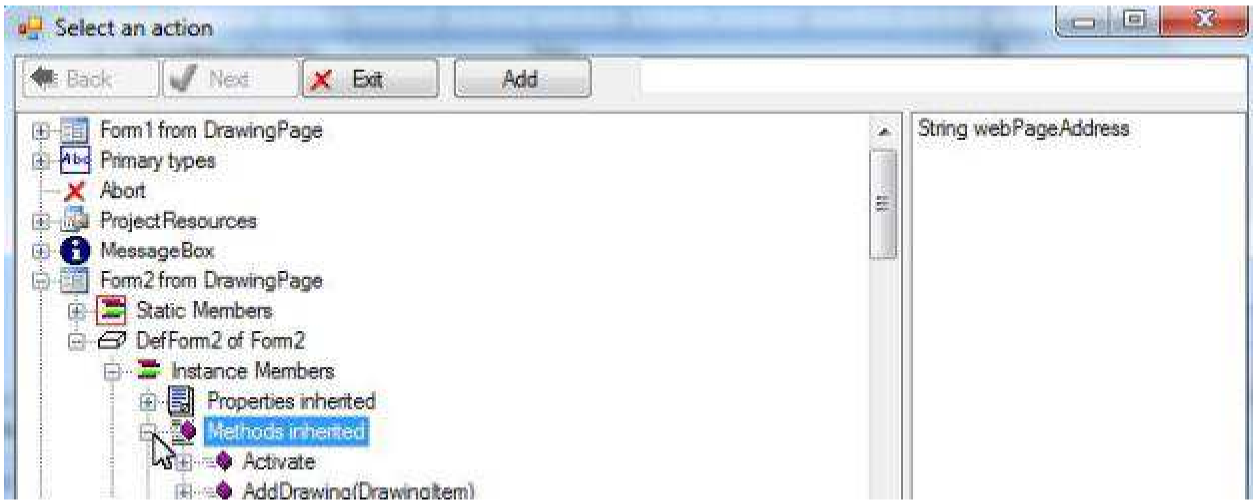
Finish editing this method:

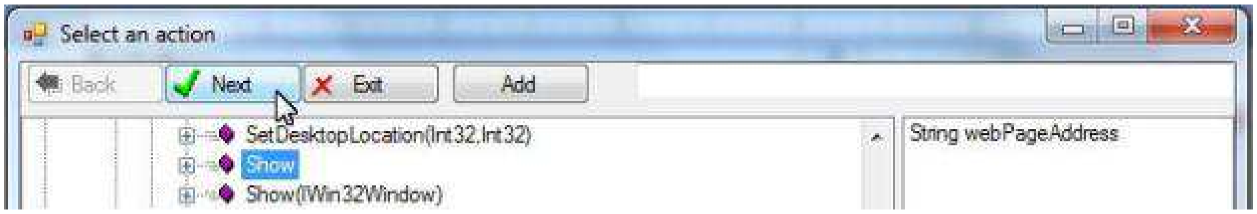


(ActionName)	WebBrowserControl1.Navigate1
HasOutput	False

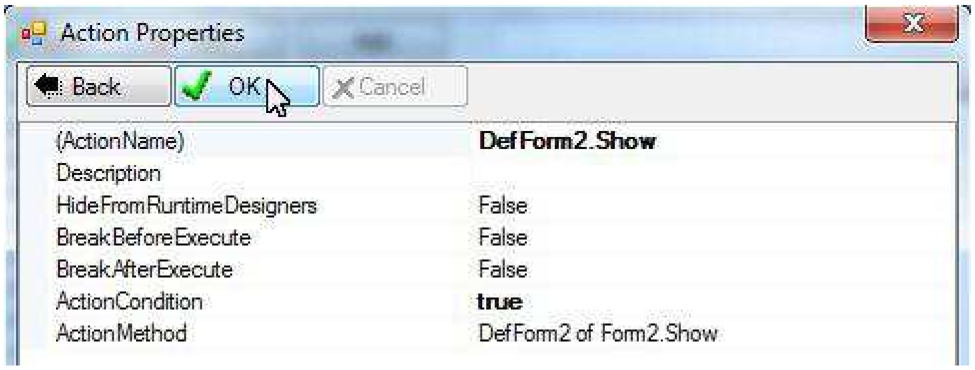
Show Form2 from Form1





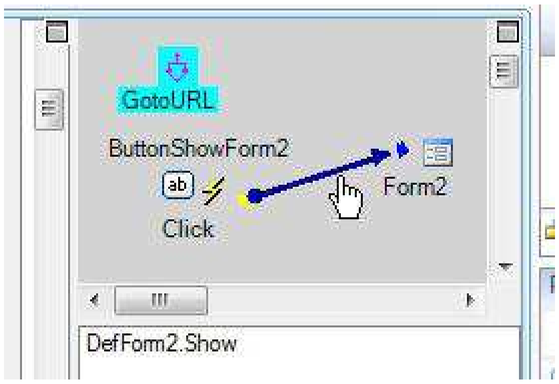


Click OK:

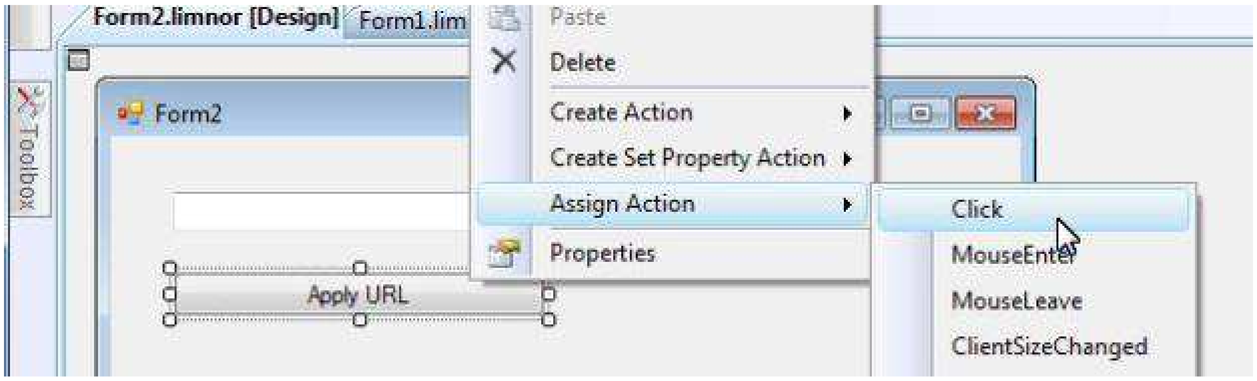


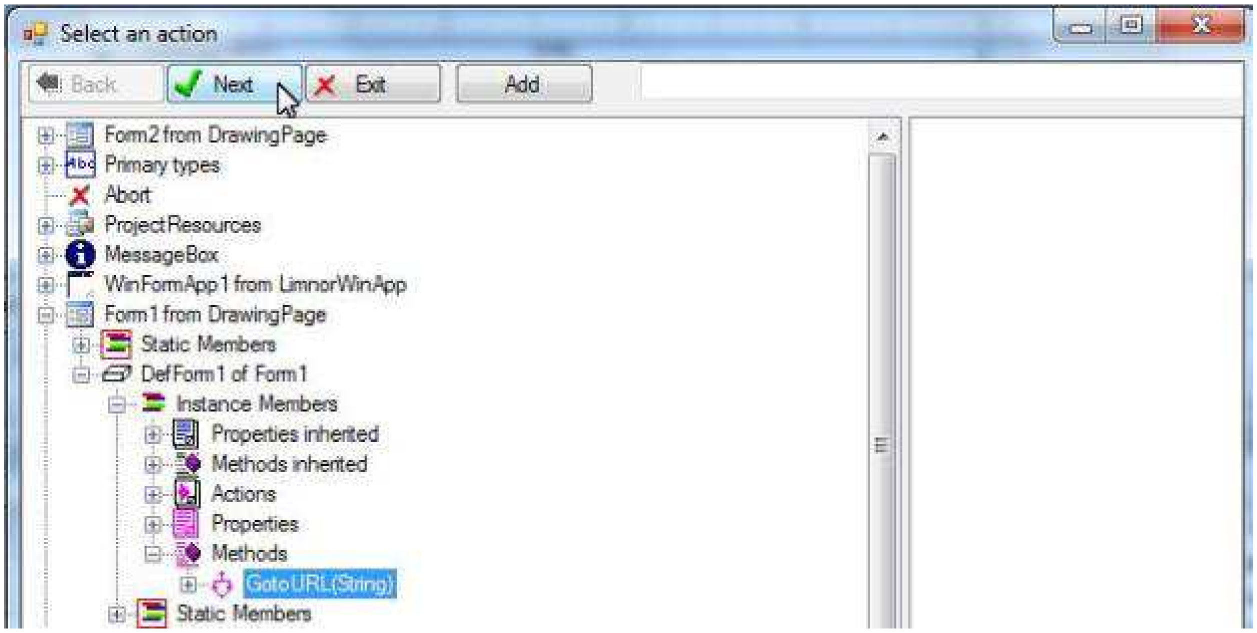
(ActionName)	**DefForm2.Show**
Description	
HideFromRuntimeDesigners	False
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	**true**
ActionMethod	DefForm2 of Form2.Show

The action is created and linked to the button:

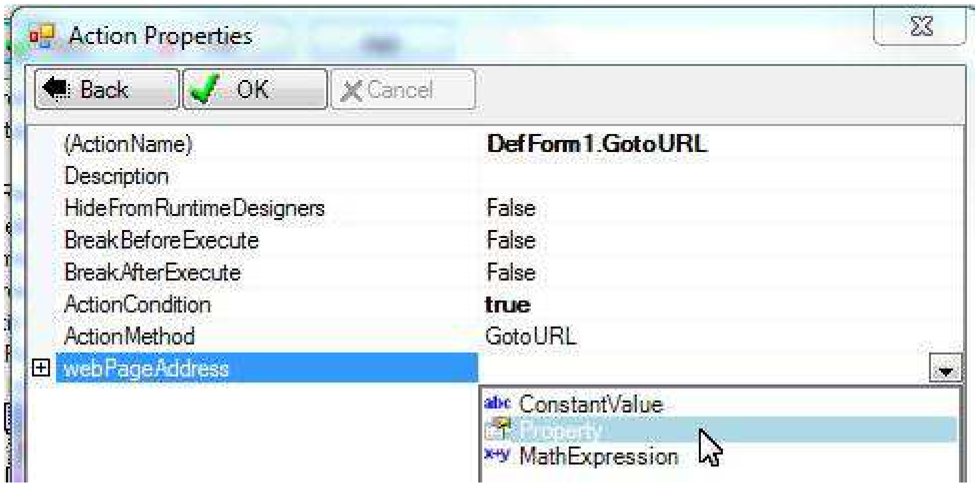


Execute GotoURL from Form2:

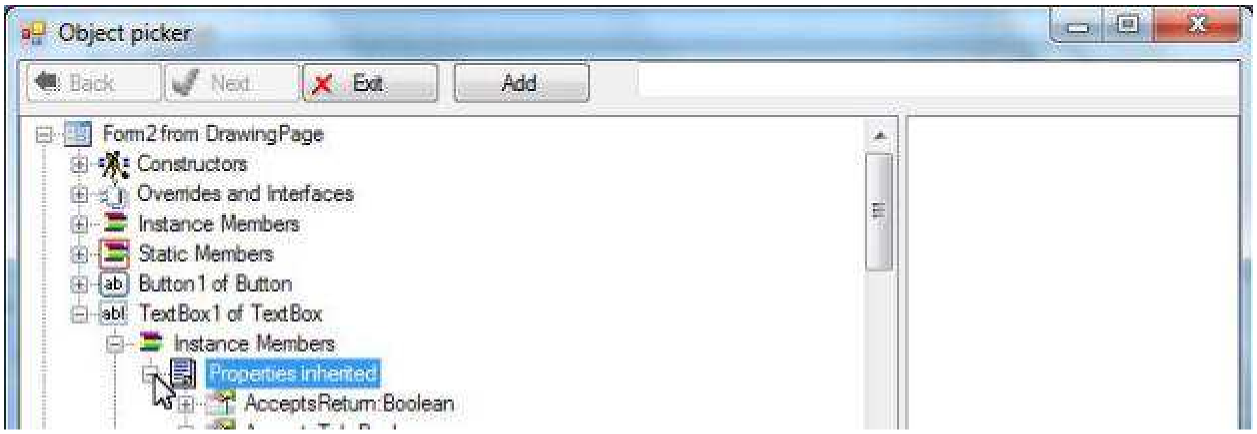


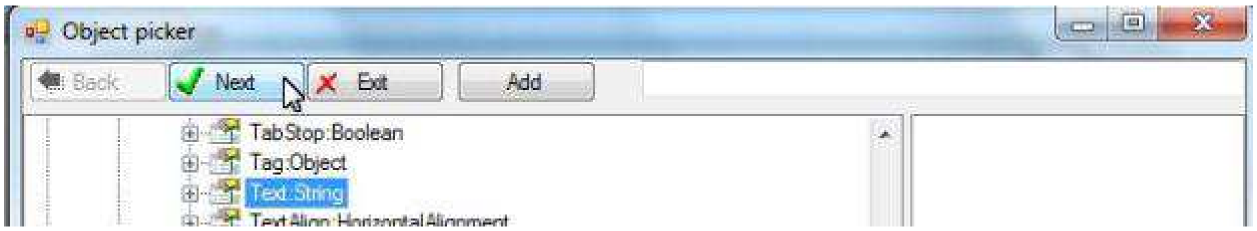


Pass the Text property of the text box to the action:

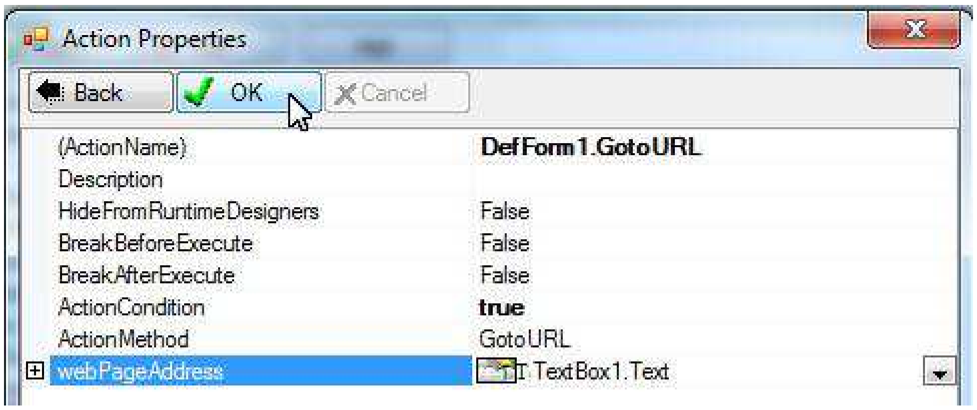


(ActionName)	DefForm1.GotoURL
Description	
HideFromRuntimeDesigners	False
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	GotoURL
webPageAddress	



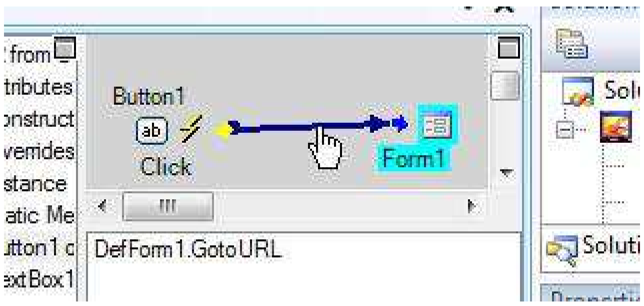


Click OK:



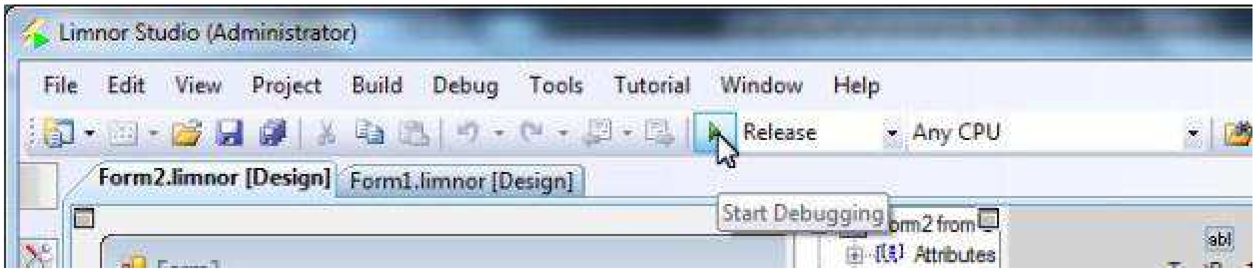
(ActionName)	DefForm1.GotoURL
Description	
HideFromRuntimeDesigners	False
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	GotoURL
webPageAddress	TextBox1.Text

The action is created and linked to the button:

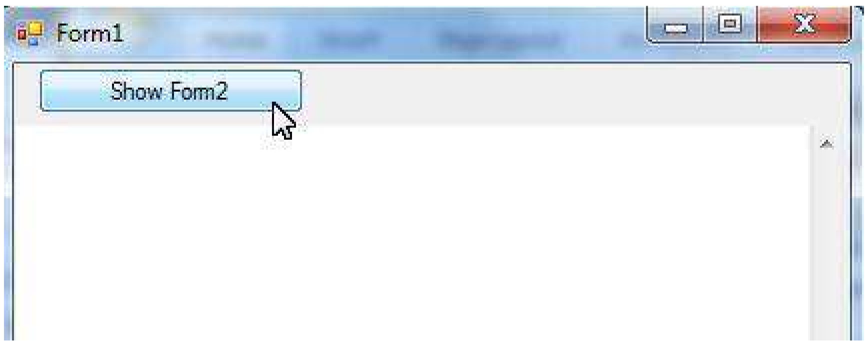


Test

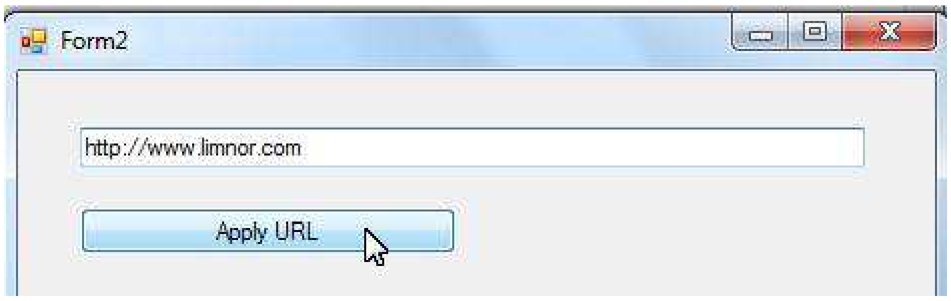
Compile and test the program:


Limnor Studio (Administrator)
File Edit View Project Build Debug Tools Tutorial Window Help
Release
Any CPU
Form2.limnor [Design]
Form1.limnor [Design]
Start Debugging
Attributes

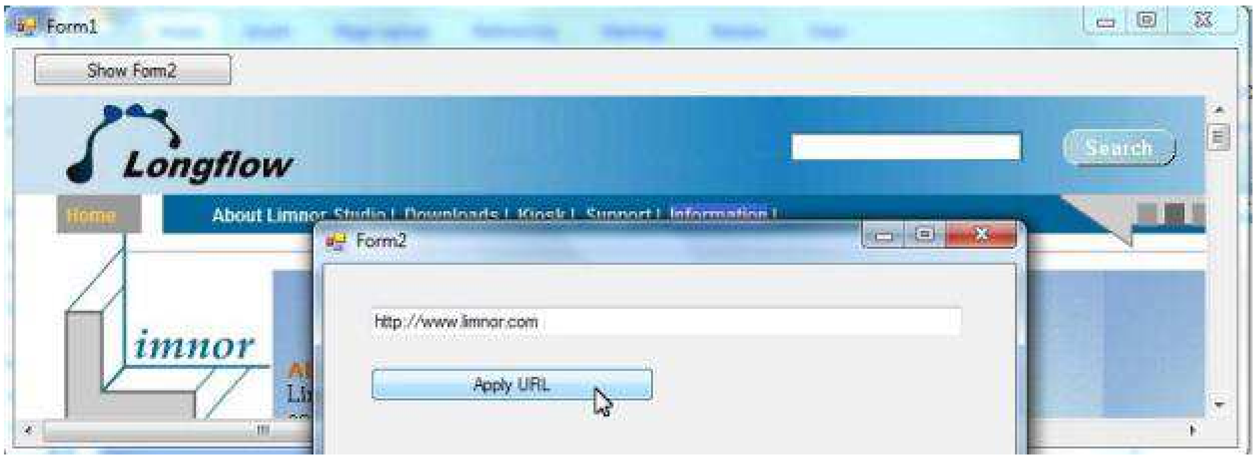
Form1 appears. Click button “Show Form2”:



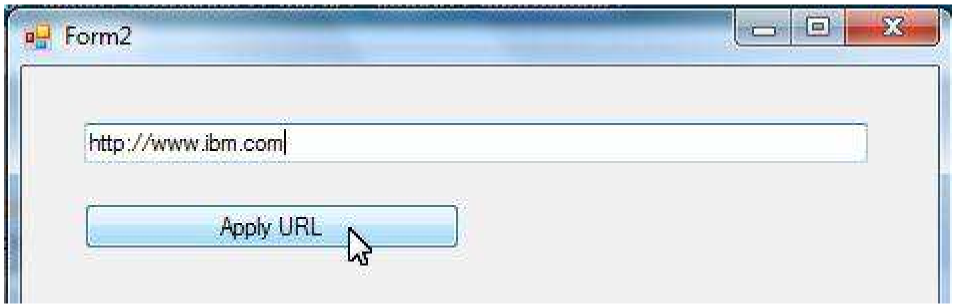
Form2 appears. Enter an URL in the text box. Click “Apply URL”:



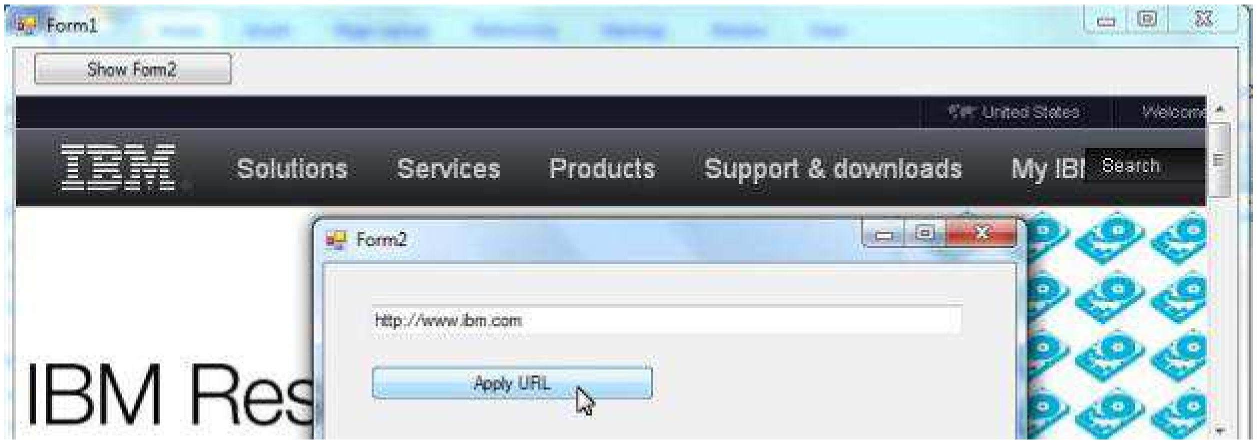
The web page appears in the web browser control on Form1:



Enter another URL. Click "Apply URL" again:



The new web page appears in Form1:



Create a dialogue box to collect user inputs

We'll show some techniques you may use to create dialogue boxes for collecting and using user inputs.

Close a dialogue box and get Dialogue Result

When a dialogue box is closed, we want to know the reason for its closing. For example, is it canceled?

A Form has a property DialogResult. We should use it to represent the reason for closing the dialogue box.

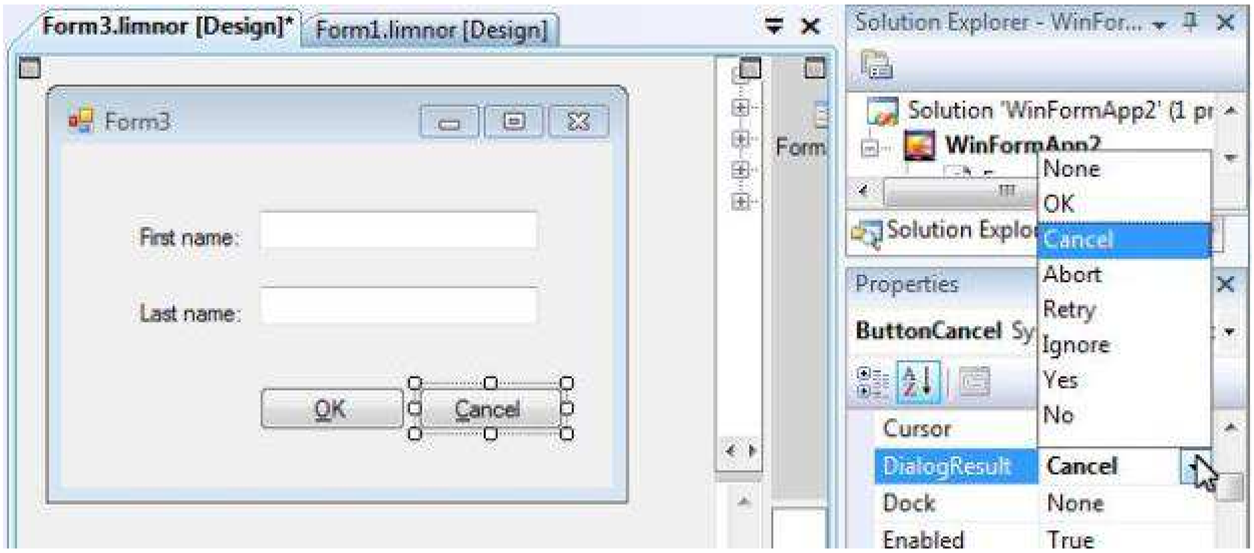
By convention, DialogResult should be set to OK when the dialogue box successfully gets data; DialogResult should be set to Cancel when the dialogue box is canceled by the user.

Note that the value of DialogResult and the meaning are programmed by the programmers designing the dialogue box. DialogResult can also be Ignore, Abort, Retry, Yes, No, and None. The programmers may use these values for other reasons if needed.

While a form is displayed as a dialogue box, if its DialogResult is set to a value other than None then the dialogue box is closed automatically, as if Close method is executed. So, usually Close action is not used to close a dialogue box. A “set property” action for DialogResult does two things in one action: give a closing reason and close the dialogue box.

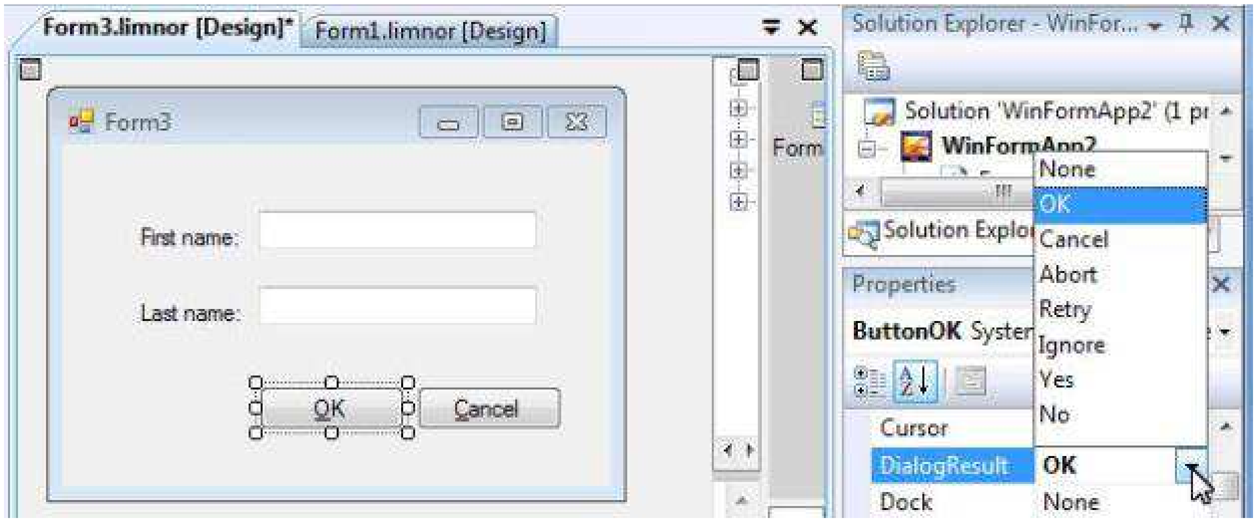
Another common way of closing a dialogue box is by buttons. One button is for one closing reason. The closing reason is set to the DialogResult property of the button.

For example, we set DialogResult to Cancel to a button used as cancel button:



At runtime when the user clicks the cancel button, the DialogResult property of Form3 is set to Cancel and the Form3 is closed. No need other actions.

An ok button is created by setting its DialogResult to OK:



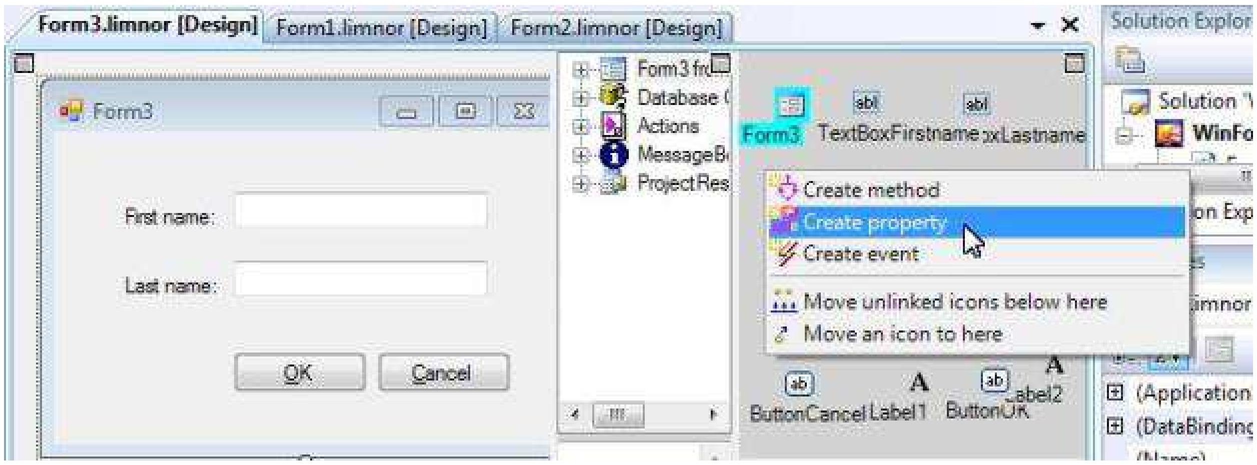
Use Properties for collecting user inputs

One idea we used here is that the dialogue boxes designed for collecting user inputs are independent of how the dialogue boxes will be used. It is following such a basic designing idea: a big system is constructed by independent small components. This is important when designing large systems.

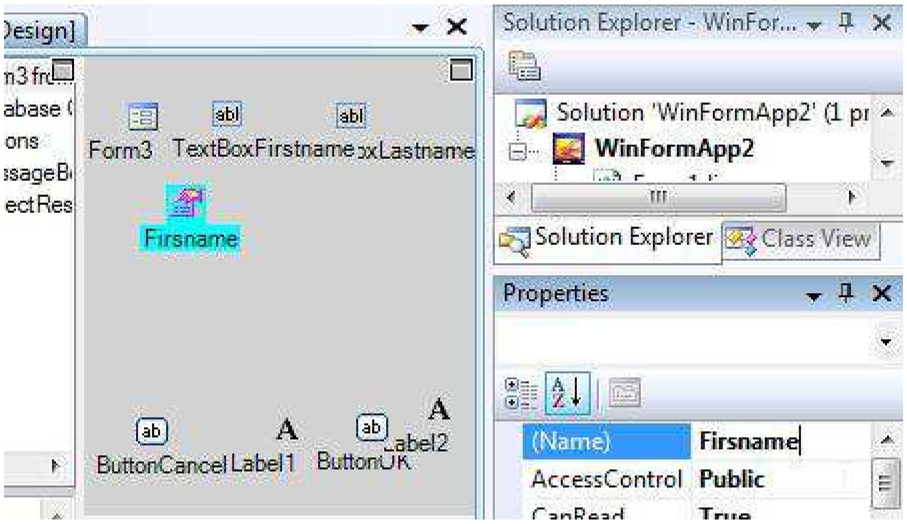
Suppose we create Form3 with two text boxes for data entry. Suppose we want to display the data the user entered in Form1.

We can do it just like we did in Form2 by executing Method1 in Form1. But in doing so, Form3 is not independent. It is coupled with Form1.

To make Form3 independent, we create two properties to hold the data the user entered. Other components will access these properties for user inputs. In our example, Form1 will access Form3, and Form3 will not access Form1 and other components.

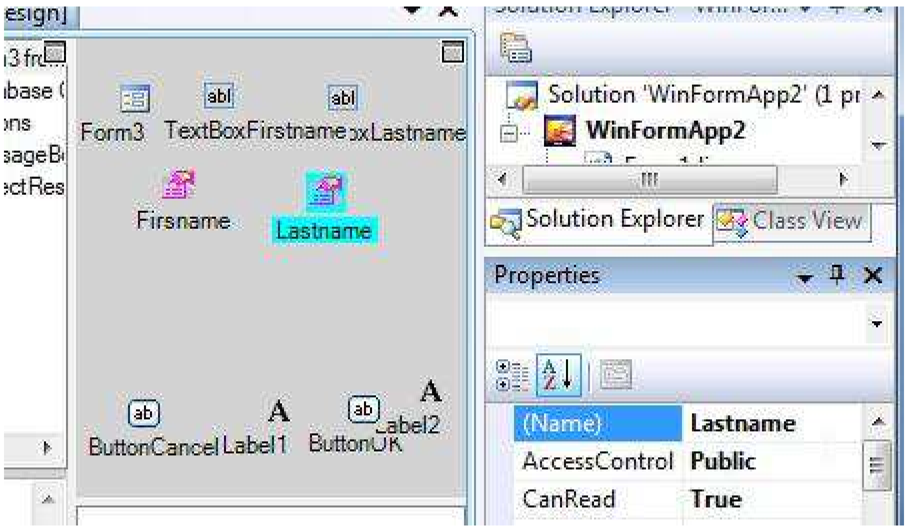


Change the name of the new property to Firstname:



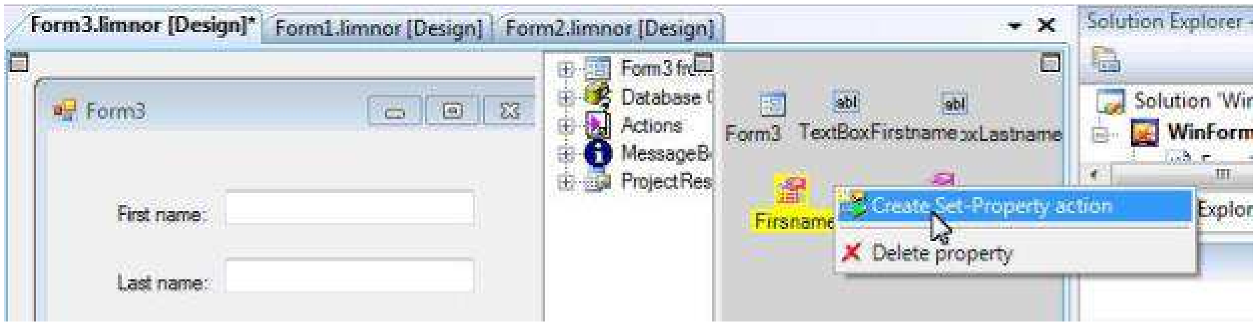
(Name)	Firsname
AccessControl	Public
CanRead	True

Create another property:

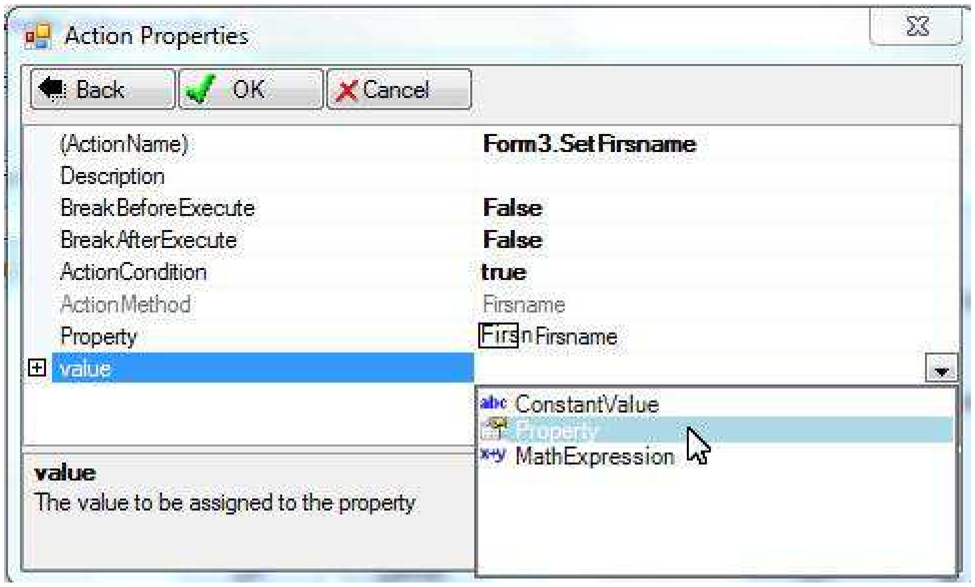


(Name)	Lastname
AccessControl	Public
CanRead	True

Create an action to set the Firstname property to the text of the text box:

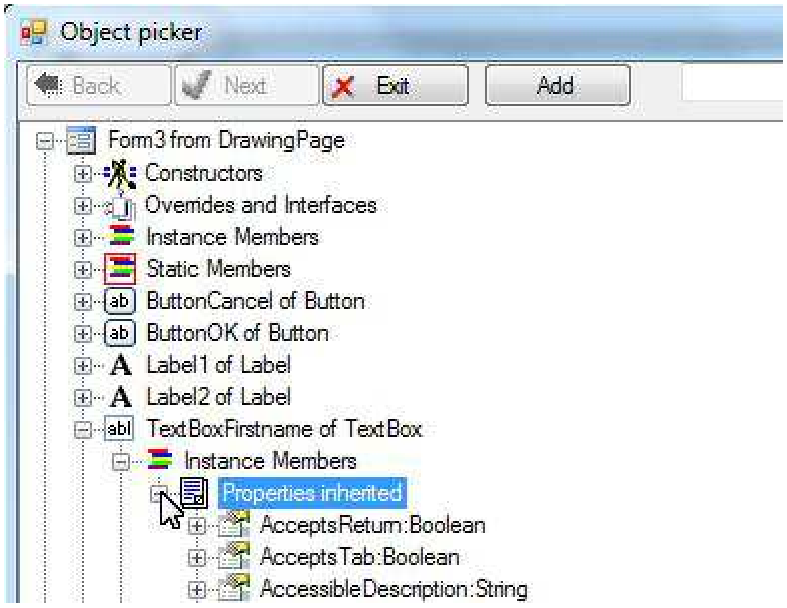


Use the Text property of the TextBox for the value of the action:

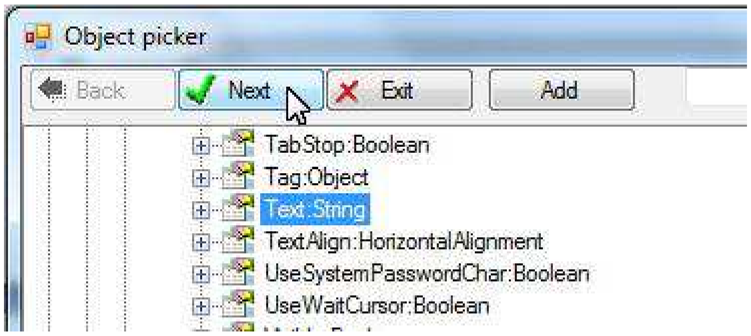


(ActionName)	Form3.SetFirsname
Description	
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	Firsname
Property	Firsname
value	

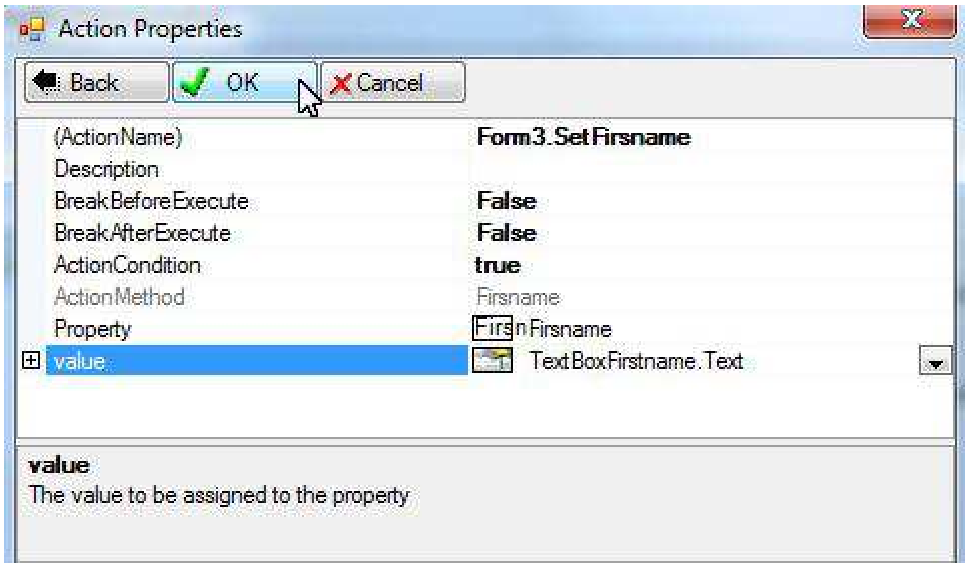
value
The value to be assigned to the property



```
Form3 from DrawingPage
  Constructors
  Overrides and Interfaces
  Instance Members
  Static Members
  ButtonCancel of Button
  ButtonOK of Button
  Label1 of Label
  Label2 of Label
  TextBoxFirstname of TextBox
    Instance Members
      Properties inherited
        AcceptsReturn:Boolean
        AcceptsTab:Boolean
        AccessibleDescription:String
```



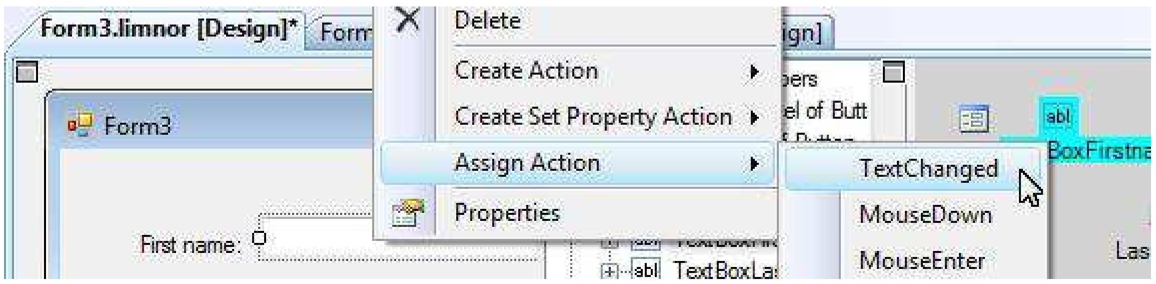
```
TabStop:Boolean
Tag:Object
Text:String
TextAlign:HorizontalAlignment
UseSystemPasswordChar:Boolean
UseWaitCursor:Boolean
```

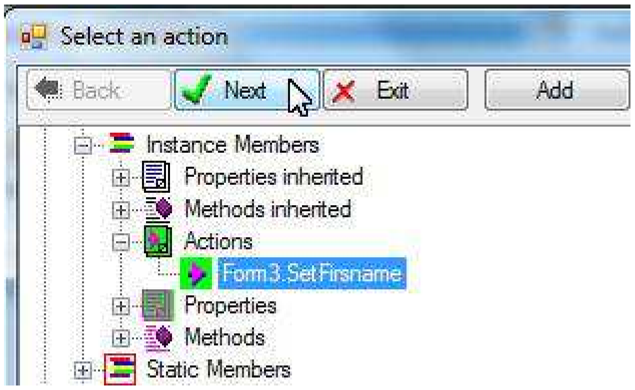


(ActionName)	**Form3.SetFirsname**
Description	
BreakBeforeExecute	**False**
BreakAfterExecute	**False**
ActionCondition	**true**
ActionMethod	Firsname
Property	Firsname
value	TextBoxFirstname.Text

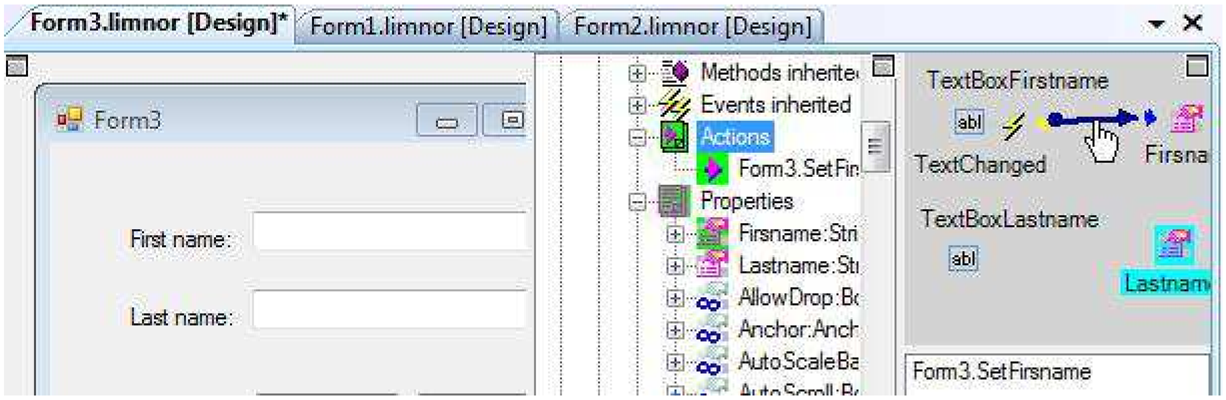
value
The value to be assigned to the property

Assign the action to the TextChanged event of the TextBox:

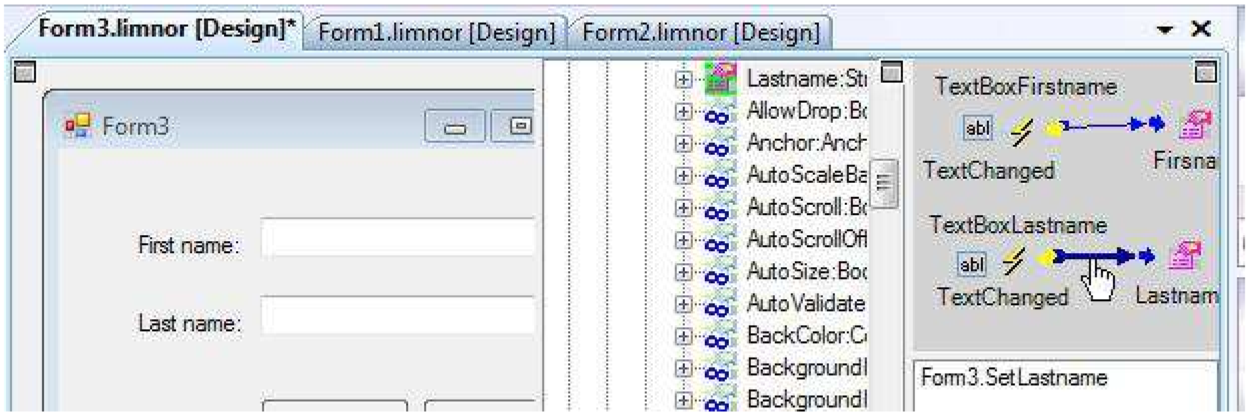




The action is assigned to TextChanged, thus whenever the user enters text the text is assigned to property Firstname:



Create another action to set the Lastname property to the Text property of the second TextBox and assign it to the TextChanged event of the second text box:

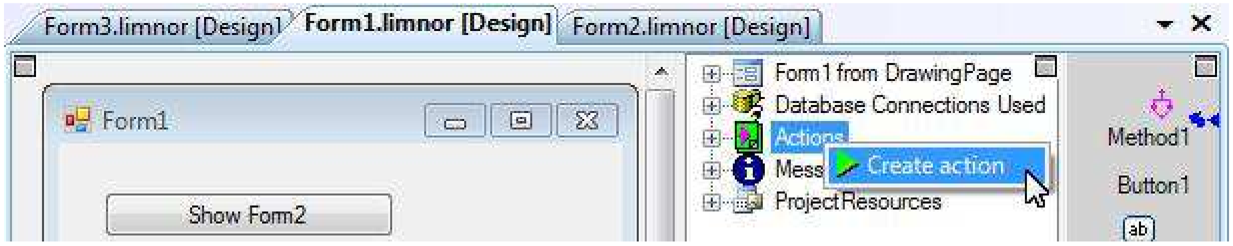


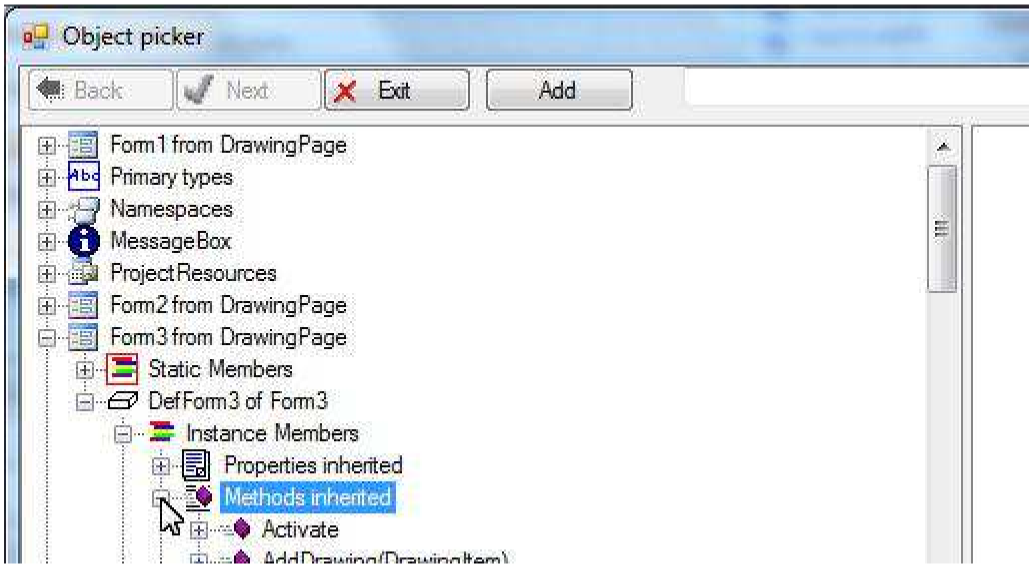
Use the dialogue box

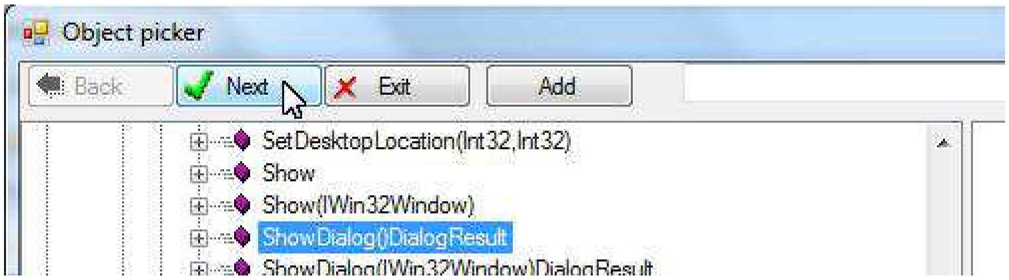
We can now use Form3 as a dialogue box to collect user inputs.

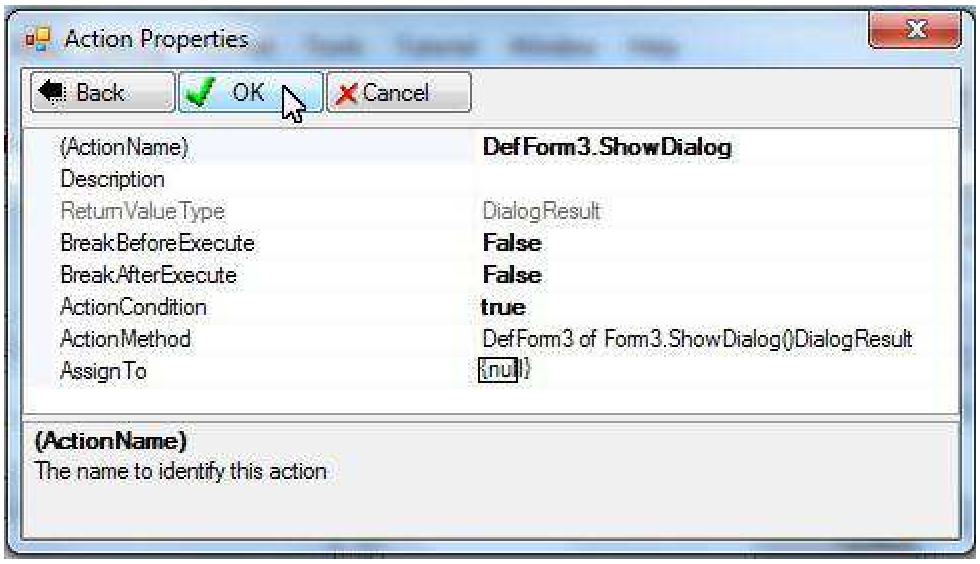
Show dialogue box

Create a ShowDialog action from Form1:









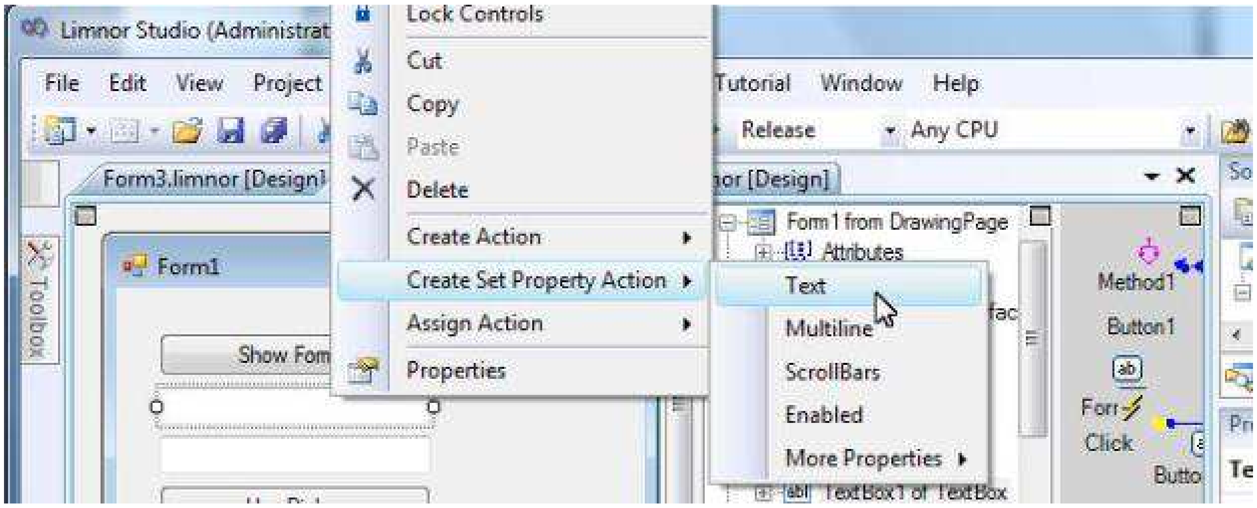
(ActionName)	**DefForm3.ShowDialog**
Description	
ReturnValueType	DialogResult
BreakBeforeExecute	**False**
BreakAfterExecute	**False**
ActionCondition	**true**
ActionMethod	DefForm3 of Form3.ShowDialog()DialogResult
AssignTo	(null)

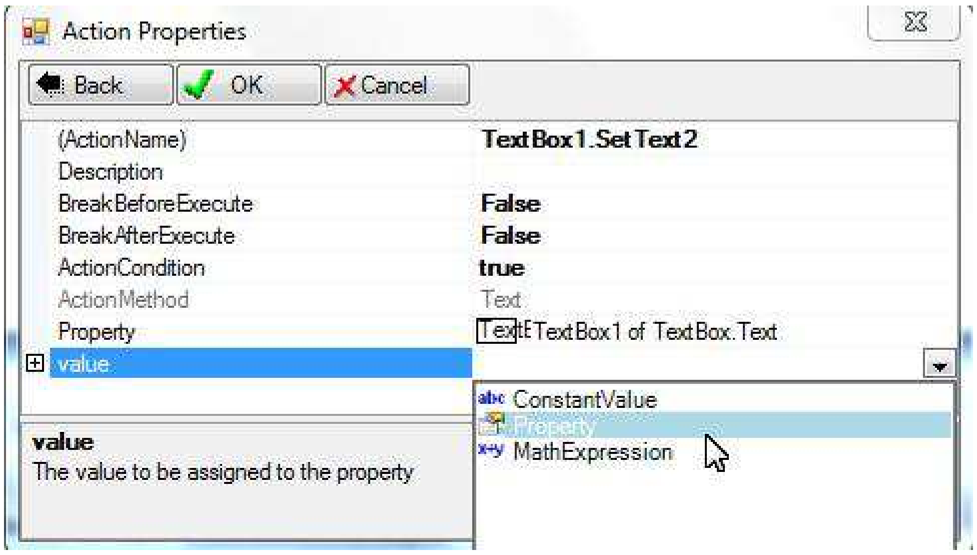
(ActionName)
The name to identify this action

Get data from dialogue box

Get Firstname

We may create an action to assign the Firstname property of Form3 to the first text box on Form1.



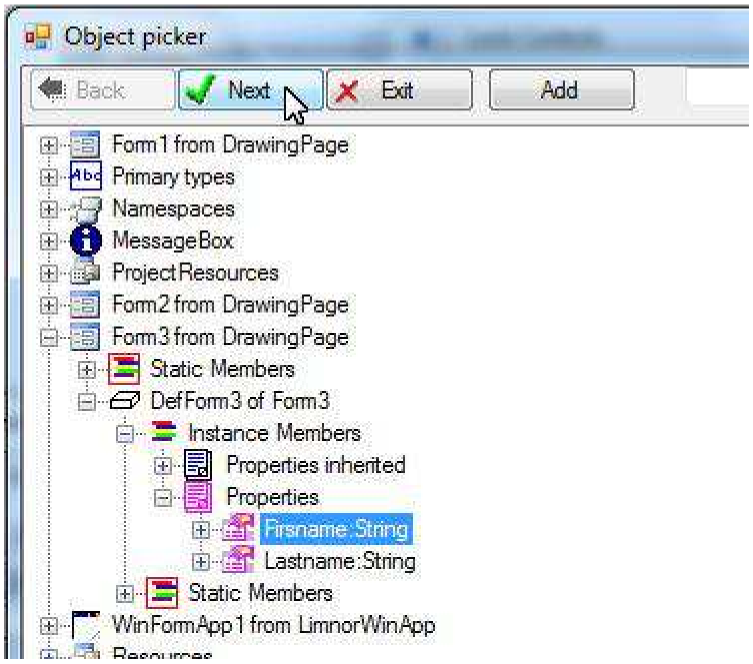

Action Properties

Back | OK | Cancel

(ActionName)	TextBox1.SetText2
Description	
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	Text
Property	TextBox1 of TextBox.Text
value	

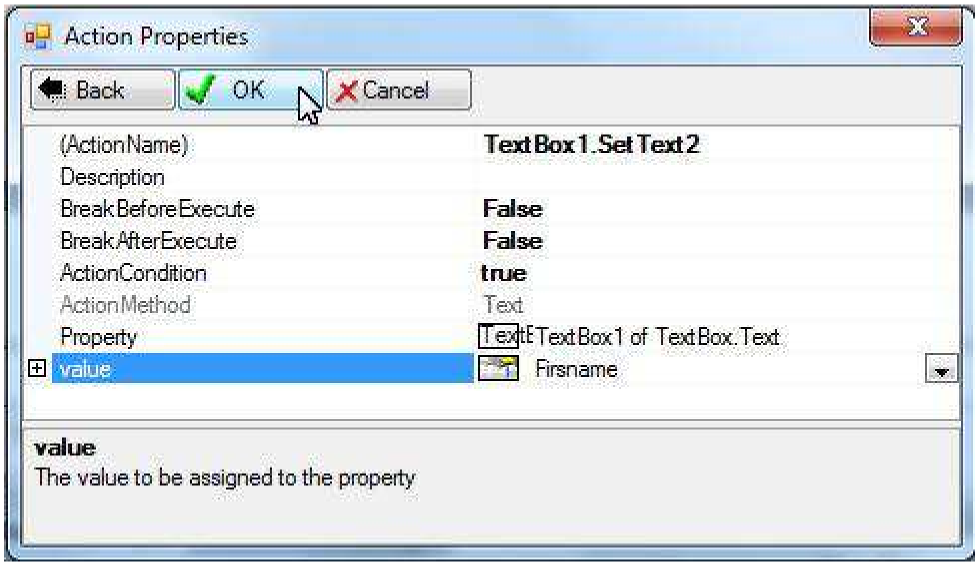
ConstantValue
Property
MathExpression

value
The value to be assigned to the property


Object picker

Back | Next | Exit | Add

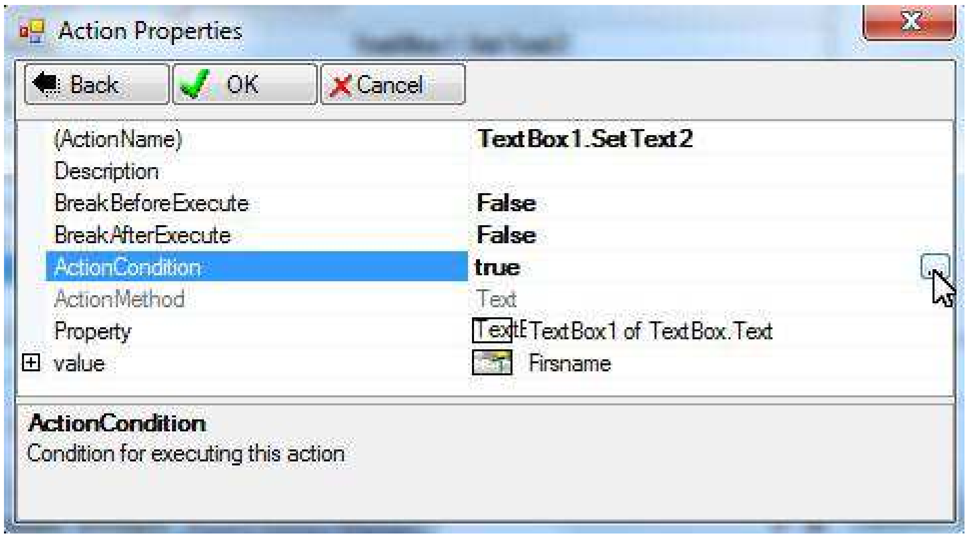
```
Form1 from DrawingPage
Primary types
Namespaces
MessageBox
ProjectResources
Form2 from DrawingPage
Form3 from DrawingPage
    Static Members
    DefForm3 of Form3
        Instance Members
            Properties inherited
            Properties
                Firsname:String
                Lastname:String
        Static Members
WinFormApp1 from LimnorWinApp
Resources
```



(ActionName)	TextBox1.SetText2
Description	
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	Text
Property	TextBox1 of TextBox.Text
value	Firsname

value
The value to be assigned to the property

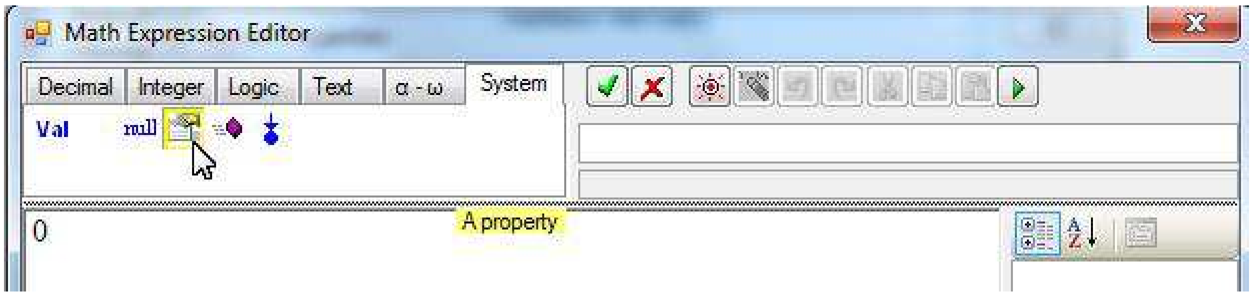
Set ActionCondition so that this action only executes when the DialogResult of Form3 is OK:



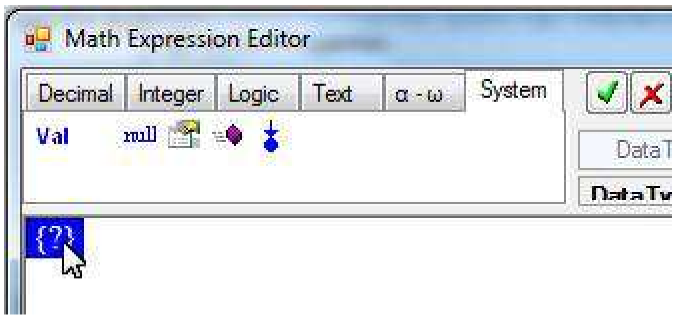
(ActionName)	TextBox1.SetText2
Description	
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	Text
Property	TextBox1 of TextBox.Text
value	Firsname

ActionCondition
Condition for executing this action

Click the Property icon:



Double-click the property element to select the DialogResult property of Form3:

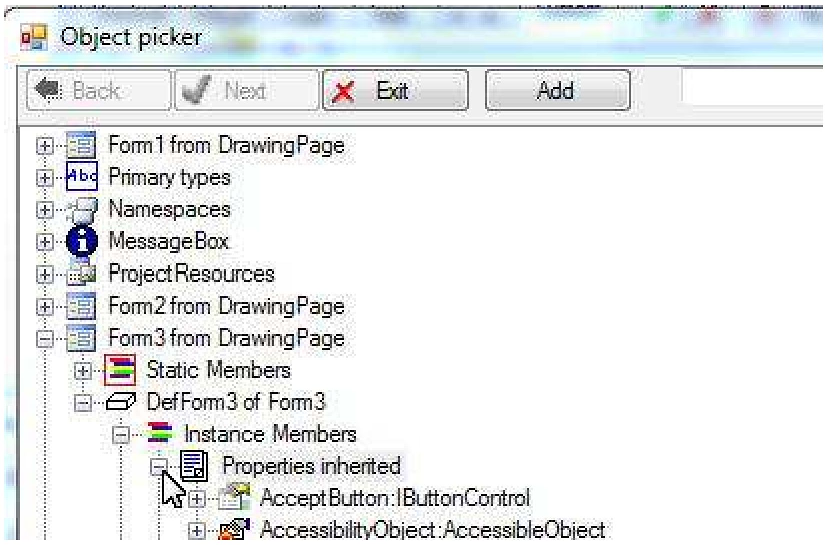

Math Expression Editor

Decimal | Integer | Logic | Text | α - ω | System

Val null

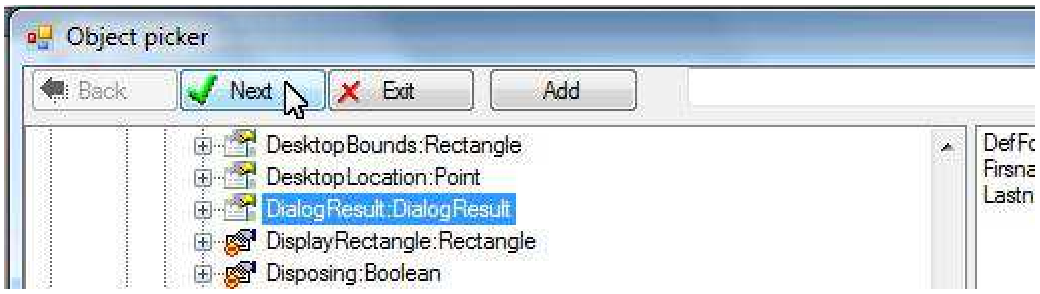
DataT

{?}


Object picker

Back | Next | Exit | Add

- Form1 from DrawingPage
- Primary types
- Namespaces
- MessageBox
- ProjectResources
- Form2 from DrawingPage
- Form3 from DrawingPage
  - Static Members
  - DefForm3 of Form3
    - Instance Members
      - Properties inherited
        - AcceptButton:IButtonControl
        - AccessibilityObject:AccessibleObject

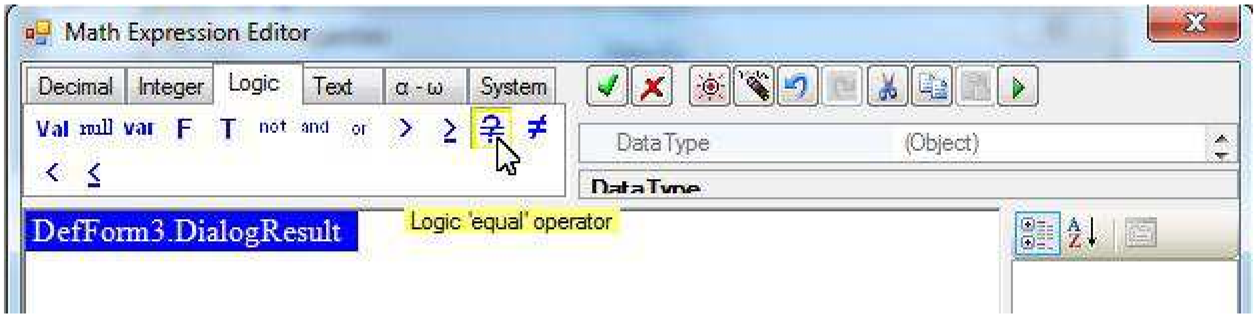

Object picker

Back | Next | Exit | Add

- DesktopBounds:Rectangle
- DesktopLocation:Point
- DialogResult:DialogResult
- DisplayRectangle:Rectangle
- Disposing:Boolean

DefFo
Firsna
Lastn

Click equality checking icon:


Math Expression Editor

Decimal | Integer | Logic | Text | α - ω | System

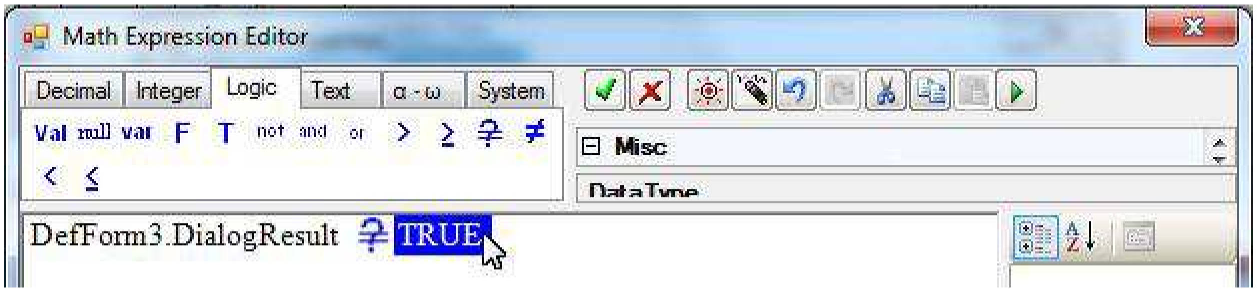
Val null var F T not and or > ≥ ≟ ≠ < ≤

DataType (Object)

DefForm3.DialogResult

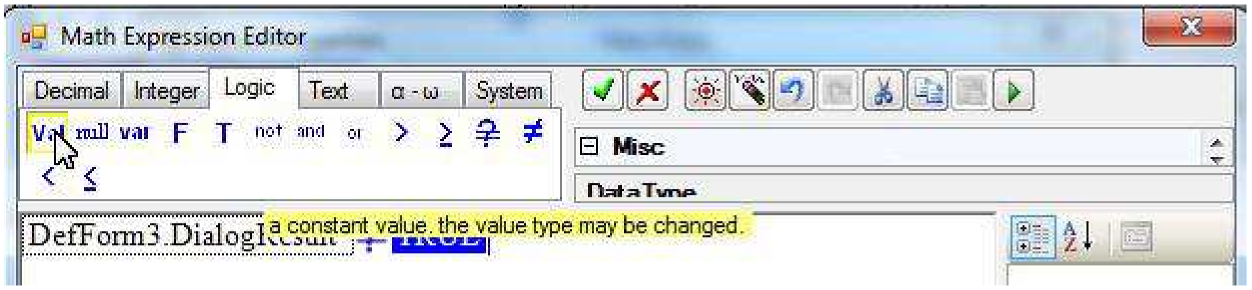
Logic 'equal' operator

Select the second element of the equality checking:

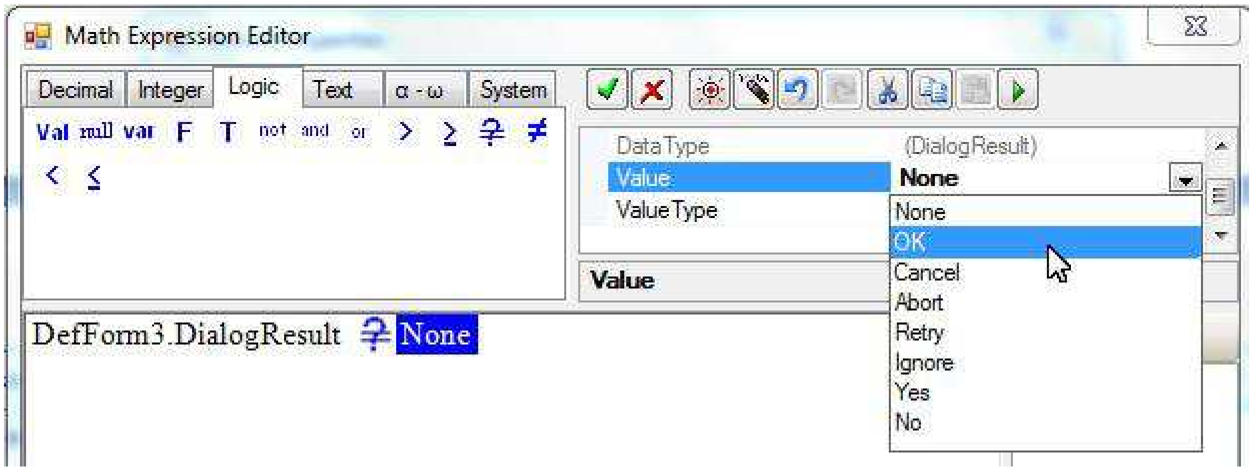


DefForm3.DialogResult ≠ TRUE

Click the constant icon:



Select OK for the value:

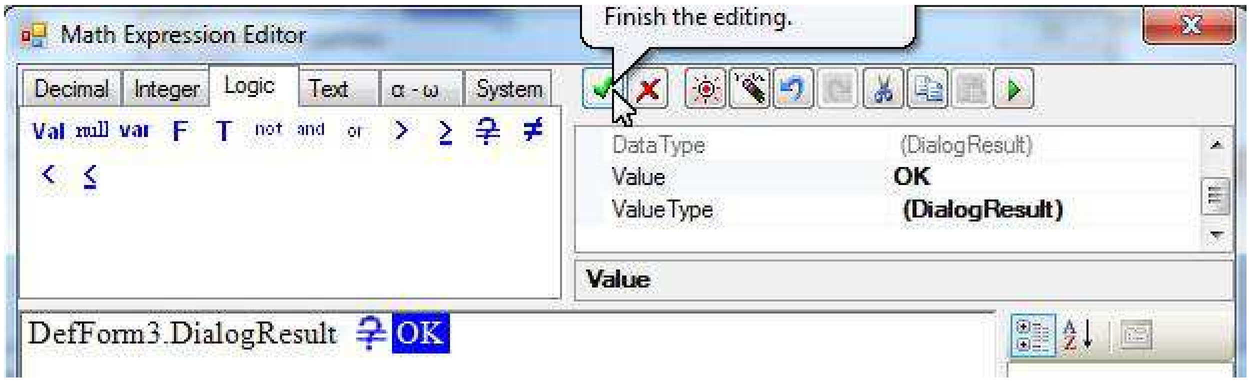


DataType	(DialogResult)
Value	None
ValueType	

None
OK
Cancel
Abort
Retry
Ignore
Yes
No

DefForm3.DialogResult ≠ None

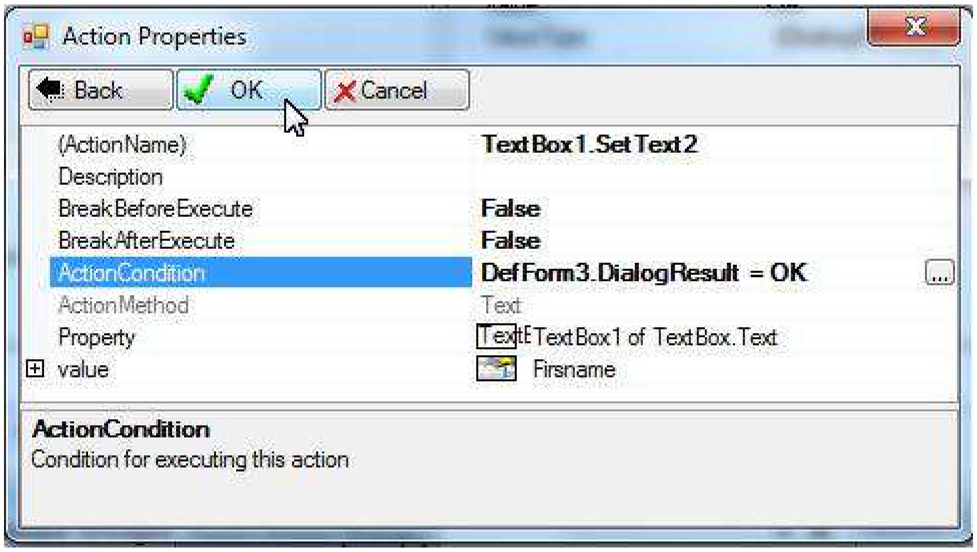
This condition checks whether the dialogue box close reason is OK:



DataType	(DialogResult)
Value	OK
ValueType	(DialogResult)

DefForm3.DialogResult ≠ OK

This is the action for getting Firstname from Form3:


Action Properties

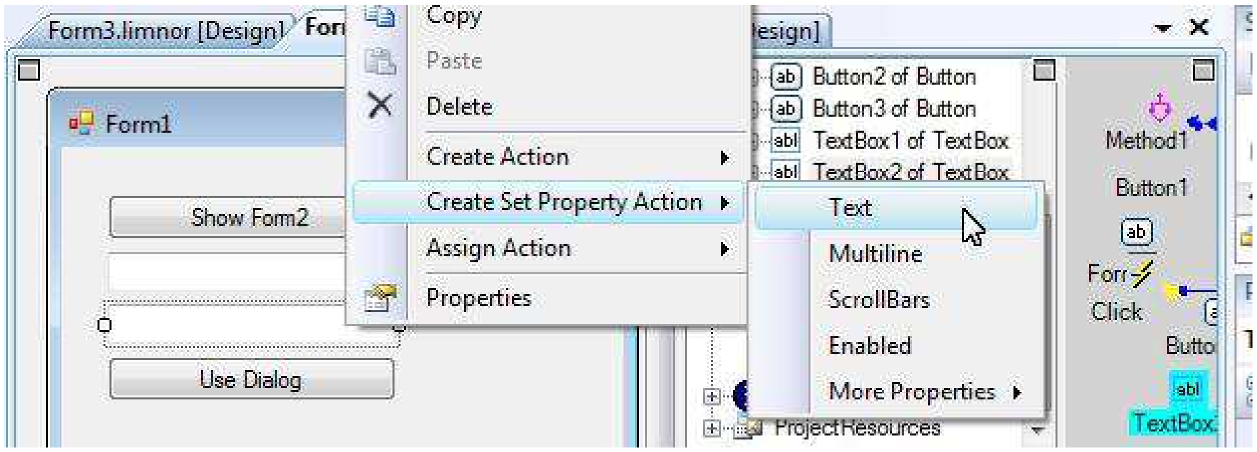
Back | OK | Cancel

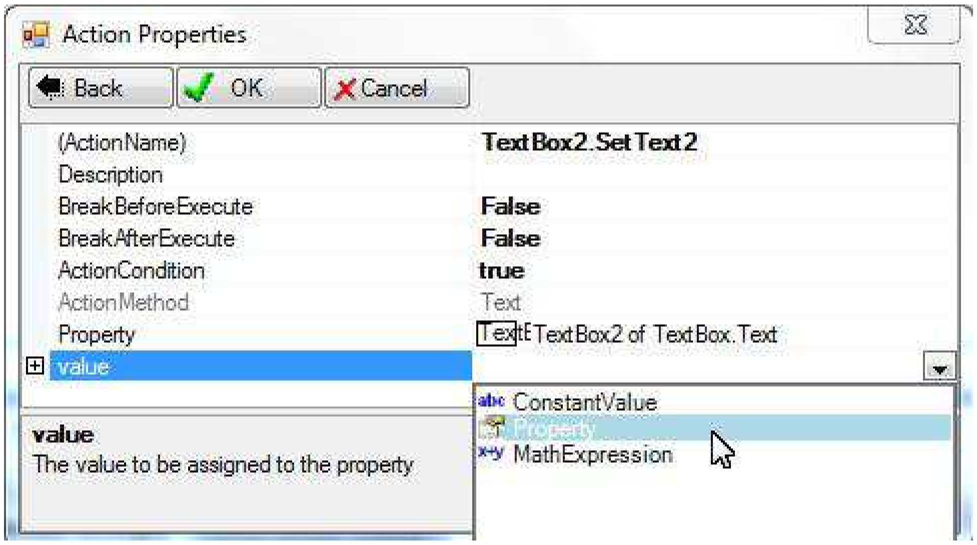
(ActionName)	TextBox1.SetText2
Description	
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	DefForm3.DialogResult = OK
ActionMethod	Text
Property	TextBox1 of TextBox.Text
value	Firsname

ActionCondition
Condition for executing this action

Get Lastname

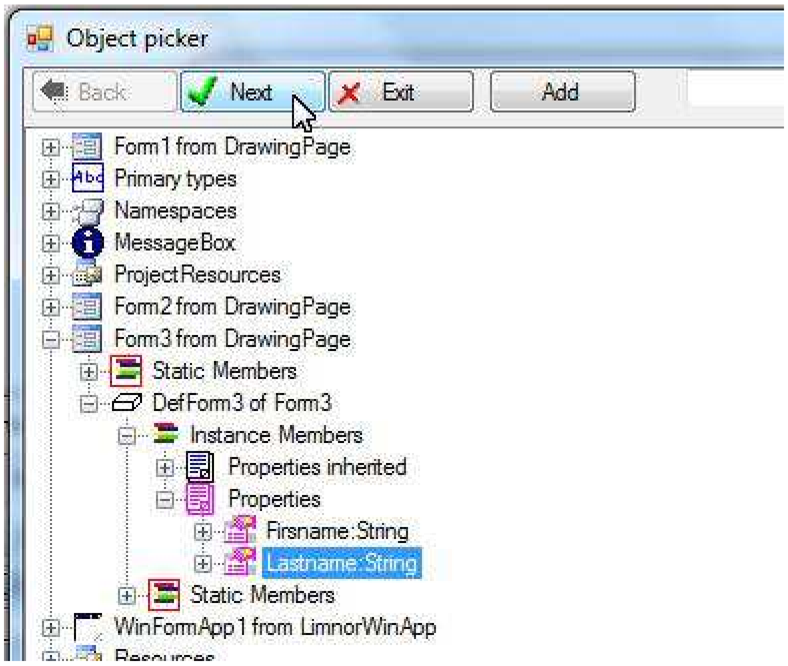
We may create another action to assign the Lastname property of Form3 to the Text property of the second text box on Form1:

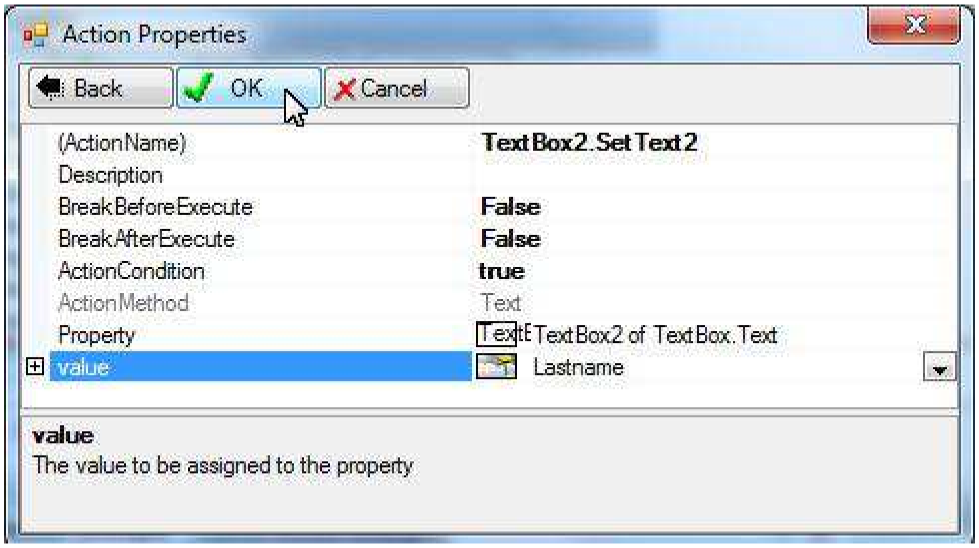




(ActionName)	TextBox2.SetText2
Description	
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	Text
Property	TextE TextBox2 of TextBox.Text
value	

value
The value to be assigned to the property

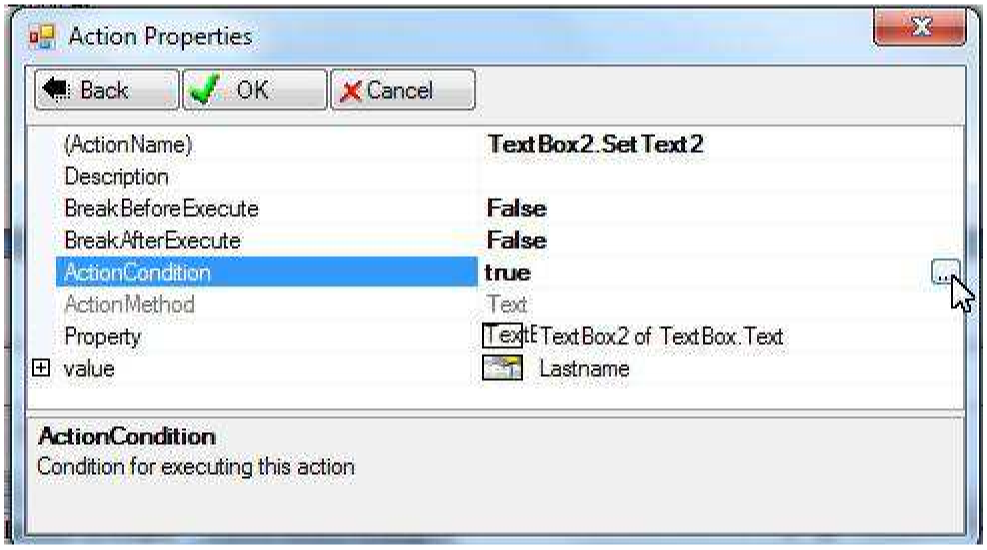




(ActionName)	TextBox2.SetText2
Description	
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	Text
Property	TextBox2 of TextBox.Text
value	Lastname

value
The value to be assigned to the property

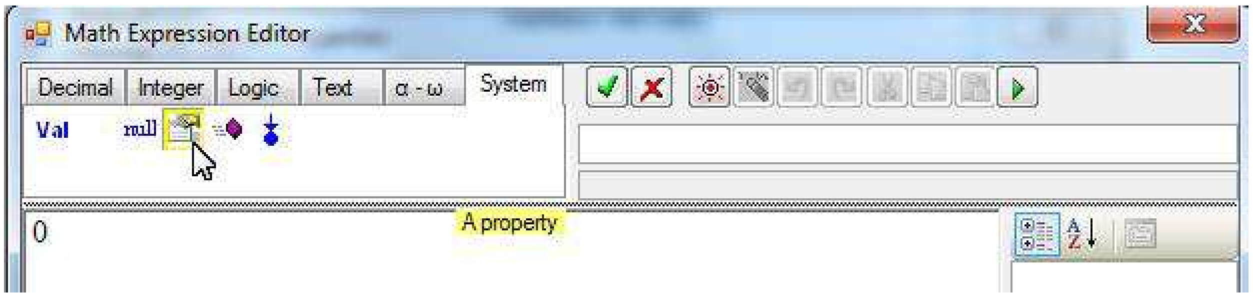
Set the ActionCondition so that this action can only be executed when the DialogResult of Form3 is OK:



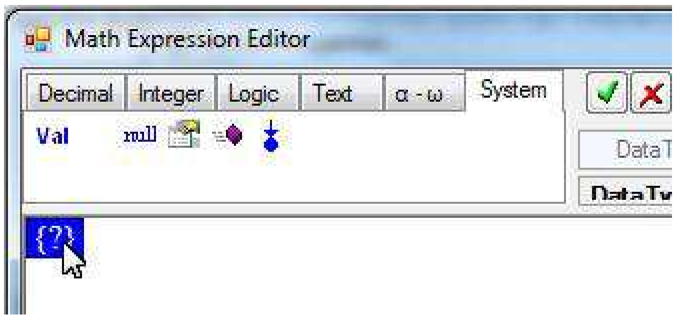
(ActionName)	TextBox2.SetText2
Description	
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	Text
Property	TextBox2 of TextBox.Text
value	Lastname

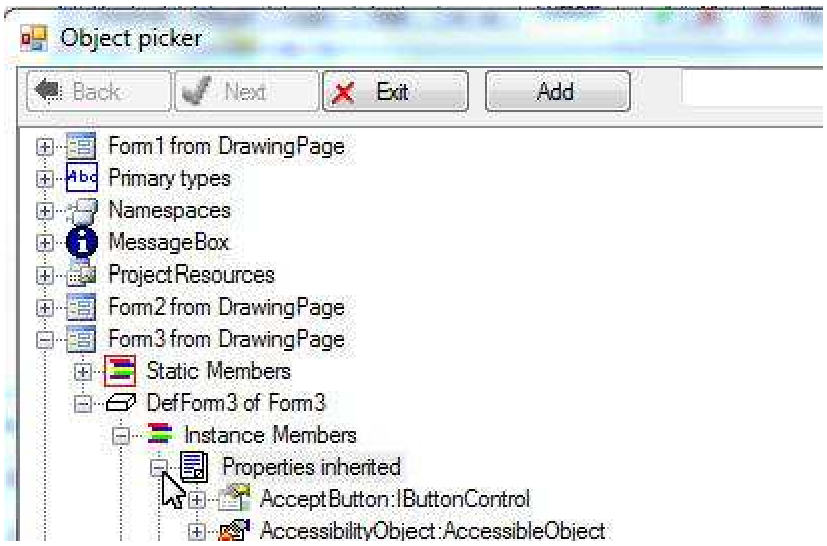
ActionCondition
Condition for executing this action

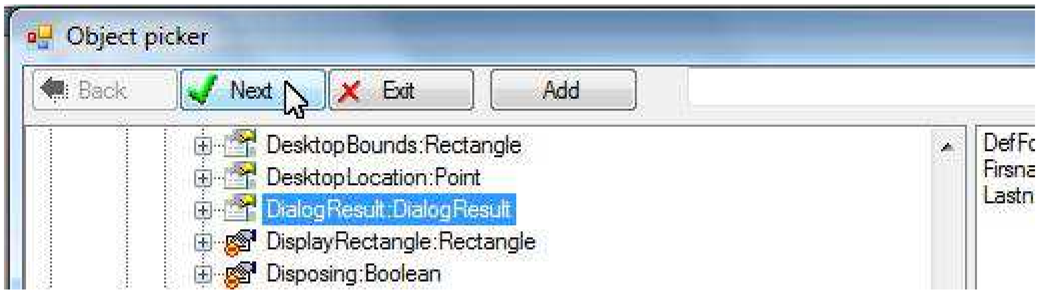
Click the Property icon:



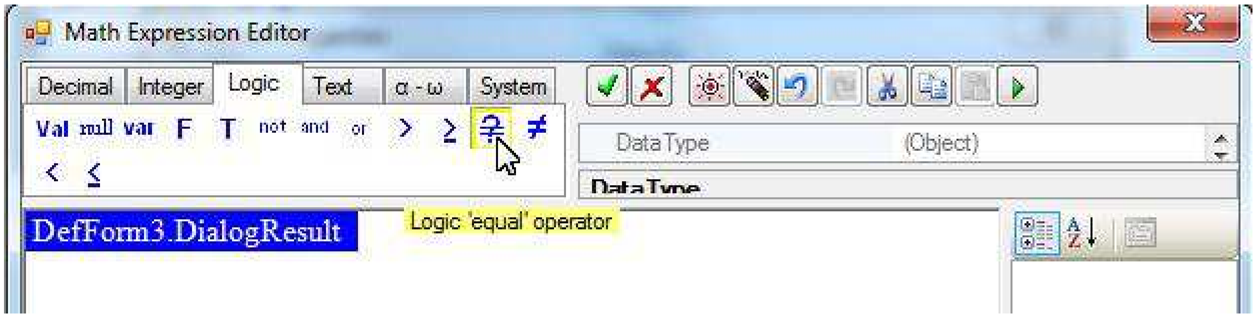
Double-click the property element to select the DialogResult property of Form3:



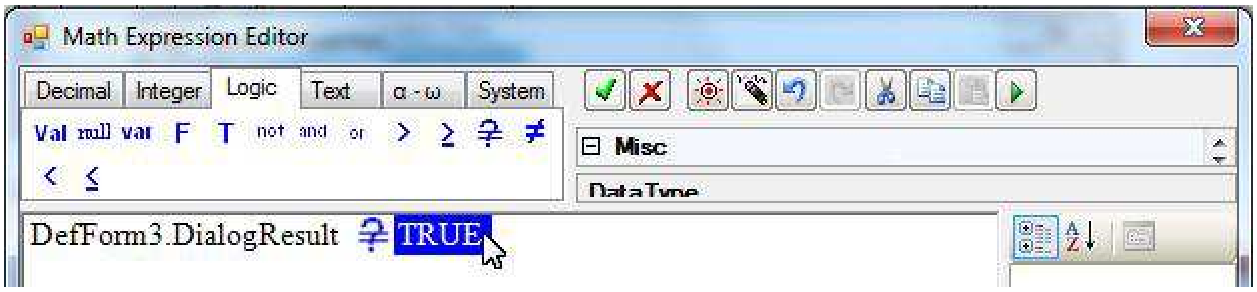




Click equality checking icon:

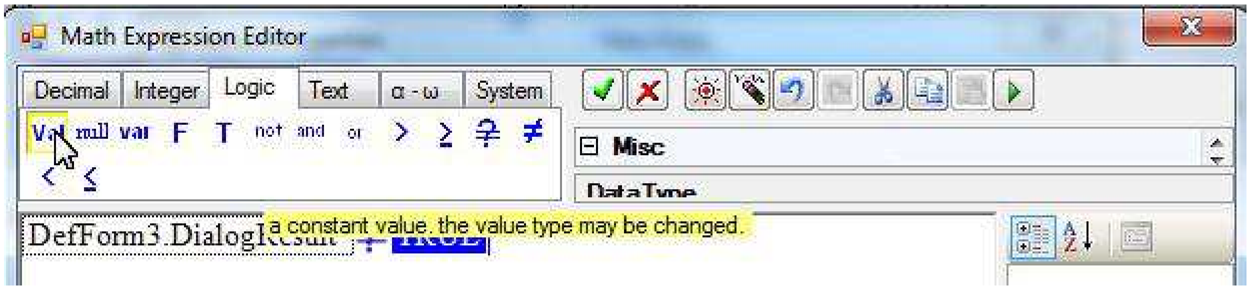


Select the second element of the equality checking:

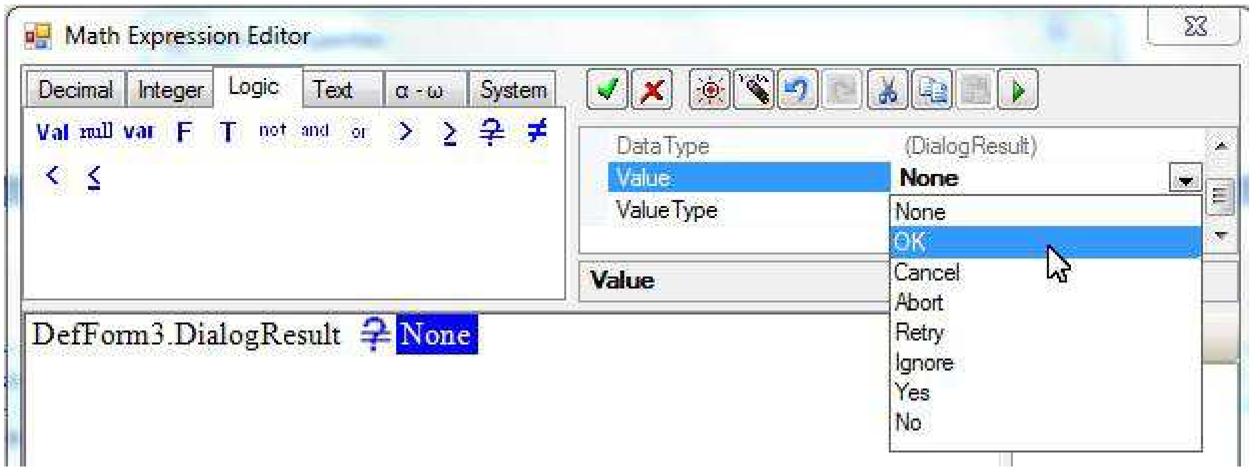


DefForm3.DialogResult ≠ TRUE

Click the constant icon:



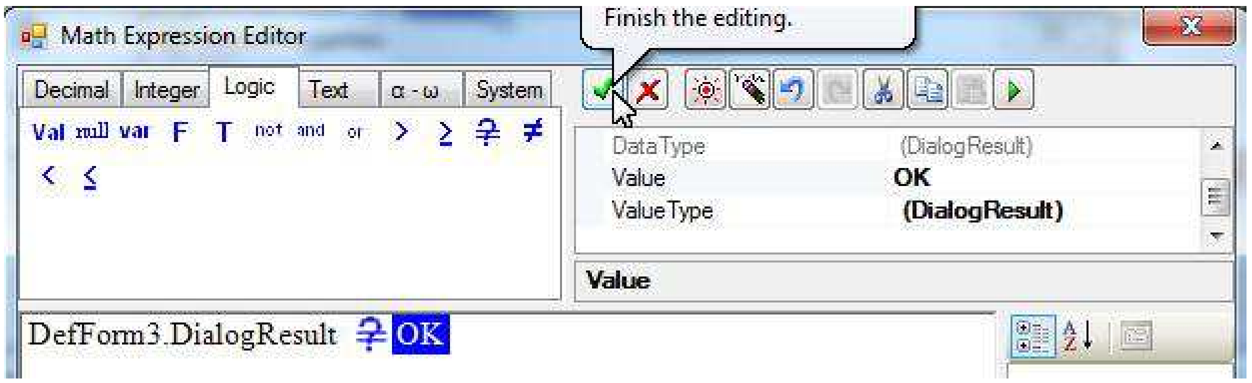
Select OK for the value:



DataType	(DialogResult)
Value	None
ValueType	

DefForm3.DialogResult ≠ None

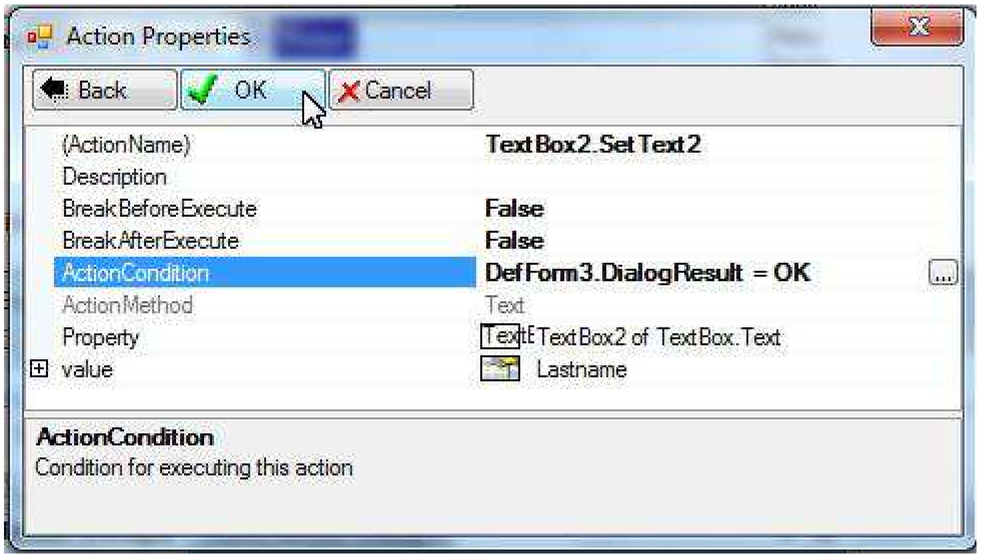
This condition checks whether the dialogue box close reason is OK:



DataType	(DialogResult)
Value	OK
ValueType	(DialogResult)

DefForm3.DialogResult ≠ OK

This is the action for getting Lastname from Form3:


Action Properties

Back OK Cancel

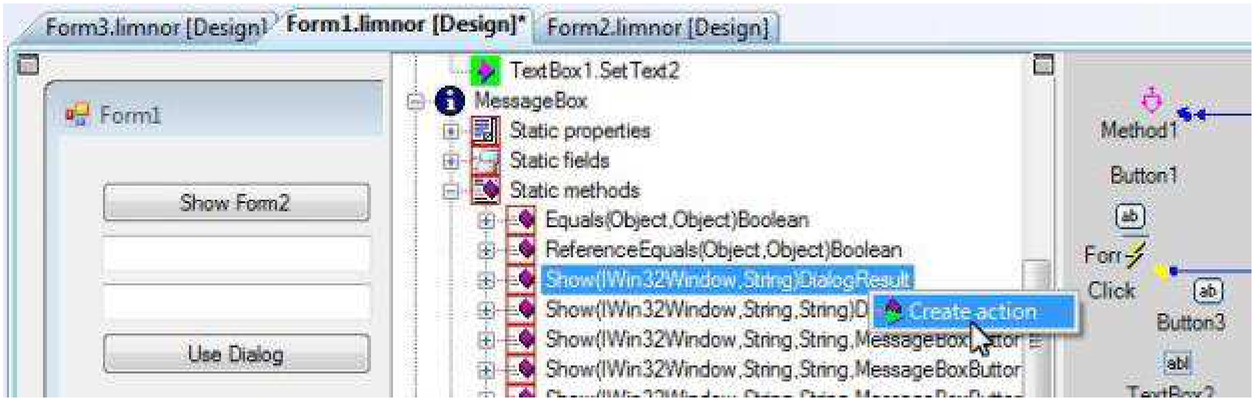
(ActionName)	TextBox2.SetText2
Description	
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	DefForm3.DialogResult = OK
ActionMethod	Text
Property	TextBox2 of TextBox.Text
value	Lastname

ActionCondition
Condition for executing this action

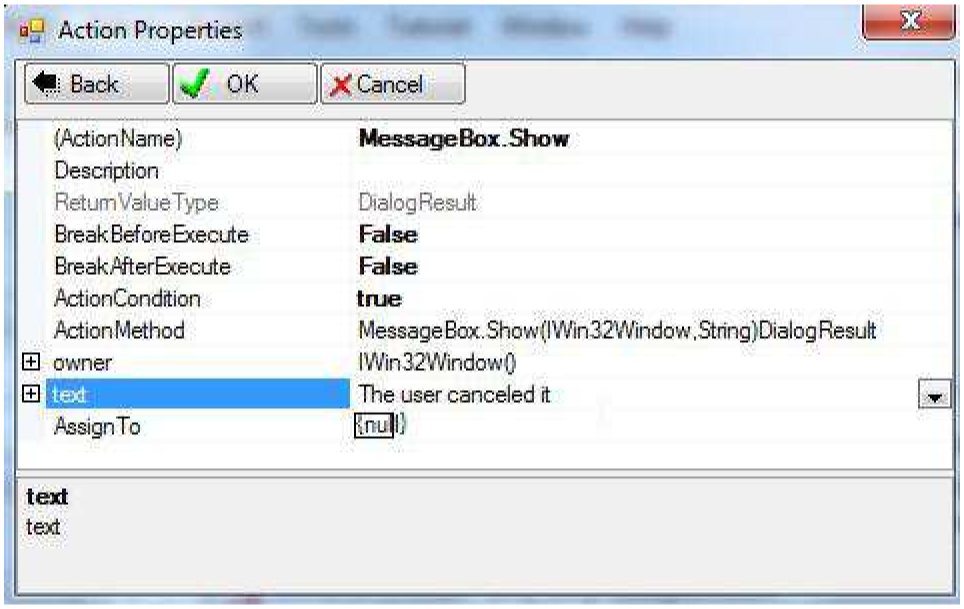
Show "User canceled" message box

Suppose that we want to show a message box when the dialogue box is canceled.

Right-click a Show method of the MessageBox class; choose "Create action":


Form3.limnor [Design]
Form1.limnor [Design]*
Form2.limnor [Design]
Form1
Show Form2
Use Dialog
TextBox1.SetText2
MessageBox
Static properties
Static fields
Static methods
Equals(Object,Object)Boolean
ReferenceEquals(Object,Object)Boolean
Show(IWin32Window,String)DialogResult
Show(IWin32Window,String,String)D
Create action
Show(IWin32Window,String,String,MessageBox
Show(IWin32Window,String,String,MessageBoxButton
Method1
Button1
Click
Button3

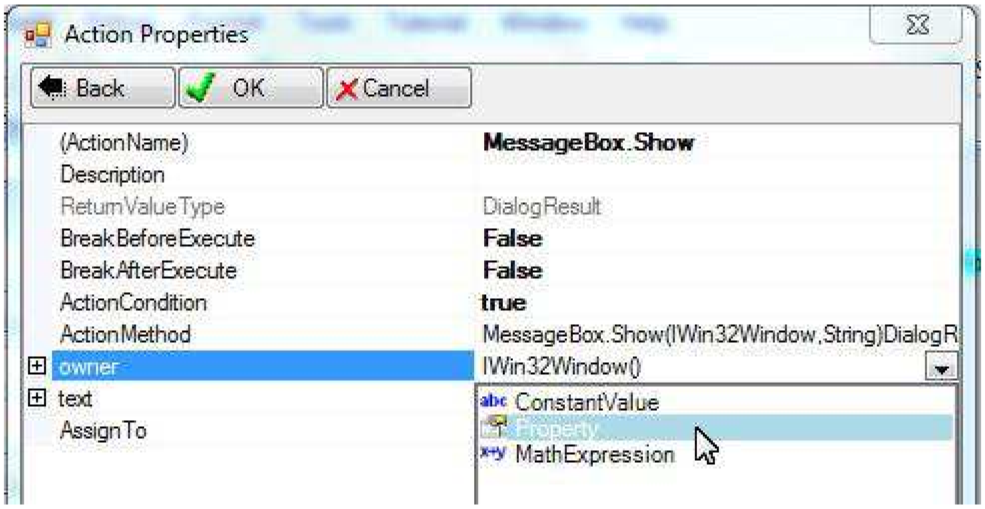
Give a message to be displayed:


Action Properties

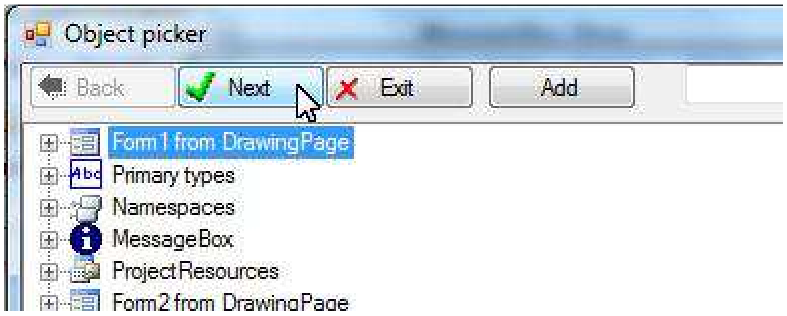
(ActionName)	**MessageBox.Show**
Description	
ReturnValueType	DialogResult
BreakBeforeExecute	**False**
BreakAfterExecute	**False**
ActionCondition	**true**
ActionMethod	MessageBox.Show(IWin32Window,String)DialogResult
owner	IWin32Window()
text	The user canceled it
AssignTo	(null)

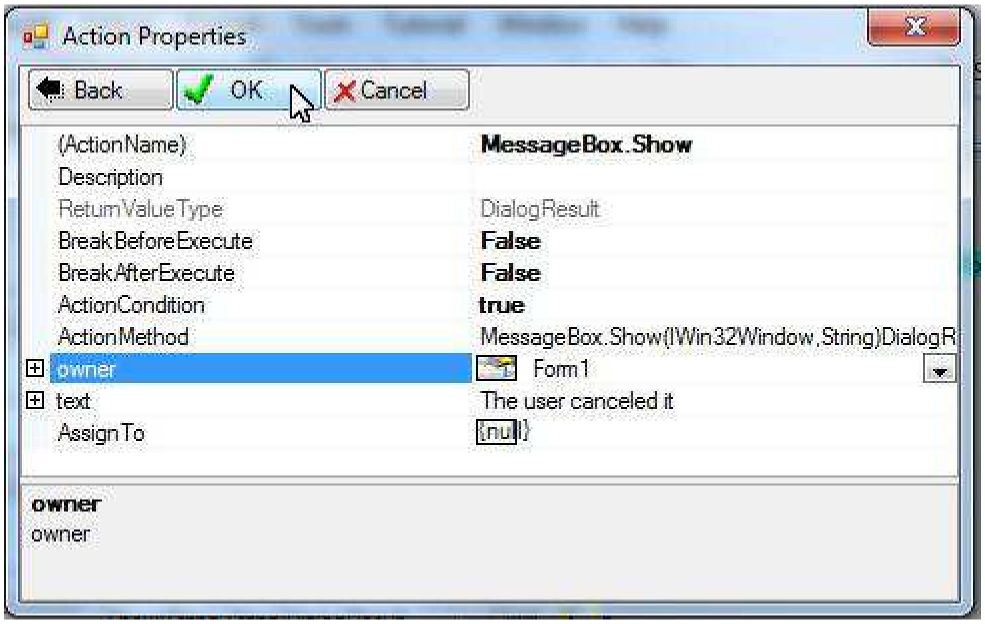
text
text

Choose Form1 as the owner of the message box:


Action Properties

(ActionName)	**MessageBox.Show**
Description	
ReturnValueType	DialogResult
BreakBeforeExecute	**False**
BreakAfterExecute	**False**
ActionCondition	**true**
ActionMethod	MessageBox.Show(IWin32Window,String)DialogR
owner	IWin32Window()
text	
AssignTo	

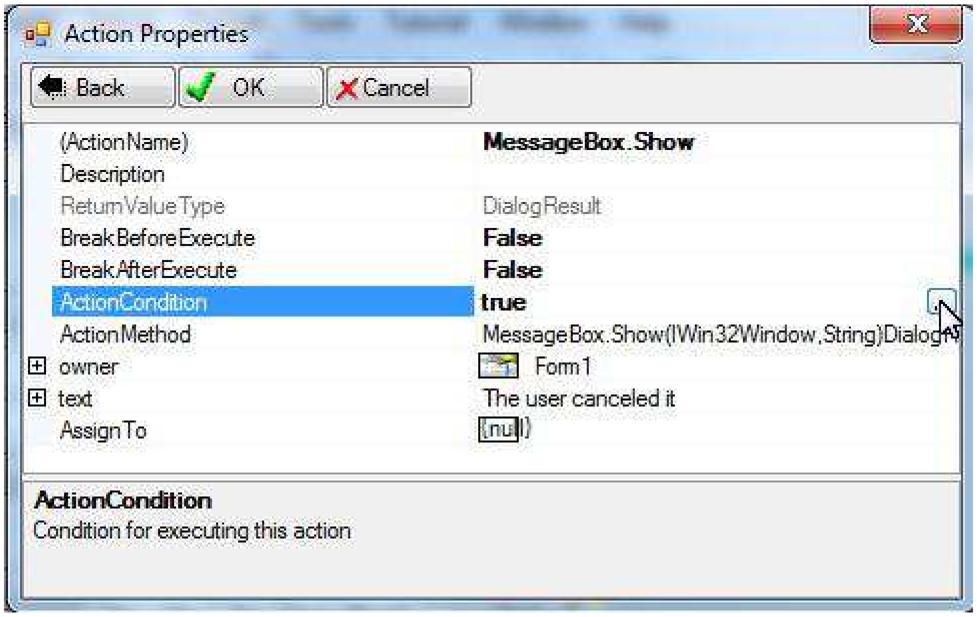




(ActionName)	**MessageBox.Show**
Description	
ReturnValueType	DialogResult
BreakBeforeExecute	**False**
BreakAfterExecute	**False**
ActionCondition	**true**
ActionMethod	MessageBox.Show(IWin32Window,String)DialogR
⊞ owner	Form1
⊞ text	The user canceled it
AssignTo	(null)

owner
owner

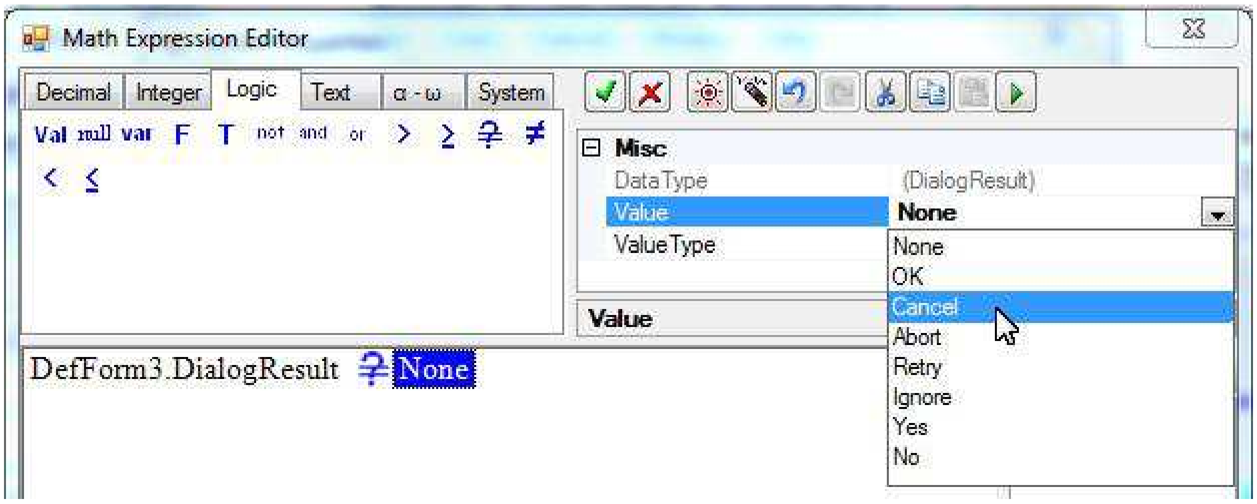
Set the ActionCondition to execute it only when the cancel button is clicked:



(ActionName)	**MessageBox.Show**
Description	
ReturnValueType	DialogResult
BreakBeforeExecute	**False**
BreakAfterExecute	**False**
ActionCondition	**true**
ActionMethod	MessageBox.Show(IWin32Window,String)Dialog
⊞ owner	Form1
⊞ text	The user canceled it
AssignTo	(null)

ActionCondition
Condition for executing this action

Like we did before, we create an expression to check DialogResult:

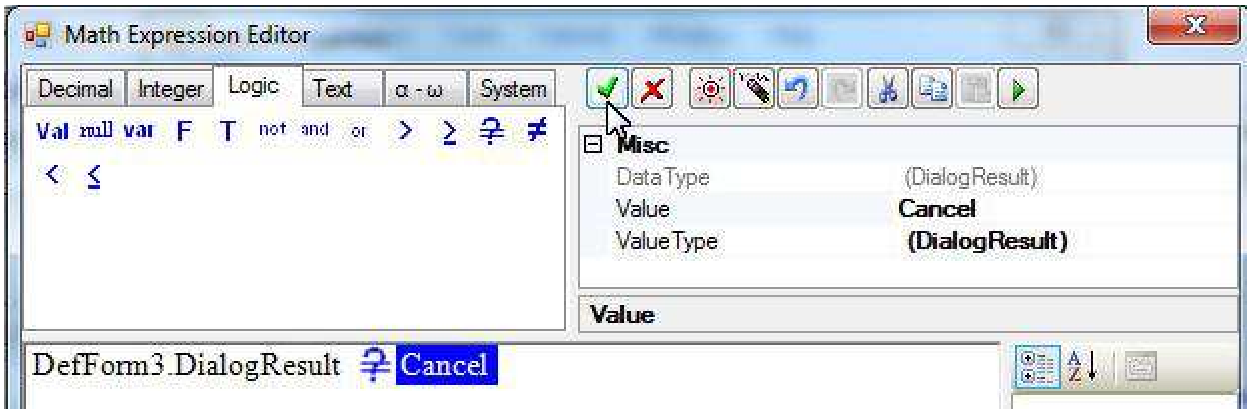


Misc
DataType (DialogResult)
Value None
ValueType

Value

None
OK
Cancel
Abort
Retry
Ignore
Yes
No

DefForm3.DialogResult ≠ None

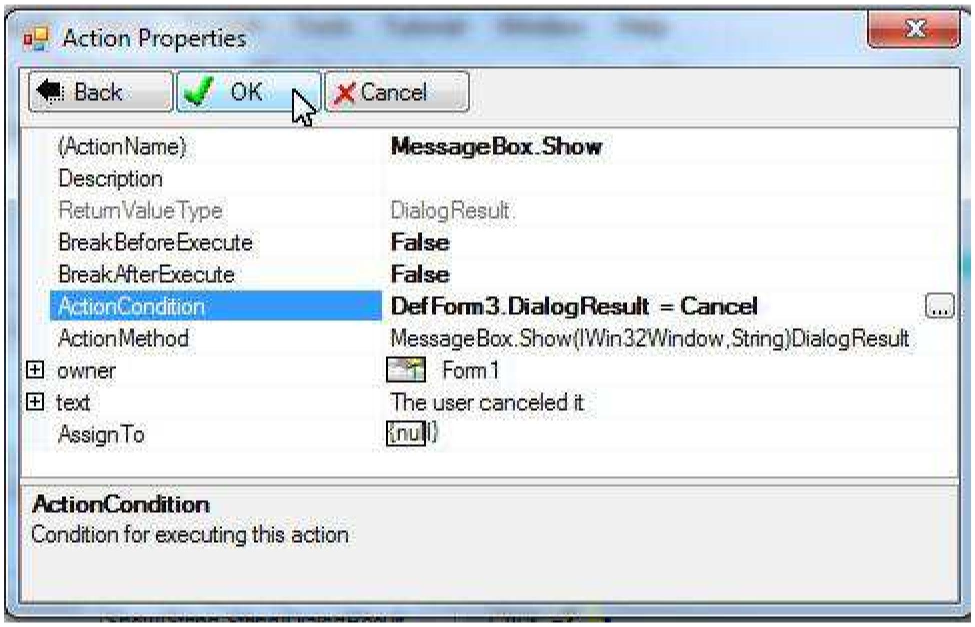


Misc
DataType (DialogResult)
Value Cancel
ValueType (DialogResult)

Value

DefForm3.DialogResult ≠ Cancel

This action displays a message box when the user cancels the dialogue box:

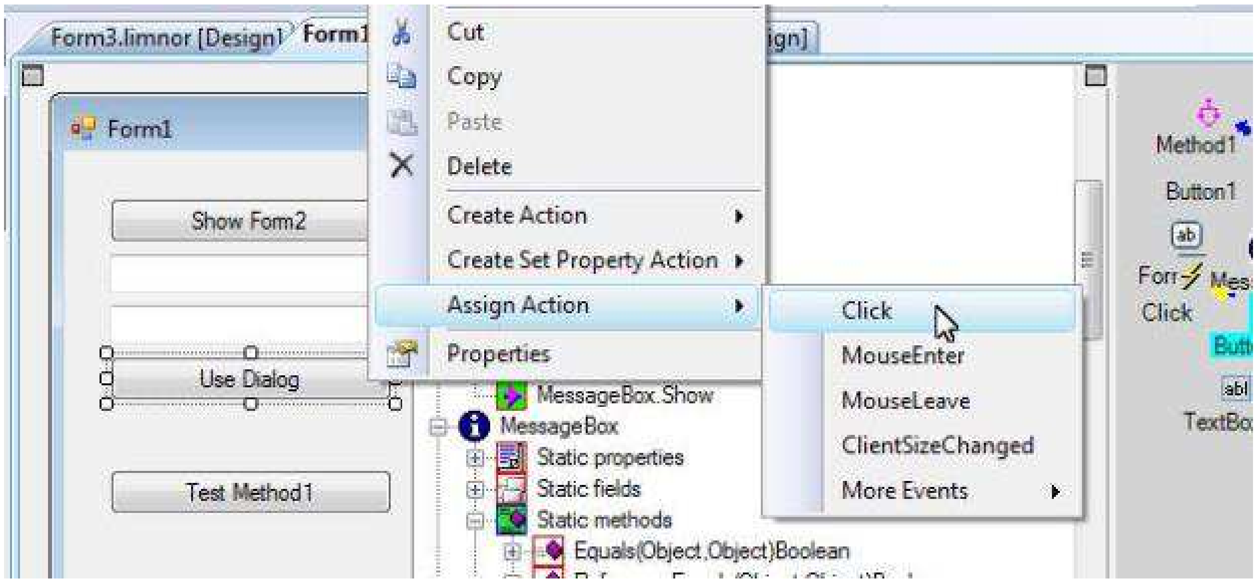


(ActionName)	MessageBox.Show
Description	
ReturnValueType	DialogResult
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	DefForm3.DialogResult = Cancel
ActionMethod	MessageBox.Show(IWin32Window,String)DialogResult
owner	Form1
text	The user canceled it
AssignTo	{null}

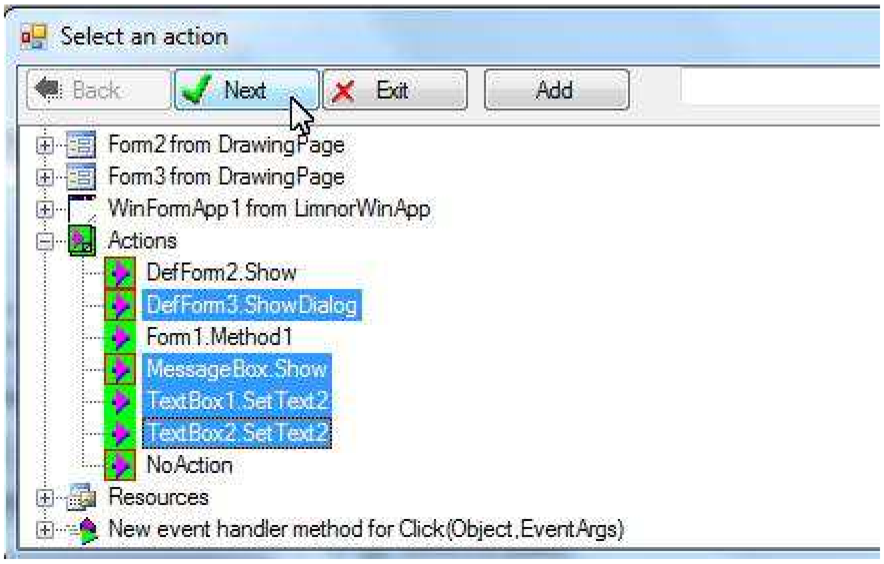
ActionCondition
Condition for executing this action

Use the dialogue actions

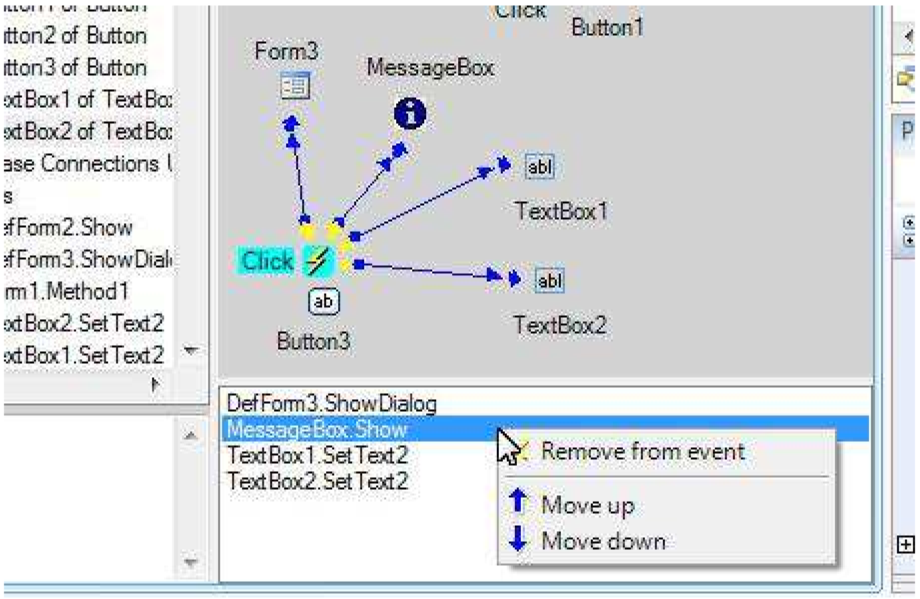
Assign the actions to a button:



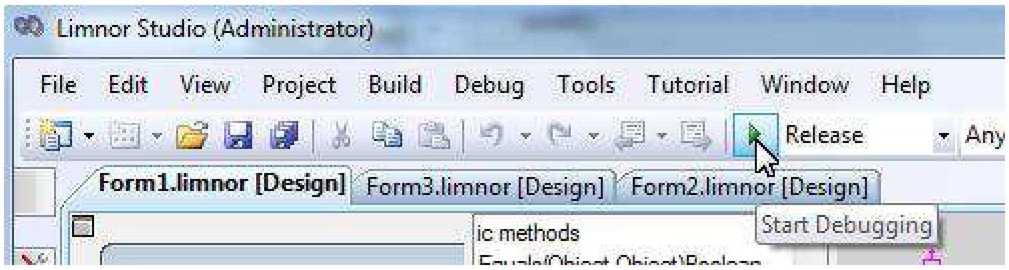
Select the actions for using the dialogue box:



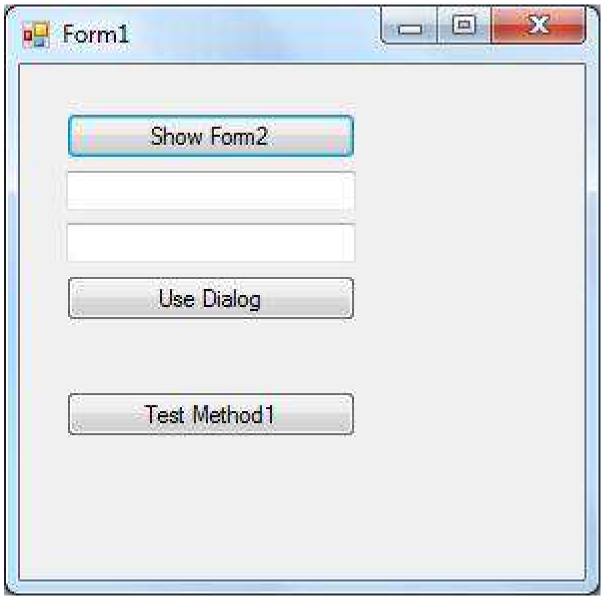
The action DefForm3.ShowDialog must be the first action assigned to the event. The order of an action in the assigned action list can be changed by selecting the action and choose “Move up” or “Move down”:



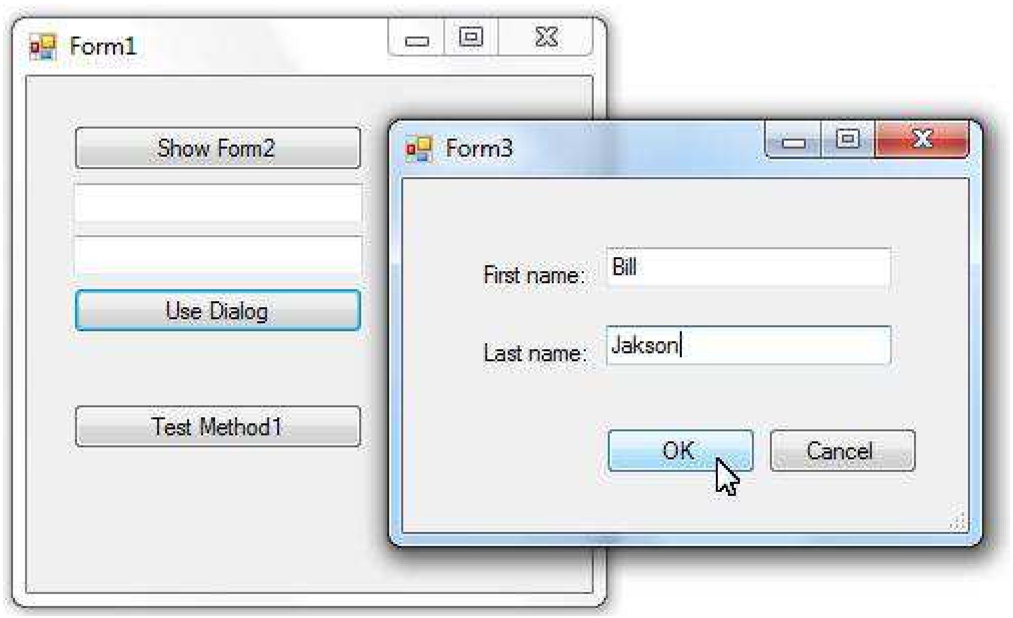
We may test the application now



Form1 appears:



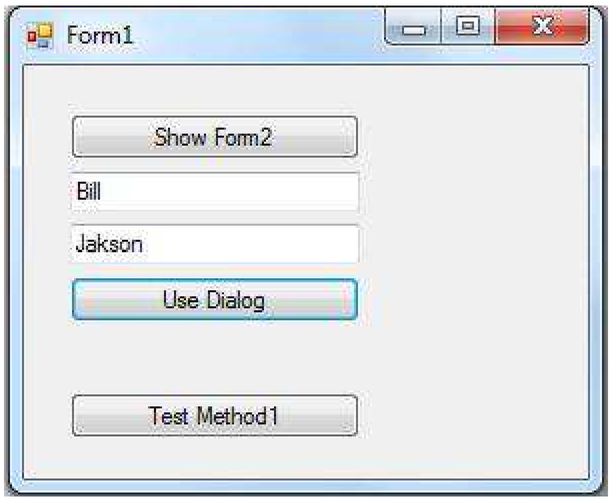
Click button “Use Dialog”, Form3 appears. Enter some data; click OK:



First name: Bill

Last name: Jakson

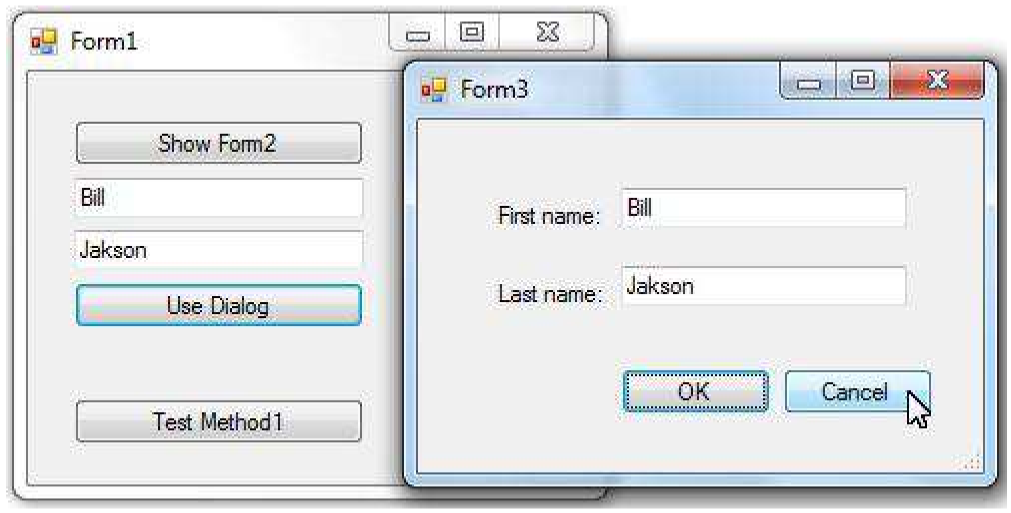
The data entered appear in Form1:



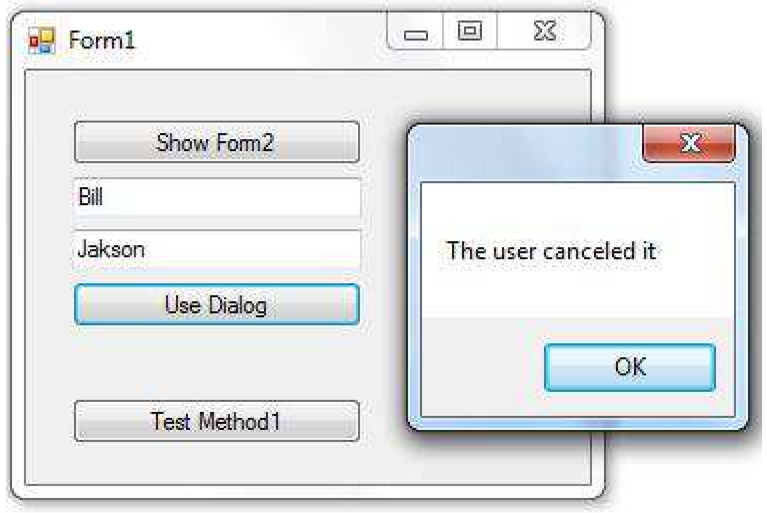
Bill

Jakson

Click button “Use Dialog”. Form3 appears again. Click “Cancel” button:



A message box appears:



The user canceled it

Create Data Validation

In many cases we do not want invalid data to be passed to the data consuming components. In our example, we will add data validation in Form3.

Insert data validation to dialogue box

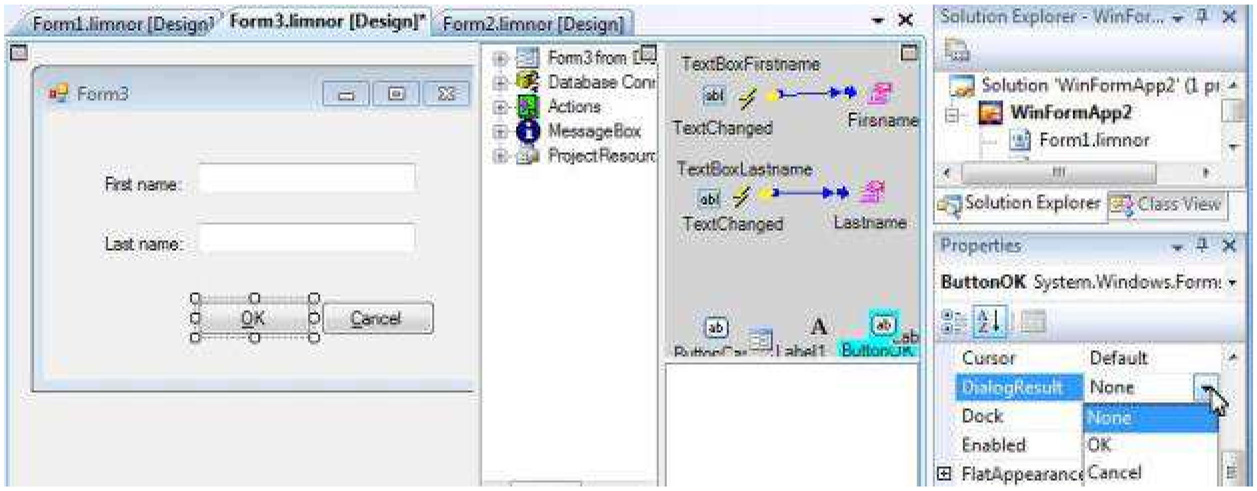
There are many ways for inserting data validations. For example, data can be verified at the time the user enters the data. TextBoxNumber control rejects invalid user inputs in this way.

Another way of data validation is to verify data when the user finishes all data entry. This is what we are going to do.

The user finishes data entry by clicking the OK button.

For Form3, it is closed with its DialogResult property being OK when the user clicks the OK button. So, no data validation is possible. This is because the DialogResult property of its OK button is set to OK.

So, to enable dialogue box level data validation, the DialogResult property of the OK button must be set to None.



Now, when the user clicks the OK button, nothing will happen. To close the dialogue, we may create a “set property” action to set the DialogResult property of Form3 to OK and assign it to the OK button. Data validations can be added as we show below.

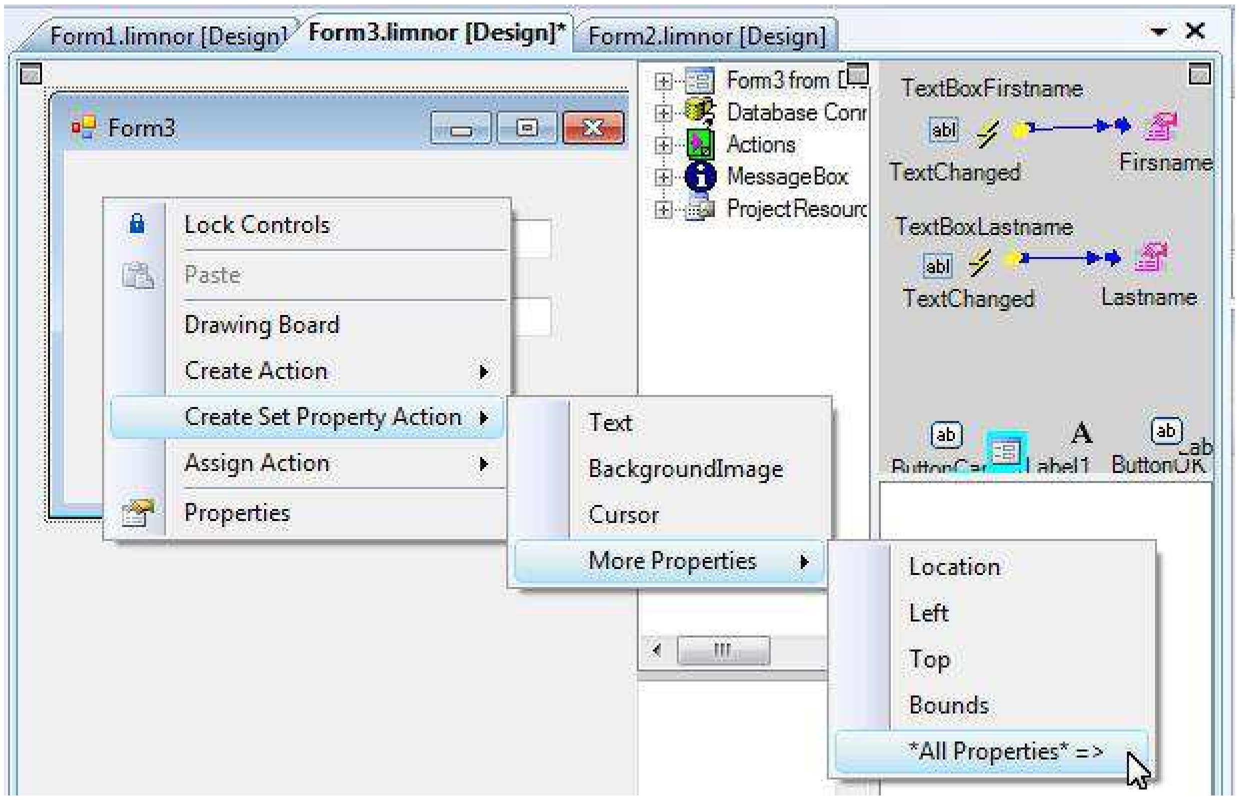
Simple data validation

In this sample, suppose we define valid data as that the contents in the two text boxes are not empty.

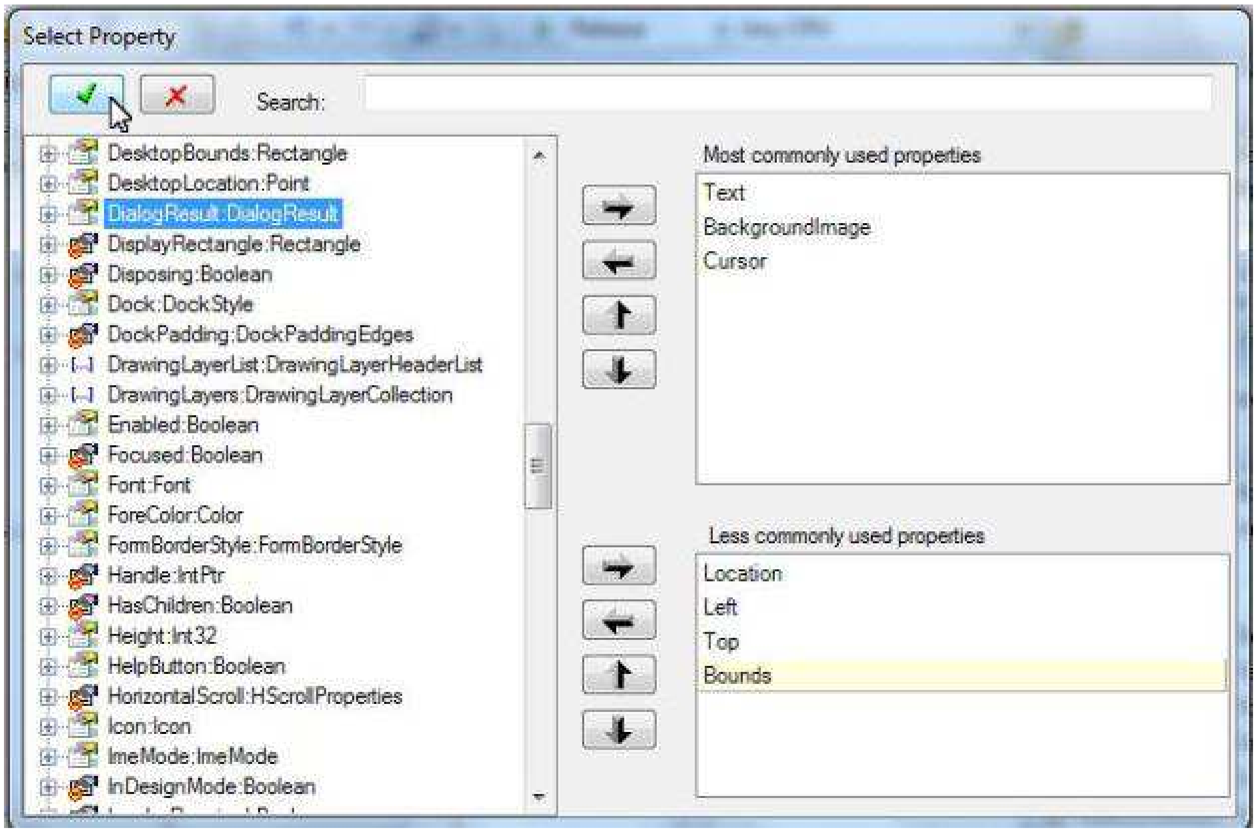
It can be expressed by the Length of the Text property being larger than 0.

We may create a “set property” action to set the DialogResult property of Form3 to OK and set its ActionCondition to verify the data.

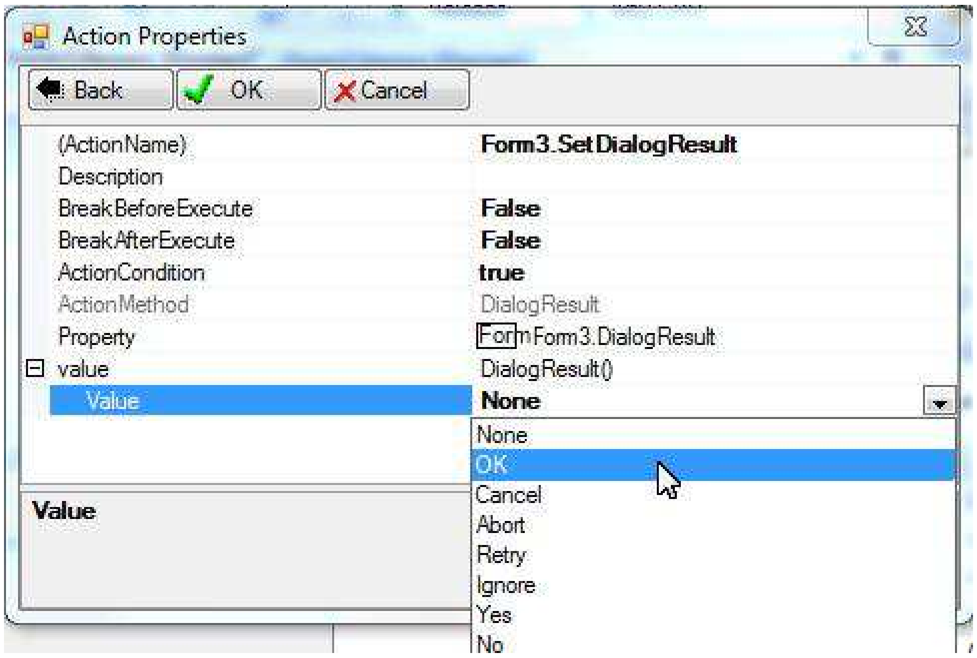
Right-click Form3, choose “Create Set Property Action”. Choose “More Properties”. Choose “All Properties”



Choose “DialogResult”:

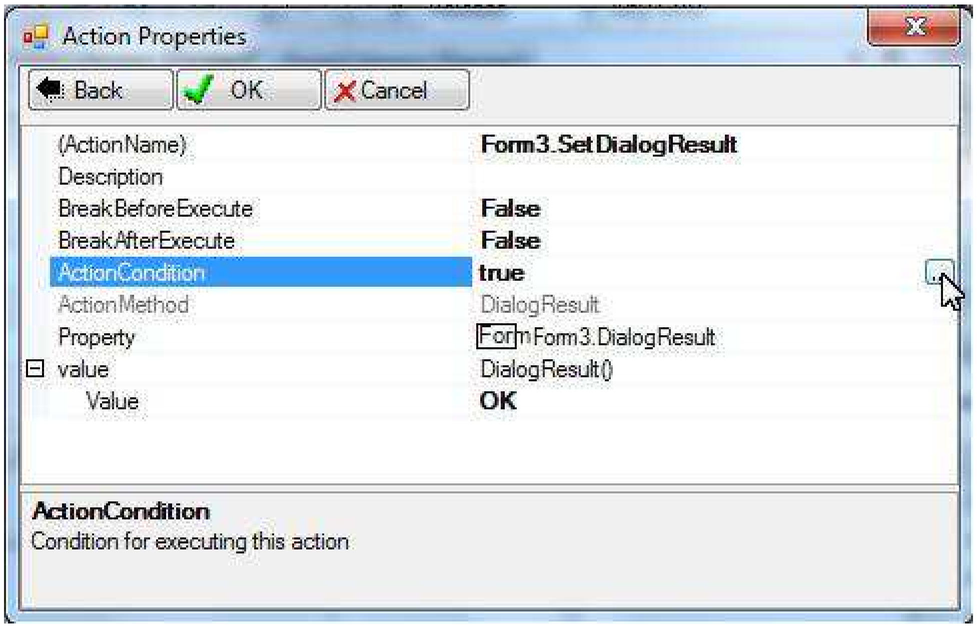


Set the value to OK:



(ActionName)	Form3.SetDialogResult
Description	
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	DialogResult
Property	Form Form3.DialogResult
value	DialogResult()
Value	None

Set ActionCondition to verify data:

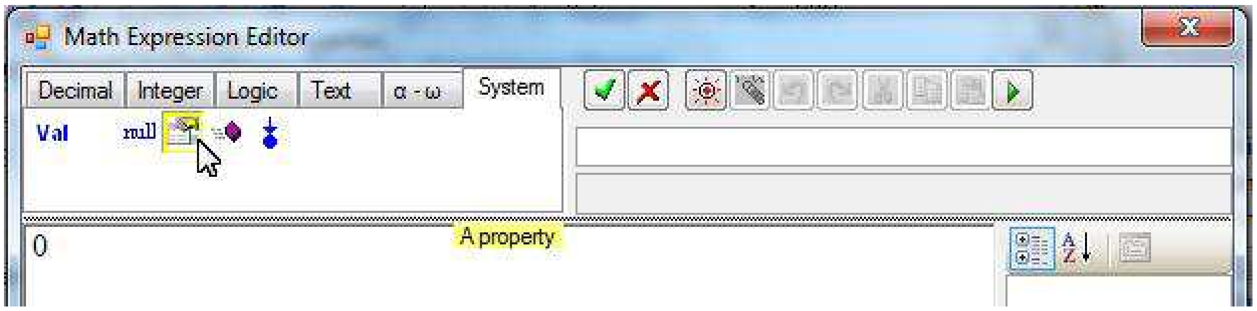


Action Properties

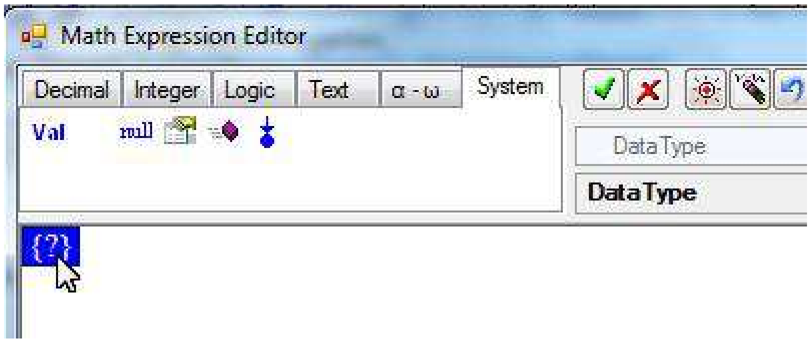
(ActionName)	Form3.SetDialogResult
Description	
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	DialogResult
Property	Form Form3.DialogResult
value	DialogResult()
Value	OK

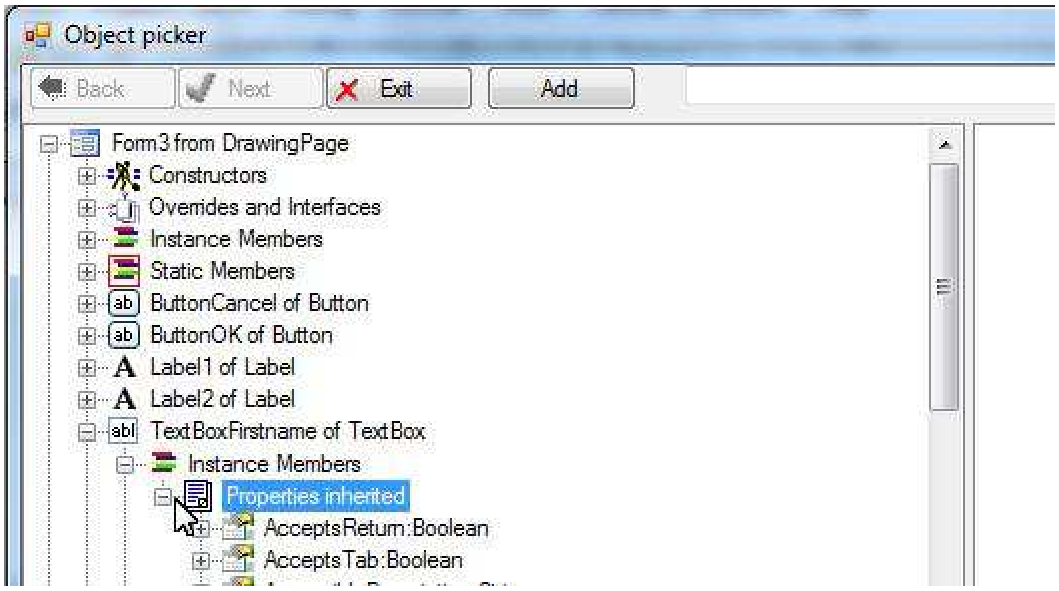
ActionCondition
Condition for executing this action

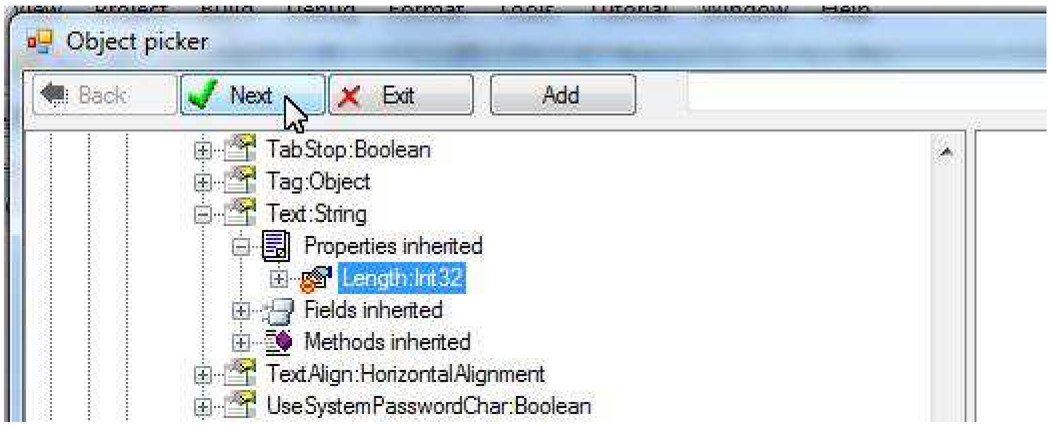
Click the Property icon:



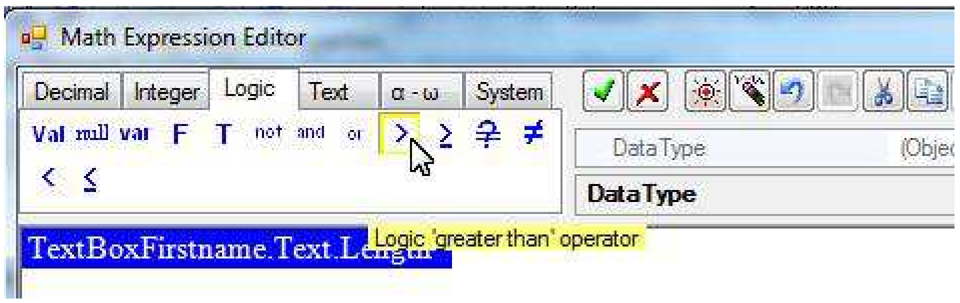
Double-click the property element to select property:



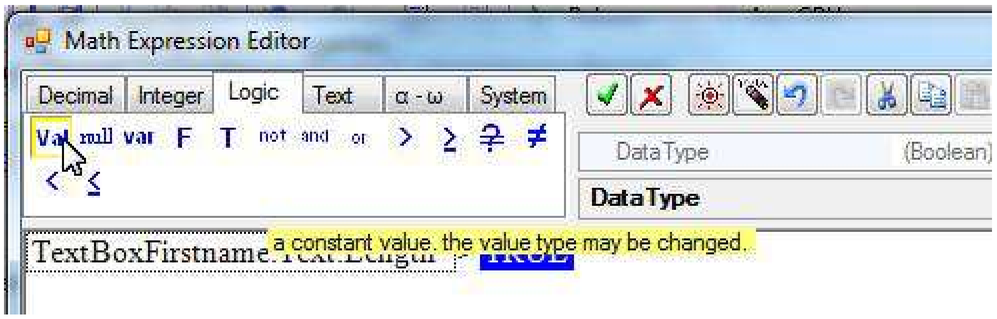




Click the "greater than" icon:


Math Expression Editor
Decimal
Integer
Logic
Text
α - ω
System
Val null var F T not and or > ≥ ≠
< ≤
DataType
(Objec
DataType
TextBoxFirstname.Text.Length
Logic 'greater than' operator

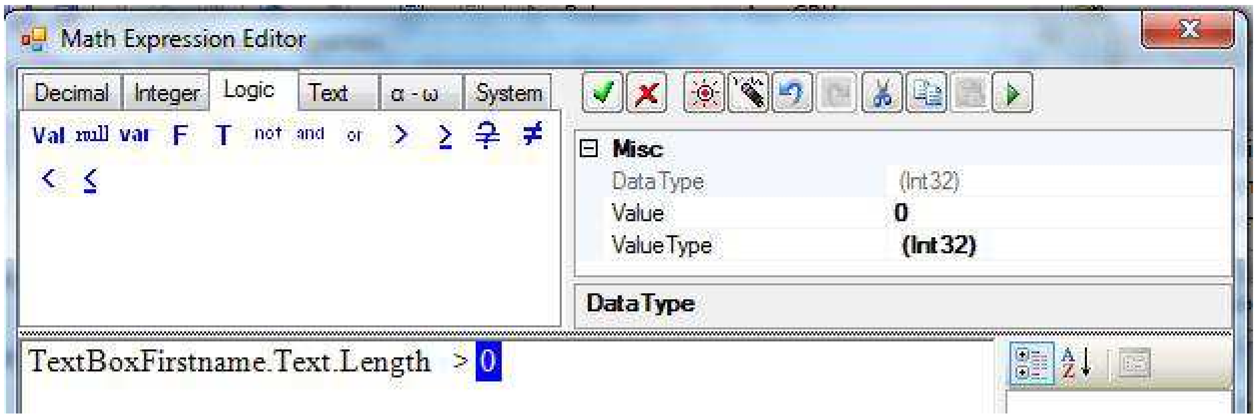
Select the second element and click the "constant" icon:



TextBoxFirstname.Text.Length > TRUE

a constant value. the value type may be changed.

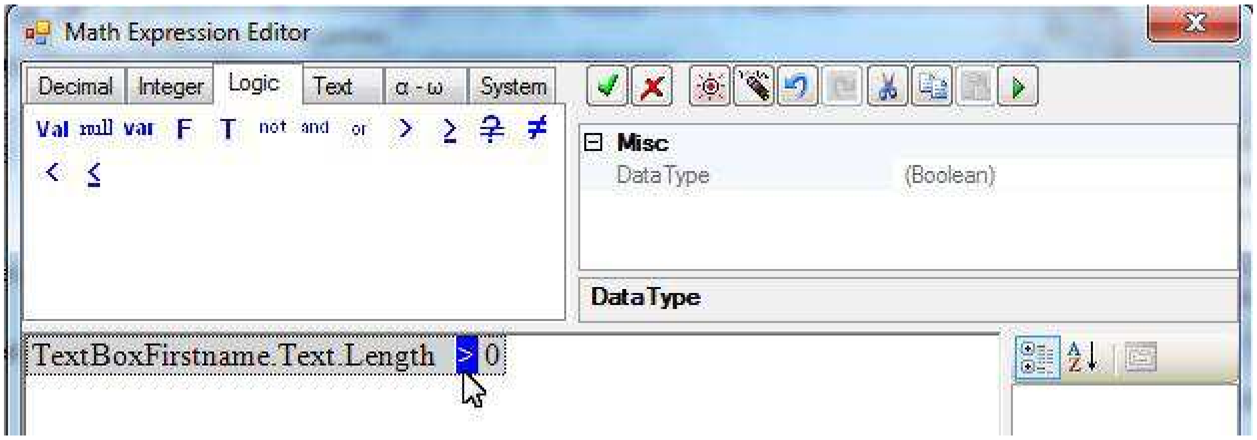
It gives us an integer 0 as the comparing constant. That is what we want. So, no more changes to the constant value are needed:



Data Type (Int32)
Value 0
Value Type (Int32)

TextBoxFirstname.Text.Length > 0

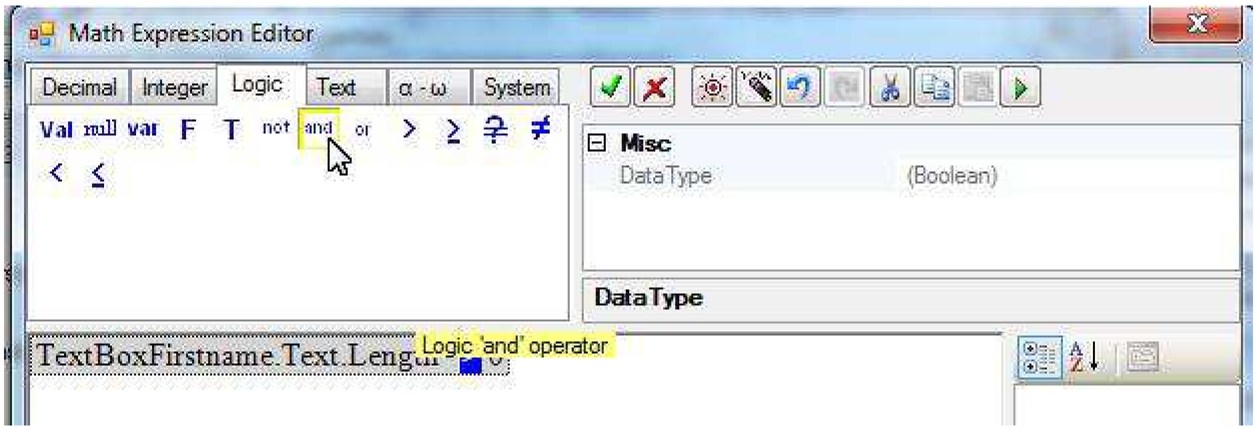
Select the “greater than” element:



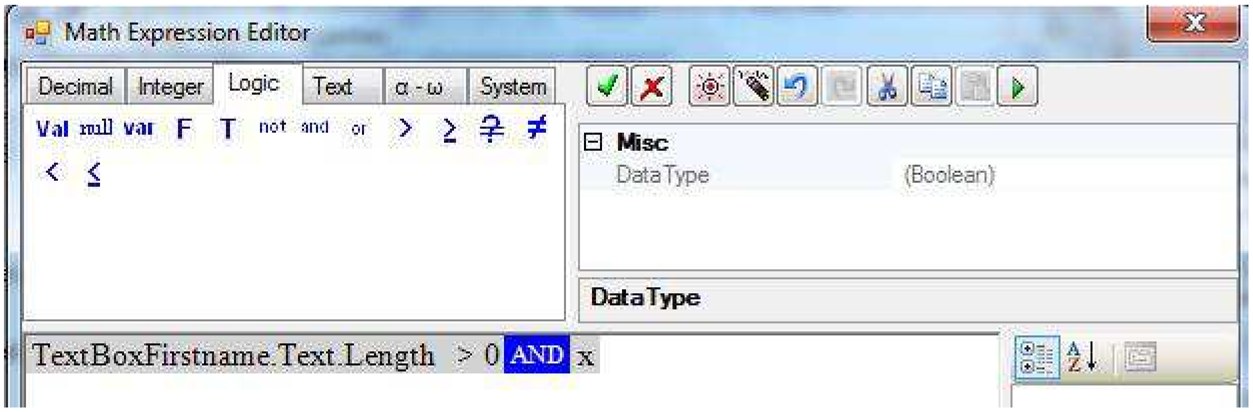
Data Type (Boolean)

TextBoxFirstname.Text.Length > 0

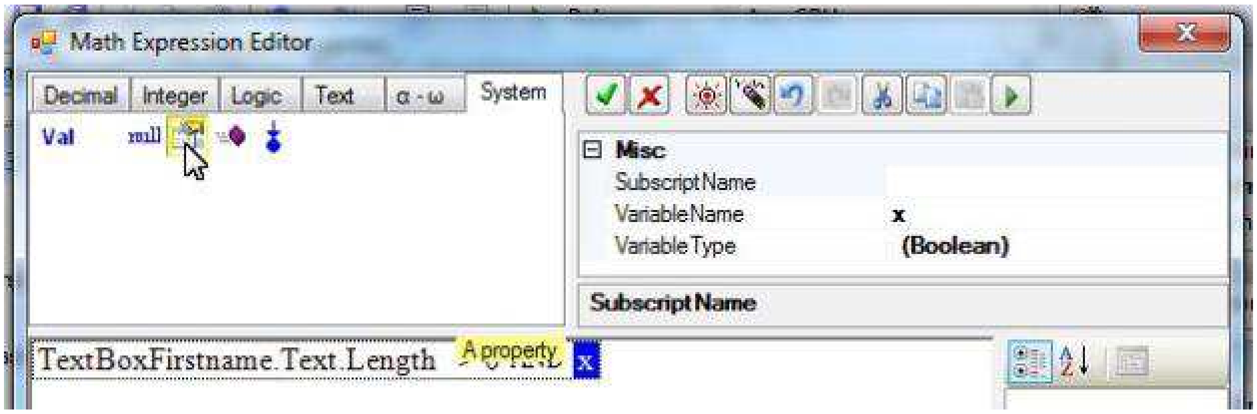
Click the logic “and” icon:



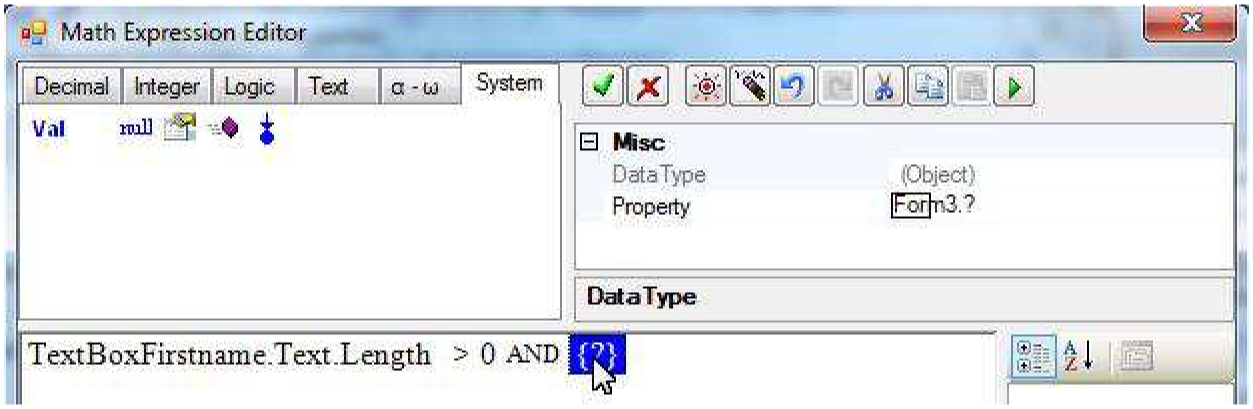
A new “AND” expression is added:



Select the new element “x” and click the Property icon:

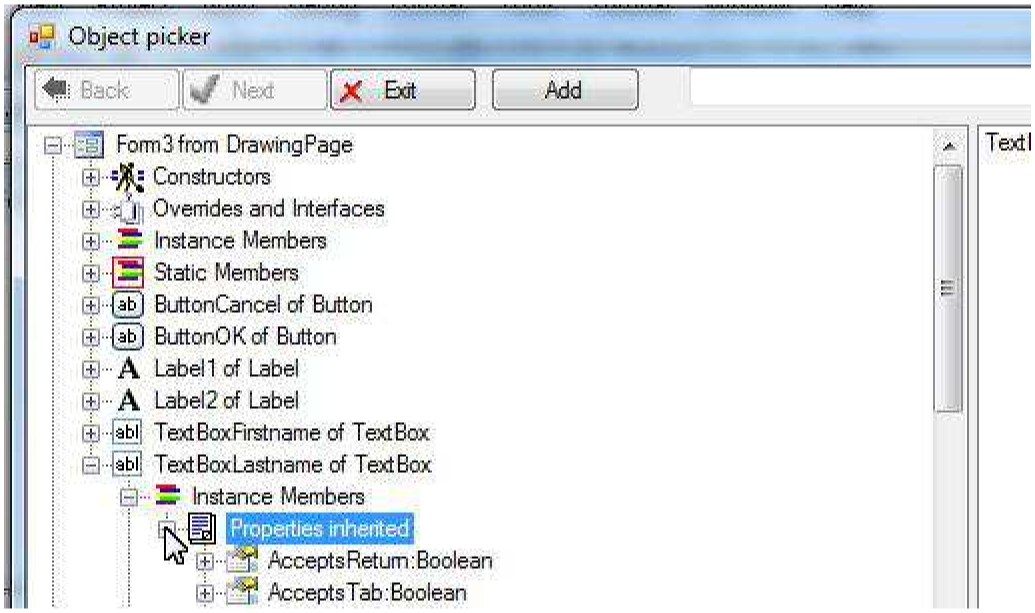


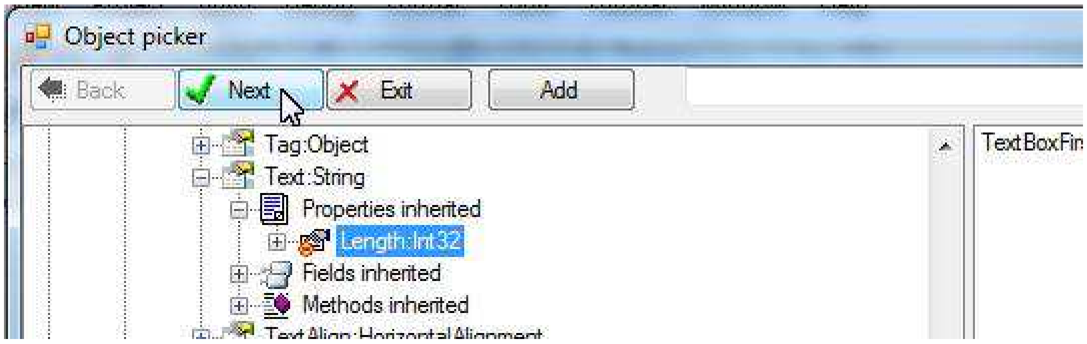
Double-click the new Property element to select property:



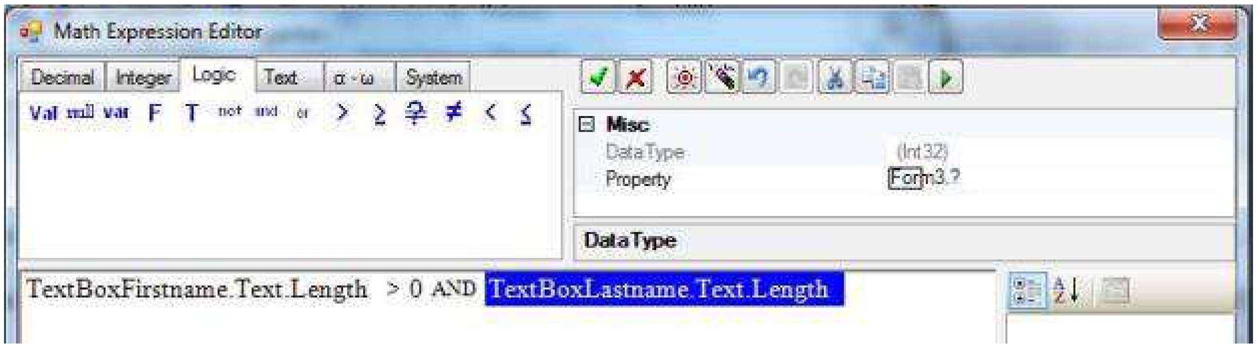
TextBoxFirstname.Text.Length > 0 AND {?}

Select the text length property of the Lastname text box:



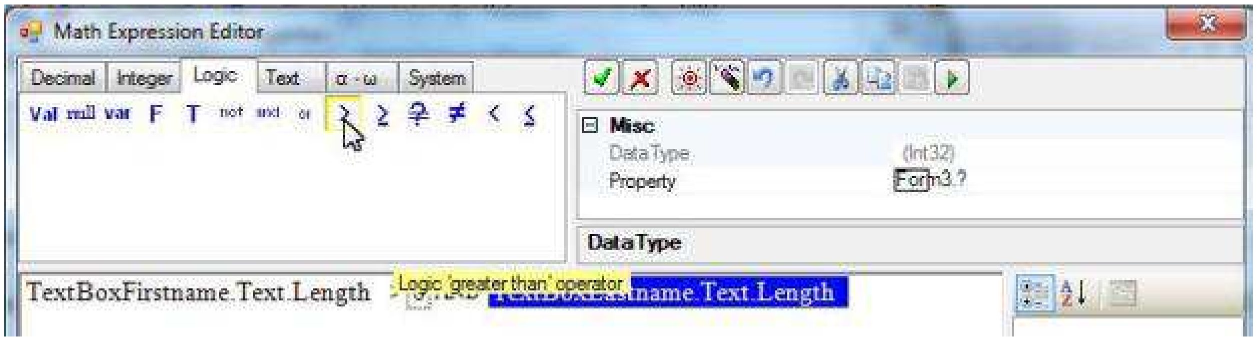


The selected property appears in the expression:

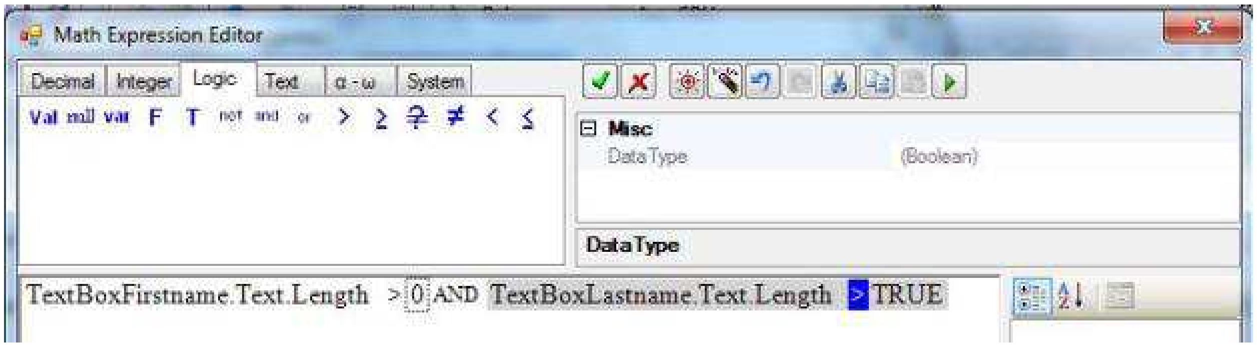


```
TextBoxFirstname.Text.Length > 0 AND TextBoxLastname.Text.Length
```

Click the “greater than” icon:

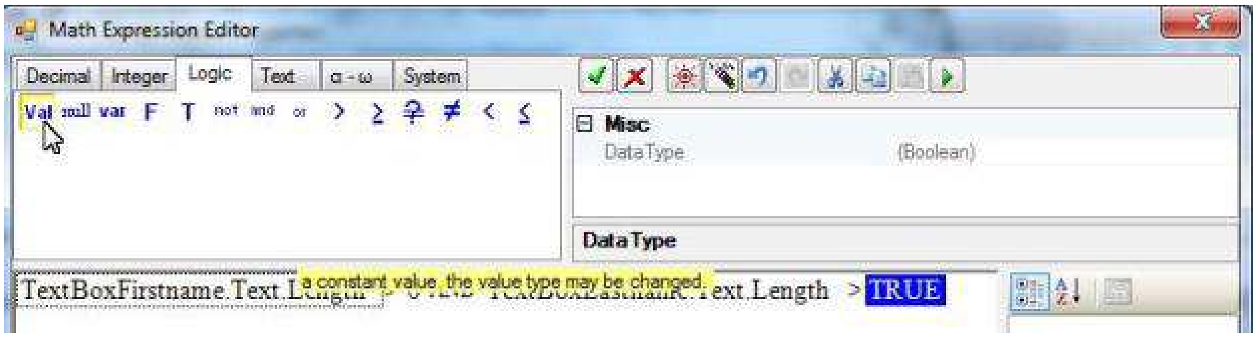


A new “greater than” is added:

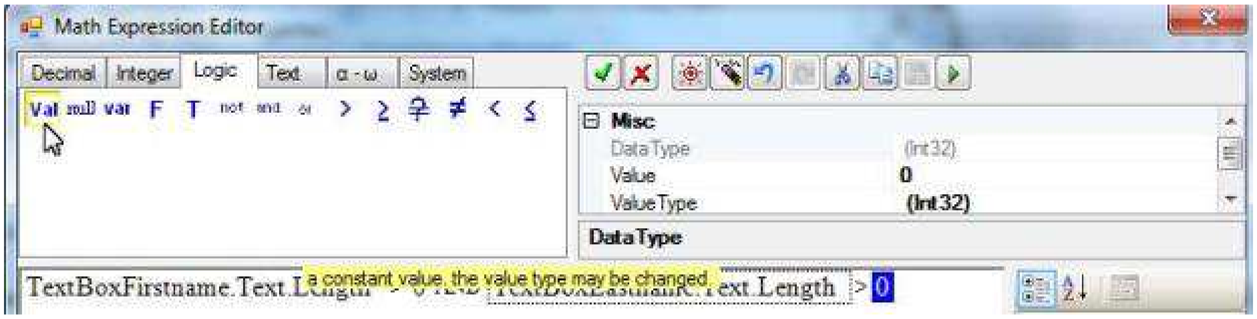


```
TextBoxFirstname.Text.Length > 0 AND TextBoxLastname.Text.Length > TRUE
```

Select the second element of the “greater than”, click the “constant” icon:



It gives us an integer 0 as the comparing constant. That is what we want. So, no more changes to the constant value are needed:

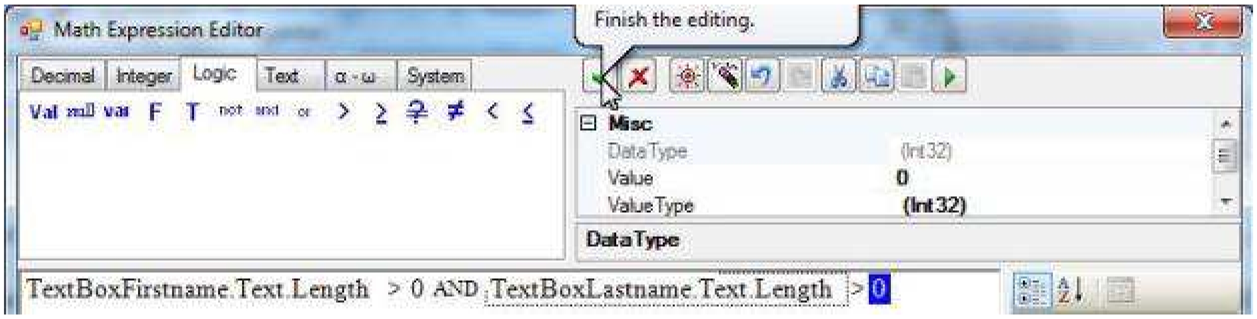


DataType	(Int32)
Value	0
ValueType	(Int32)

TextBoxFirstname.Text.Length > 0 AND TextBoxLastname.Text.Length > 0

a constant value. the value type may be changed.

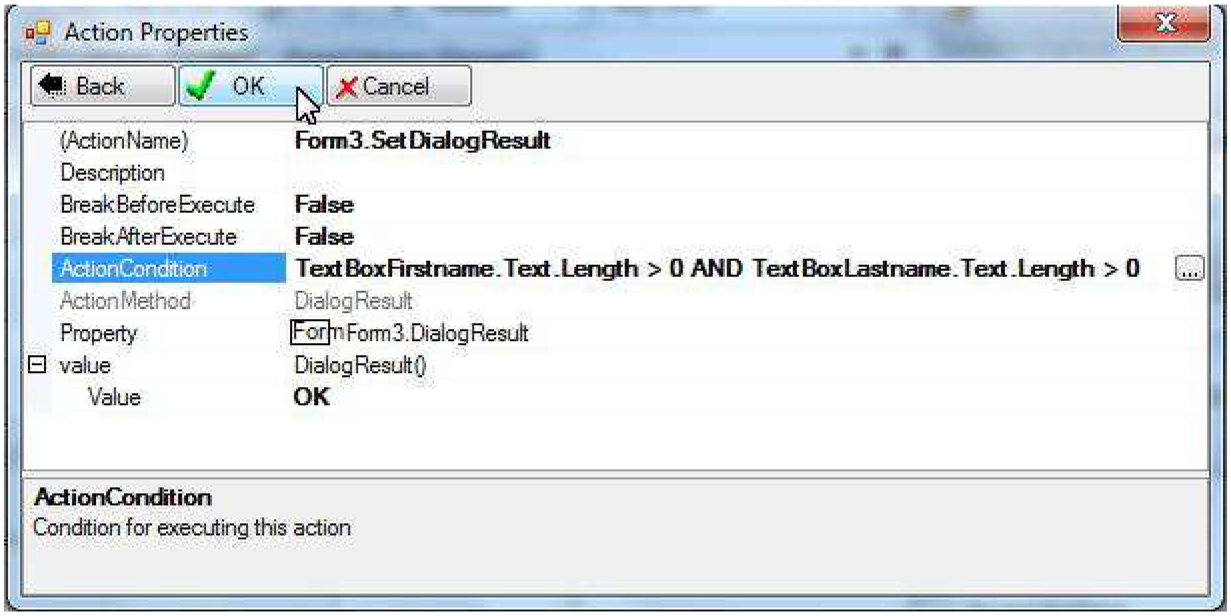
This is the expression for data validation:



DataType	(Int32)
Value	0
ValueType	(Int32)

TextBoxFirstname.Text.Length > 0 AND TextBoxLastname.Text.Length > 0

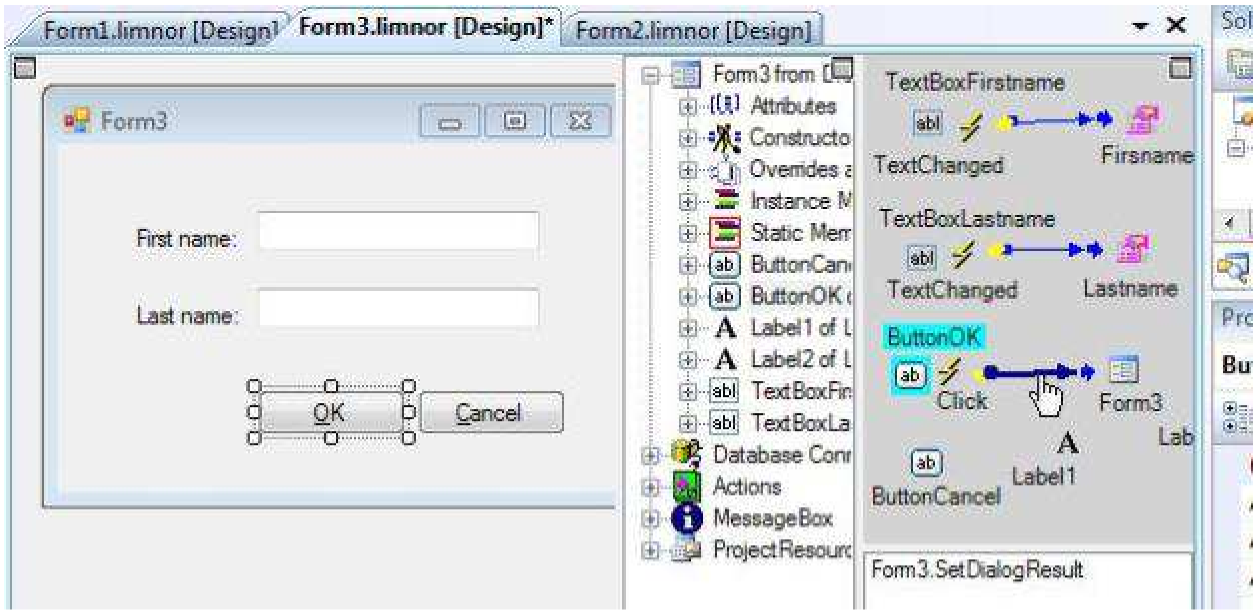
This action sets the DialogResult property of Form3 to OK. Doing so will also close Form3. This action only executes when both text boxes are not empty:



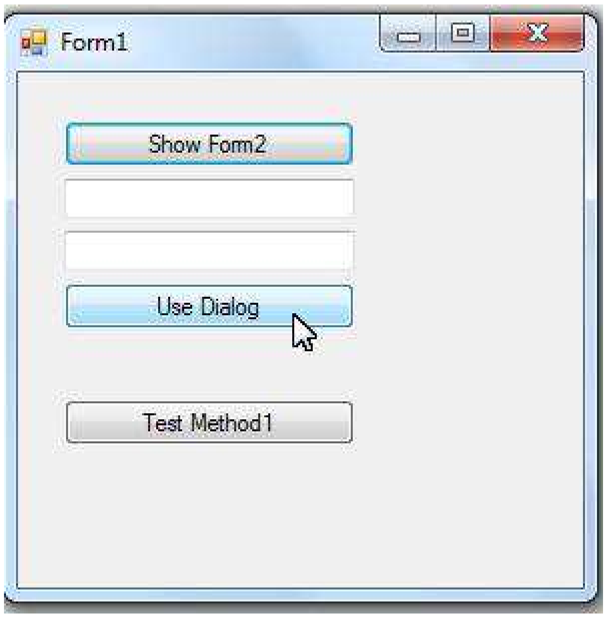
(ActionName)	Form3.SetDialogResult
Description	
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	TextBoxFirstname.Text.Length > 0 AND TextBoxLastname.Text.Length > 0
ActionMethod	DialogResult
Property	Form Form3.DialogResult
value	DialogResult()
Value	OK

ActionCondition
Condition for executing this action

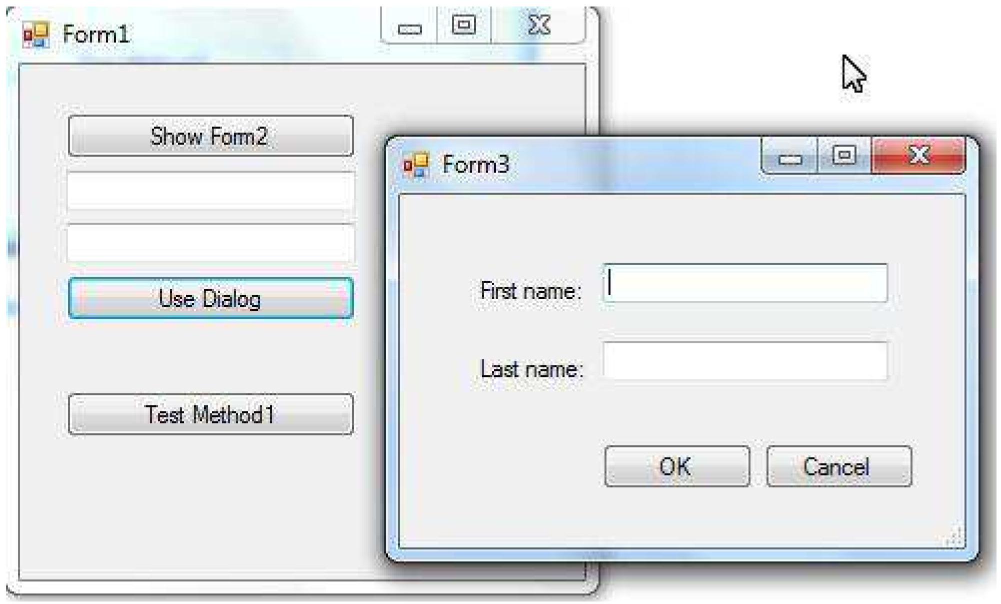
Assign this action to the OK button:



That is all for this simple data validation. We may test it now.



Click button “Use Dialog”. Form3 appears:

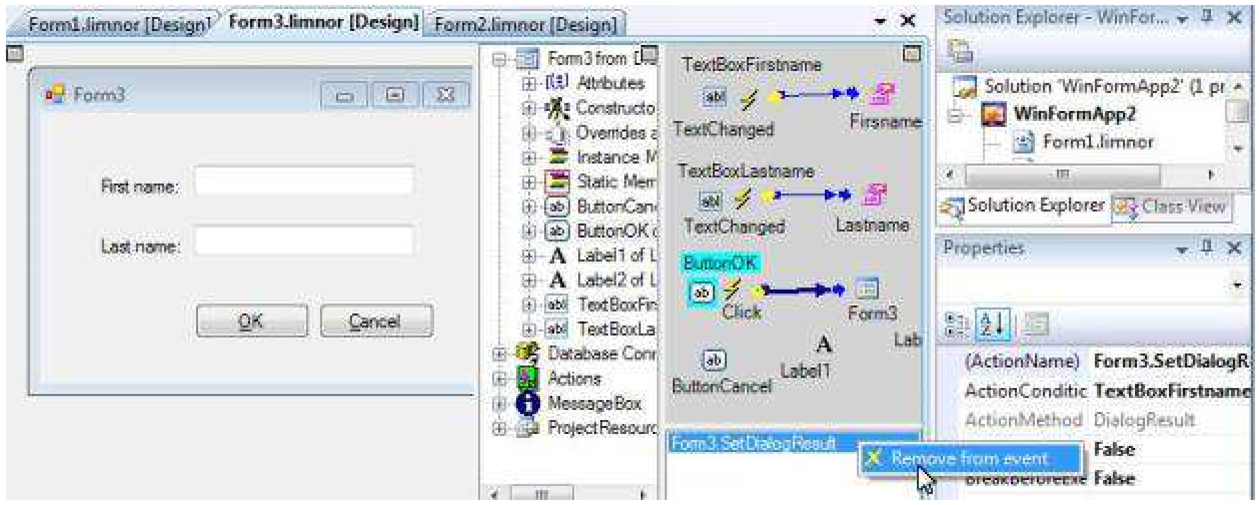


We can see that if any text box is empty then nothing will happen by clicking the OK button.

More complex data validation

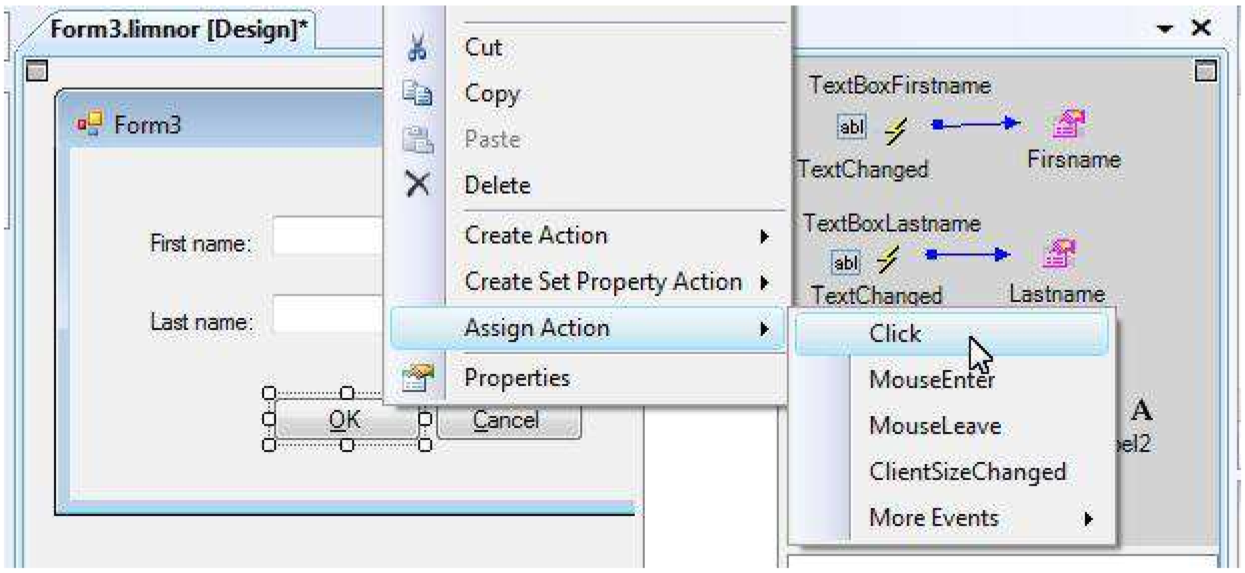
More complex data validation usually requires using more actions. It is easier to be done in a method.

First, let’s remove the action assigned to the OK button:

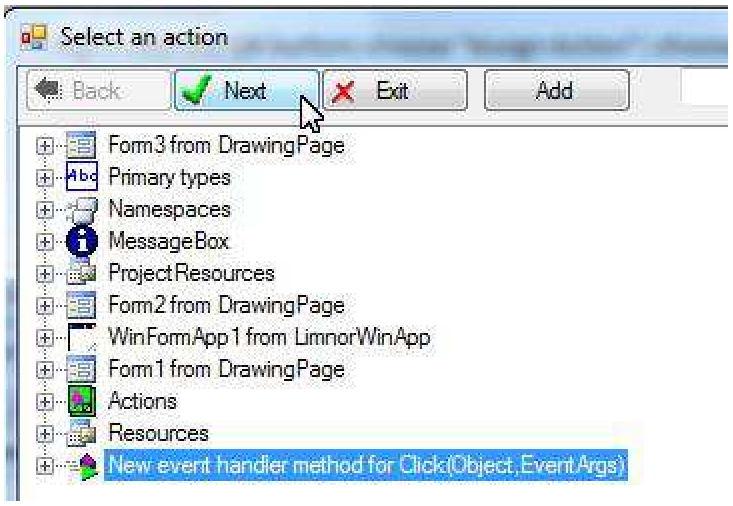


We can create a method to do data validation. Use the method to create an action and assign the action to the OK button. But since we only need to do the validation by the OK button, we may use an event handler method of the OK button to do it. It saves us the operations of creating action and assigning action.

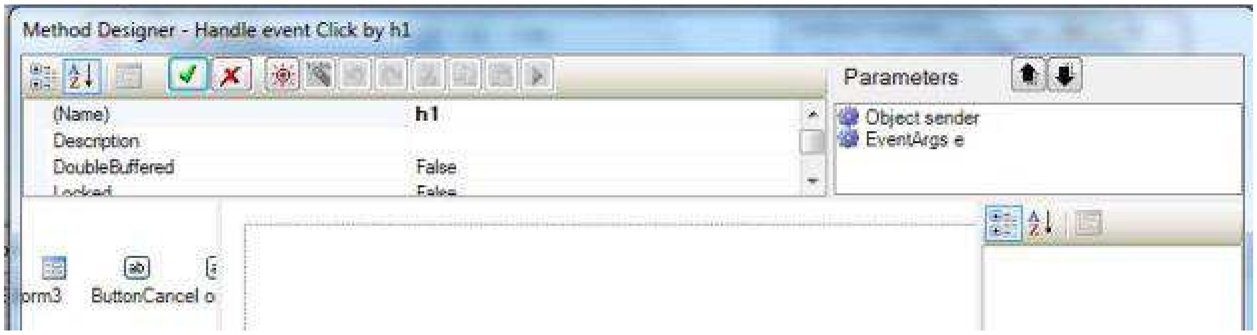
Right-click the OK button; choose “Assign Action”; choose “Click” event:



Instead of selecting actions, select “New event handler method for Click”

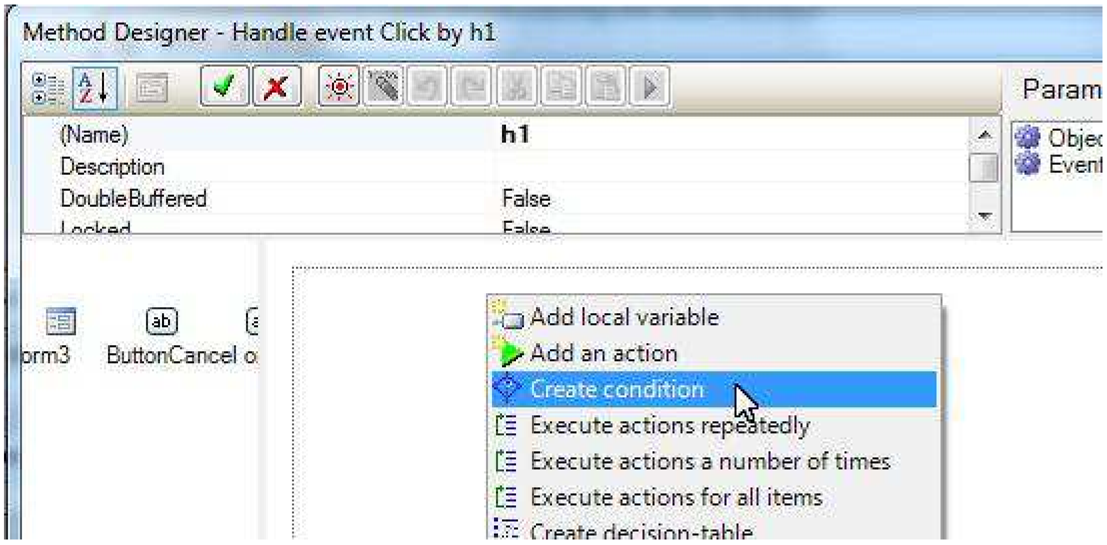


The Method Editor appears for editing this new method:

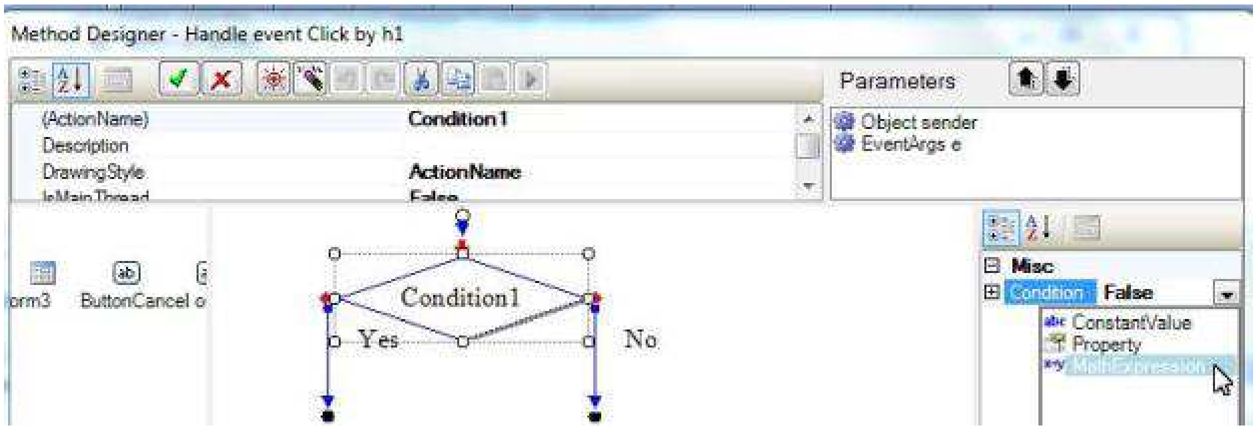


Suppose we want the data validation process works in such a way: if the Firstname text box is empty then show a message box; else if the Lastname text box is empty then show another message box; else set the DialogResult of Form3 to OK

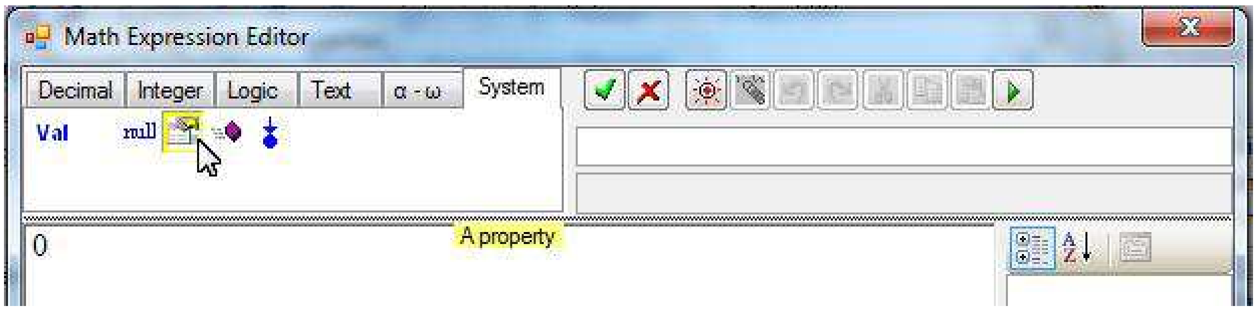
Right-click the Action Pane; choose “Create condition”:



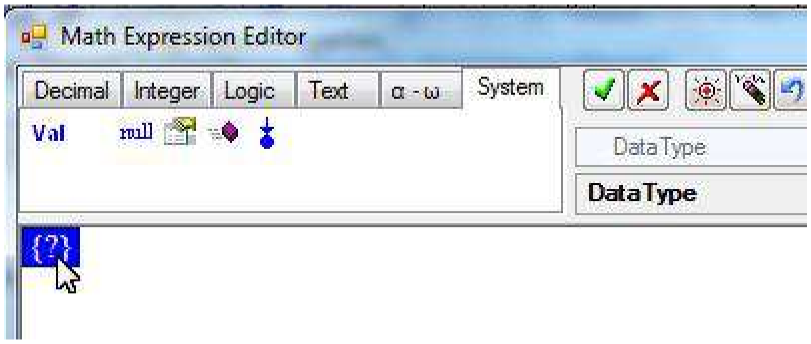
Set the Condition to an expression:

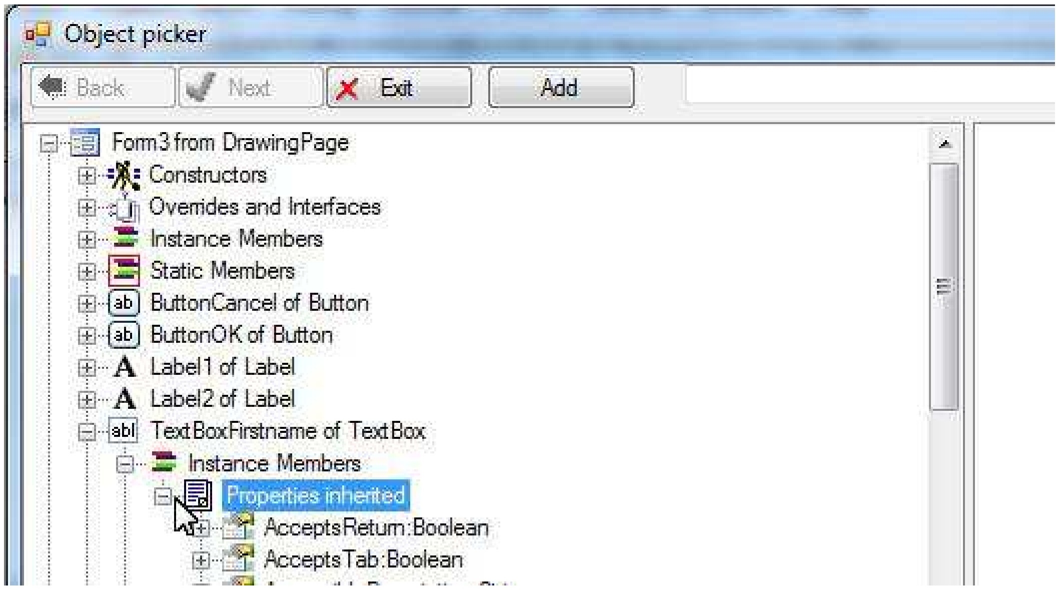


Click the Property icon:

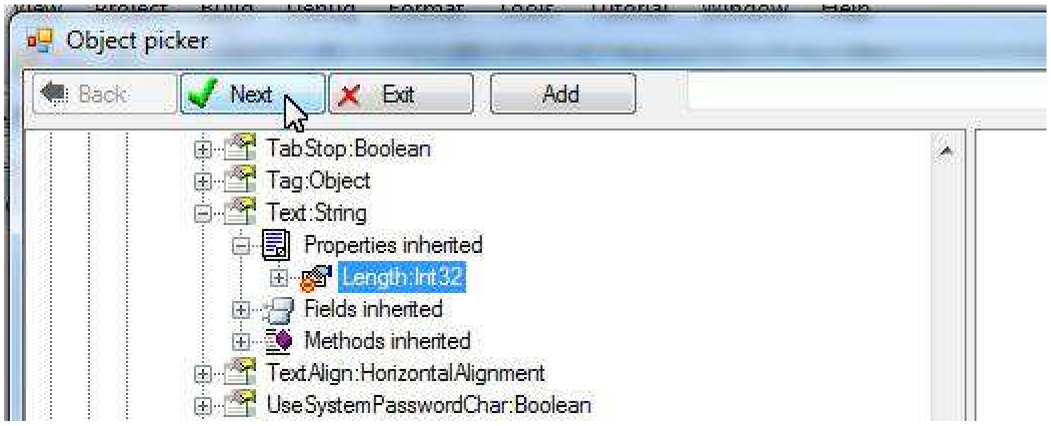


Double-click the property element to select property:



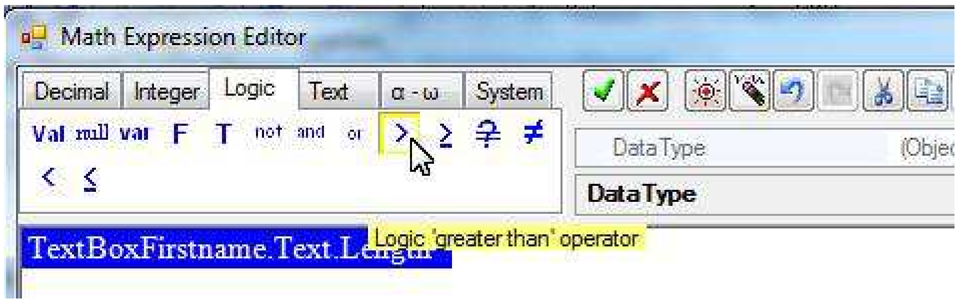


Form3 from DrawingPage
Constructors
Overrides and Interfaces
Instance Members
Static Members
ButtonCancel of Button
ButtonOK of Button
Label1 of Label
Label2 of Label
TextBoxFirstname of TextBox
Instance Members
Properties inherited
AcceptsReturn:Boolean
AcceptsTab:Boolean



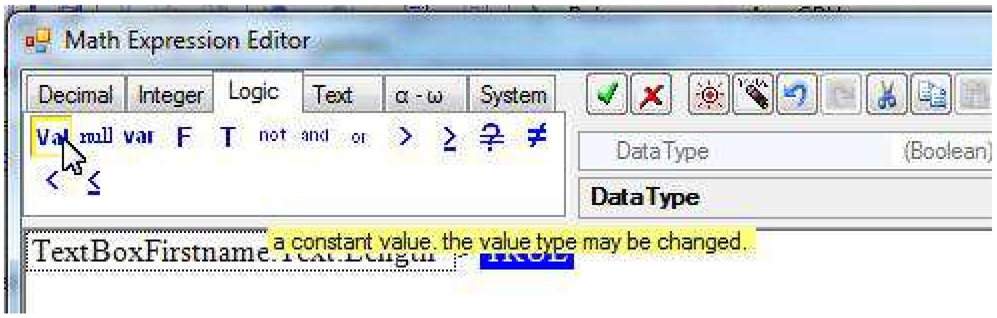
TabStop:Boolean
Tag:Object
Text:String
Properties inherited
Length:Int32
Fields inherited
Methods inherited
TextAlign:HorizontalAlignment
UseSystemPasswordChar:Boolean

Click the “greater than” icon:



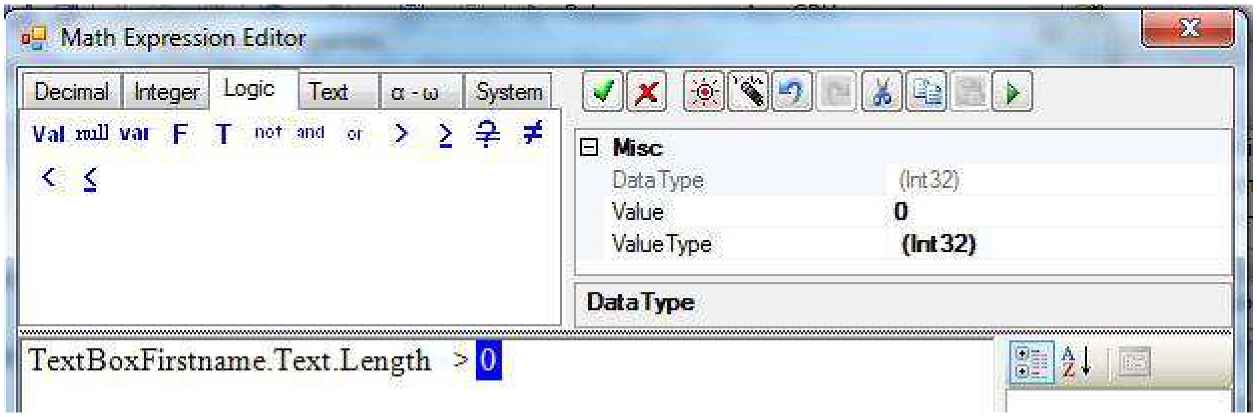
TextBoxFirstname.Text.Length

Select the second element and click the “constant” icon:


Math Expression Editor
Decimal
Integer
Logic
Text
α - ω
System
Val null var F T not and or > ≥ ≠ < ≤
Data Type
(Boolean)
Data Type
a constant value. the value type may be changed.

TextBoxFirstname.Text.Length > TRUE

It gives us an integer 0 as the comparing constant. That is what we want. So, no more changes to the constant value are needed:



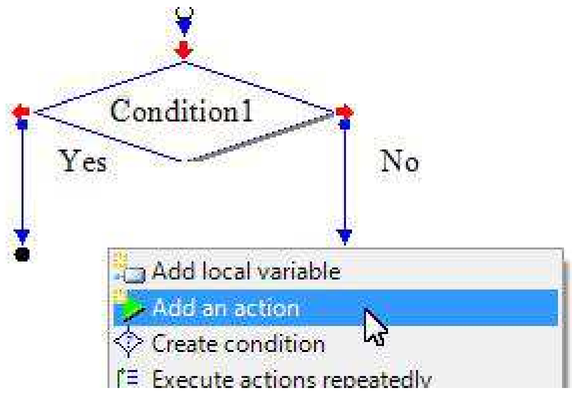
Misc

Data Type	(Int32)
Value	0
Value Type	(Int32)

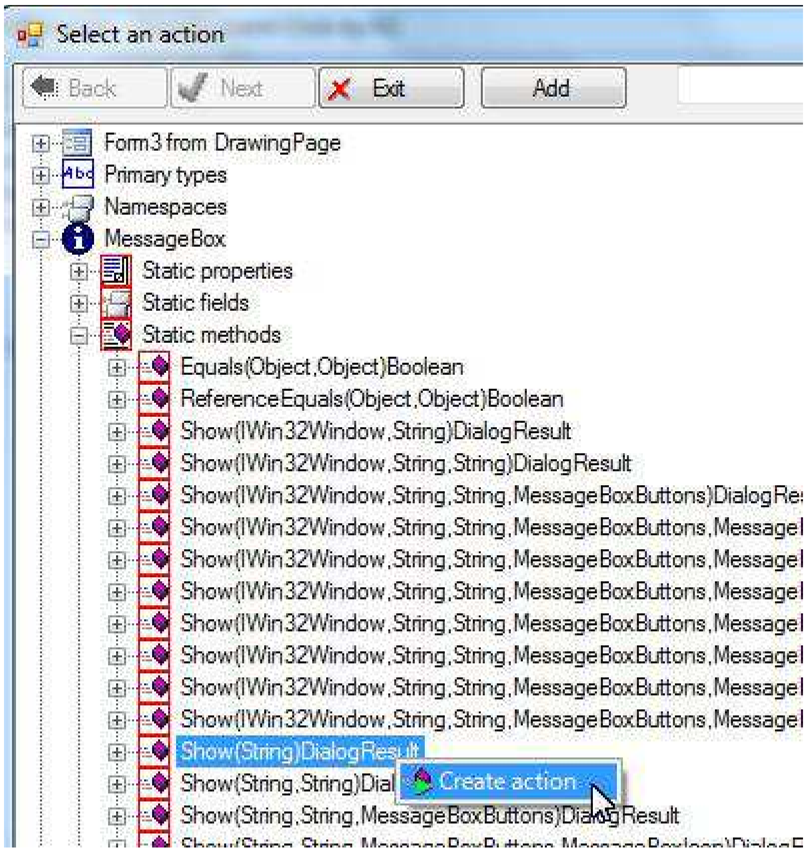
Data Type

TextBoxFirstname.Text.Length > 0

To add an action to display a message box, right-click the Method Pane, choose “Add an action”:

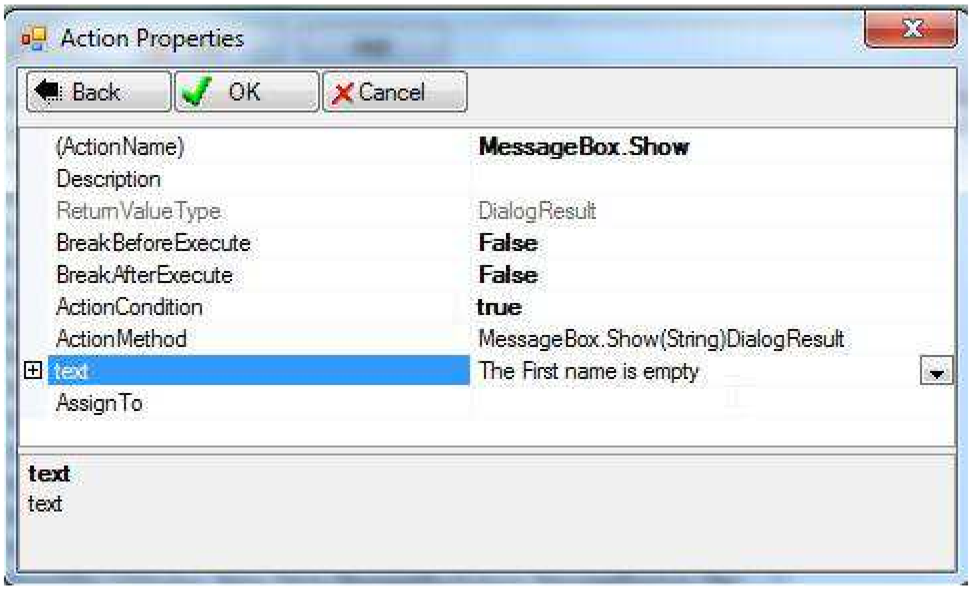


Right-click a Show method, choose “Create action”:



Form3 from DrawingPage
Primary types
Namespaces
MessageBox
Static properties
Static fields
Static methods
Equals(Object,Object)Boolean
ReferenceEquals(Object,Object)Boolean
Show(IWin32Window,String)DialogResult
Show(IWin32Window,String,String)DialogResult
Show(IWin32Window,String,String,MessageBoxButtons)DialogRe
Show(IWin32Window,String,String,MessageBoxButtons,Message
Show(IWin32Window,String,String,MessageBoxButtons,Message
Show(IWin32Window,String,String,MessageBoxButtons,Message
Show(IWin32Window,String,String,MessageBoxButtons,Message
Show(IWin32Window,String,String,MessageBoxButtons,Message
Show(IWin32Window,String,String,MessageBoxButtons,Message
Show(IWin32Window,String,String,MessageBoxButtons,Message
Show(String)DialogResult
Show(String,String)Dial
Show(String,String,MessageBoxButtons)DialogResult

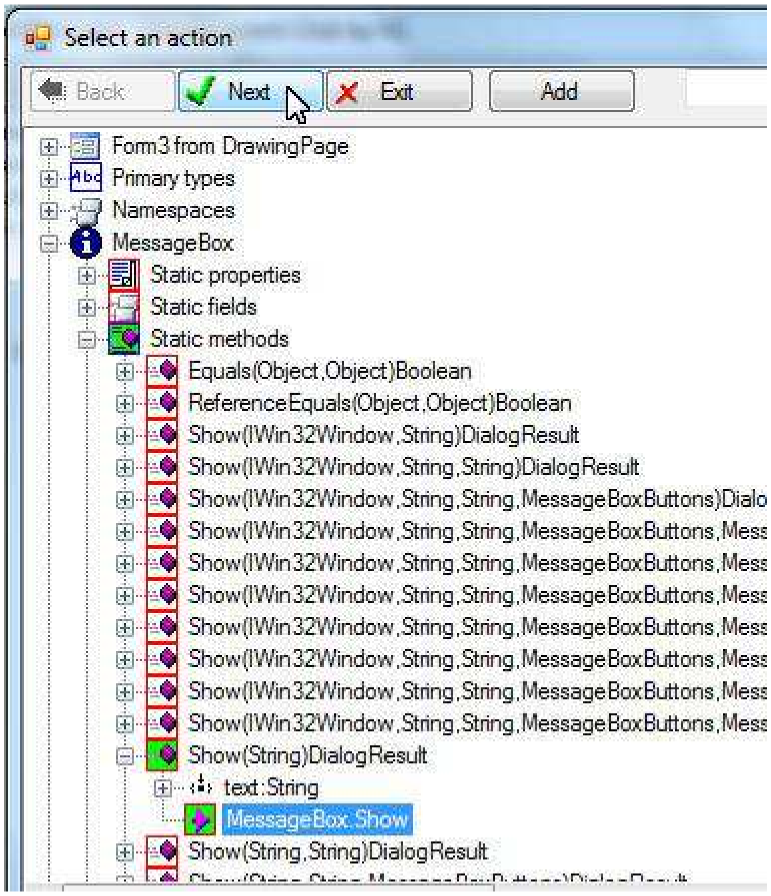
Give a message to the action:



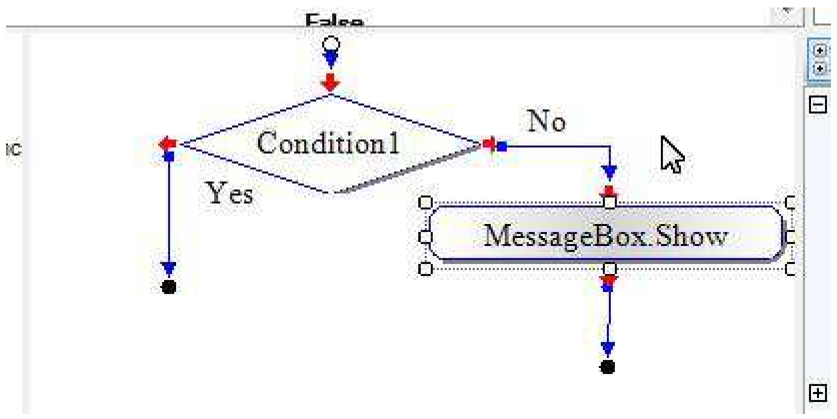
(ActionName)	**MessageBox.Show**
Description	
ReturnValueType	DialogResult
BreakBeforeExecute	**False**
BreakAfterExecute	**False**
ActionCondition	**true**
ActionMethod	MessageBox.Show(String)DialogResult
text	The First name is empty
AssignTo	

text
text

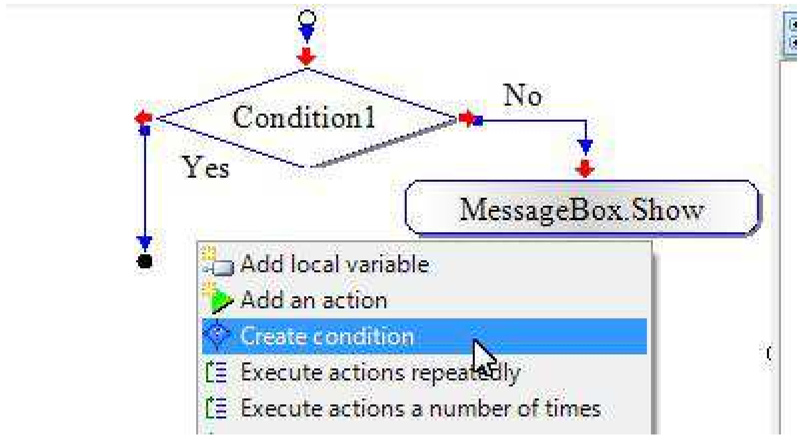
While the new action is selected, click Next:


Select an action
Back
Next
Exit
Add
Form3 from DrawingPage
Primary types
Namespaces
MessageBox
Static properties
Static fields
Static methods
Equals(Object,Object)Boolean
ReferenceEquals(Object,Object)Boolean
Show(IWin32Window,String)DialogResult
Show(IWin32Window,String,String)DialogResult
Show(IWin32Window,String,String,MessageBoxButtons)Dialo
Show(IWin32Window,String,String,MessageBoxButtons,Mess
Show(IWin32Window,String,String,MessageBoxButtons,Mess
Show(IWin32Window,String,String,MessageBoxButtons,Mess
Show(IWin32Window,String,String,MessageBoxButtons,Mess
Show(IWin32Window,String,String,MessageBoxButtons,Mess
Show(IWin32Window,String,String,MessageBoxButtons,Mess
Show(IWin32Window,String,String,MessageBoxButtons,Mess
Show(String)DialogResult
text:String
MessageBox.Show
Show(String,String)DialogResult

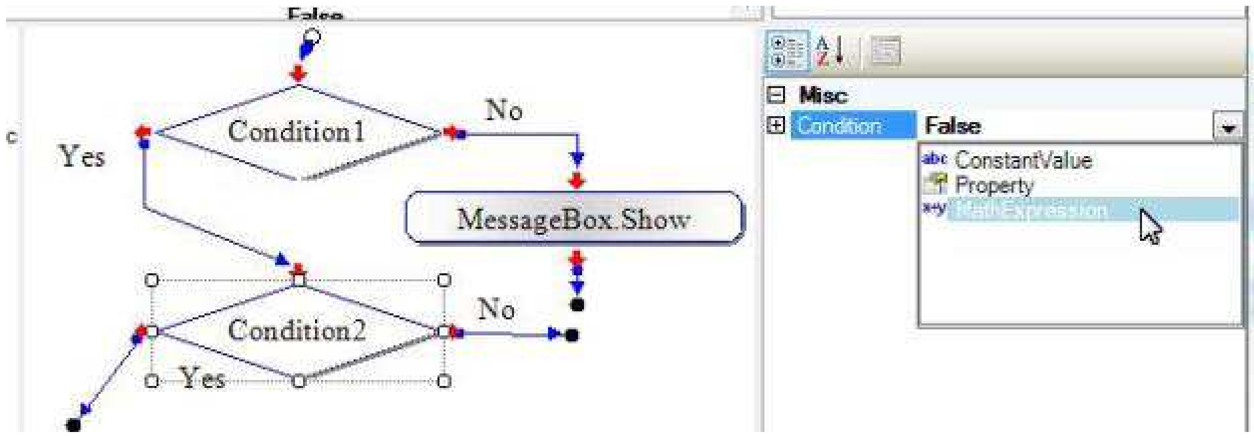
Link the new action to the No port of the condition:



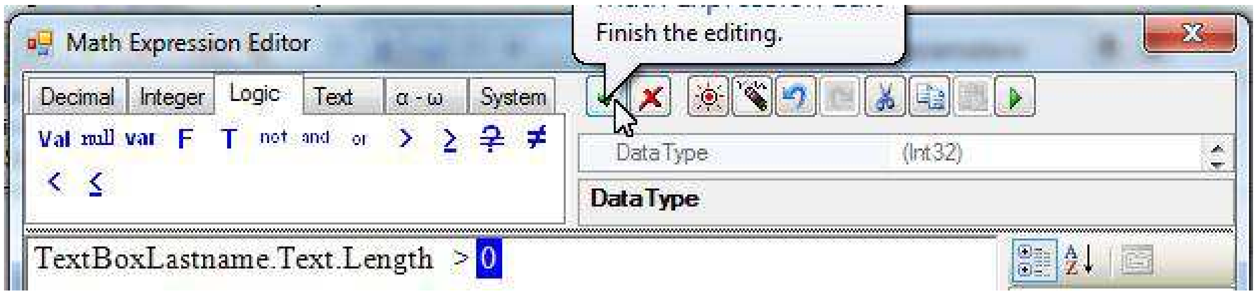
Right-click the Action Pane and choose “Create condition”:



Link the new condition to the Yes port of the first condition. Set its condition to an expression.

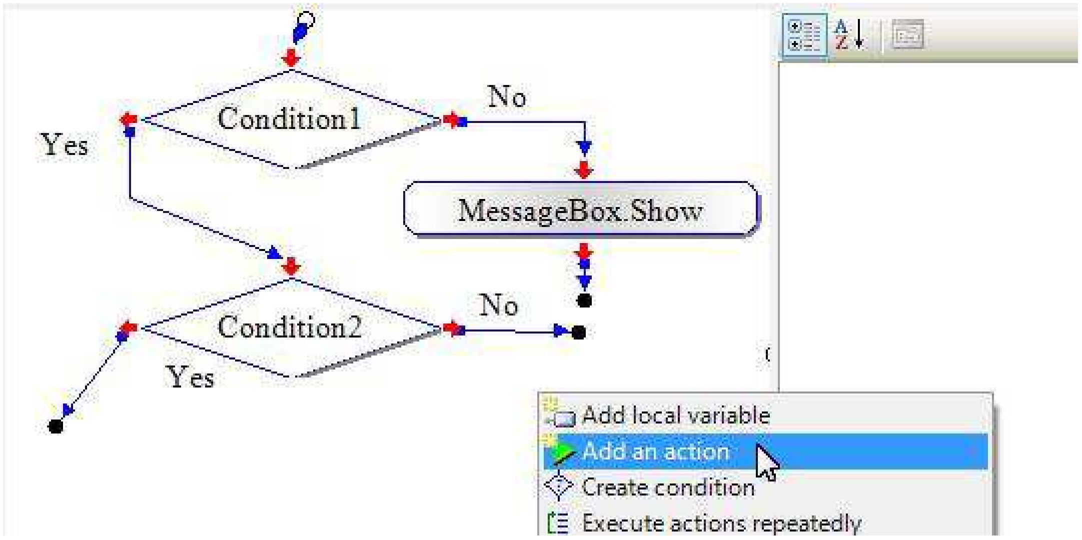


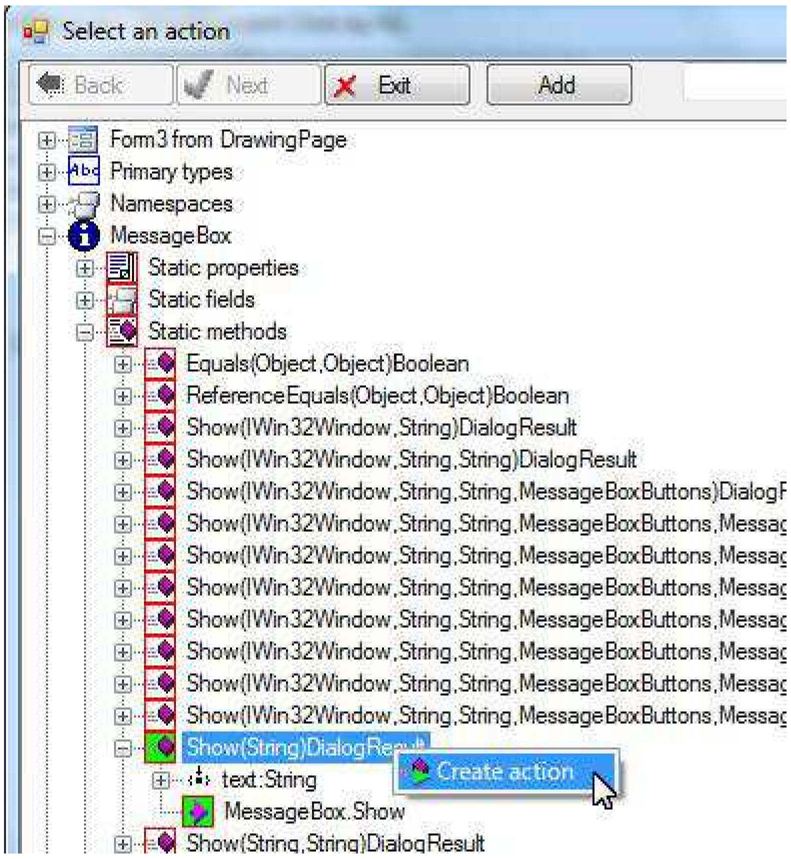
Create an expression to check the length of the Lastname text box:



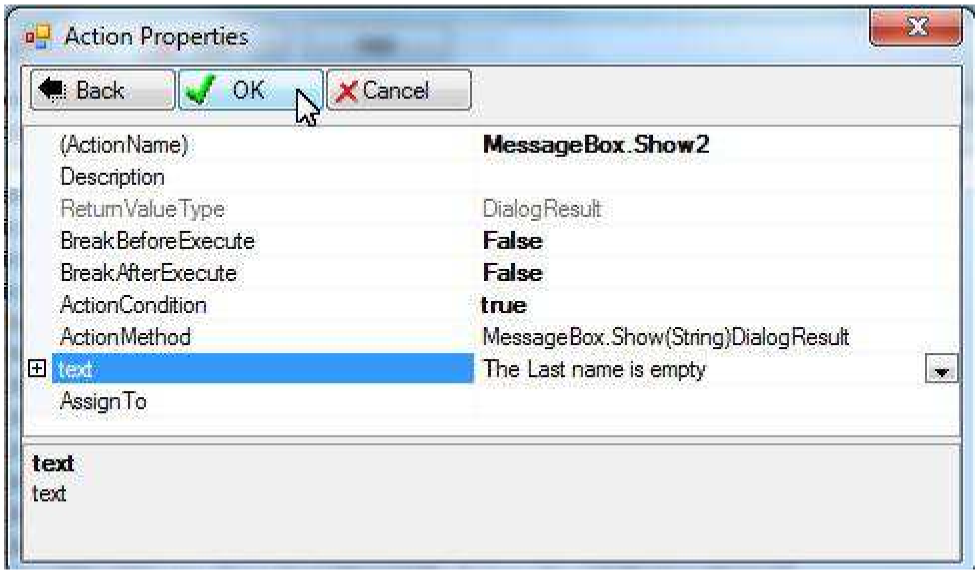
TextBoxLastname.Text.Length > 0

Add another action to show another message box:





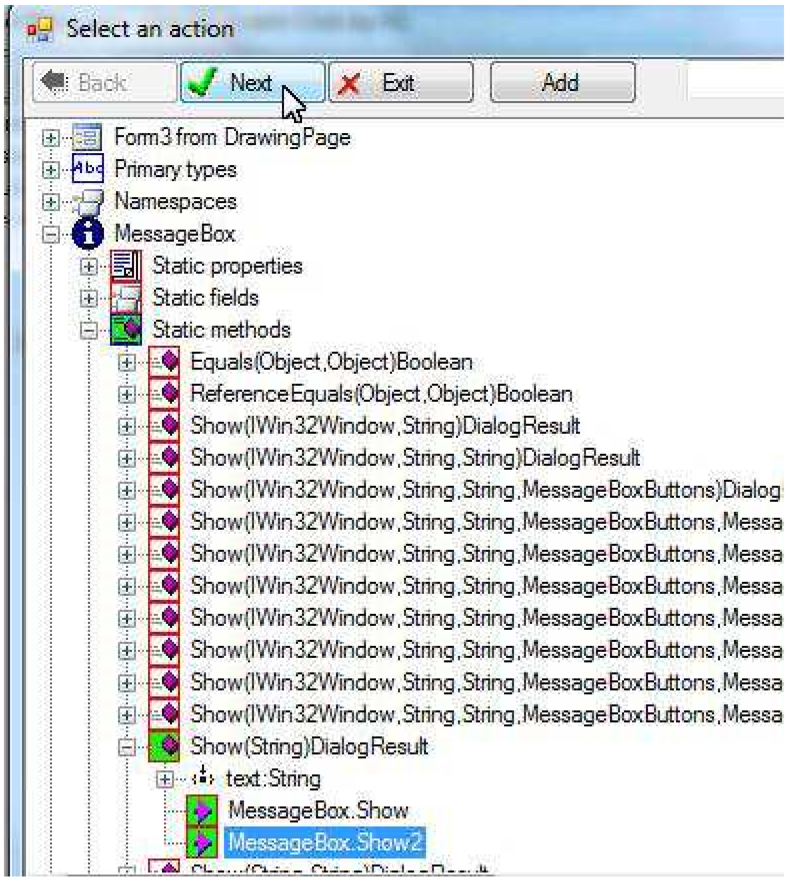
Form3 from DrawingPage
Primary types
Namespaces
MessageBox
Static properties
Static fields
Static methods
Equals(Object,Object)Boolean
ReferenceEquals(Object,Object)Boolean
Show(IWin32Window,String)DialogResult
Show(IWin32Window,String,String)DialogResult
Show(IWin32Window,String,String,MessageBoxButtons)DialogR
Show(IWin32Window,String,String,MessageBoxButtons,Messag
Show(IWin32Window,String,String,MessageBoxButtons,Messag
Show(IWin32Window,String,String,MessageBoxButtons,Messag
Show(IWin32Window,String,String,MessageBoxButtons,Messag
Show(IWin32Window,String,String,MessageBoxButtons,Messag
Show(IWin32Window,String,String,MessageBoxButtons,Messag
Show(IWin32Window,String,String,MessageBoxButtons,Messag
Show(String)DialogResult
text:String
MessageBox.Show
Show(String,String)DialogResult



(ActionName)	MessageBox.Show2
Description	
ReturnValueType	DialogResult
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	MessageBox.Show(String)DialogResult
text	The Last name is empty
AssignTo	

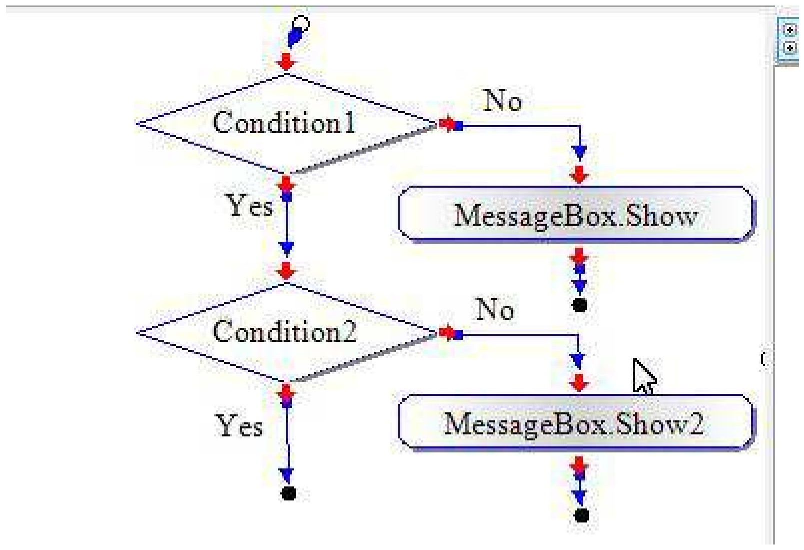
text
text

While the new action is selected, click Next:

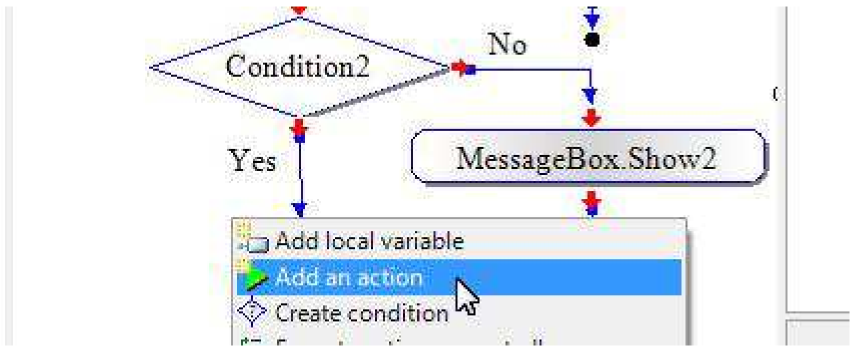


```
Form3 from DrawingPage
Primary types
Namespaces
MessageBox
    Static properties
    Static fields
    Static methods
        Equals(Object,Object)Boolean
        ReferenceEquals(Object,Object)Boolean
        Show(IWin32Window,String)DialogResult
        Show(IWin32Window,String,String)DialogResult
        Show(IWin32Window,String,String,MessageBoxButtons)Dialog
        Show(IWin32Window,String,String,MessageBoxButtons,Messa
        Show(IWin32Window,String,String,MessageBoxButtons,Messa
        Show(IWin32Window,String,String,MessageBoxButtons,Messa
        Show(IWin32Window,String,String,MessageBoxButtons,Messa
        Show(IWin32Window,String,String,MessageBoxButtons,Messa
        Show(IWin32Window,String,String,MessageBoxButtons,Messa
        Show(IWin32Window,String,String,MessageBoxButtons,Messa
        Show(String)DialogResult
            text:String
            MessageBox.Show
            MessageBox.Show2
```

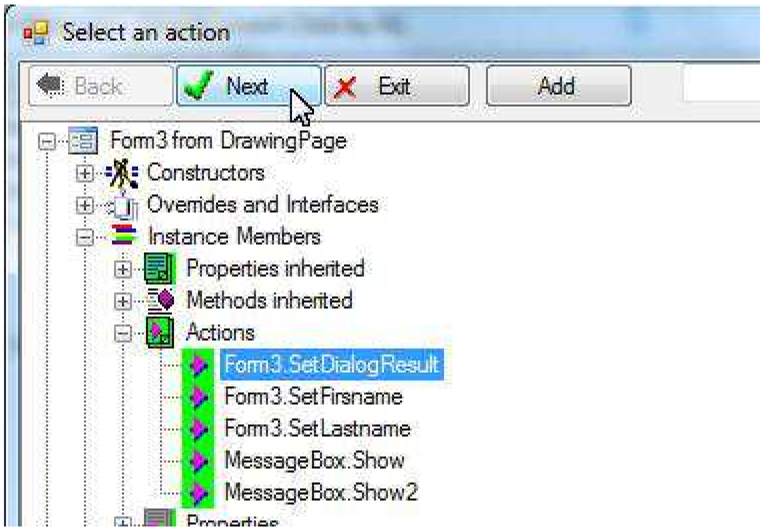
Link the new action to the No port of the second condition:



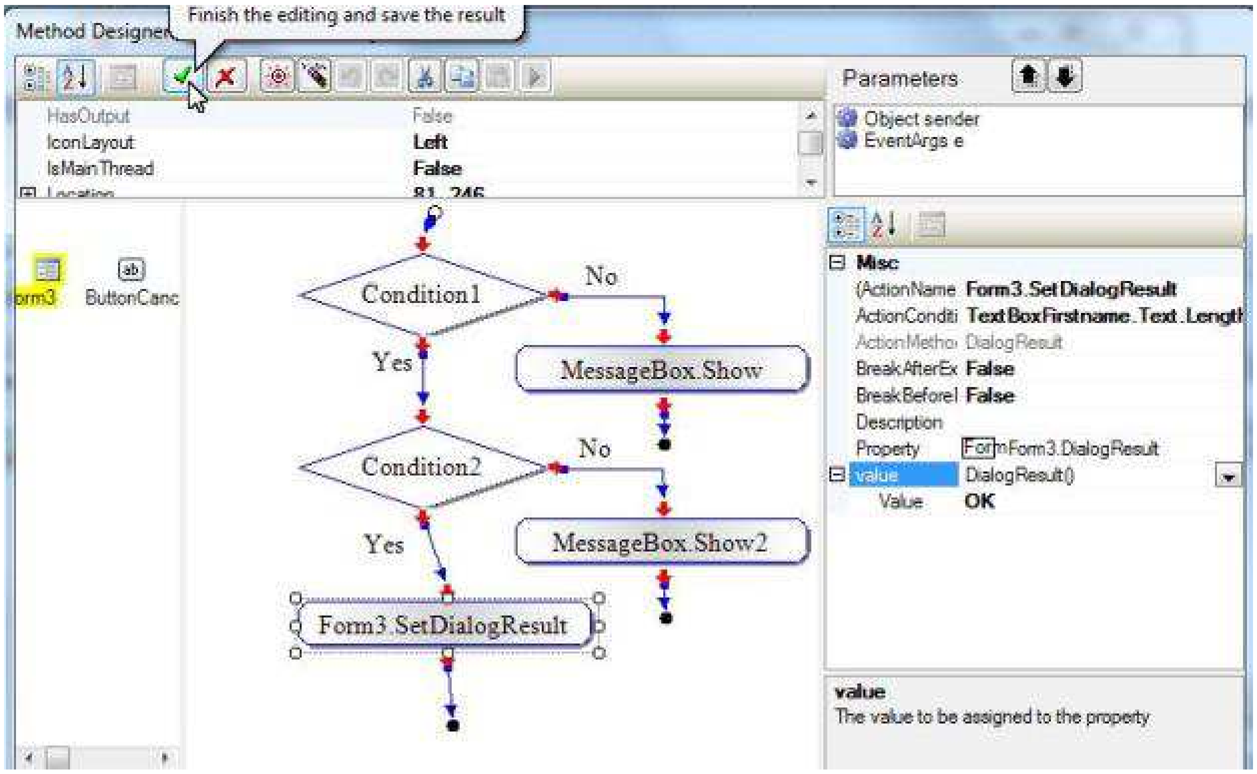
Right-click the Action Pane, choose “Add an action”:



Choose the action Form3.SetDialogResult; click Next



The action appears in the Action Pane. Link it to the Yes port of the second condition:

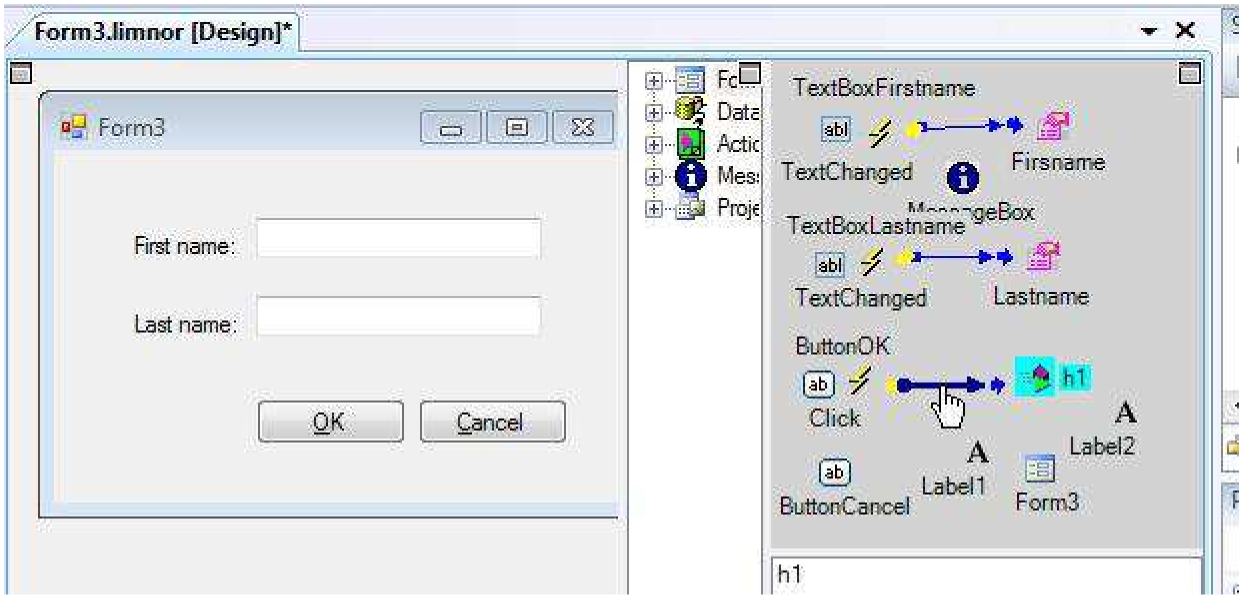


HasOutput	False
IconLayout	Left
IsMainThread	False
Location	81, 246

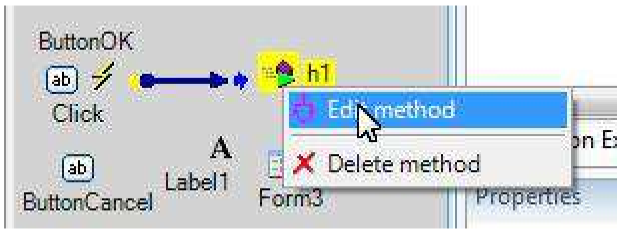
Misc	
(ActionName	Form3.SetDialogResult
ActionConditi	TextBoxFirstname.Text.Length
ActionMetho	DialogResult
BreakAfterEx	False
BreakBeforel	False
Description	
Property	Form3.DialogResult
value	DialogResult()
Value	OK

value
The value to be assigned to the property

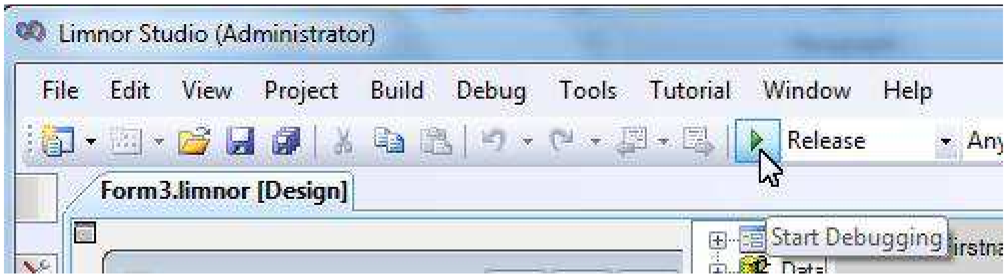
When the user clicks the OK button method h1 is executed:



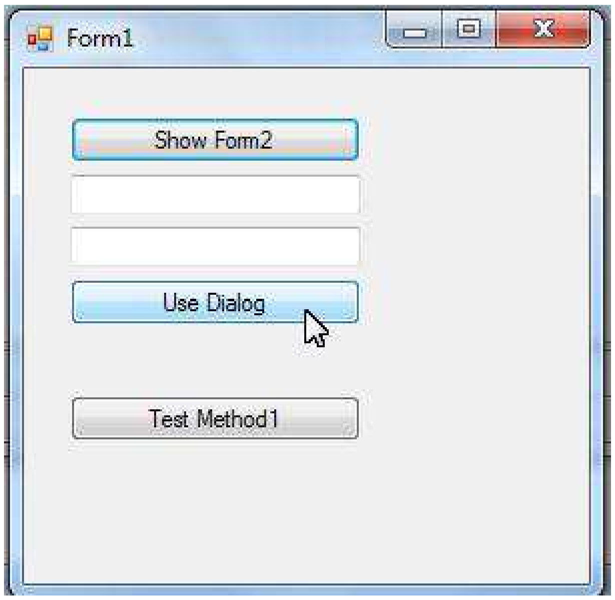
The event handler method can be modified by right-clicking it and choosing “Edit method”:



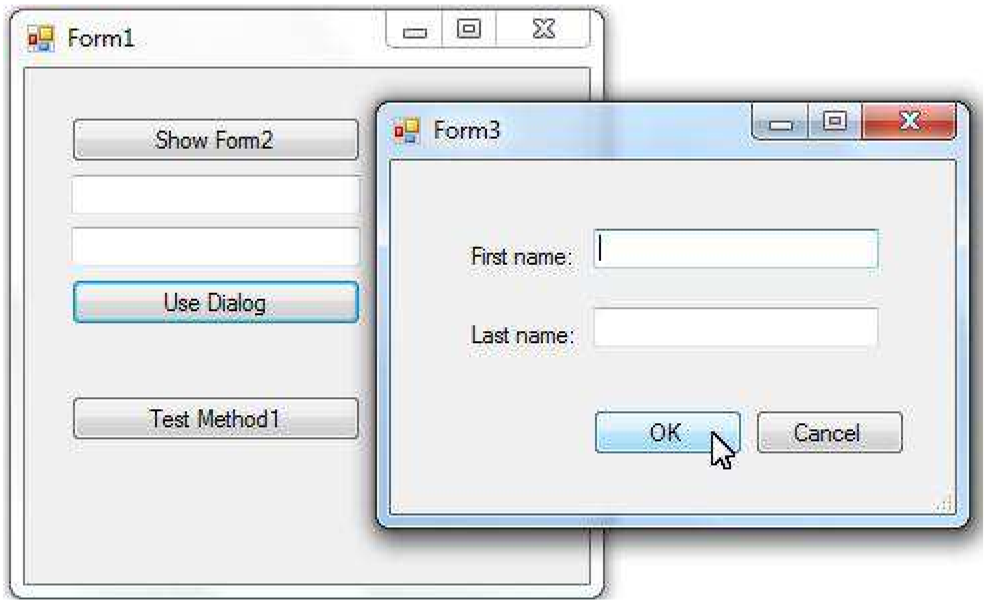
We may test it now



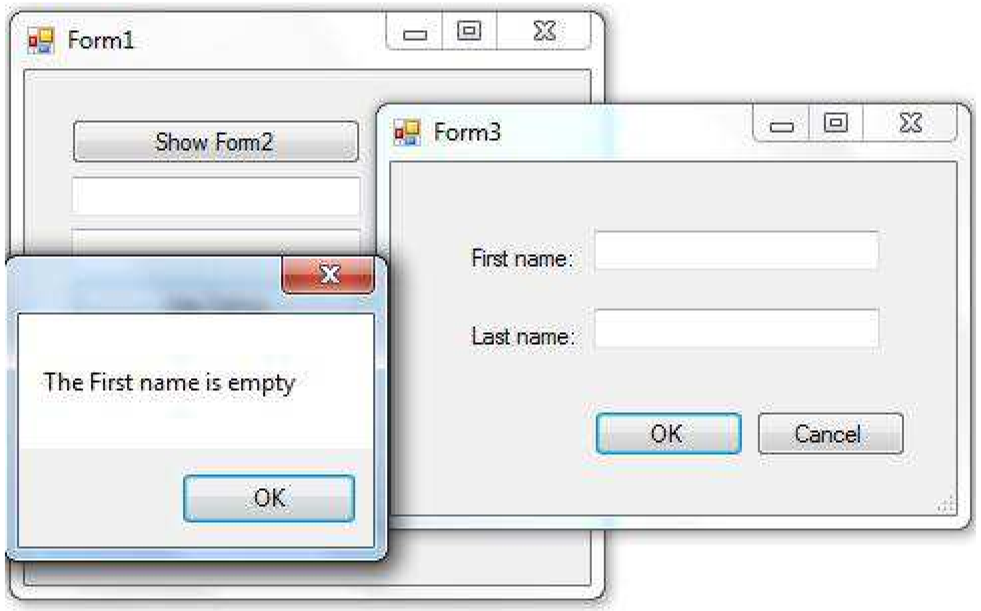
Form1 appears.



Click “Use Dialog”. Form3 appears.

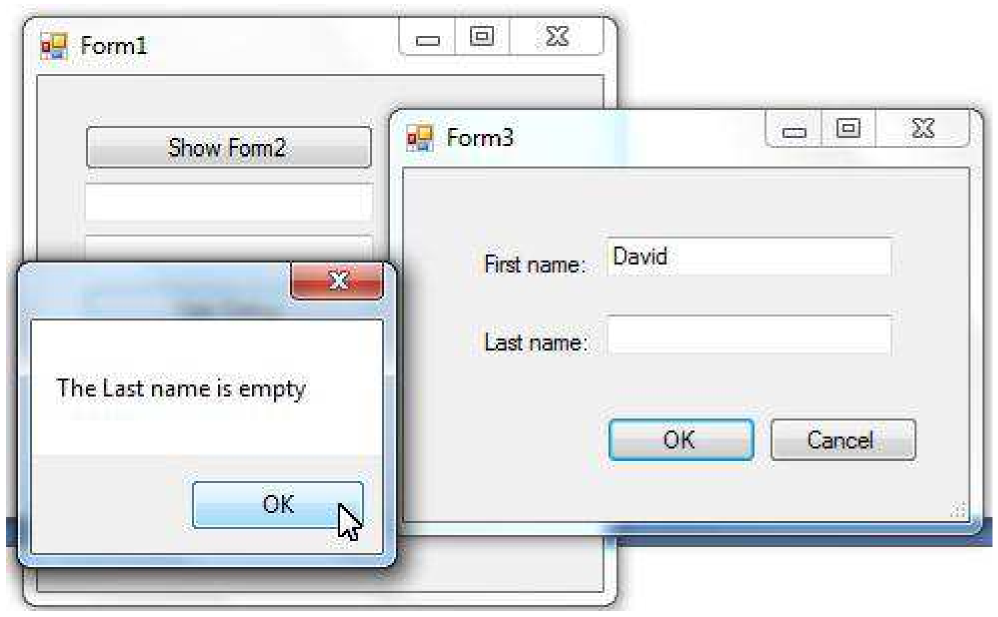


Click OK. A message box appears. Form3 does not close.

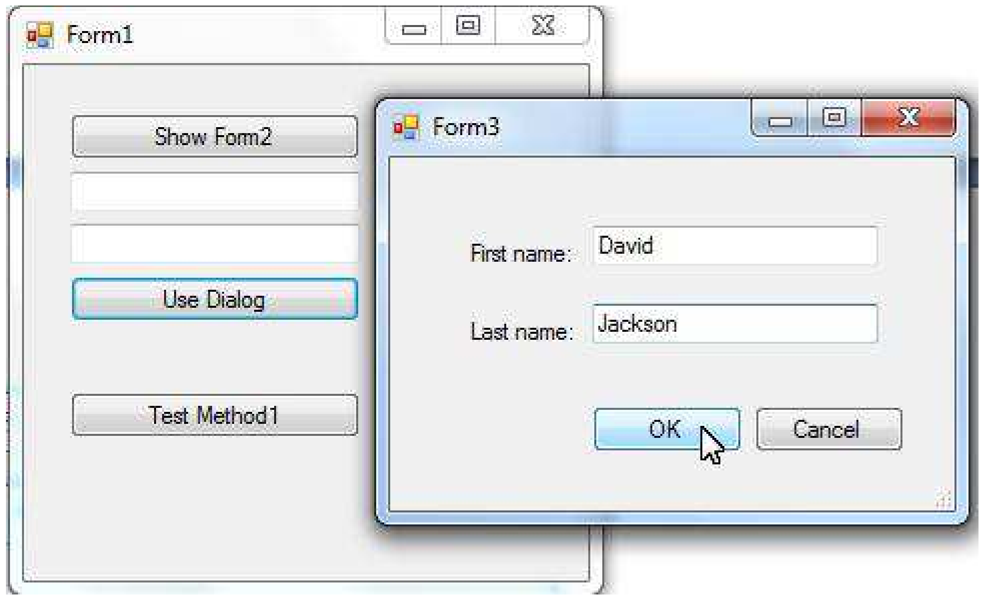


The First name is empty

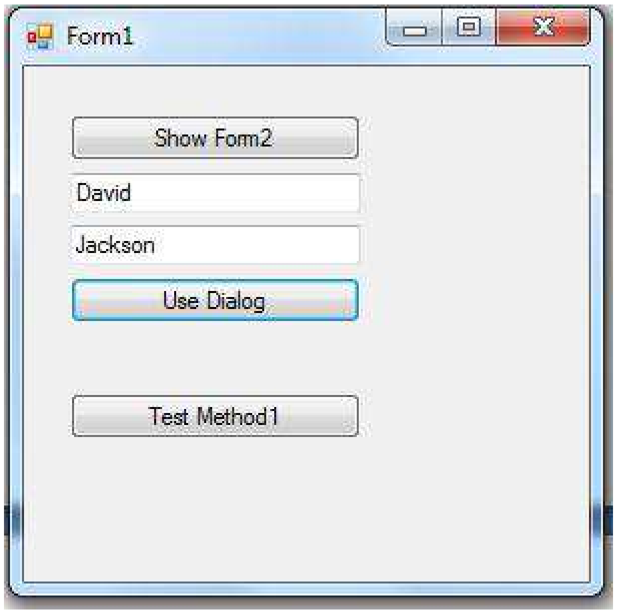
Enter some text in the first text box. Click OK again. Another message box appears.



Enter text in both text boxes. Click OK:



This time Form3 closes and the data are displayed in Form1:



David

Jackson

Feedback

Please send your feedback to support@limnor.com