

# How to Use MessageBox

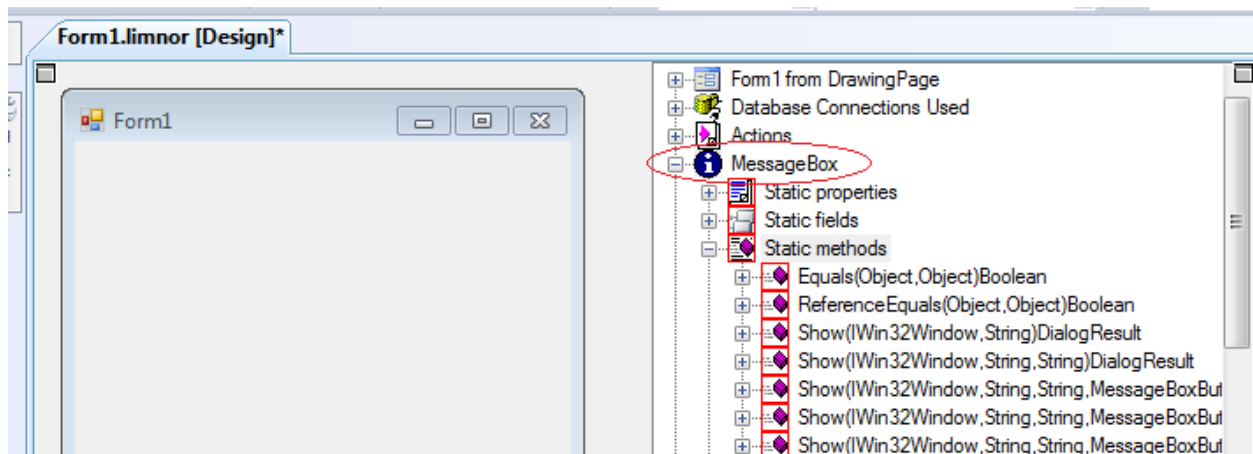
---

## Contents

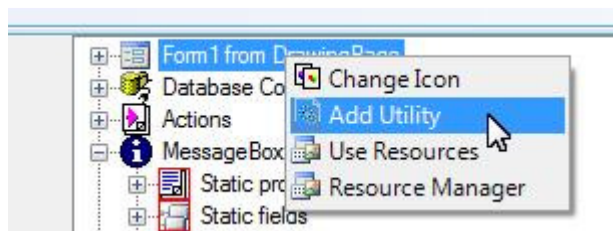
MessageBox Class.....	1
Use MessageBox without checking result .....	4
Check MessageBox Return Value.....	8
Use a property to save MessageBox return.....	9
Check MessageBox return value within a method.....	17
Use MessageBox owner.....	21

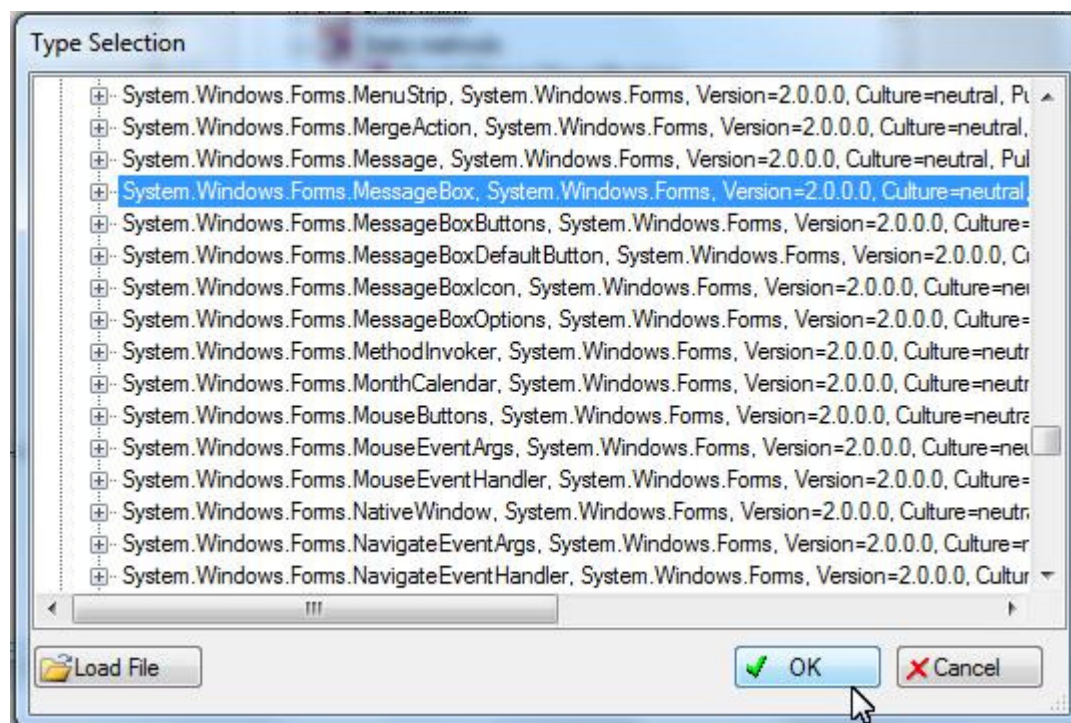
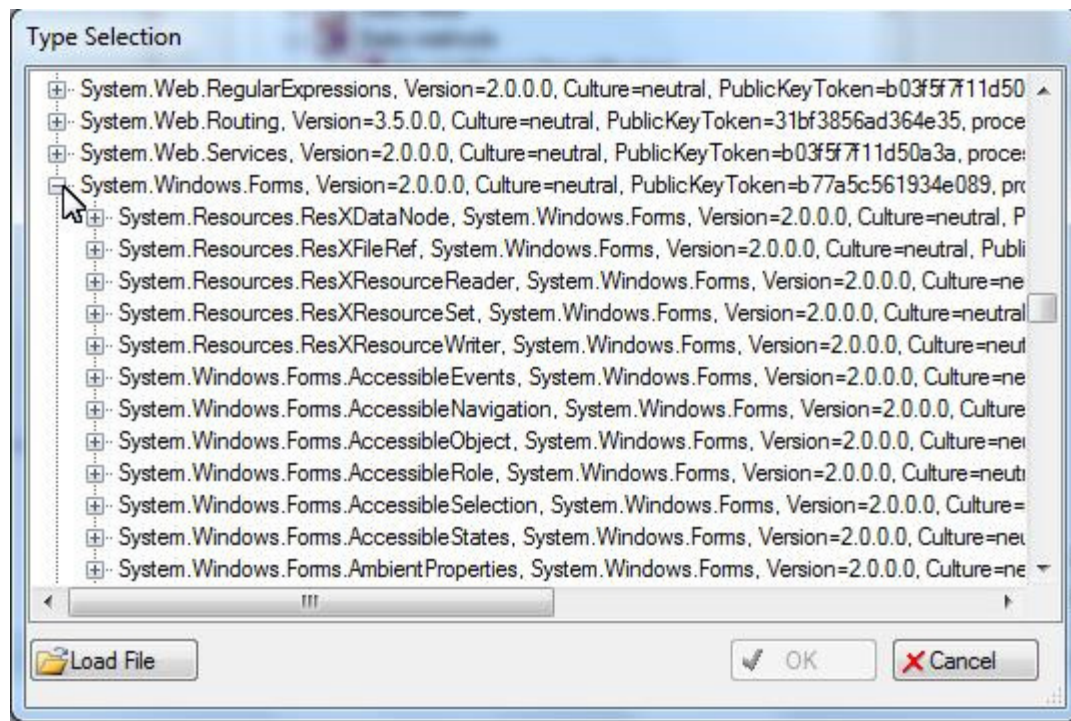
## MessageBox Class

The MessageBox is a static class. We are supposed to use its static methods to create actions. For convenience it is included in the Object Explorer.

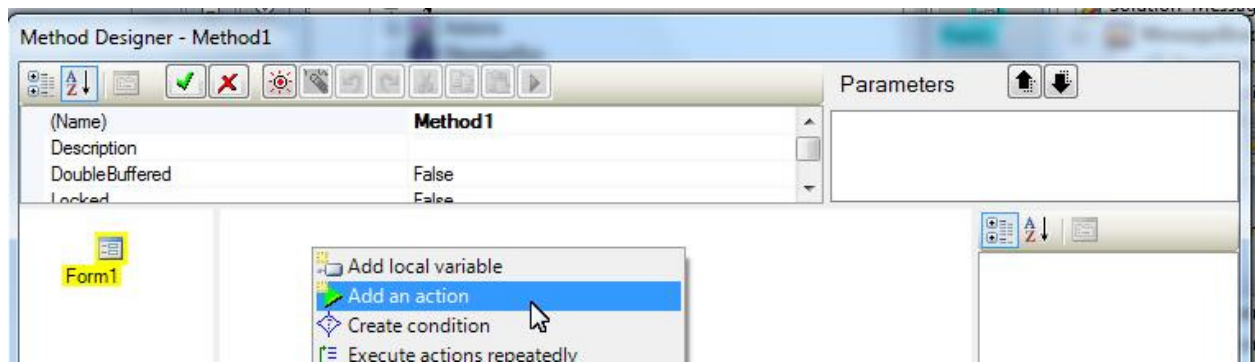


It can also be found under System.Windows.Forms namespace:

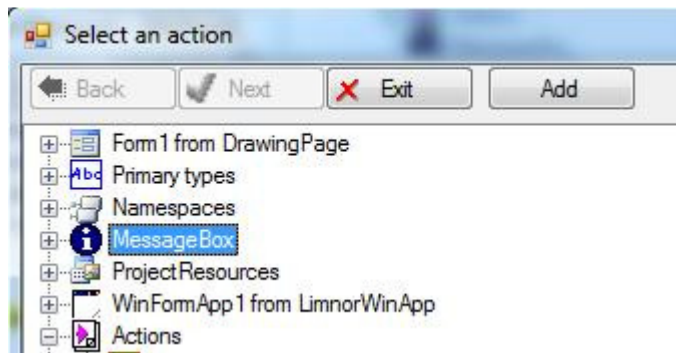




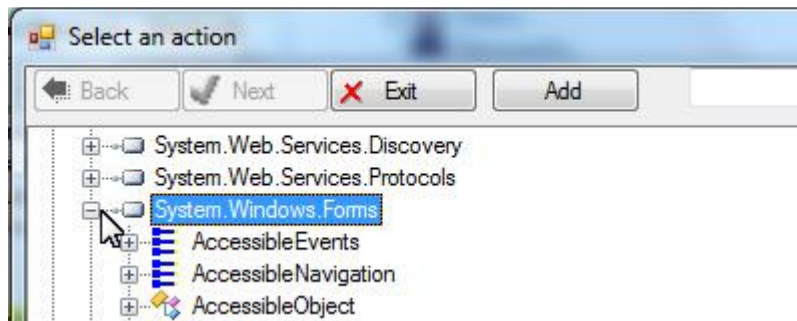
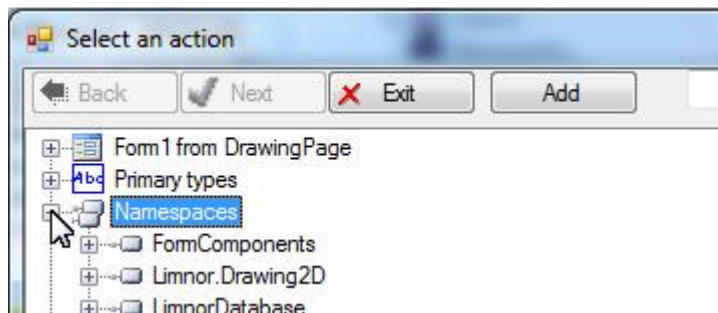
Within a Method Editor, it can also be found under namespace System.Windows.Forms:



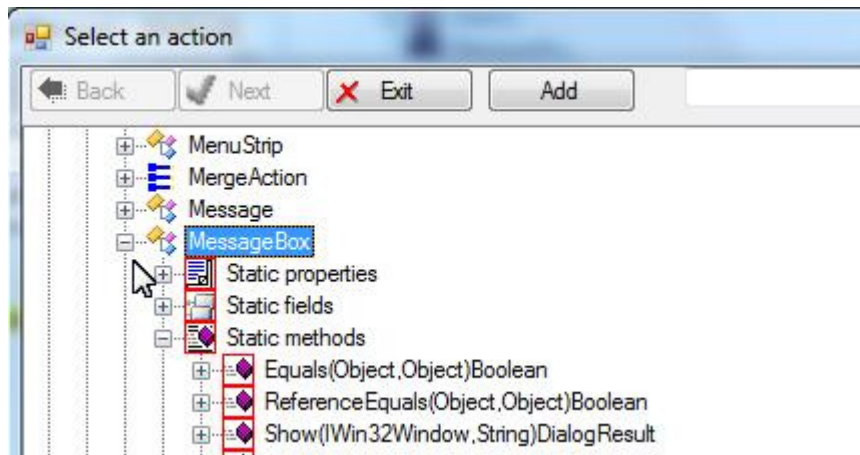
We can see that the MessageBox is there for convenience:



If it is not there then we may find it under namespace System.Windows.Forms:

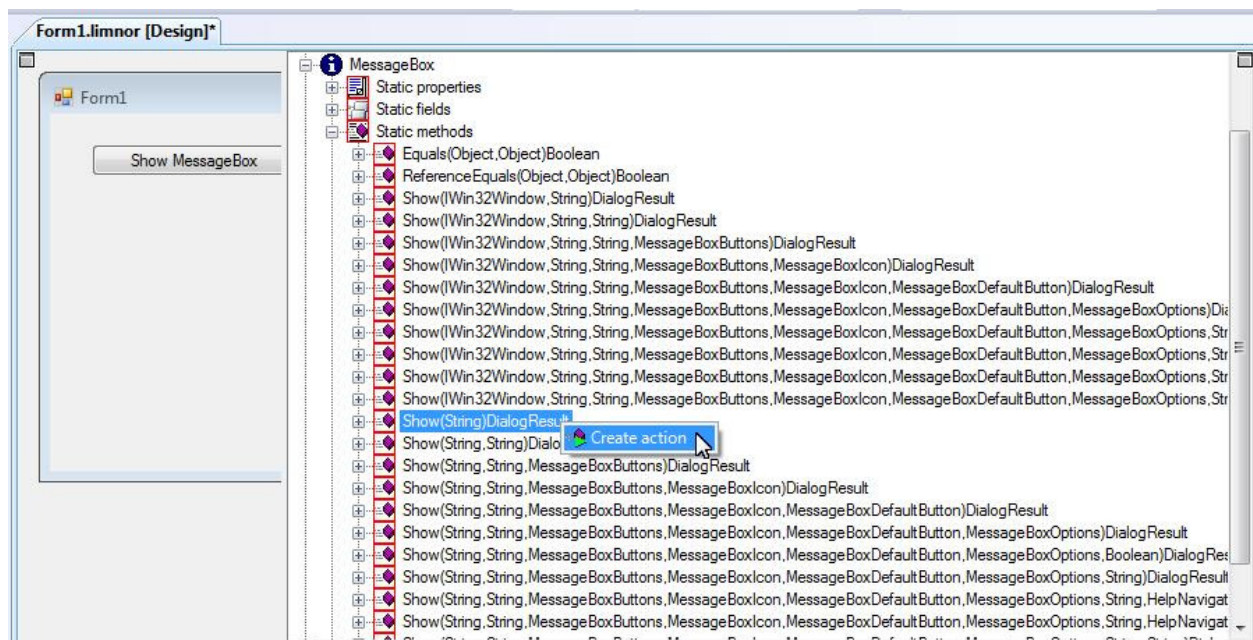




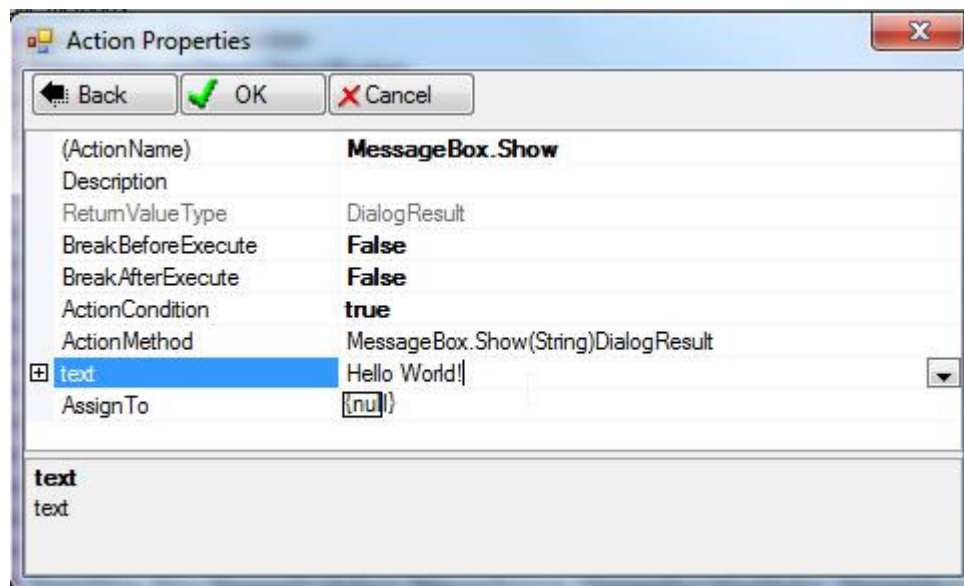


## Use MessageBox without checking result

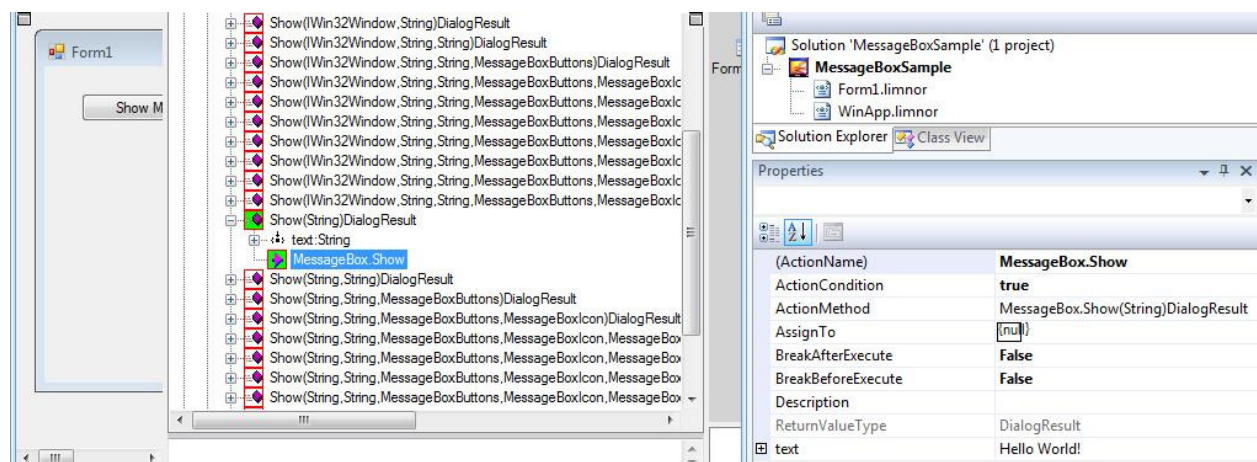
We just want to display a message, not asking the user to make a choice. The simplest method just needs one parameter for the message to be displayed:



Give the value to the “text” parameter of the action. An action parameter can be a constant, a value from a property, or a value of an expression. Here we simply type in “Hello World!”



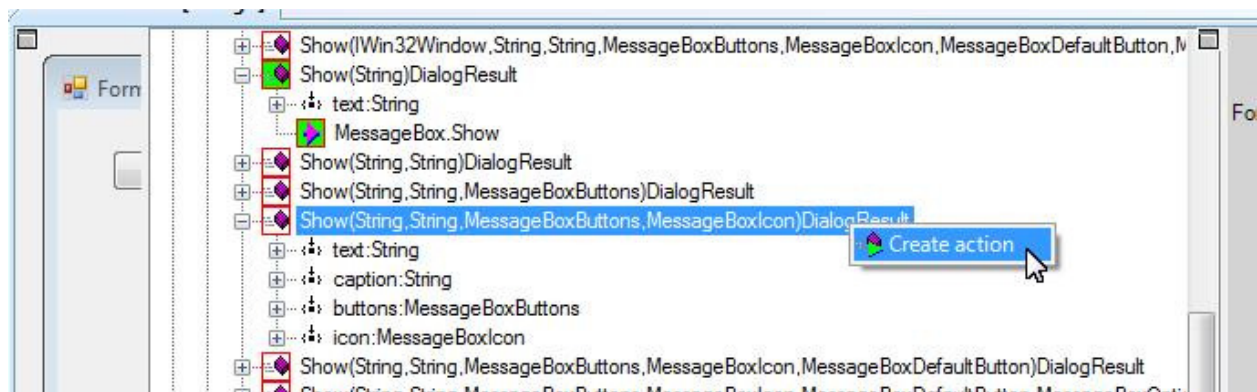
The action is created:



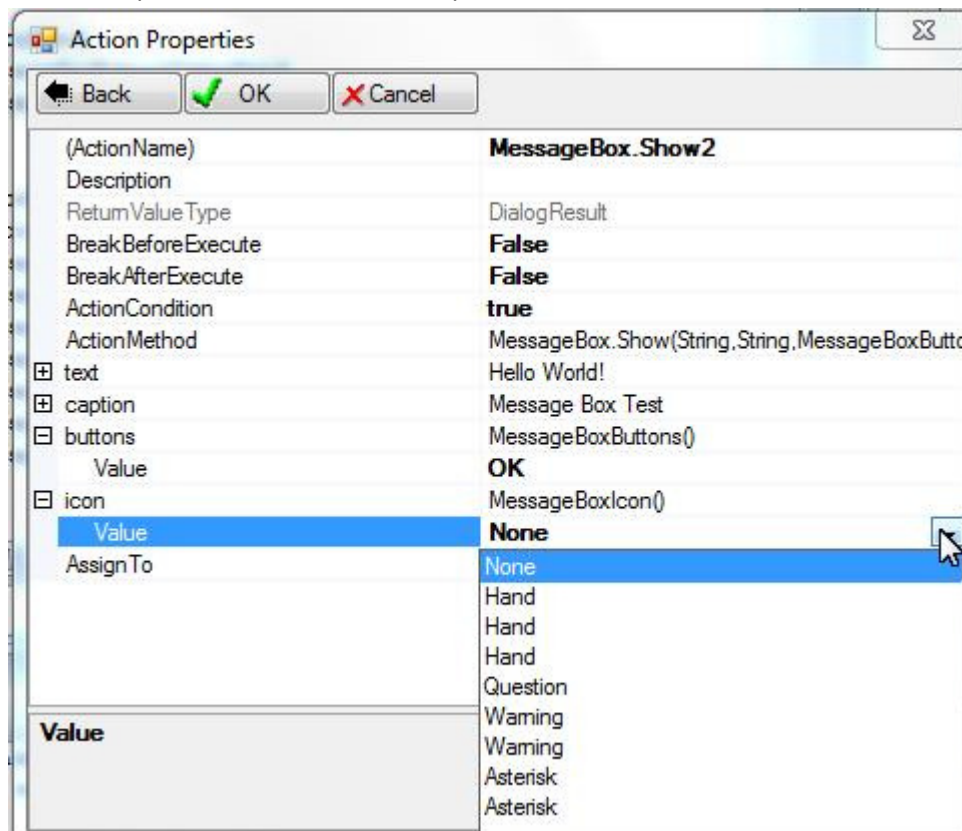
Assign this action to a button and test it:







We may use title and icon in the message box:



The “Icon” parameter of the action represents the icon to be used in the action.



We believe that there is a bug in the .Net Framework as we can see that the dropdown list repeats some values. Below are the enumerator values and icons used in the .Net Framework

Enumerator Value	Equivalent Value	Icon
None	None	
Hand	Hand	
Error		
Stop		
Question	Question	
Exclamation	Warning	
Warning		
Asterisk	Asterisk	
Information		

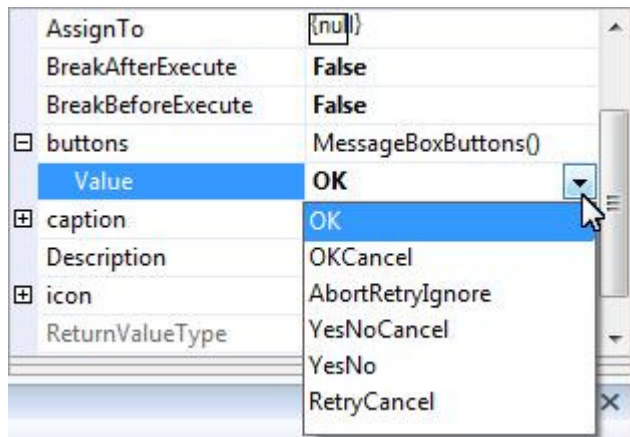
The icon may be different in different operating systems.

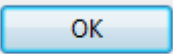

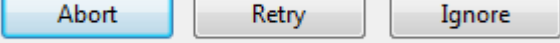
Suppose we select Asterisk for the icon. The test result is as below



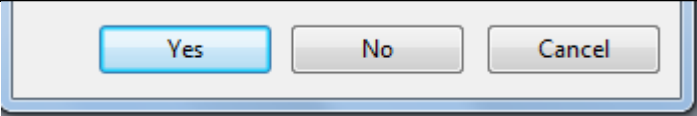

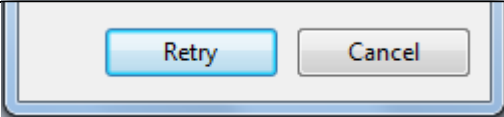
## Check MessageBox Return Value

In the previous sample action there is a “buttons” parameter. It represents the buttons to be used in the MessageBox. The MessageBox returns a value corresponding to the button the user clicks.



buttons parameter	Buttons shown	Possible return
OK		OK
OKCancel		OK Cancel
AbortRetryIgnore		Abort Retry Ignore

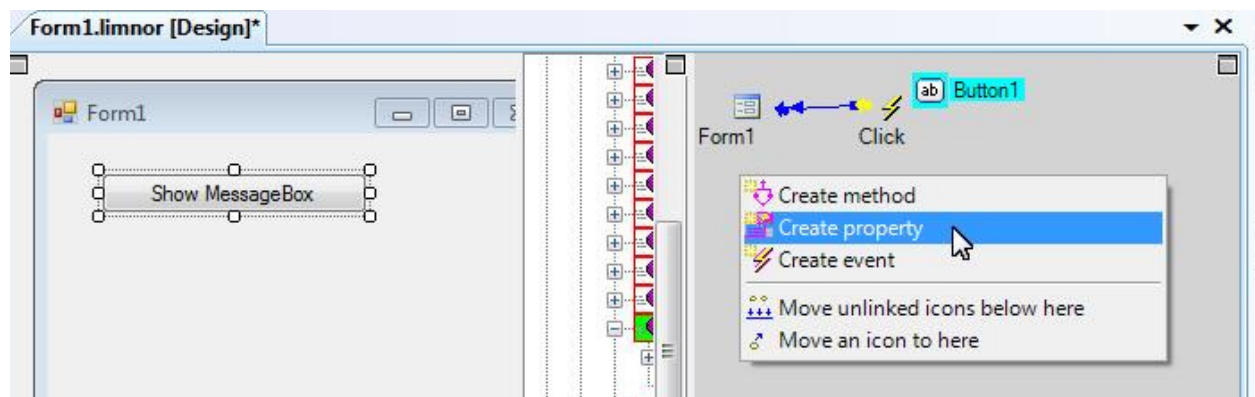


YesNoCancel		Yes No Cancel
YesNo		Yes No
RetryCancel		Retry Cancel

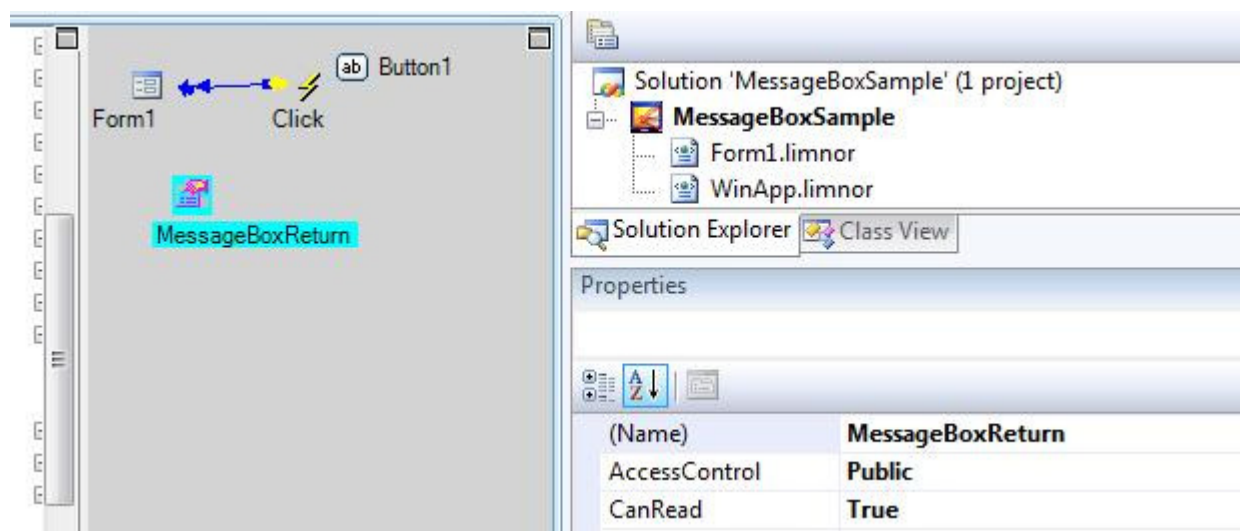
Below we show some examples of checking the return value of the MessageBox

## Use a property to save MessageBox return

Create a property

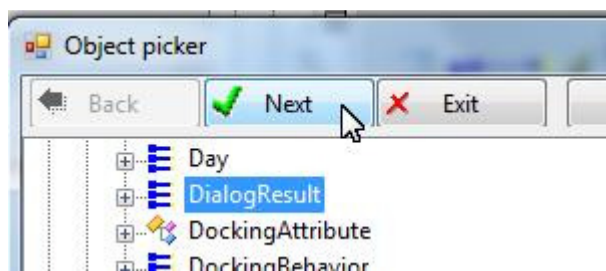
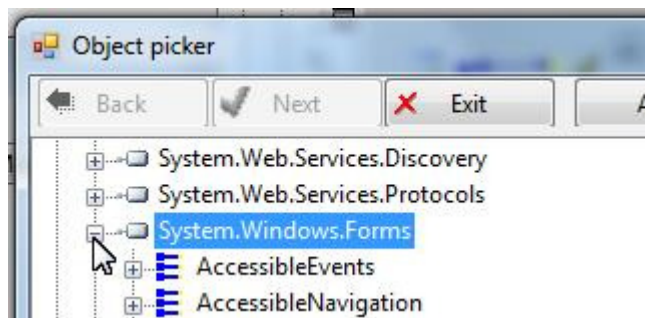
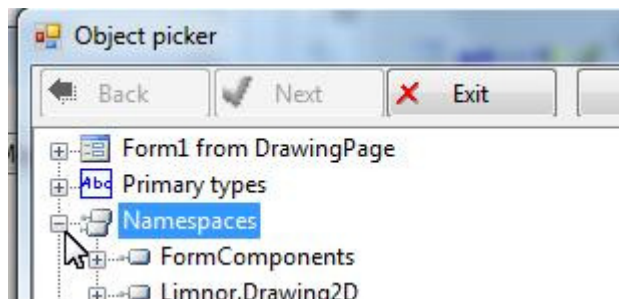


Change its name:

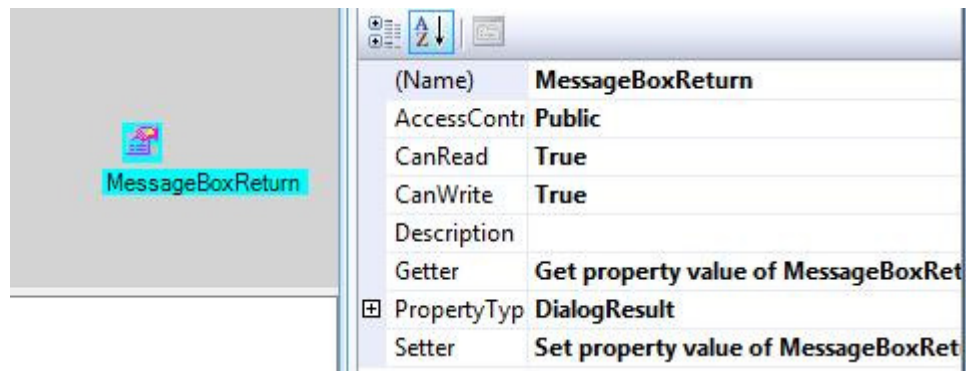


Change its PropertyType to DialogResult

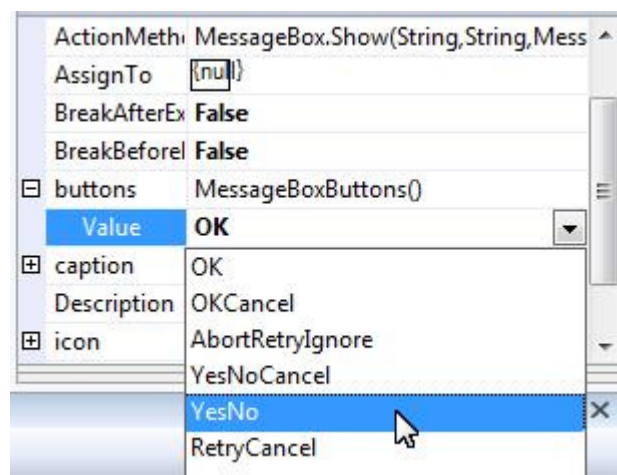
(Name)	<b>MessageBoxReturn</b>
AccessControl	<b>Public</b>
CanRead	<b>True</b>
CanWrite	<b>True</b>
Description	
Getter	<b>Get property value of MessageBoxReturn</b>
<b>PropertyType</b>	<b>String</b>
Setter	<b>Set property value of MessageBoxReturn</b>



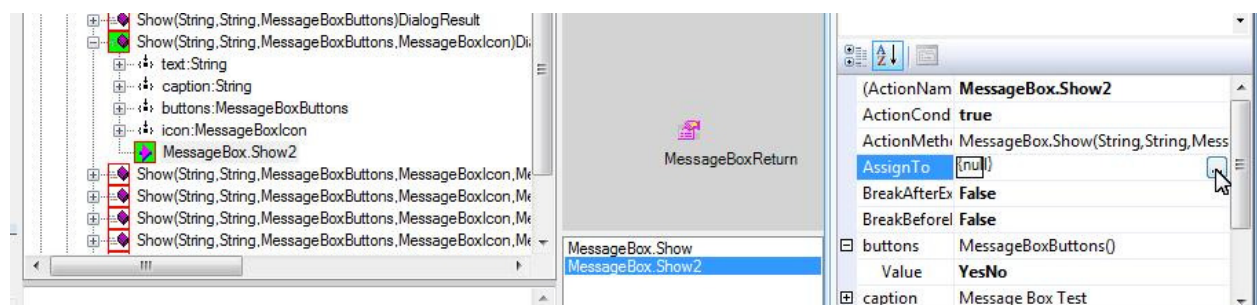
This property can be used to save MessageBox return value

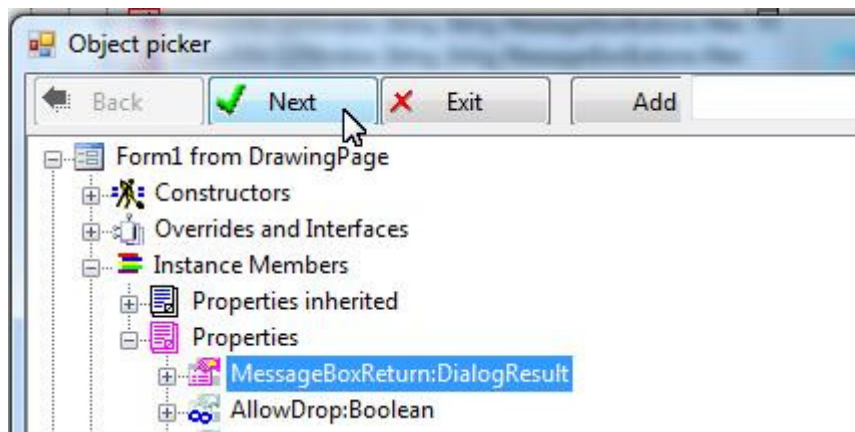


Let's modify action **MessageBox.Show2** we created previously to use **YesNo** buttons:



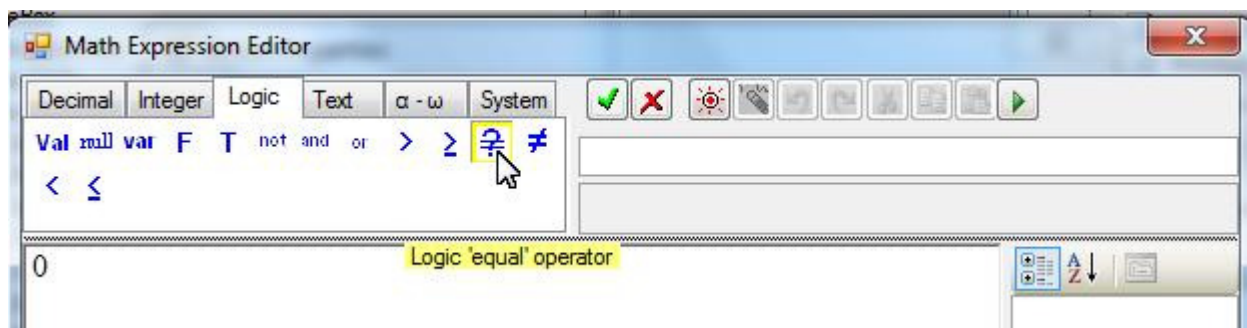
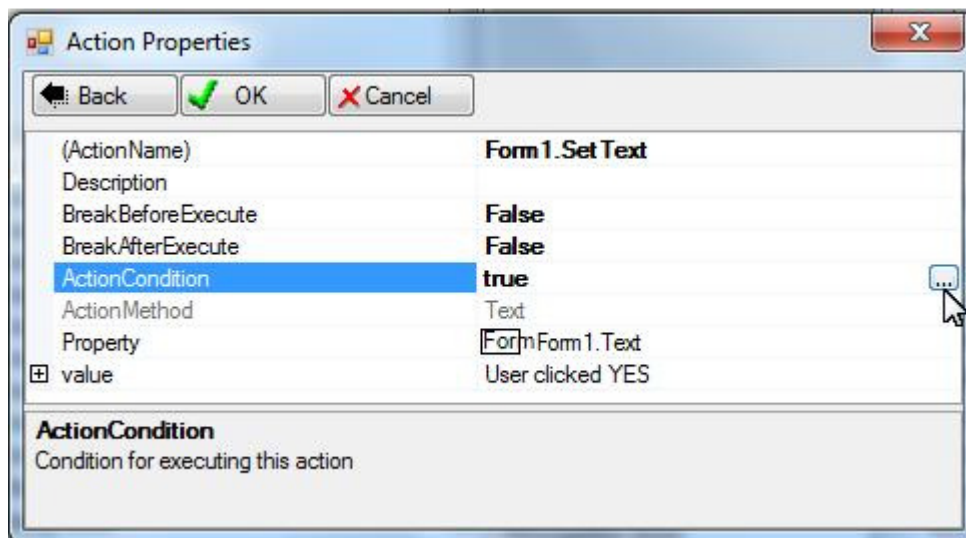
Modify its "AssignTo" property to assign the return value to the property we just created:



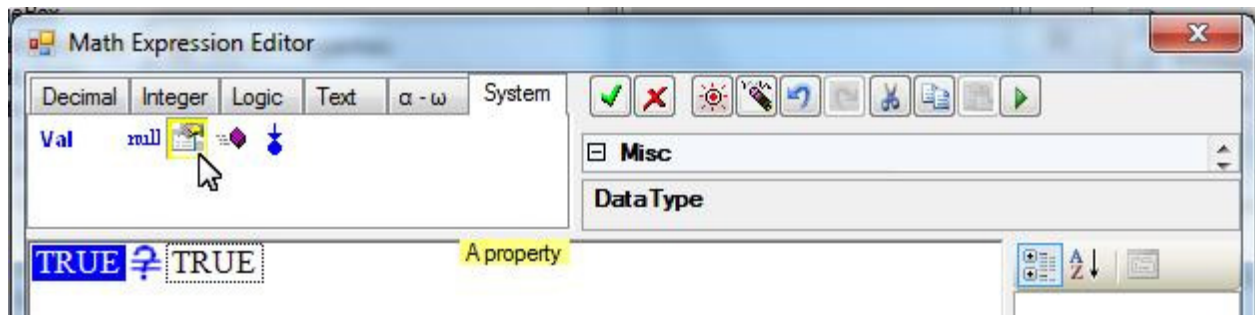


After executing action `MessageBox.Show2` examine property `MessageBoxReturn` to know which button the user clicked. For example, use `MessageBoxReturn` in action conditions.

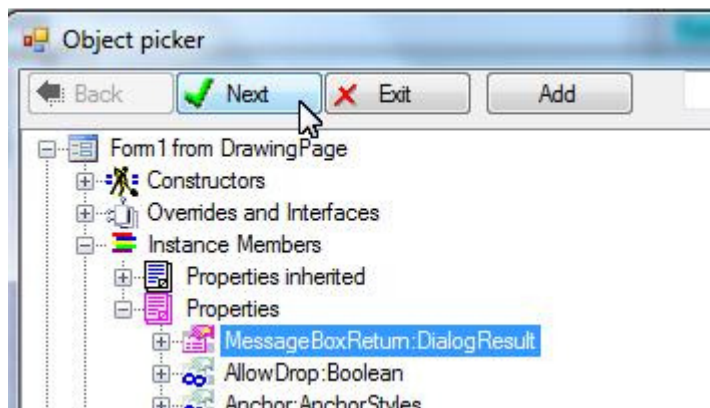
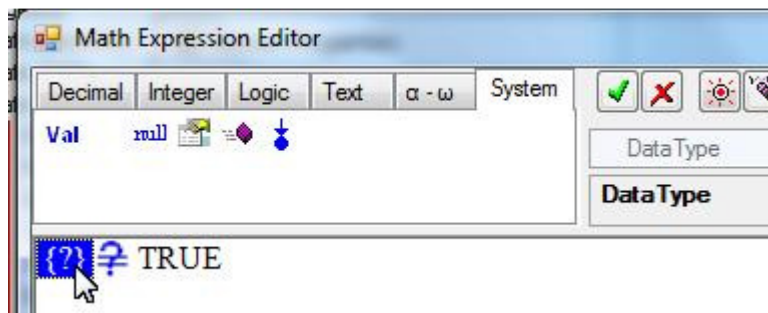
For a simple example, we create two actions to set the `Text` property of the Form. One action sets `Text` to "User clicked YES". The other to "User clicked NO". We use `MessageBoxReturn` in the `ActionCondition` to determine which action should execute.



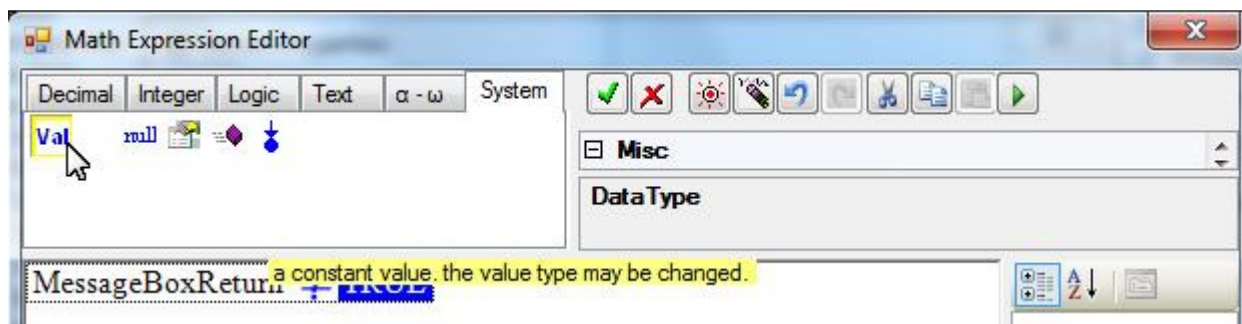




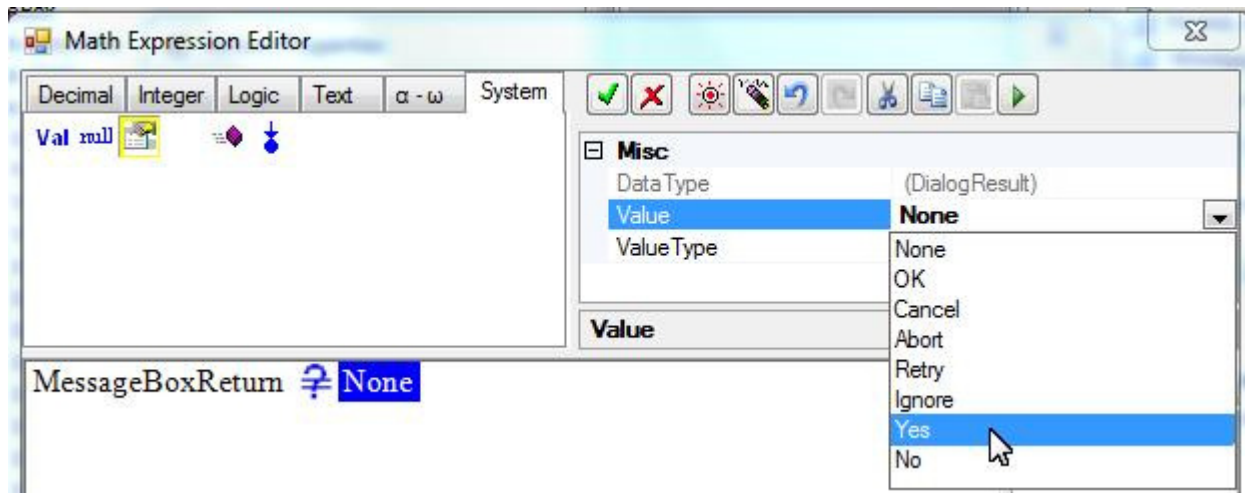
Double-click symbol {?}



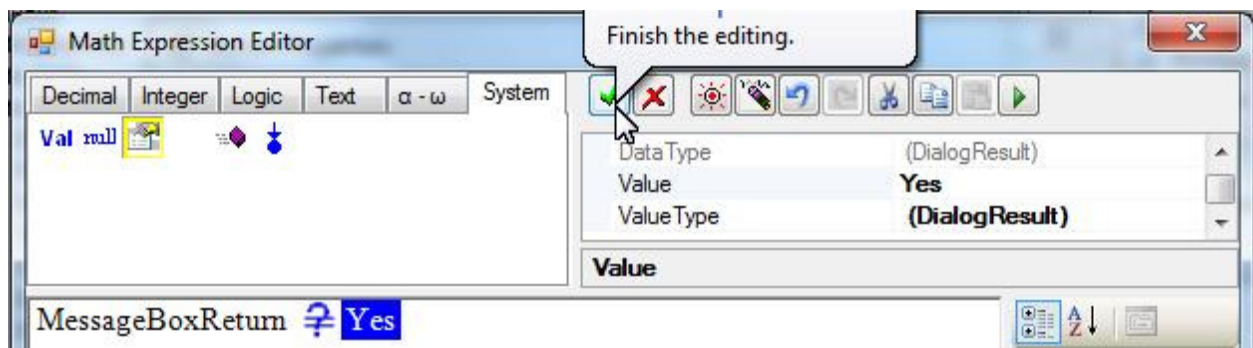
Select the second element and click "Val":



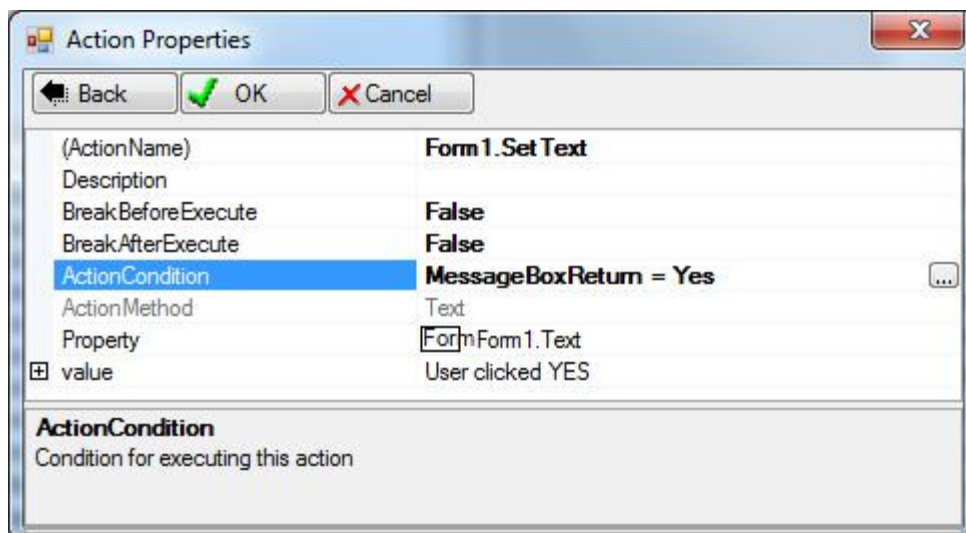
Choose Yes:



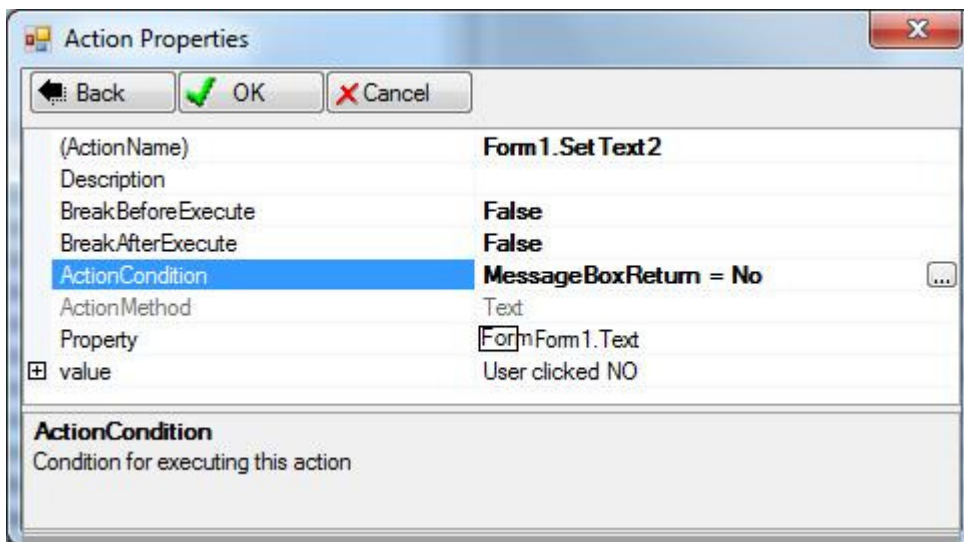
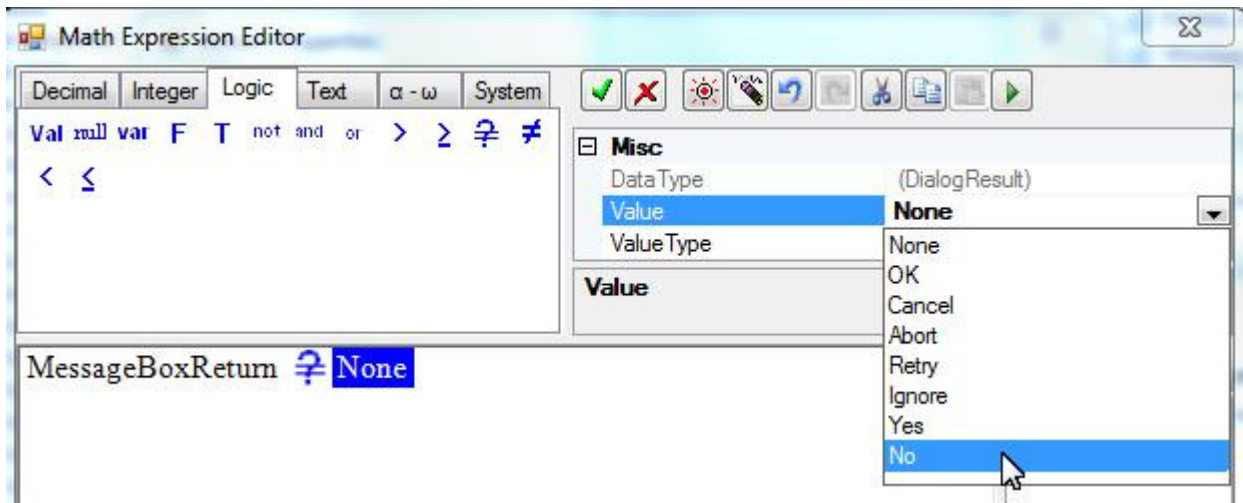
This is the action condition:



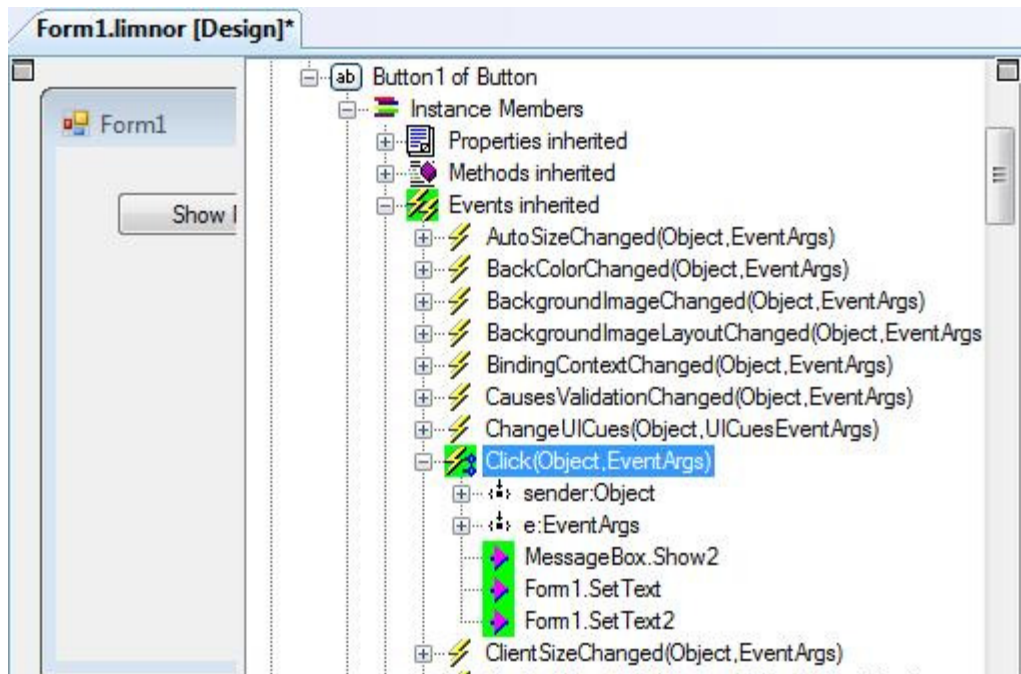
We made the first action to show "User clicked YES":



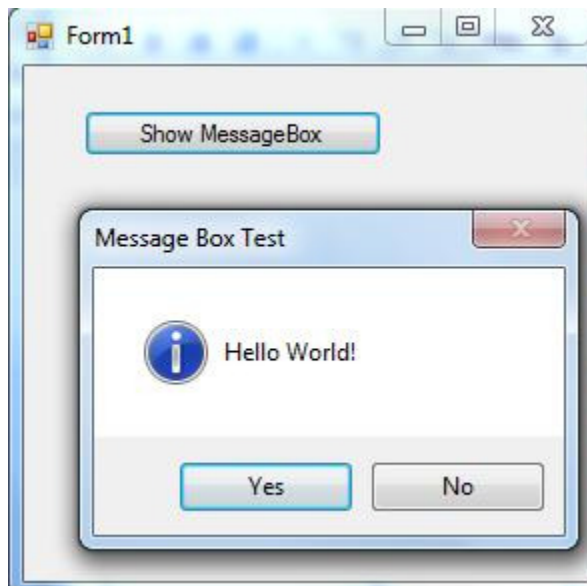
In the same way, we can make the second action to show "User clicked NO":



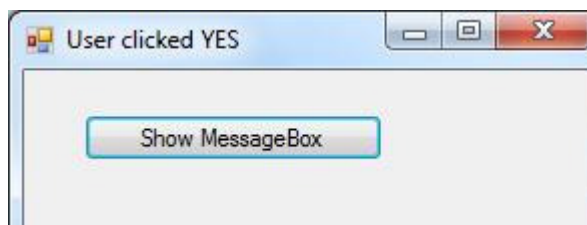
Assign the 3 actions to the button:



When the button is clicked, a message box appears with Yes and No buttons:

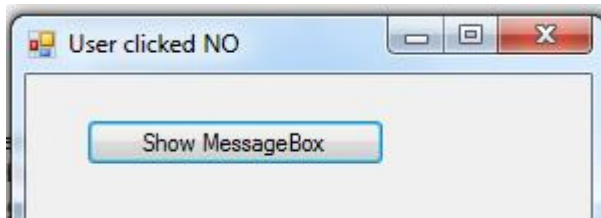


Click Yes, we see the title changed to “User clicked YES”:





Click No, we see the title changed to “User clicked No”:



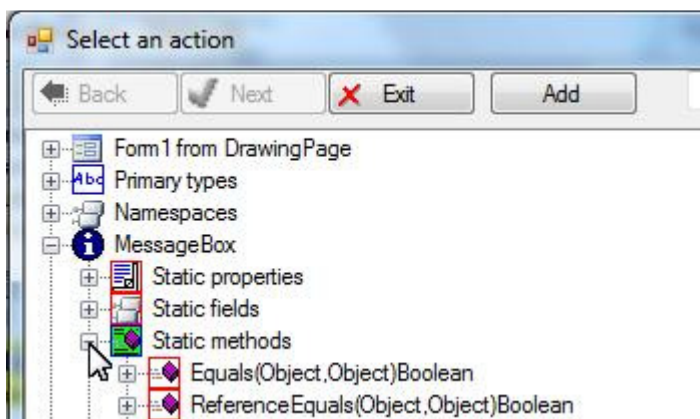
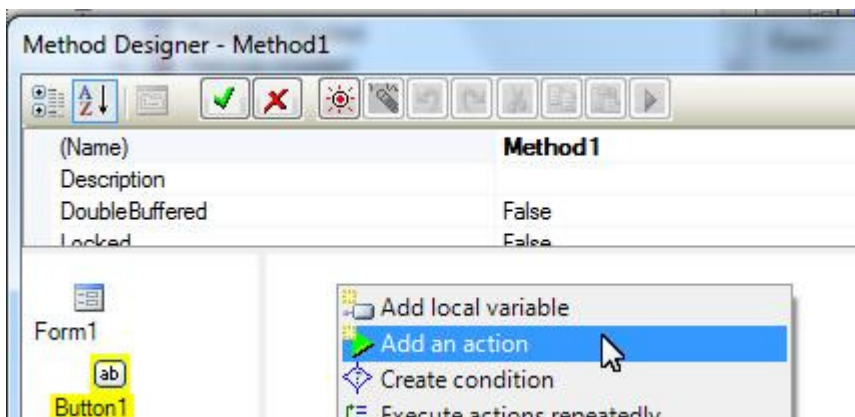
## Check MessageBox return value within a method

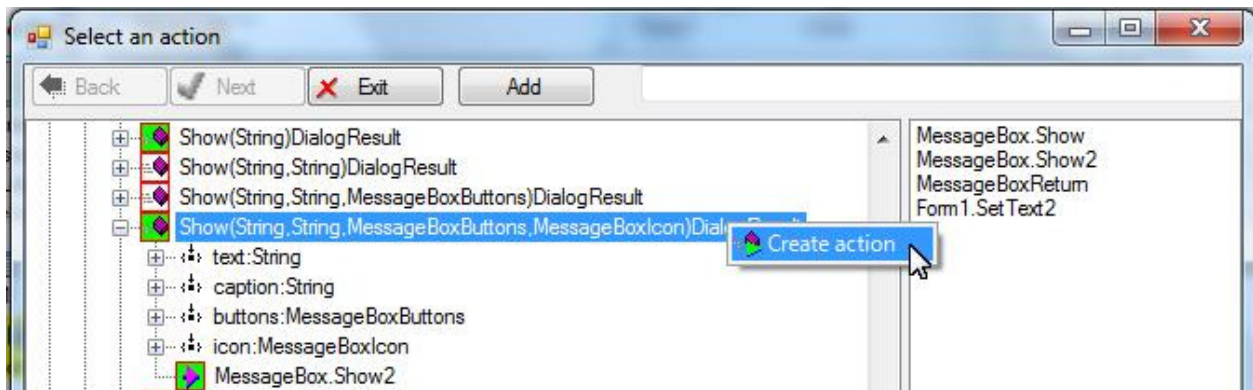
If a MessageBox is used within a method then additional options are available. For example, use a local variable to save the message box return value instead of a property.

If many actions are involved then setting action condition for all actions can be tedious.

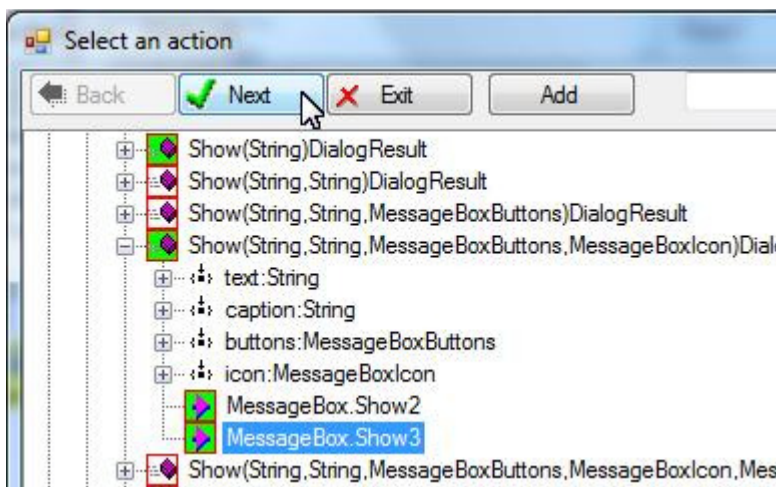
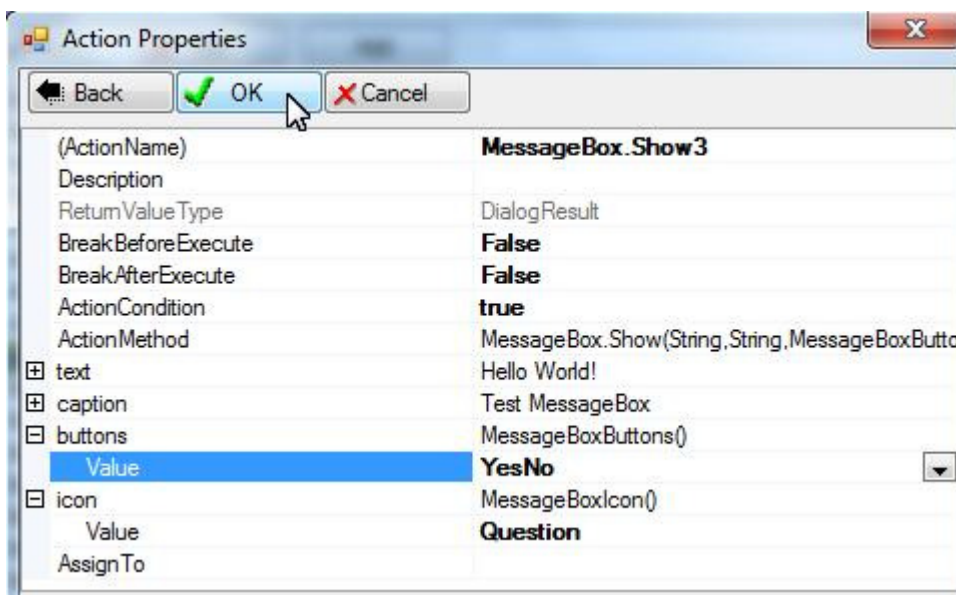
A condition action can be used to branch the execution paths without setting action conditions for every action. You **do not** need to save the MessageBox return value to a property or a local variable.

Let's add a MessageBox action in a method:



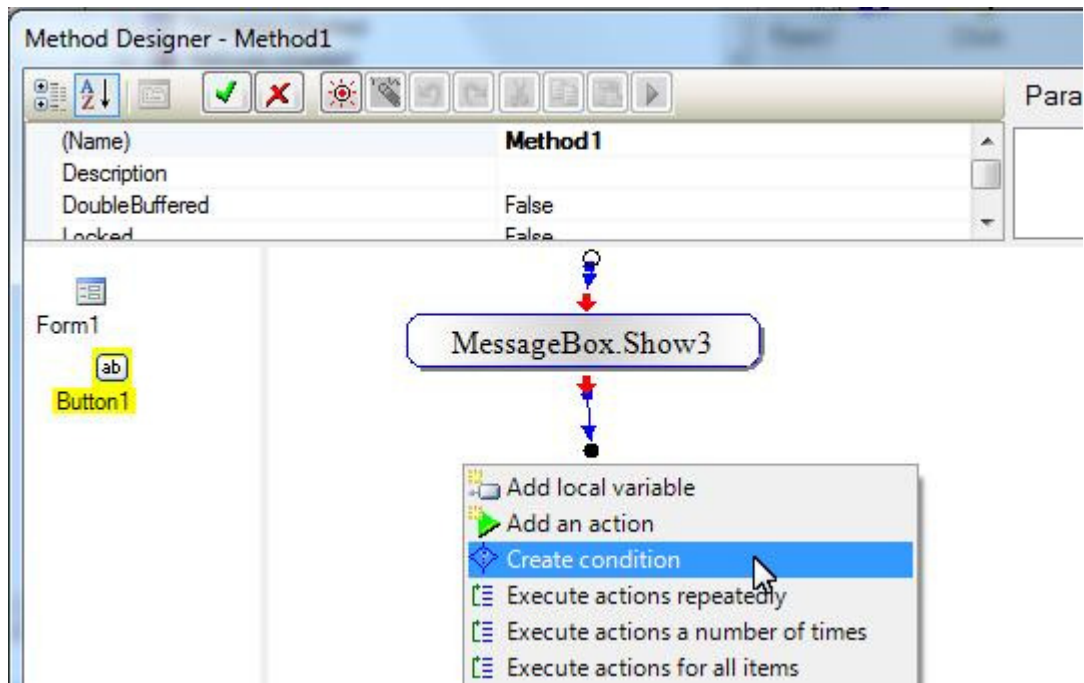


Use buttons YesNo:

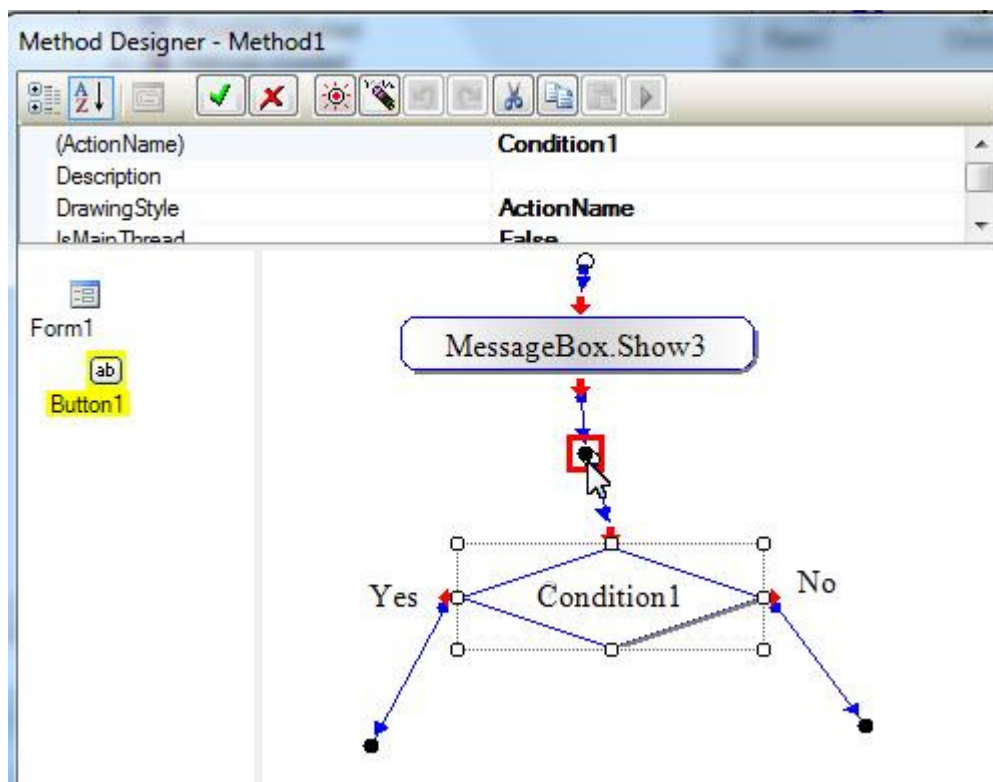


The message box action appears in the Action Pane.

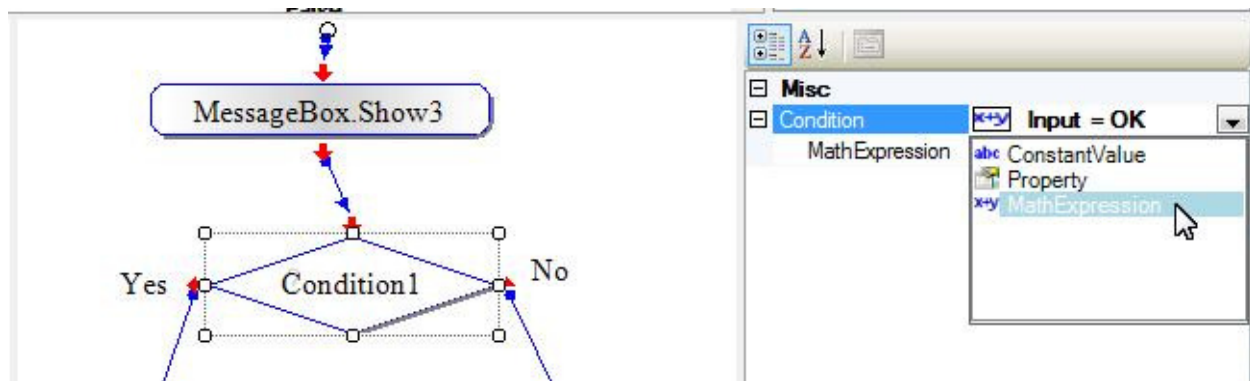
Add a Condition action:



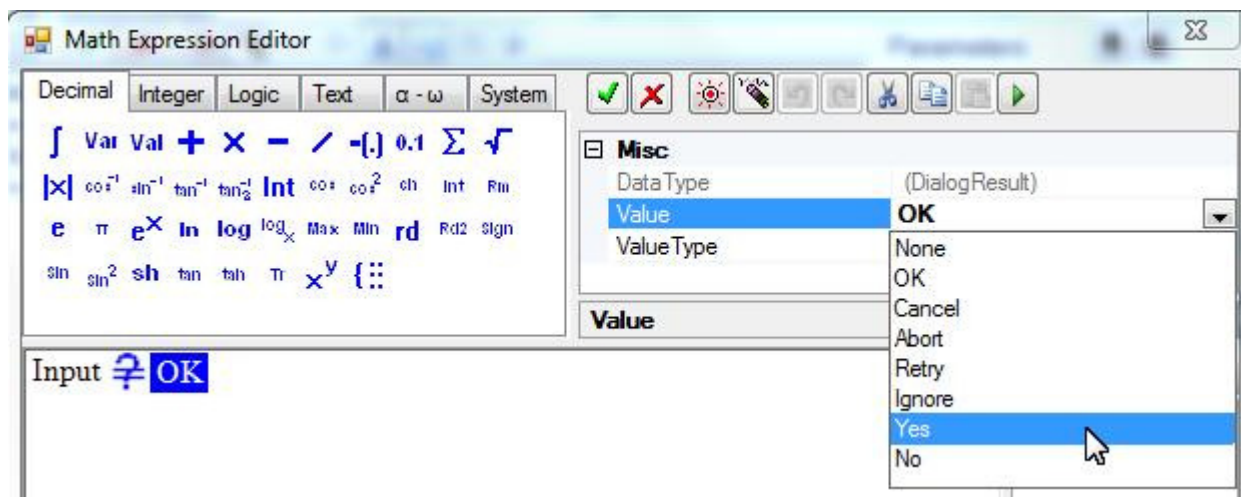
Link the input of the condition action to the output of the message box action:



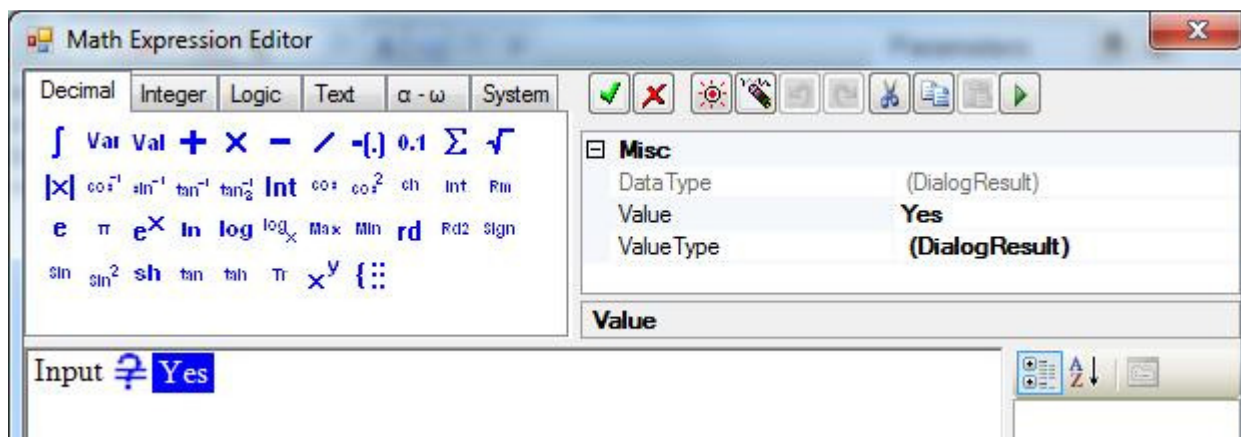
Once the condition action is linked to the message box action, its condition is automatically set to the output of the message box. We may modify the condition:



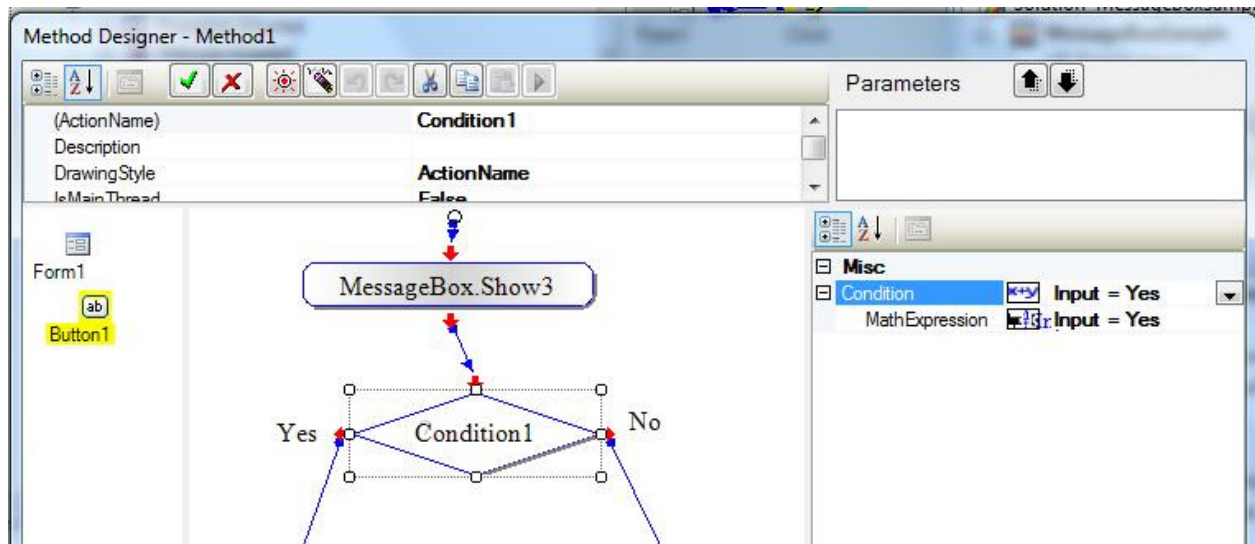
Select Yes because the message box uses Yes and No buttons for this message box action:



This is the condition for the condition action:





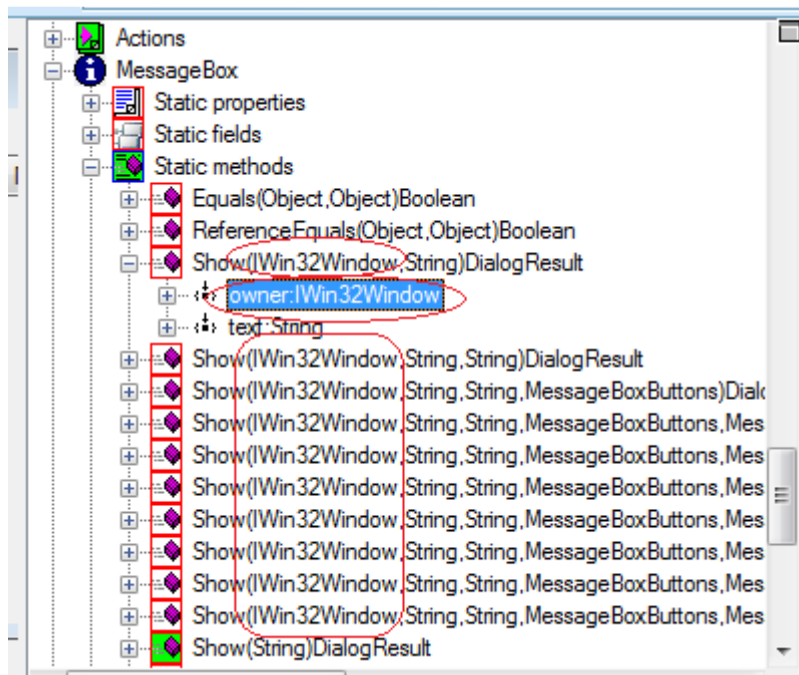


All actions to be executed when the user clicks Yes button should be linked to the Yes port of the actions.

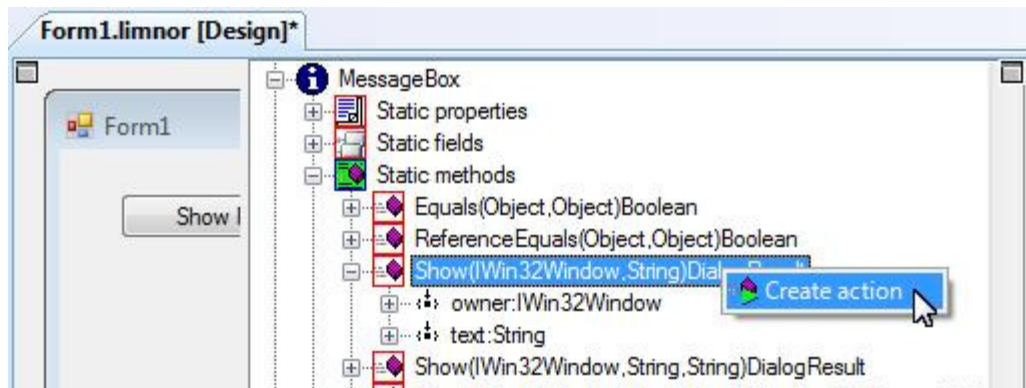
All actions to be executed when the user clicks No button should be linked to the No port of the actions.

## Use MessageBox owner

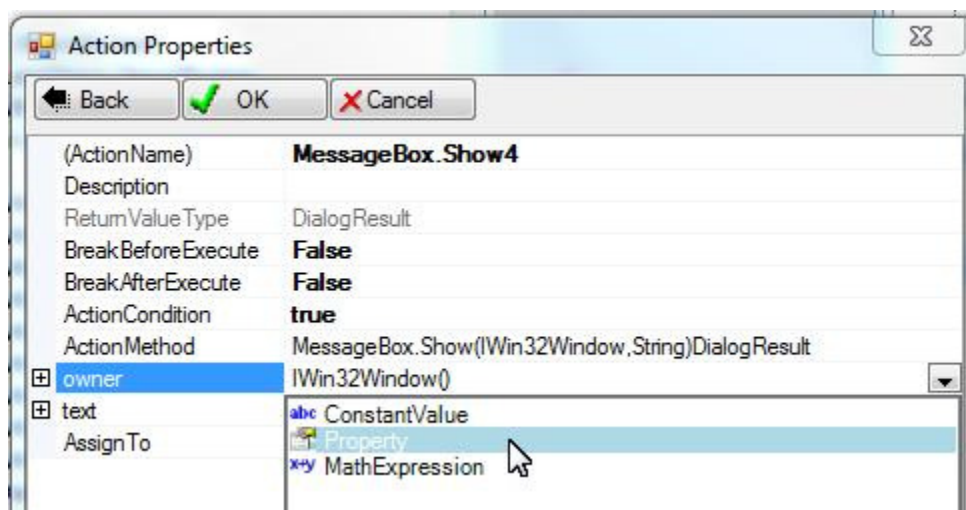
A MessageBox may have an owner if we select a method with an owner parameter:



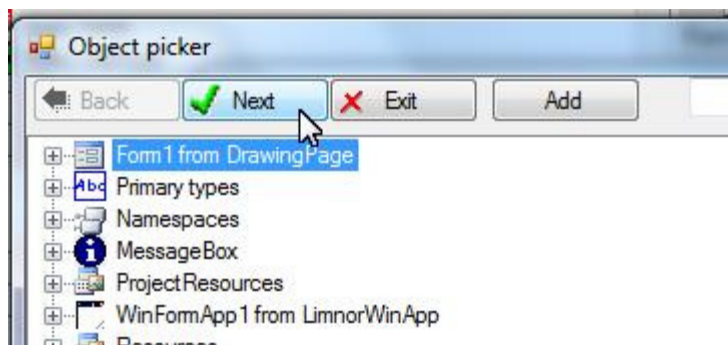
The owner is a form. If the owner is used then the message box will be owned by the form and displayed on top of the owner. Whenever it is possible an owner should be used for creating a message box action.



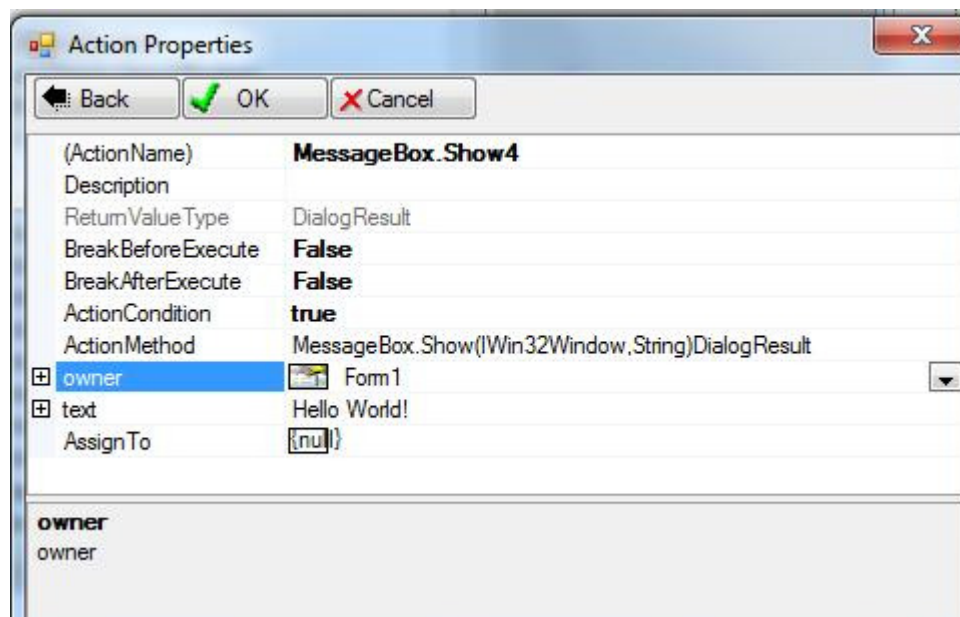
Select Property for the "owner":



Select the form as the owner:



This is a message box action using owner:

The image shows a Windows-style dialog box titled "Action Properties". At the top, there are three buttons: "Back" with a left arrow, "OK" with a green checkmark, and "Cancel" with a red X. Below these buttons is a table of properties. The first row is "(ActionName)" with the value "MessageBox.Show4". The second row is "Description" which is empty. The third row is "ReturnValue Type" with the value "DialogResult". The fourth row is "BreakBeforeExecute" with the value "False". The fifth row is "BreakAfterExecute" with the value "False". The sixth row is "ActionCondition" with the value "true". The seventh row is "ActionMethod" with the value "MessageBox.Show(IWin32Window,String)DialogResult". Below these are three expandable sections, each with a plus icon in a square. The first is "owner" with the value "Form 1" and a dropdown arrow. The second is "text" with the value "Hello World!". The third is "AssignTo" with the value "{null}". At the bottom of the dialog is a section labeled "owner" containing the text "owner".

Property	Value
(ActionName)	MessageBox.Show4
Description	
ReturnValue Type	DialogResult
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	MessageBox.Show(IWin32Window,String)DialogResult
owner	Form 1
text	Hello World!
AssignTo	{null}

**owner**  
owner