Limnor Studio – User's Guide

Part – I

Objects

Contents

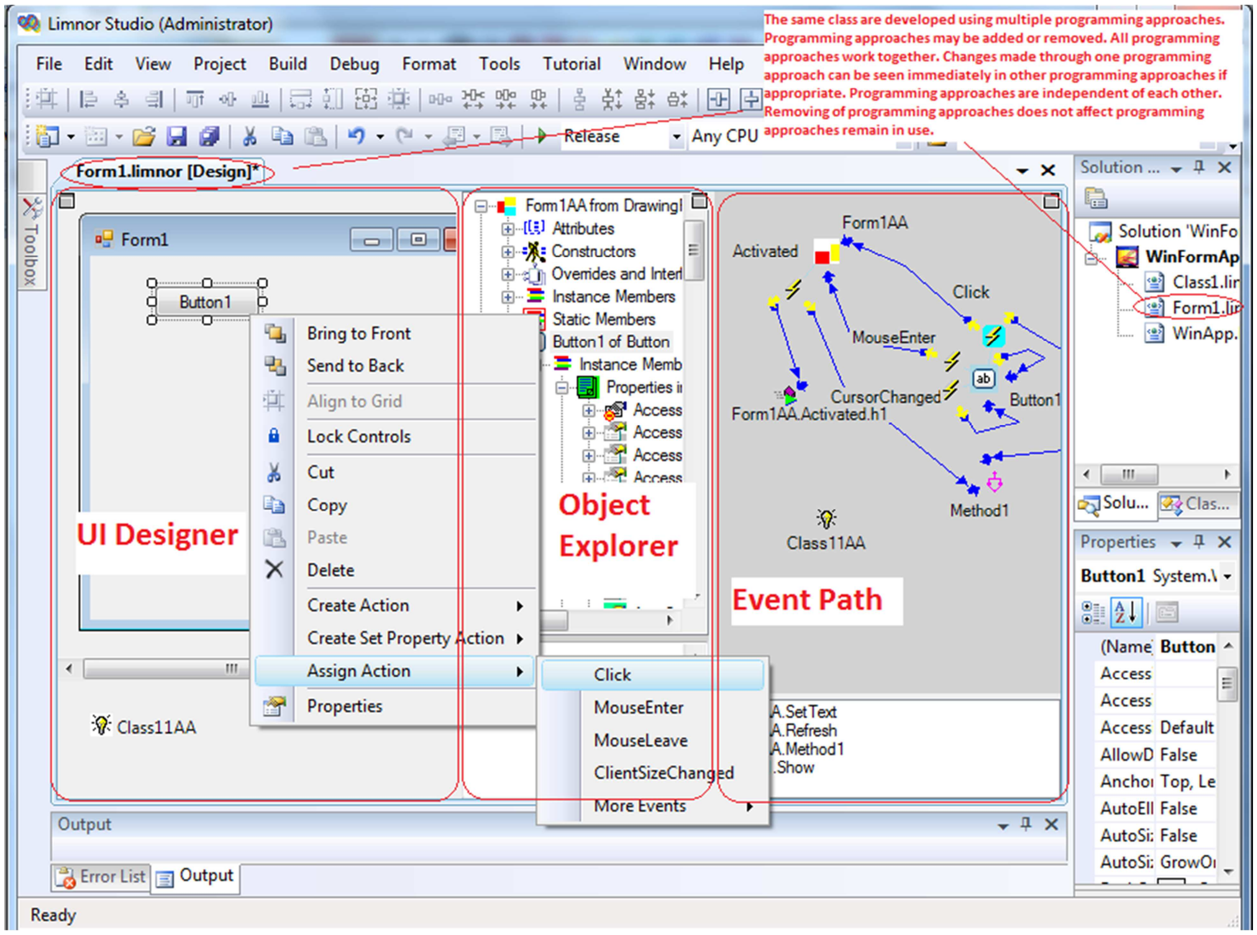
I. Introduction

Limnor Studio is a codeless visual programming environment with patented technologies. It can be used to develop computer software without using texture computer programming languages. The users do not need to learn complex syntax of computer programming languages. Computer programming is no longer a special privilege of computer programmers who have mastered one or more computer languages. Using Limnor Studio is like using an editing tool such as word processor software, spread sheet software, or other daily office software. Programming is done using concepts from daily life.

Limnor Studio does not sacrifice programming power to achieve its goal of codeless and visual programming. It does not use a proprietary format for programming entities (types). It uses directly the Microsoft .Net Framework types as its programming entities. It generates .Net types as programming results. It also generates C# source code as another form of programming results. It may seamlessly use programming results made in other .Net programming tools and languages. Other .Net programming languages may seamlessly use programming results made by Limnor Studio. For web applications it also generates JavaScript, PHP, HTML and ASPX code from your visual programming.

Limnor Studio provides additional support and simplifications in some specific areas such as database applications, 2D drawings, kiosk applications, web services, web site, etc.

Limnor Studio provides a built-in copy-protection system to protect your work against software piracy. You may add licensing in the software you developed using Limnor Studio. Users of your software have to get licenses from you in order to use your software.



The same class are developed using multiple programming approaches. Programming approaches may be added or removed. All programming approaches work together. Changes made through one programming approach can be seen immediately in other programming approaches if appropriate. Programming approaches are independent of each other. Removing of programming approaches does not affect programming approaches remain in use.

II. Programming Entity – Object

In this chapter some basic computer programming concepts are introduced. The concepts are described using concepts from daily life, not from computer science.

II.1. Everything is an object

All programming entities are called **objects**. A number is an object; a window is an object; a button on a window is an object; an image is an object; …; just everything is an object. Think about how we understand the world: by “things” - objects. A tree is an object, a person is an object, a house is an object, …, anything we can understand we understand it as a “thing”. A “thing” is referred to as an “object” in programming.

The following are some basic objects in computer:

- Integers and decimals. These are for numeric values. For example, 23.9.
- Strings. A string is formed by a sequence of letters. For example, “Hello World!”. In documentations usually a string is enclosed in quotations, but the quotations used to enclose the string are not part of string.
- Boolean. A Boolean represents logic True or False.

For a Windows Form application, following are some common objects:

- Form. A form represents one window.
- Button.
- Label
- TextBox

Every **object** in a computer is defined by 3 characteristics:

1. **Properties**. An object may need some quantities or values to describe its characteristics. For example, a Button has a Location property and a Size property to describe its location and size. A window has a Text property indicating its caption. A TextBox also has a Text property but it represents the text contents of the TextBox. Height and Weight can be properties of a person.
2. **Methods**. A method indicates one kind of things an object is capable of doing. For example, a Form has a Show method and a Hide method for showing and hiding the form. Run and Sing can be two methods of a person.
3. **Event**. An event represents that a specific thing happens to the object. For example, a button has a Click event which occurs when the user clicks the mouse on the button. A Form has a Closing event which occurs when the form is going to close. Birthday can be an event of a person.

The properties, methods and events are also called the **members** of an object.

To do programming is to create objects and put together objects to form software. Limnor Studio allows you visually manipulate objects. Limnor Studio uses a special way of putting the objects together by creating actions and assigning actions to events. Actions will be described in other chapters.

II.2. Class and Instance

The word “Human” refers to one type of living things on earth. Michael Jackson is one instance of human. Ludwig van Beethoven is another instance of human. Every one of us is an instance of human. In computer programming, “Human” is a **class** or called a **type**, every individual human being is an **instance** of the human class.

We may use term “object” to refer to both class and instance, but mostly it refers to an instance.

To create a new computer program, the first thing is to create one or more new classes. Then instances of classes may be created.

II.3. Instance Member, Static Member and Static Class

It is mentioned before that an object is defined by its members, which are Properties, Methods and Events.

“Height” can be a property of “Human”. We can say that the Height property of Michael Jackson is 5 feet and 7 inches. But we cannot give a value for the Height property of “Human”. That is, the Height property is only meaningful when it is associated with an instance. Such members are called instance members.

“Number of Legs” can be another property of “Human”. This property does not need to be associated with a specific human being (an instance of human class). Such a member is called a static member. (This example is for illustration only. In programming usually changeable data are defined as properties, static or instance. Unchangeable data usually are defined as constants)

If we define “Earth” as a class then there is only one instance of this class. When creating a computer application, an “Application” class is created to represent the application. There also could only be one instance of Application. For such special classes, to prevent invalid operation of creating more than one instance, no instance is allowed to be created. All its members are static members. Such a class is called a static class.

II.4. Class Inheritance

Suppose “Human” is a class. “Maya” can be a certain type of “Human”. We say “Maya” is a class derived from “Human” class. “Maya” class inherits all members of “Human” class. “Maya” class may have its own members not defined in “Human” class. For example, “Head Flattened” can be a property of “Maya” class for indicating whether the head was flattened in childhood. “Head Flattened” property does not exist in the “Human” class.

Suppose “Human” class has properties “Height” and “Weight”. Then “Maya” class also has “Height” and “Weight” because “Maya” class inherits all members from “Human” class. “Maya” class does not have to re-define “Height” and “Weight” properties.

To do programming is to create new classes derived from existing classes. An existing class that has most of the members we want the new class to have should be chosen and then new members are added to the new class.

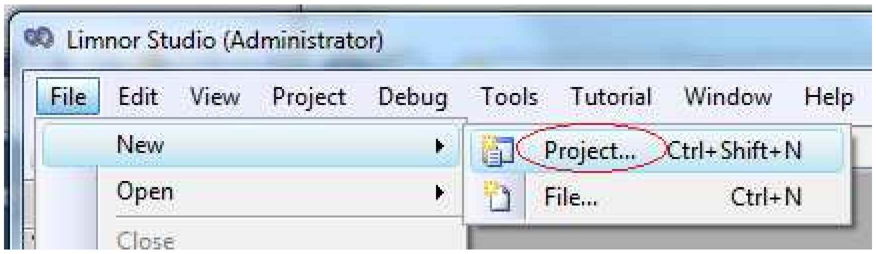
II.5. Components of a Class

A class may contain other objects. For example, a person class may contain a nose object. The contained object is also called a component; it must be an instance of a class.

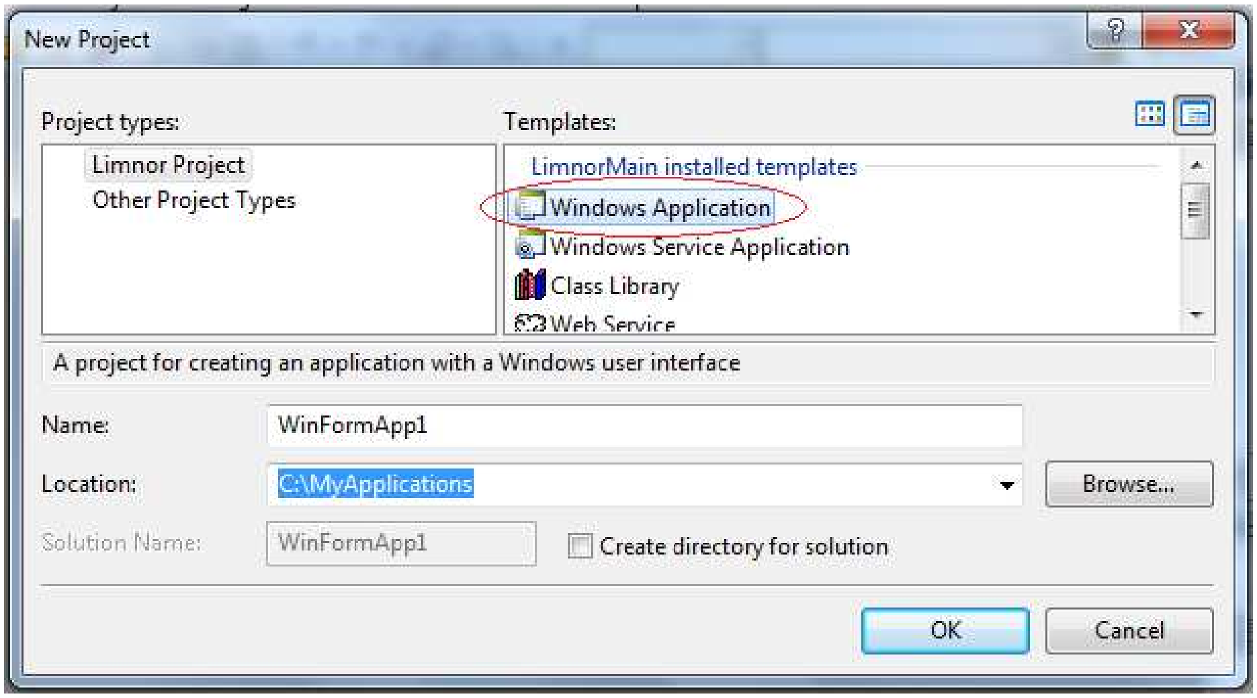
II.6. Create the first program

We now create a program using Limnor Studio and see how the concepts we introduced previously are materialized: **object**, **class**, **instance**, **component**, **member**, **property**, **static class**, and **inheritance/derive**.

Create a new project:



Let’s choose “Windows Application” for creating an executable program (EXE file) which may have windows for graphic user interfaces:



A project for creating an application with a Windows user interface

Name: WinFormApp1

Location: C:\MyApplications

Solution Name: WinFormApp1

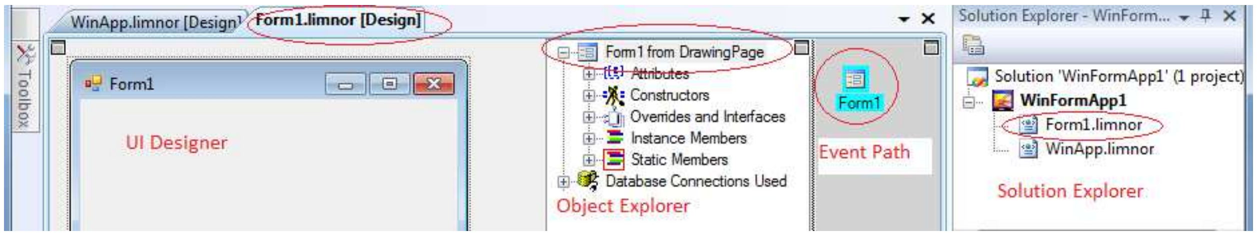
Create directory for solution

A project named WinFormApp1 is created. Two new classes are created: Form1 and WinFormApp1.

All project files for classes are listed in the **Solution Explorer** under the project. In this example, the project is WinFormApp1. Under WinFormApp1 we can see Form1.limnor and WinApp.limnor, each is a file for a new class.

Each class may be developed via many designers. Current version of Limnor Studio ships with 3 designers. UI Designer is for developing graphic user interface. Object Explorer shows the hierarchy structure of the class and its members. Event Path shows action-event relationships.

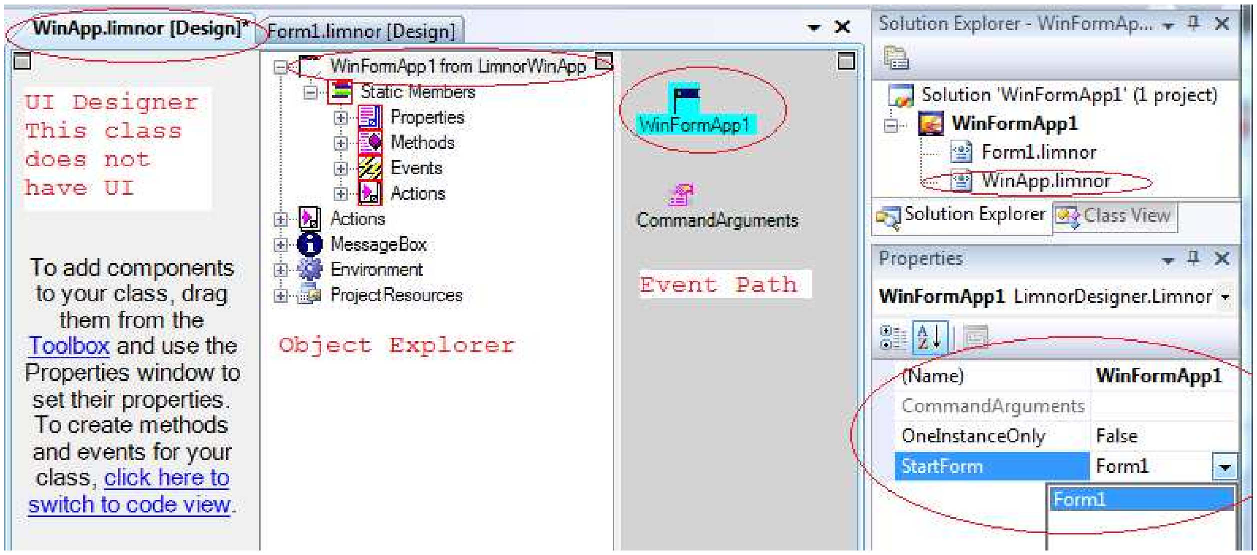
Let’s examine class Form1 first.



The root node of the Object Explorer shows "Form1 from DrawingPage", which means class Form1 is **derived** from an existing class named DrawingPage.

UI Designer shows that Form1 is an empty window.

Let's examine the class, WinFormApp1.


WinApp.limnor [Design]*
Form1.limnor [Design]
UI Designer This class does not have UI
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.
WinFormApp1 from LimnorWinApp
Static Members
Properties
Methods
Events
Actions
Actions
MessageBox
Environment
ProjectResources
Object Explorer
WinFormApp1
CommandArguments
Event Path
Solution Explorer - WinFormAp...
Solution 'WinFormApp1' (1 project)
WinFormApp1
Form1.limnor
WinApp.limnor
Solution Explorer
Class View
Properties
WinFormApp1 LimnorDesigner.Limnor
(Name) WinFormApp1
CommandArguments
OneInstanceOnly False
StartForm Form1
Form1

Class WinFormApp1 represents the application itself. It has 3 **properties**:

- CommandArguments – It is a string array representing the command arguments. For example, if the application is started with command "WinFormApp1.exe 123 abc" then this property is a 2 dimension array; the first array item is "123" and the second item is "abc".
- OneInstanceOnly – it is a Boolean value indicating whether only one instance of the application can be running.
- StartForm – This property refers to an instance of a Form class. This Form instance will be used as the first form to be displayed when this application starts. By default, this property is set to the default instance of class Form1. "Default instance" is a new concept we have not introduced yet.

Default form instance

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 technical 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.

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

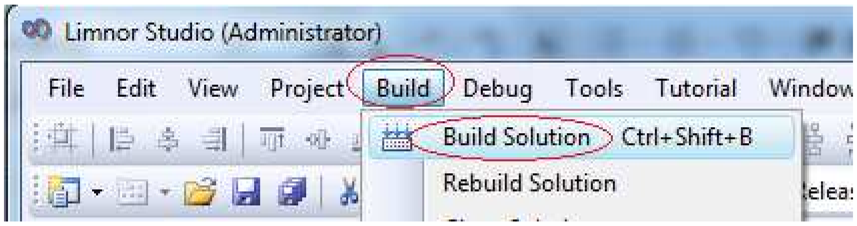
We see that Form1 is used as the StartForm for WinFormApp1. Note that Form1 is a name for a class. Here Form1 actually refers to the **default instance** of Form1, not class Form1.

For more information on using default instance, see
http://www.limnor.com/support/UseDefaultFormInstance.pdf

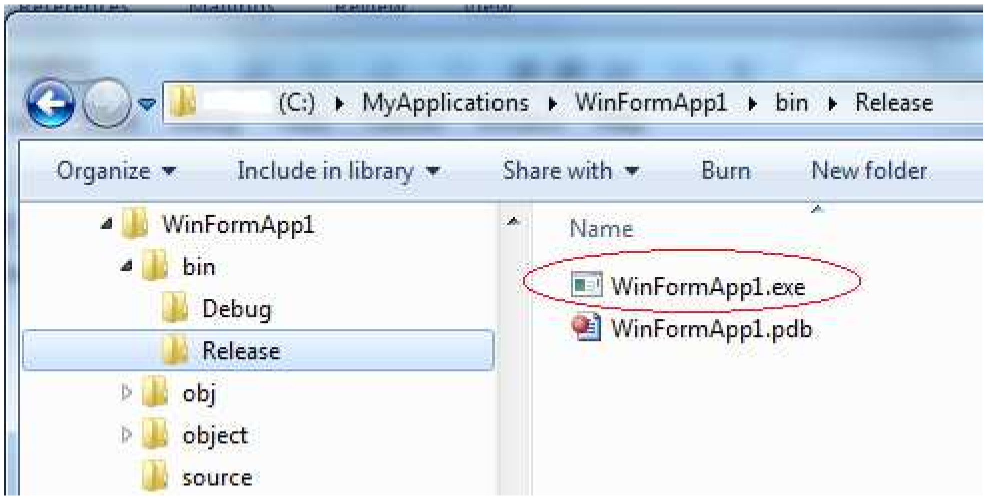
WinFormApp1 is a **static class** because it represents the application itself, and no instances are allowed.

Let's compile the application and run it.

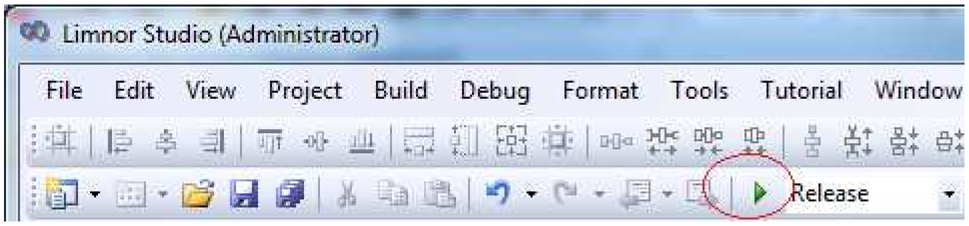
Choose menu "Build Solution" to compile the application.



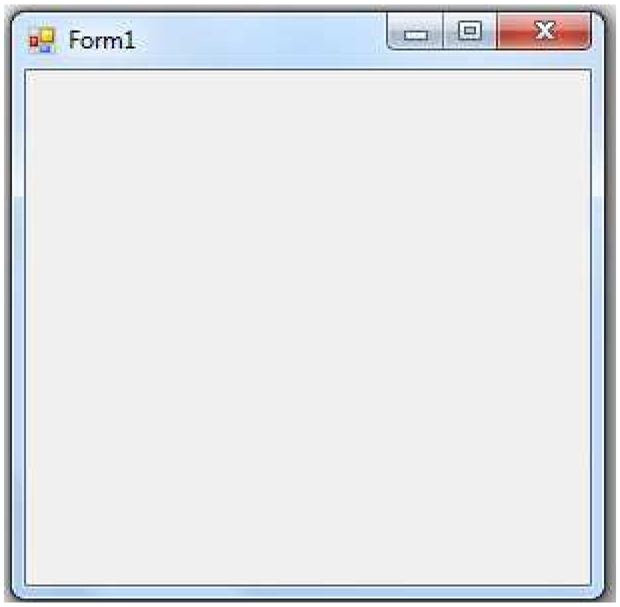
The compilation generates WinFormApp1.exe:



Double-click file WinFormApp1.exe to run it. Or click run button ▶ to compile and debug the project:

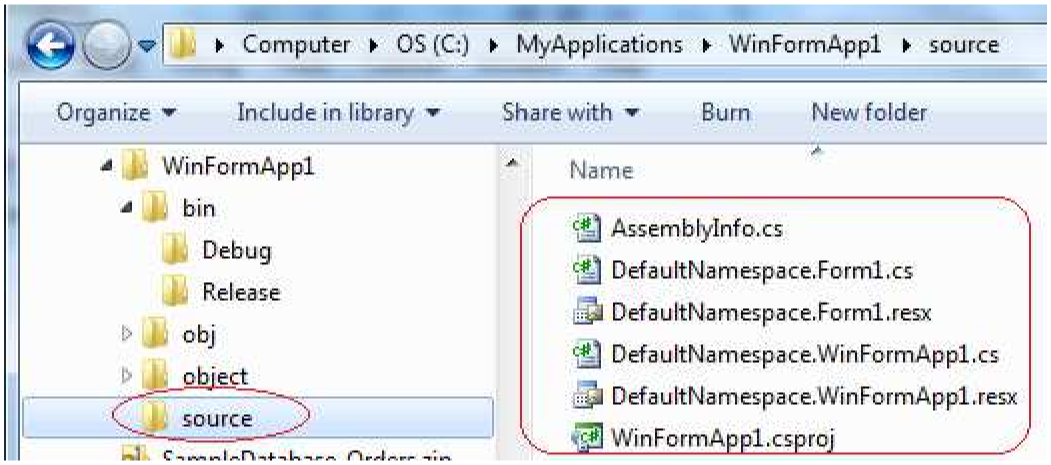


A window appears:



This is the default instance of class Form1.

The compilation also generates C# source code for the application:


Computer ▸ OS (C:) ▸ MyApplications ▸ WinFormApp1 ▸ source
Organize
Include in library
Share with
Burn
New folder
WinFormApp1
bin
Debug
Release
obj
object
source
Name
AssemblyInfo.cs
DefaultNamespace.Form1.cs
DefaultNamespace.Form1.resx
DefaultNamespace.WinFormApp1.cs
DefaultNamespace.WinFormApp1.resx
WinFormApp1.csproj

If you want your application to run under Linux or other non-Windows platforms then you may take these source files and compile those files using MONO compilers for corresponding platforms. See http://www.mono-project.com/Main_Page

III. Visual Programming via Actions and Events

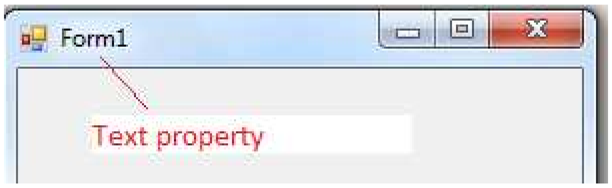
To let computer do something is to create actions and execute actions.

There are 3 types of actions.

- Method execution action – such an action executes a method of an object.
- Set property action – such an action sets value of a property of an object.
- Fire-event action – such an action generates an event.

III.1. Use Set Property Action

The Form class has a Text property, which is the caption of the form:

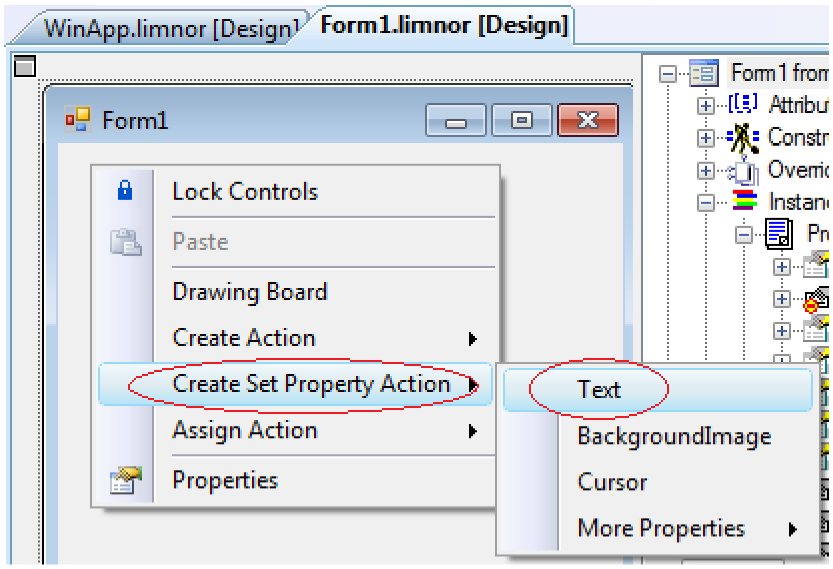


We may create a Set-Property action to change the Text property of the form at runtime.

All 3 designers, UI Designer, Object Explorer, and Event Path, can be used to create set-property actions. You may choose to use any one of them to do it. We show how each designer is used to do it.

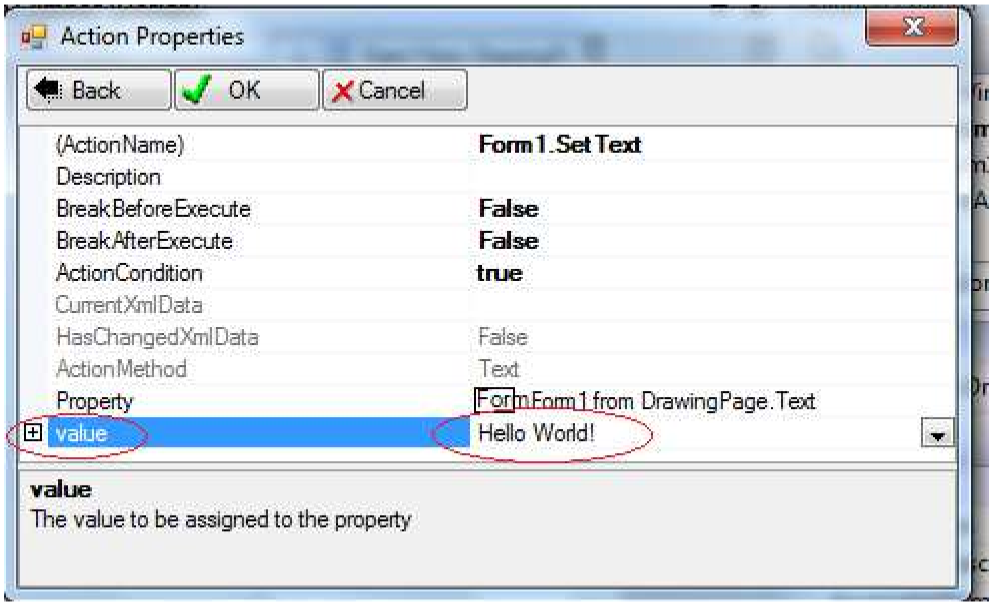
III.1.1. Via UI Designer

Right-click on class Form1 in the UI Designer, choose “Create Set Property Action”, choose “Text” property.



In this operation, we right-click on Form1 because Form1 is the owner of the property we want to set. When "Create Set Property Action" is selected, the most commonly used properties are displayed. If the property we want to set is not in the list then we may choose "More Properties" to find the property we want to use.

A dialogue box appears to allow us to specify the value for the property. In this example, we simply give a constant string "Hello World!"

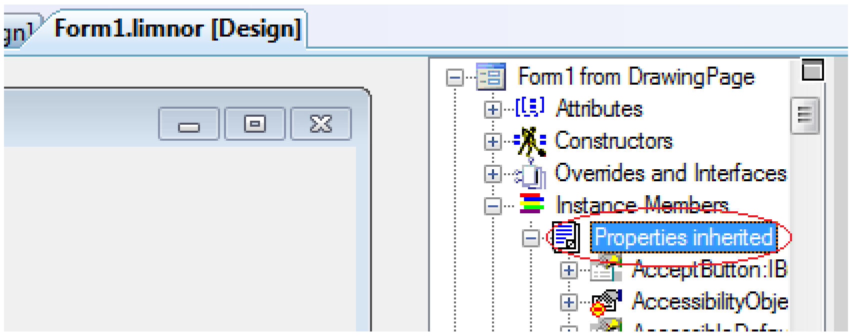


(ActionName)	Form1.SetText
Description	
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
CurrentXmlData	
HasChangedXmlData	False
ActionMethod	Text
Property	FormForm1 from DrawingPage.Text
value	Hello World!

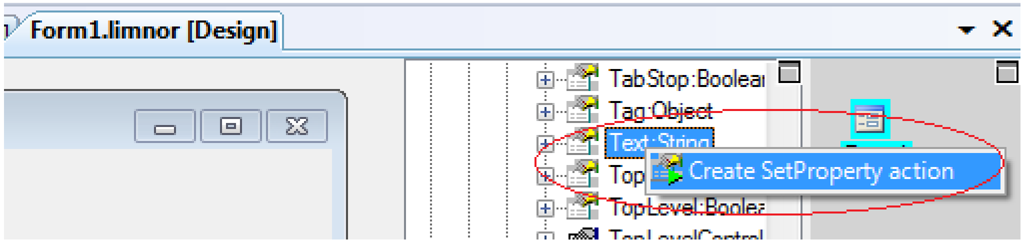
value
The value to be assigned to the property

III.1.2. Via Object Explorer

Expand "Properties inherited" node under Form1 in the Object Explorer:



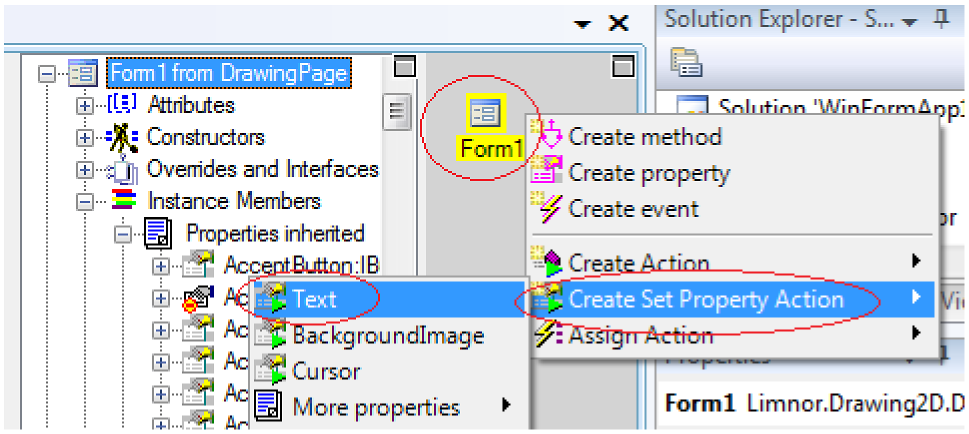
Scroll down to find Text property. Right-click it and choose “Create SetProperty action”:



As via UI Designer, “Action Properties” dialogue box appears to specify the value for the property.

III.1.3. Via Event Path

Right-click icon Form1; choose “Create Set Property Action”; and choose Text property:

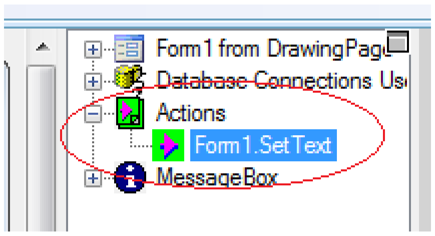


As via UI Designer, “Action Properties” dialogue box appears to specify the value for the property.

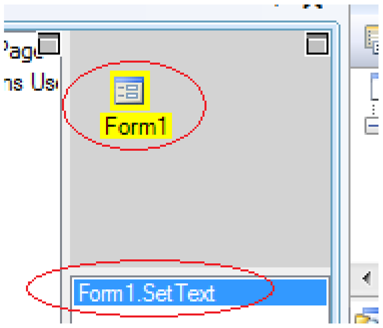
III.1.4. View Actions

All actions created can be found in Object Explorer and Event Path.

In the Object Explorer, all public actions are listed under Actions node:



In the Event Path, select an icon, all actions belonging to that icon are displayed in the list:



In this example, icon Form1 represents class Form1. Action Form1.SetText is to set the Text property of class Form1 and thus the action executer is Form1. So, this action belongs to the icon Form1.

III.2. Assign Actions to Events

We have created an action to change the caption of class Form1 to “Hello World!” The question now is that how this action get executed? When to execute this action?

The answer is to pick an event or events as the times to execute this action.

An event of a class represents a special occasion when a specific thing happens on the class. For example, we may define “Reach Driving Age” as an event for the “Human” class. For a kid in United States this event occurs on the day when the kid reaches 15 and half. For a kid in another nation this event may occur at another age.

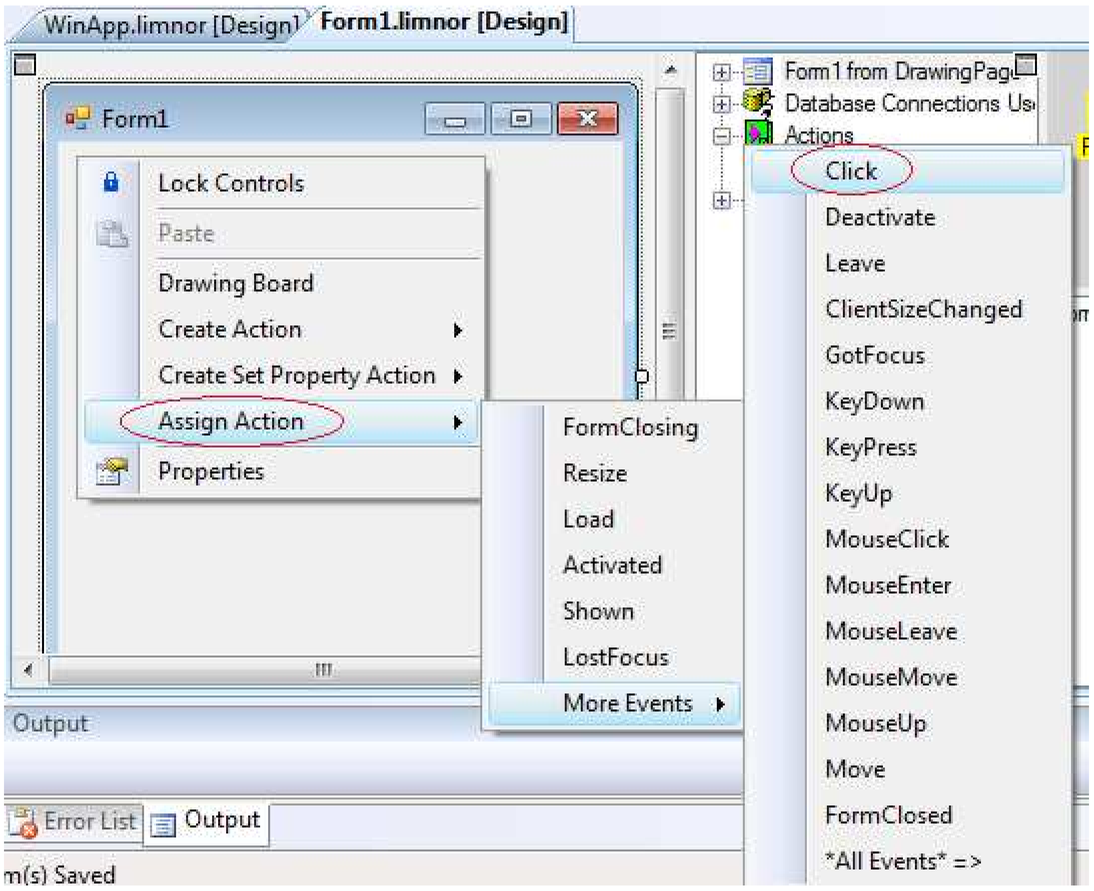
For a Form class, many events are defined for programming purpose. For example, “Resize” event occurs when the size of the Form is changed; “MouseMove” event occurs when the mouse pointer moves over the form; “Click” event occurs then the user clicks the mouse while the mouse pointer is over the form.

Suppose we want the action Form1.SetText to be executed when Click event of the Form occurs. We may assign this action to the Click event of Form1.

All 3 designers, UI Designer, Object Explorer, and Event Path, allow us to assign actions to events. You may choose one of them to do it. We show how each designer is used to do it.

III.2.1. Via UI Designer

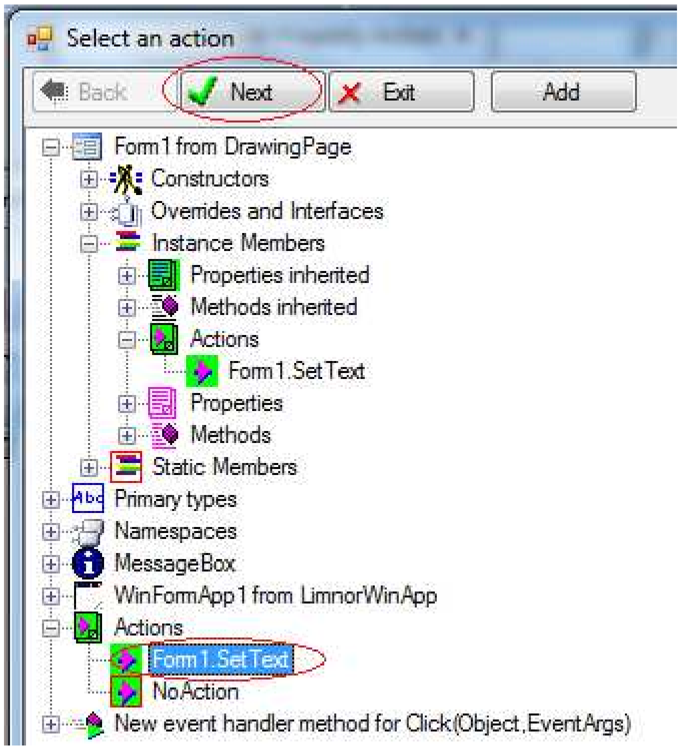
Right-click Form1, choose “Assign Actions”, choose “Click” event:



If the event we want to use is not in the menu then we may click "*All Events*=>" to find the event.

Once we select the event we want to use, a dialogue box appears to let us choose the actions to be assigned to the event.

Select the actions from the Actions node, click Next:

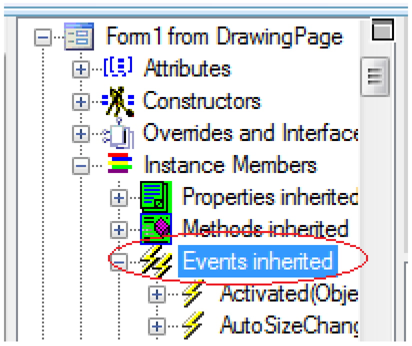


Form1 from DrawingPage
Constructors
Overrides and Interfaces
Instance Members
Properties inherited
Methods inherited
Actions
Form1.SetText
Properties
Methods
Static Members
Primary types
Namespaces
MessageBox
WinFormApp1 from LimnorWinApp
Actions
Form1.SetText
NoAction
New event handler method for Click(Object,EventArgs)

We are done assigning the action to the event.

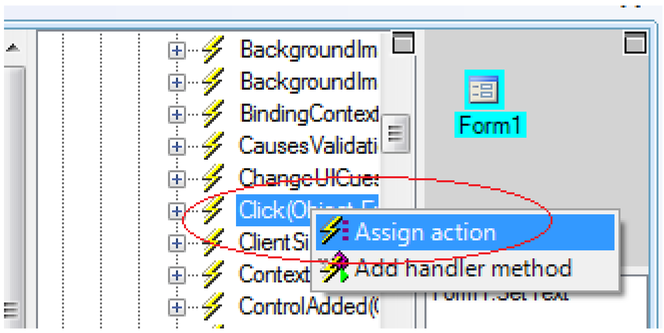
III.2.2. Via Object Explorer

To do "Assign actions to event" operation in the Object Explorer, first expand the "Events inherited" node of the class owning the event we want to use:



Form1 from DrawingPage
Attributes
Constructors
Overrides and Interface
Instance Members
Properties inheritec
Methods inherited
Events inherited
Activated(Obje
AutoSizeChan

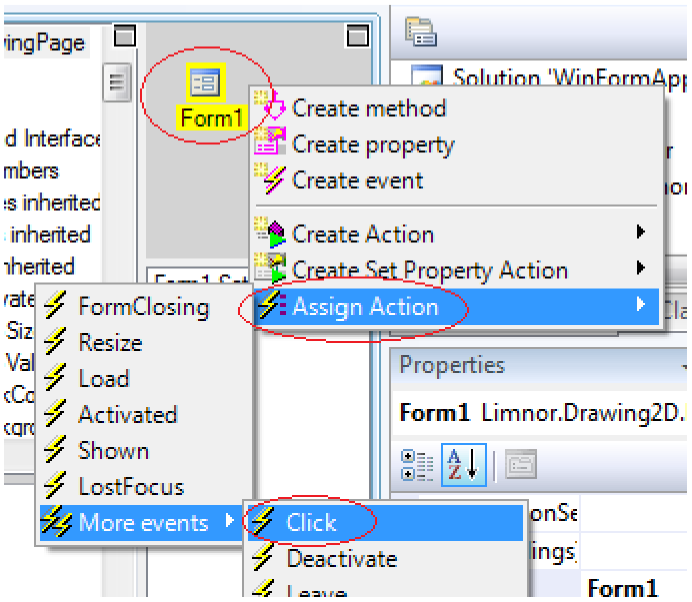
Scroll down to find "Click" event, right-click it and choose "Assign action"



As via UI Designer, Action Selection dialogue box appears to select actions to be assigned to the event.

III.2.3. Via Event Path

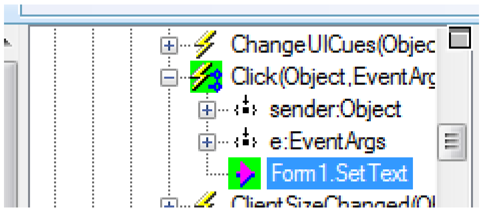
To do "Assign actions to event" operations in the Event Path, right-click the icon representing the class owning the event we want to use; choose "Assign Action"; choose the Click event:



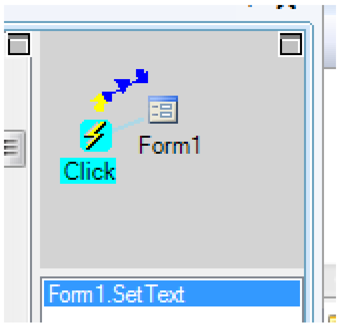
As via UI Designer, Action Selection dialogue box appears to select actions to be assigned to the event.

III.2.4. View Action-Event Relationships

In the Object Explorer, actions assigned to an event are displayed under the event:

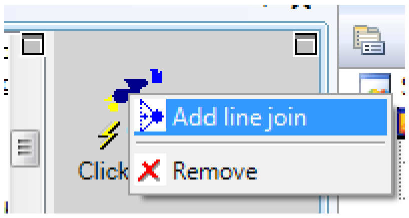


In the Event Path, a link line is drawn from the event to the action executer:

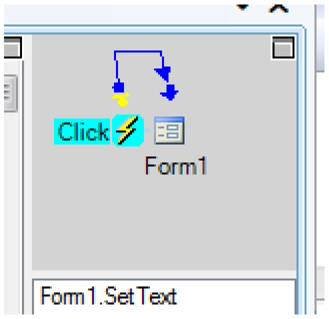


In this example, the action executer of the action Form1.SetText is Form1. So, a line is drawn from the Click event to icon Form1.

By default a straight line is drawn from the event to the action executer. More line segments can be created by right-clicking the line and choose “Add line join”:

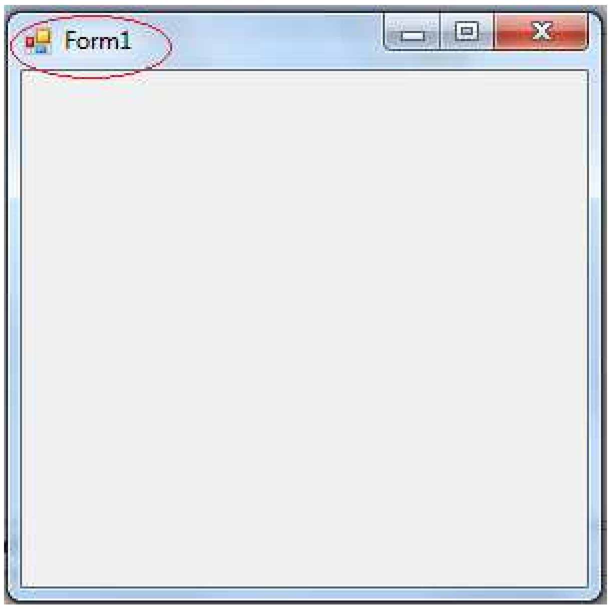


For example, we add two line-joins to the line and re-arrange the icons to make it look better (maybe):

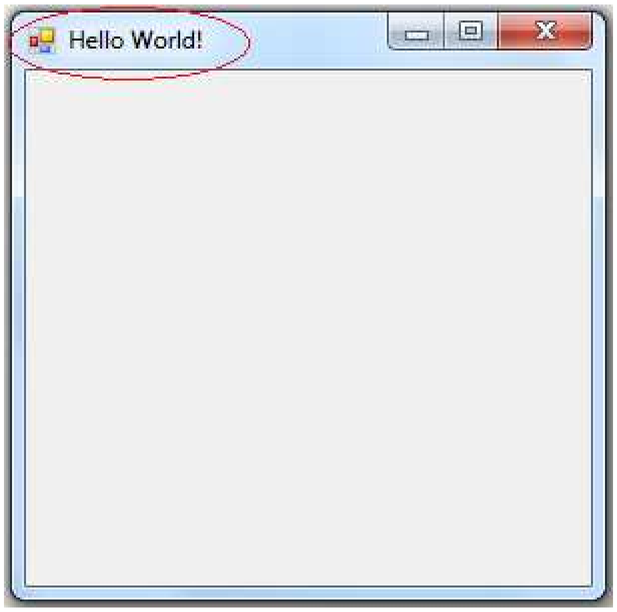


III.2.5. Test Action Execution and Debug

Actions only can be executed at runtime. Click run button ▶ to compile and debug the project. Form1 appears:

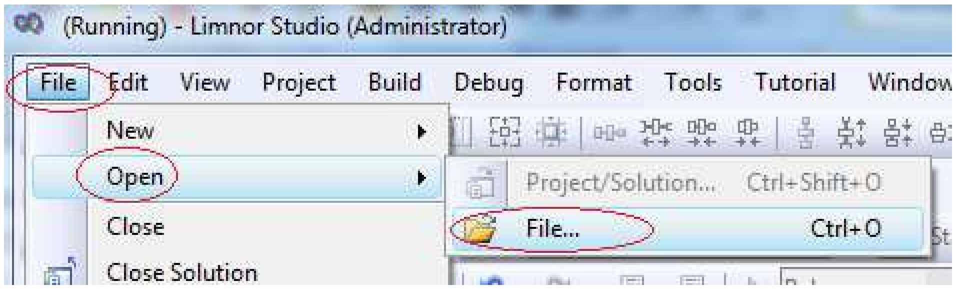


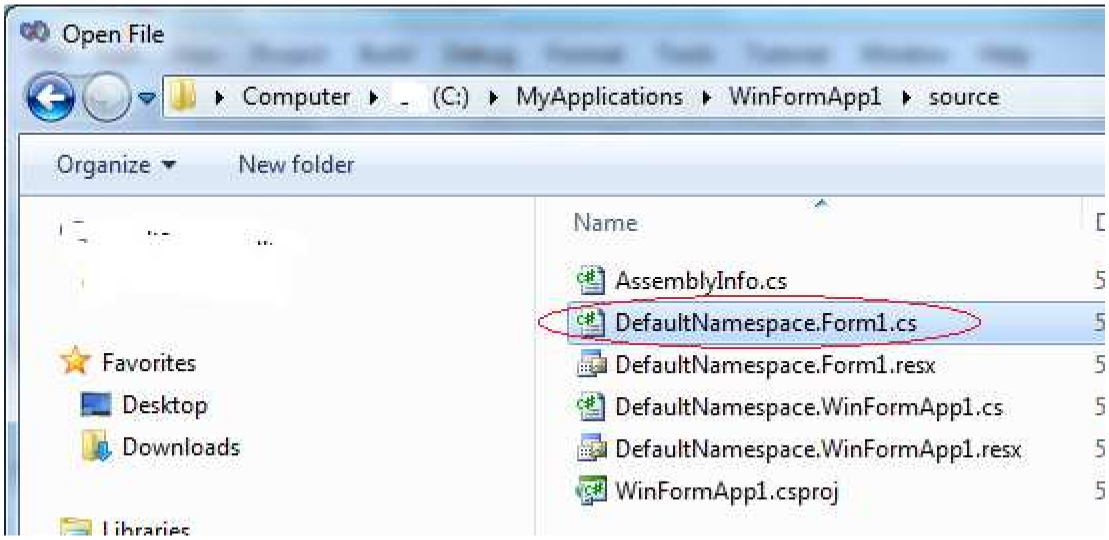
Note that the caption is “Form1” which is the value of the Text property used at the design time. Now click the mouse on the form. Notice that the caption becomes “Hello World!” indicating that the action is executed:

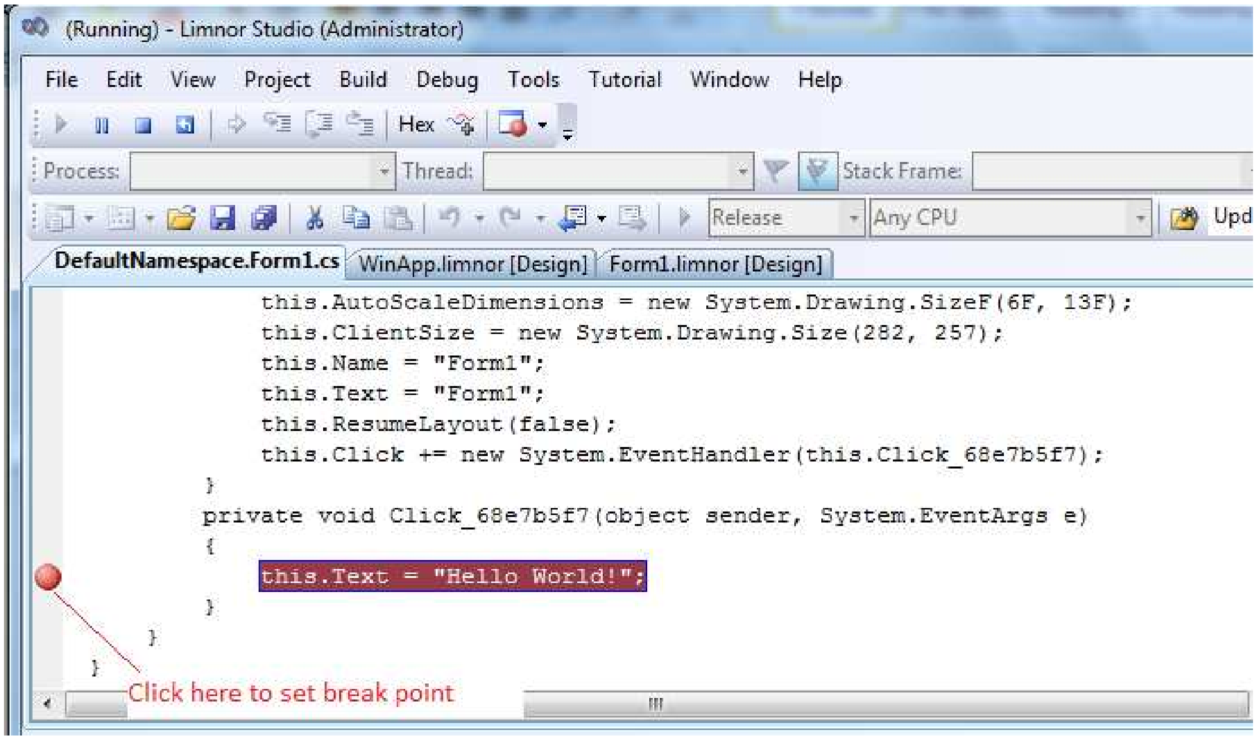


Because C# source code files are created by the compilation, we may use these files to do debugging if needed. If you do not want to use C# to do debugging then you may skip this section. We will provide visual debugging without using C# in other documents.

While Form1 is still displayed, load the C# source code file for class Form1:

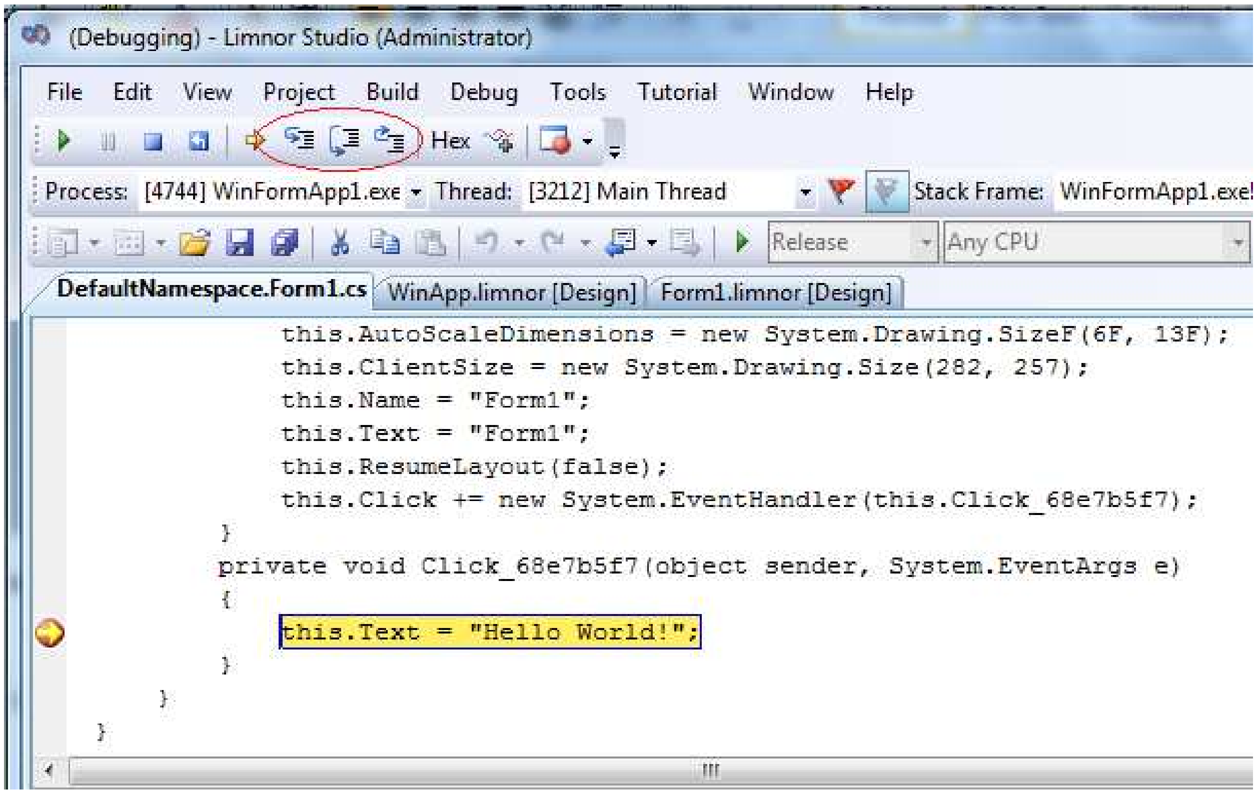






```
                this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
                this.ClientSize = new System.Drawing.Size(282, 257);
                this.Name = "Form1";
                this.Text = "Form1";
                this.ResumeLayout(false);
                this.Click += new System.EventHandler(this.Click_68e7b5f7);
            }
            private void Click_68e7b5f7(object sender, System.EventArgs e)
            {
                this.Text = "Hello World!";
            }
        }
    }
```

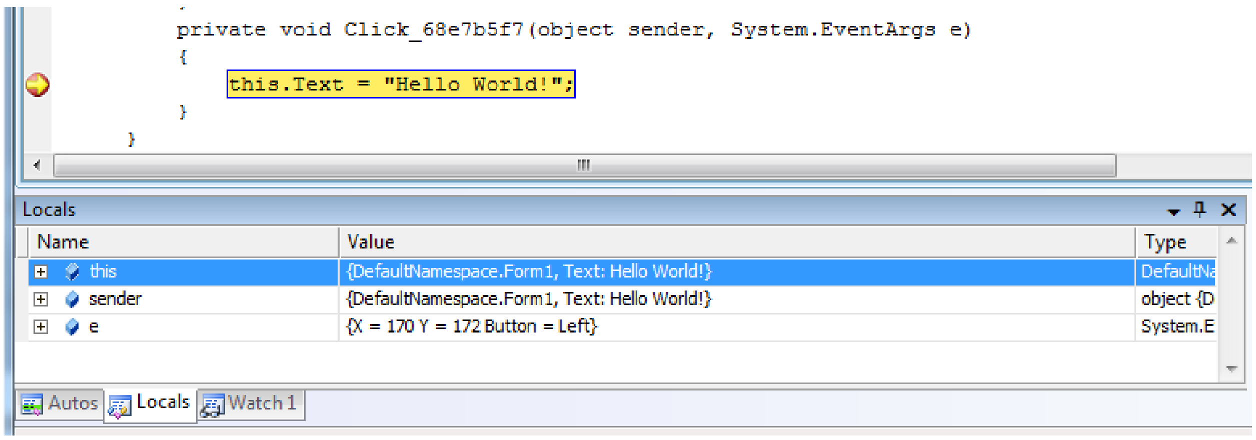
Now click on Page1, the execution will stop at the break point:


(Debugging) - Limnor Studio (Administrator)
File Edit View Project Build Debug Tools Tutorial Window Help
Hex
Process: [4744] WinFormApp1.exe
Thread: [3212] Main Thread
Stack Frame: WinFormApp1.exe!
Release
Any CPU
DefaultNamespace.Form1.cs
WinApp.limnor [Design]
Form1.limnor [Design]

```
                this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
                this.ClientSize = new System.Drawing.Size(282, 257);
                this.Name = "Form1";
                this.Text = "Form1";
                this.ResumeLayout(false);
                this.Click += new System.EventHandler(this.Click_68e7b5f7);
            }
            private void Click_68e7b5f7(object sender, System.EventArgs e)
            {
                this.Text = "Hello World!";
            }
        }
    }
```

Debug buttons can be used to step into the execution.

Variables can be examined:



```
        private void Click_68e7b5f7(object sender, System.EventArgs e)
        {
            this.Text = "Hello World!";
        }
    }
```

Locals

Name	Value	Type
this	{DefaultNamespace.Form1, Text: Hello World!}	DefaultNa
sender	{DefaultNamespace.Form1, Text: Hello World!}	object {D
e	{X = 170 Y = 172 Button = Left}	System.E

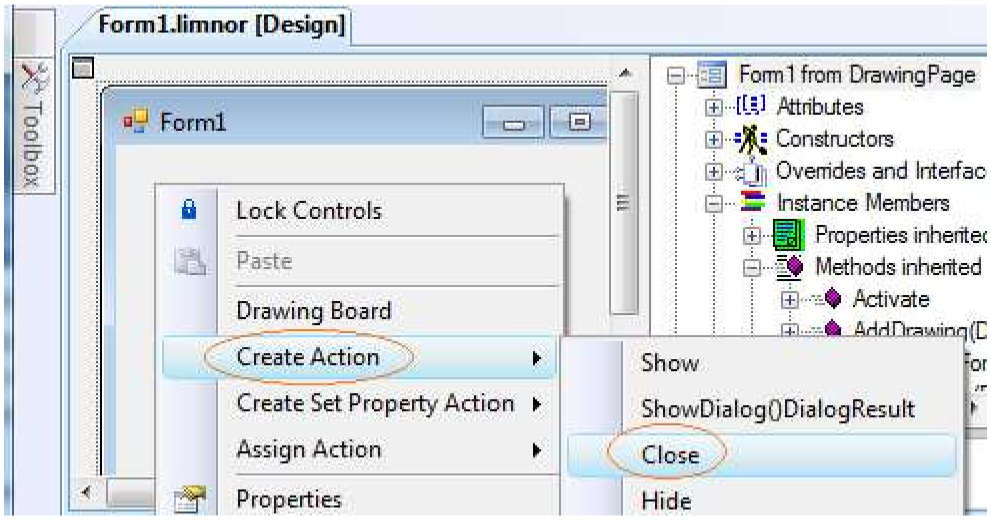
III.3. Use Method Execution Action

A method of an object represents a work capability of the object. To use this capability, create an action using the method and assign the action to an event.

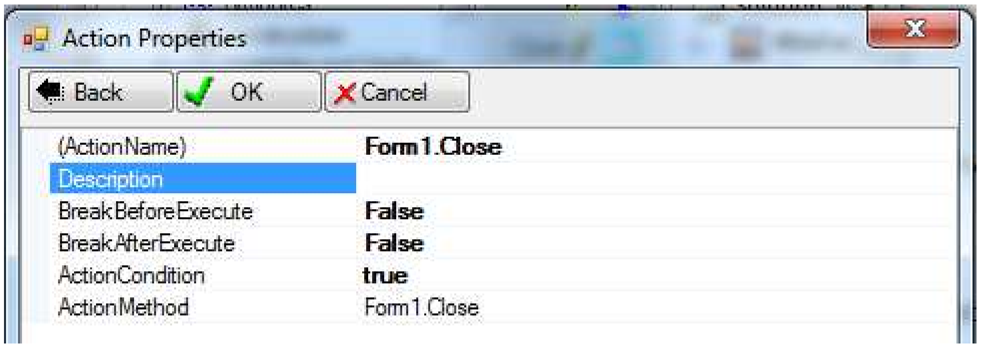
For example, a Form object has a Close method. This method closes the form object and unloads the related resources from the memory. We use this method as an example to show how to create a method execution action.

III.3.1. Create Action via UI Designer

Right-click the form, choose “Create action”. Choose “Close” method:



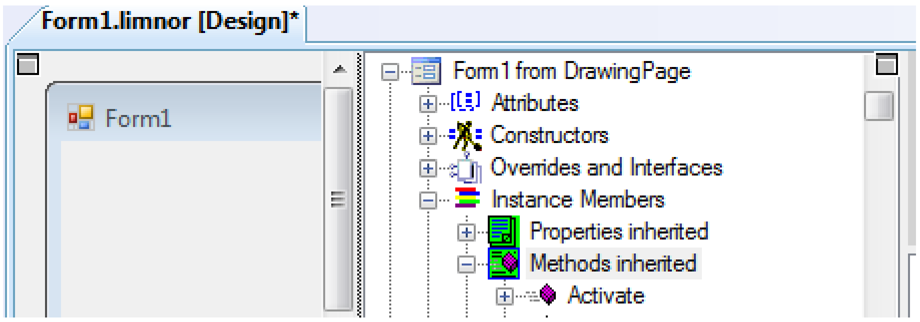
The action does not need parameters. Simply click OK to finish creating the action:



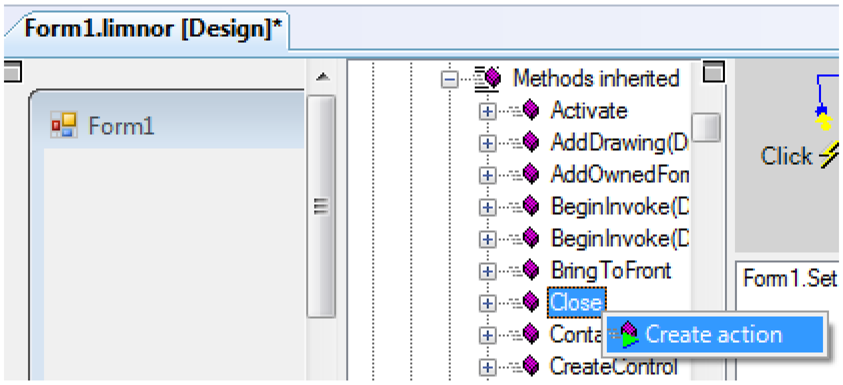
(ActionName)	Form1.Close
Description	
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	Form1.Close

III.3.2. Create Action via Object Explorer

Expand the “Methods inherited” node of the Form object



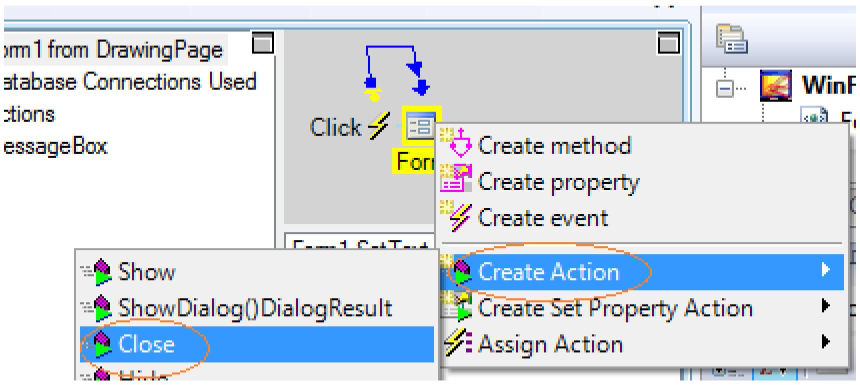
Scroll down and find “Close” method. Right-click it and choose “Create action”:



As creating action via UI Designer, the Action Dialogue box appears. Click OK to finish creating the action.

III.3.3. Create Action via Event Path

Right-click Form1 icon, choose “Create Action”. Choose “Close” method:



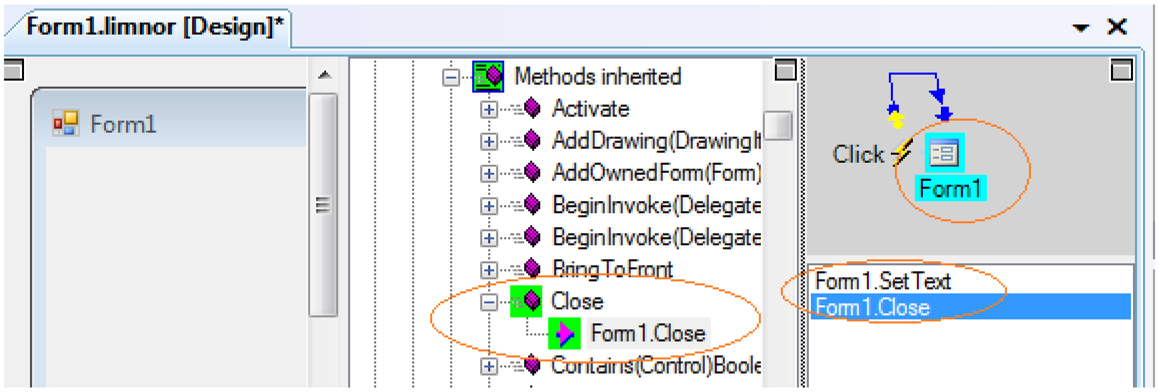
As creating action via UI Designer, the Action Dialogue box appears. Click OK to finish creating the action.

III.3.4. View Actions

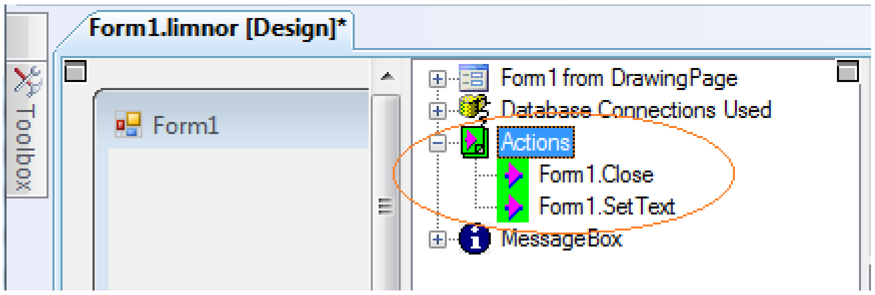
All actions created can be found in Object Explorer and Event Path.

In the Event-Path, select icon Form1, all actions executed by class Form1 are displayed in the list.

In the Object Explorer, Close action is displayed under the Close Method.

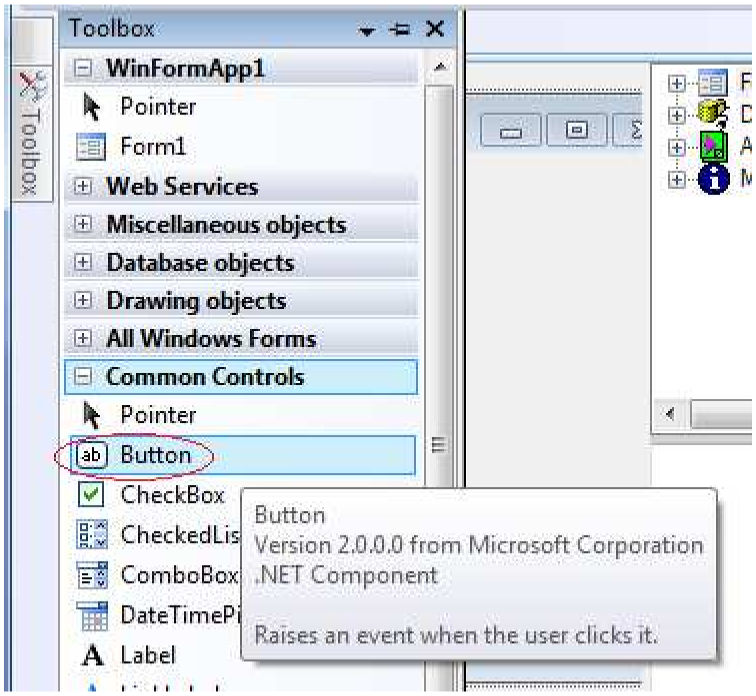


All actions created for the class are displayed under Actions in the Object Explorer:

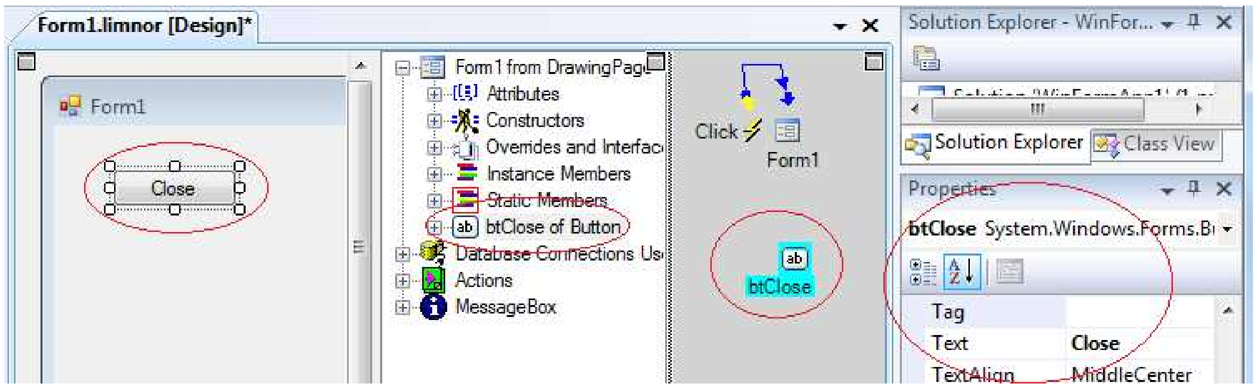


III.4. Assign Actions to Events

We want to use a button to execute the Close action. Drop a Button to the form:



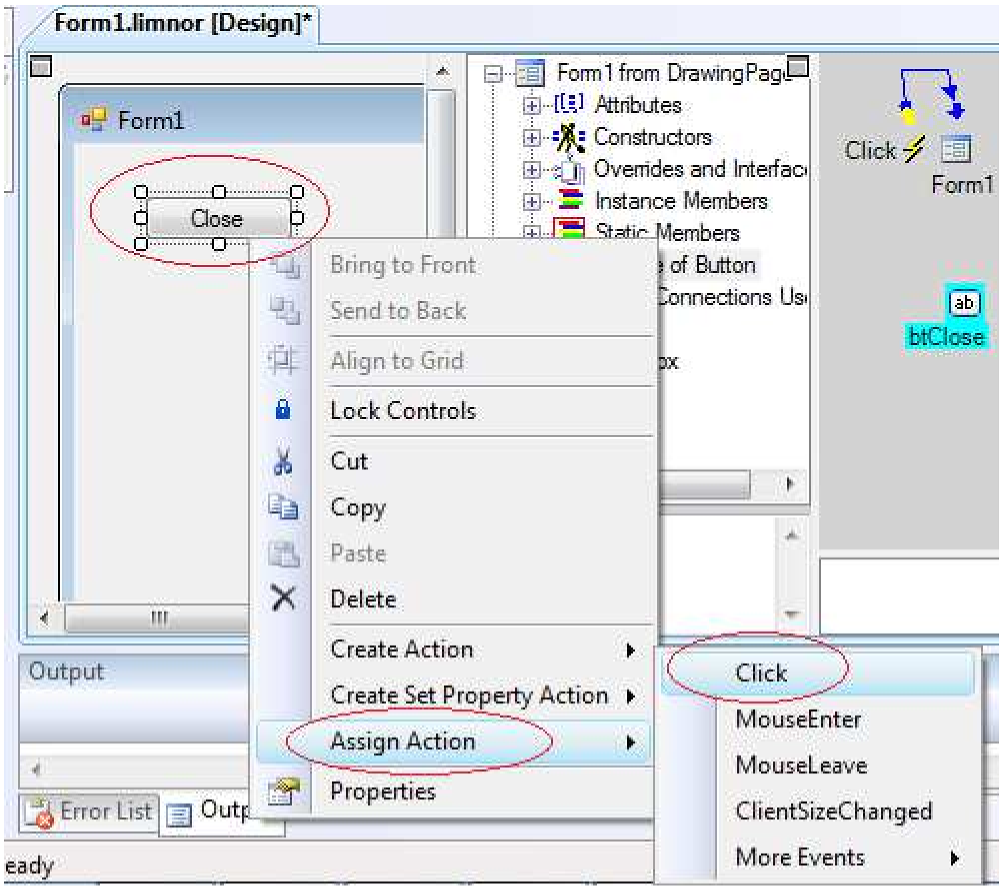
We name the button btClose, and set its Text property to "Close":



We assign the Close action to the Click event of the button. It can be done in all designers. We will show how it is done using UI Designer, Object Explorer and Event Path.

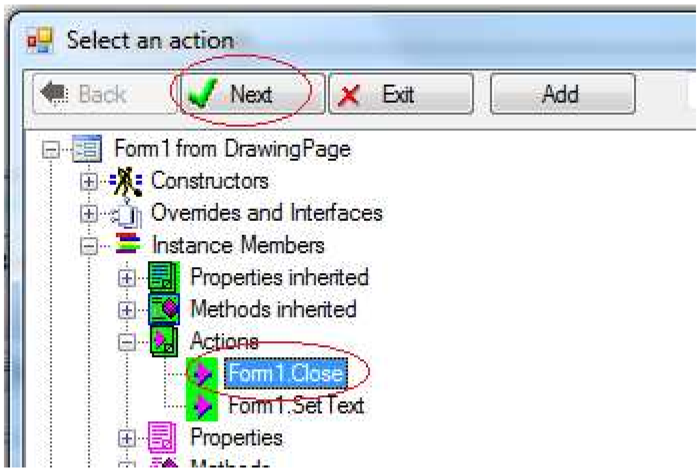
III.4.1. Via UI Designer

Right-click the button, choose "Assign action". Choose "Click" event.



The menu lists commonly used events. If the event we want to use is not in the menu then we may choose "More Events" to find the event.

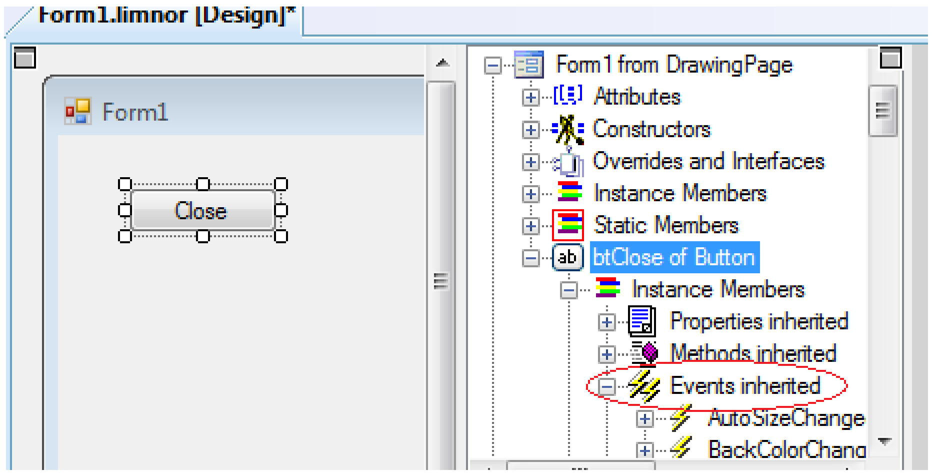
Select the action and click Next:



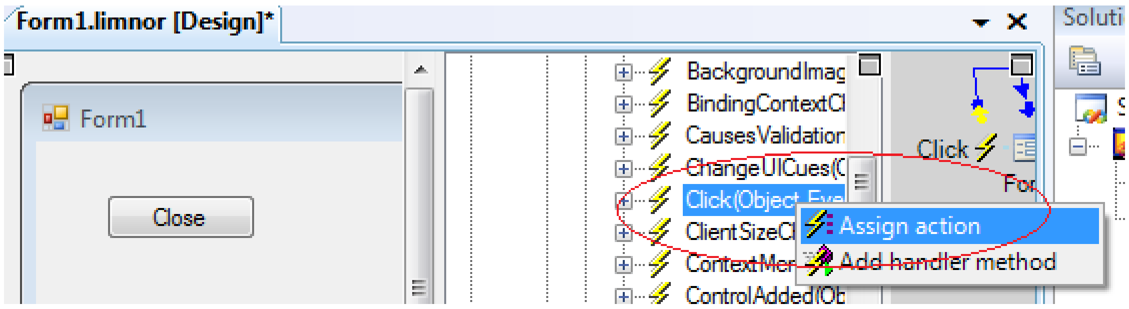
More than one action can be selected if we want. Many actions can be assigned to one event.

III.4.2. Via Object Explorer

Expand the "Events inherited" node of the button:



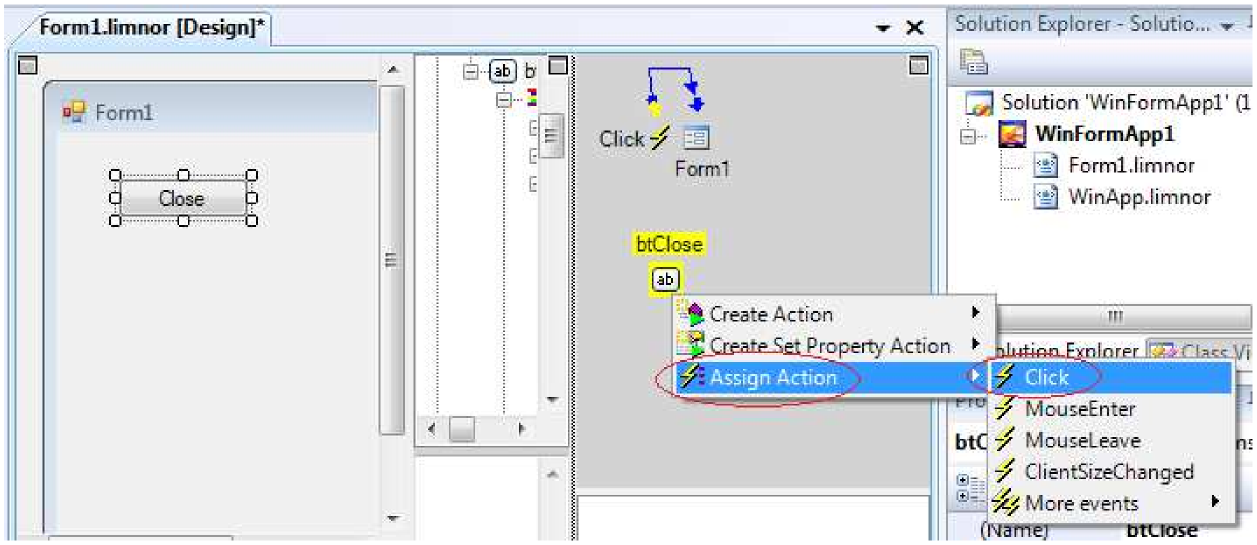
Scroll down and find Click event:



As assigning action using UI Designer, select the Close action and click Next.

III.4.3. Via Event Path

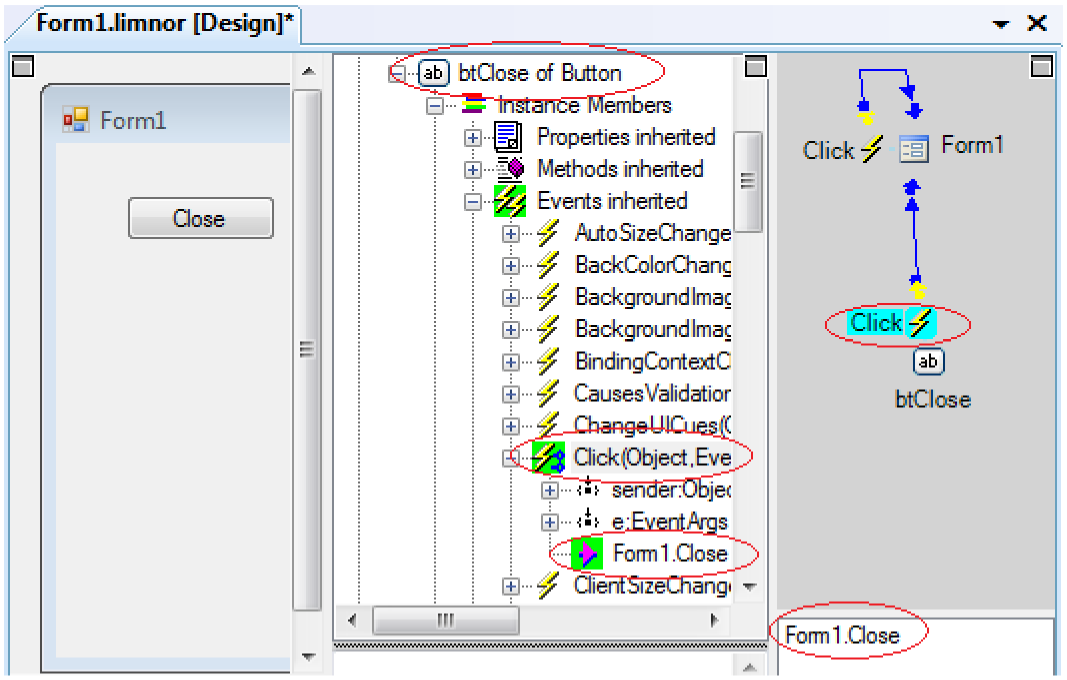
Right-click the icon representing the button; choose "Assign Action"; choose "Click" event:



As assigning action using UI Designer, select the Close action and click Next.

III.4.4. View Action-Event Relationships

Object Explorer and Event Path show the action-event relationships.

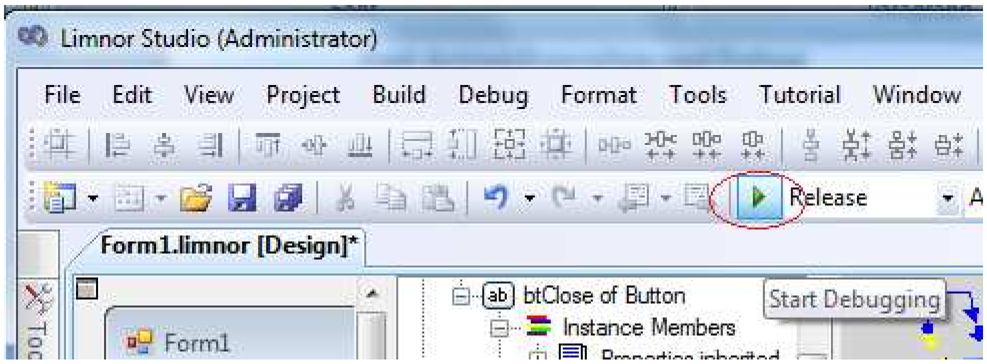


In the Object Explorer, actions assigned to an event are displayed under the event. We can see action Form1.Close shows under Click event of the button btClose.

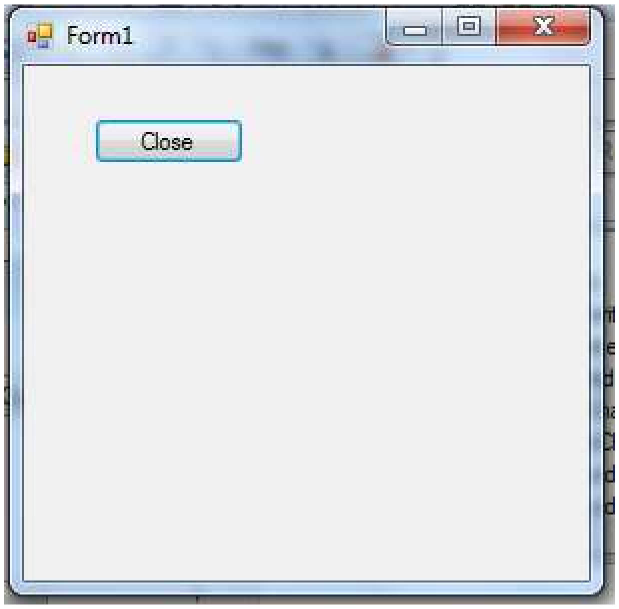
In the Event Path, an event icon represents the Click event of the button. A line starts from the event icon and ends at the icon Form1. The line indicates that when the Click event occurs Form1 will execute some actions. The actions to be executed by Form1 are displayed at the bottom.

III.4.5. Test the Application

Actions only can be executed at runtime. Click run button to compile and debug the project.


Limnor Studio (Administrator)
File Edit View Project Build Debug Format Tools Tutorial Window
Release
Form1.limnor [Design]*
Form1
btClose of Button
Instance Members
Start Debugging

The application runs. Page1 appears:



Click the button. The page closes. Because this is the only form of the application, the application also shuts down when the form closes.

IV. Create and Use Class Instances

To use instance members an instance must be created. An instance can only be created within a class and used as a component of the class.

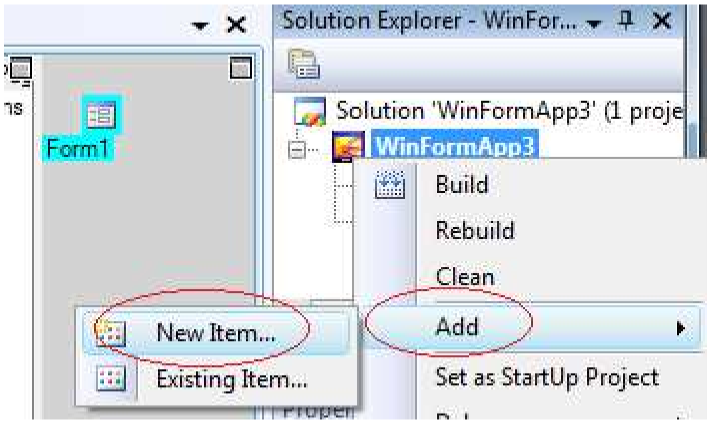
For developing Form application, refer to document “UseDefaultFormInstance” http://www.limnor.com/support/UseDefaultFormInstance.pdf which describes a much more simplified ways of form programming. It is also much safer, avoiding programming bugs.

If you do not want to create multiple instances of a form then DO NOT use the techniques described below. Use the technique described in http://www.limnor.com/support/UseDefaultFormInstance.pdf instead.

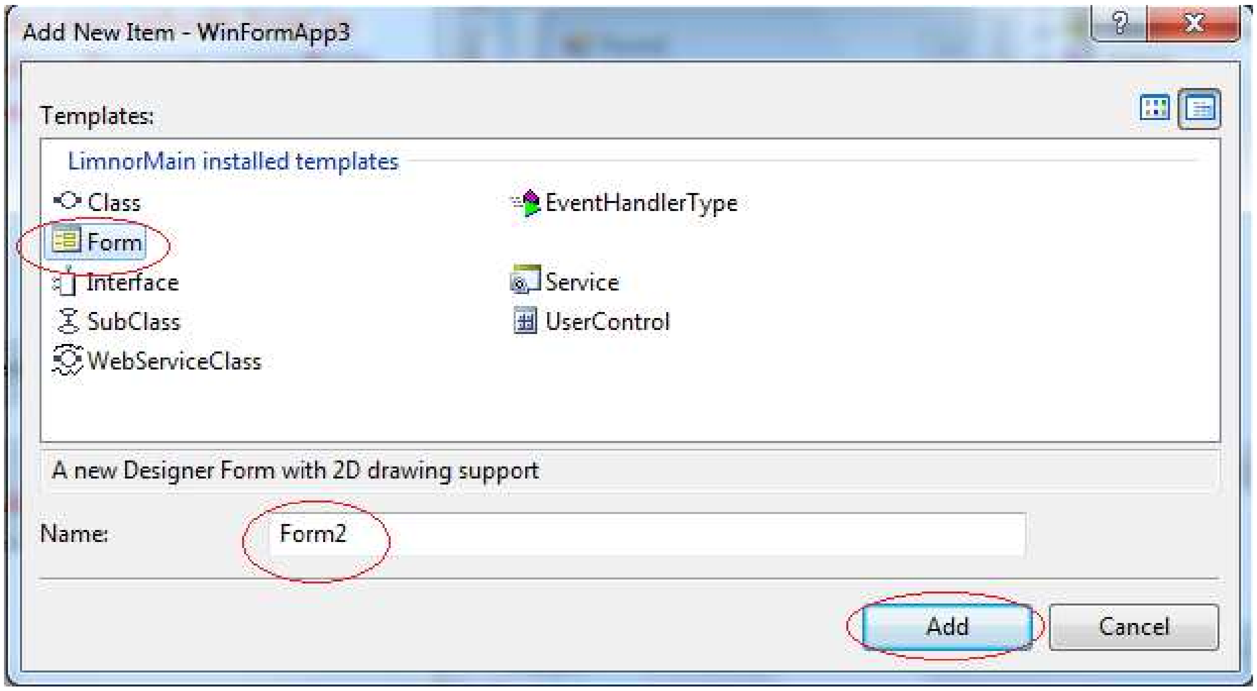
In 90% of situations you do not want to create multiple instances of a form.

IV.1. Add a New Form Class

Right-Click on the Project, choose “Add” and “New Item…”.



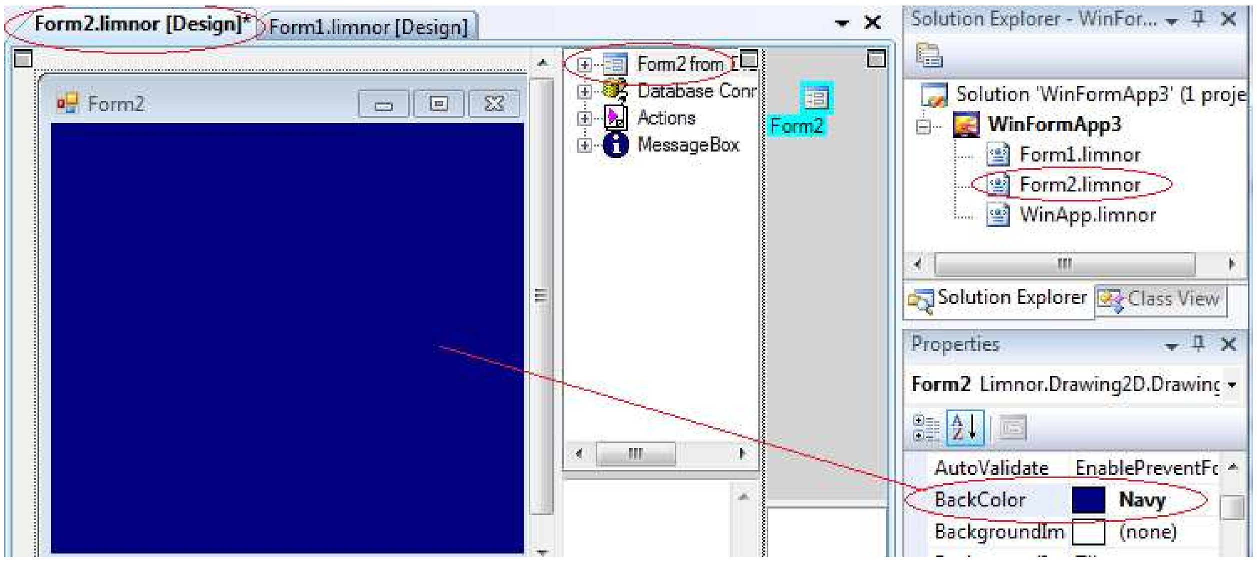
Choose Form template, choose a unique name, click button Add:



A new Designer Form with 2D drawing support

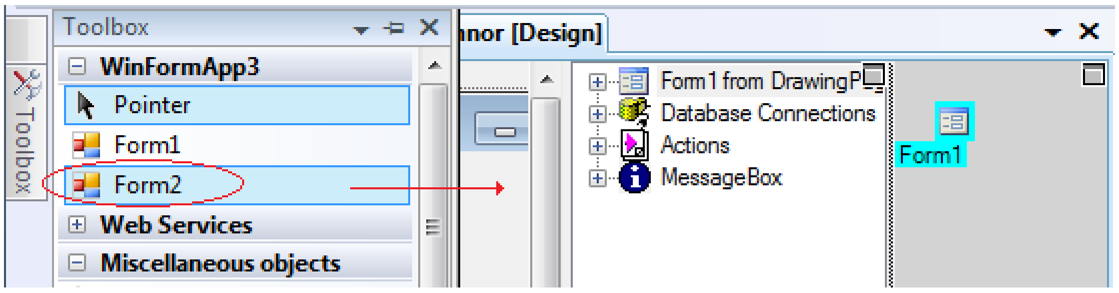
Name: Form2

A new Form class is added to the project. To make it easy to distinguish from class Form1, we set its background color to Navy:

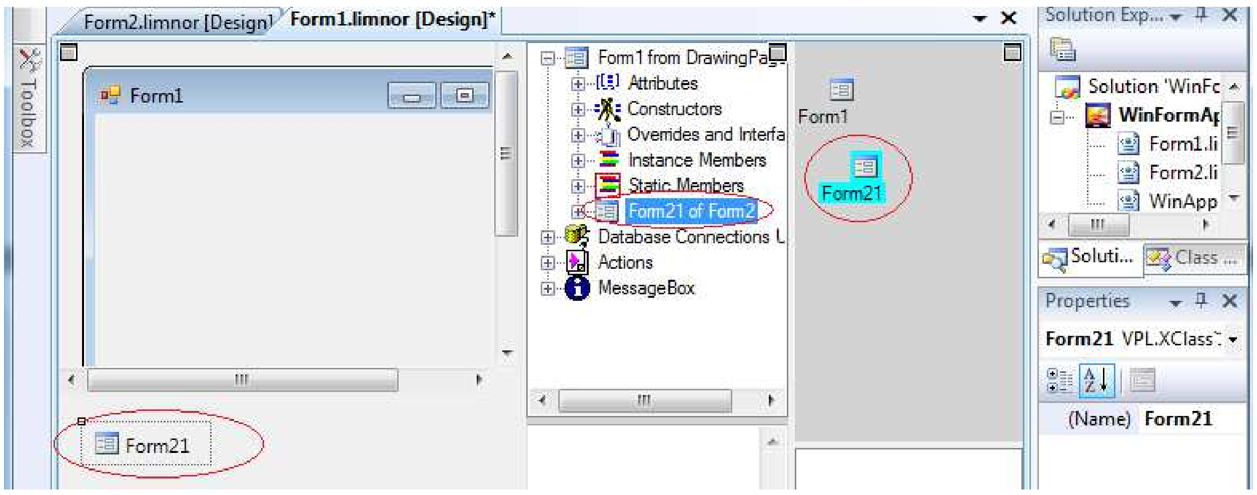


IV.2. Create and Use Instance

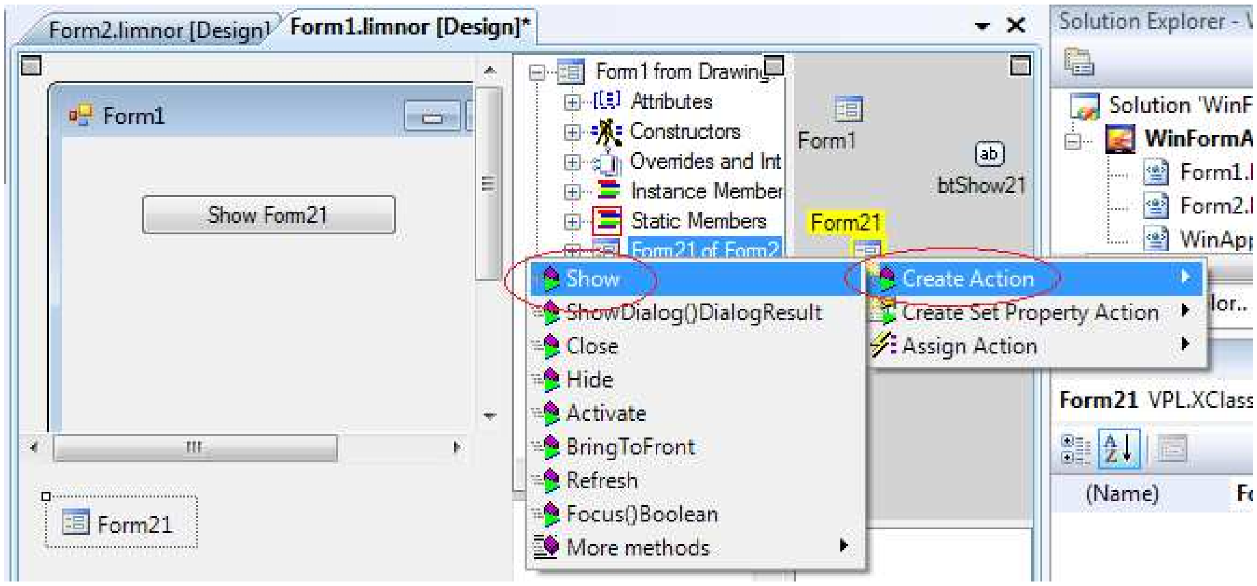
We want to show Form2 from Form1. We may add an instance of class Form2. Drop Form2 from the Toolbox to Form1:



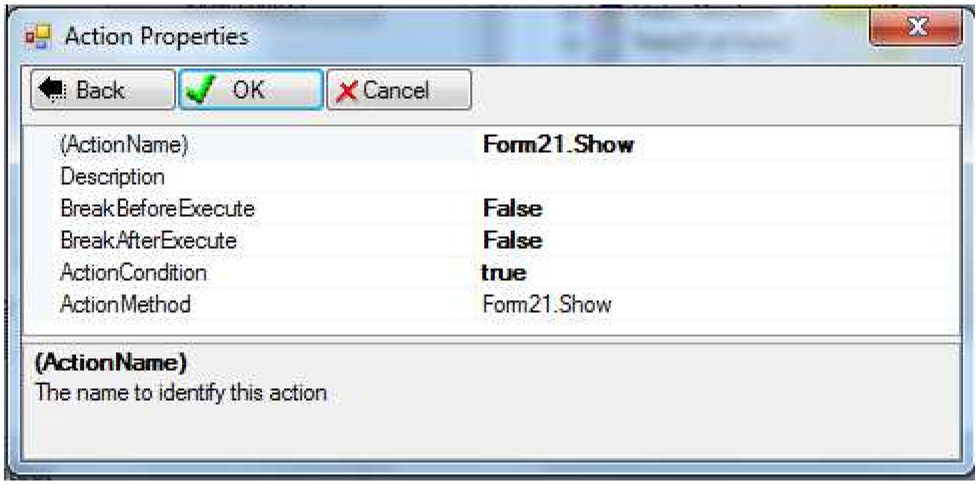
An instance of class Form2, named Form21, is created in class Form1:



To display the instance Form21, create a Show action. Right-click on icon Form21, choose “Create Action”. Choose method Show:



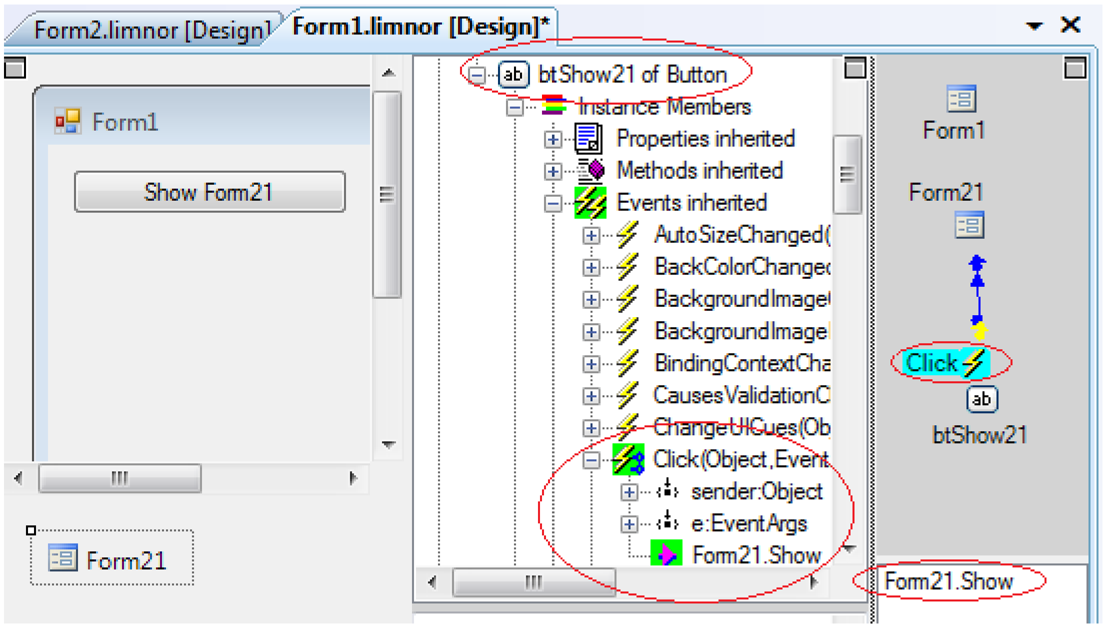
The Show method does not require a parameter:



(ActionName)	Form21.Show
Description	
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	Form21.Show

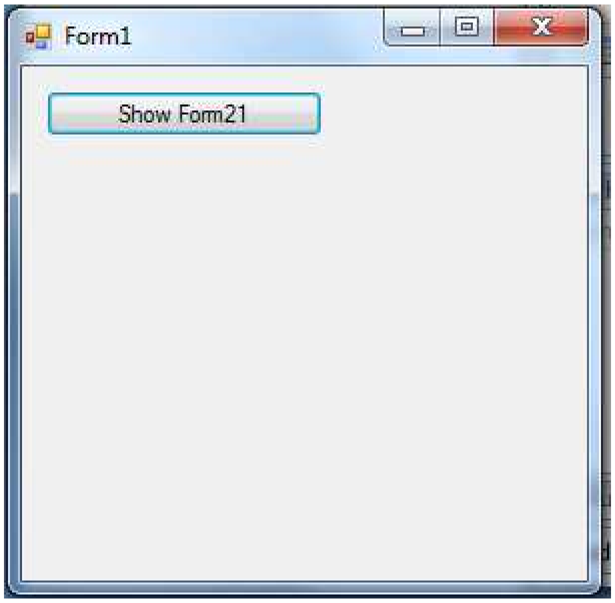
(ActionName)
The name to identify this action

Assign the new action, Form21.Show, to the Click event of button “Show Form21”:

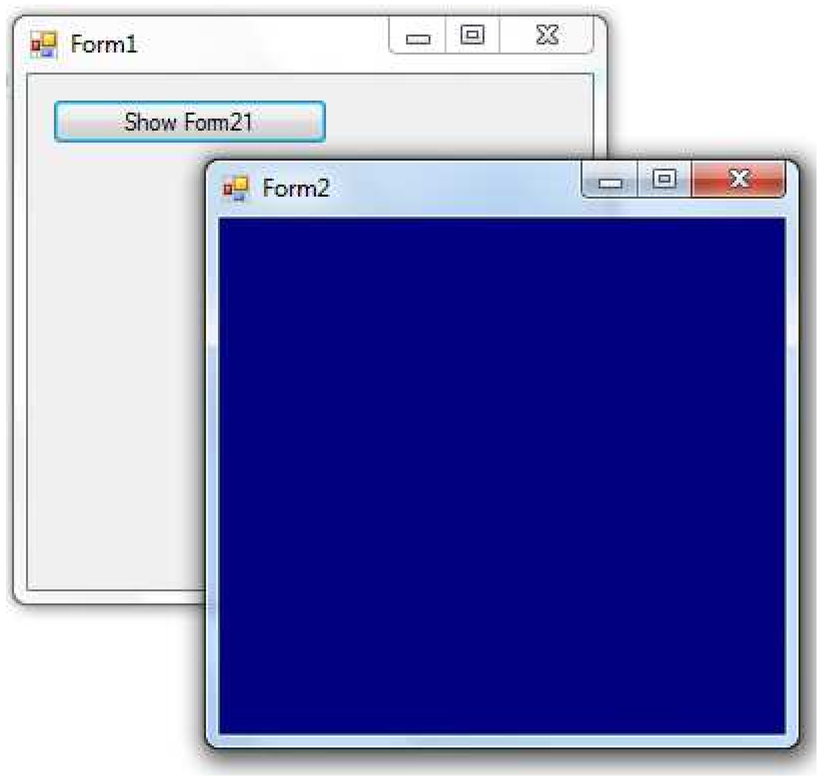


Click run button to compile and debug the project.

The application starts. Page1, which is an instance of class Form1, appears:



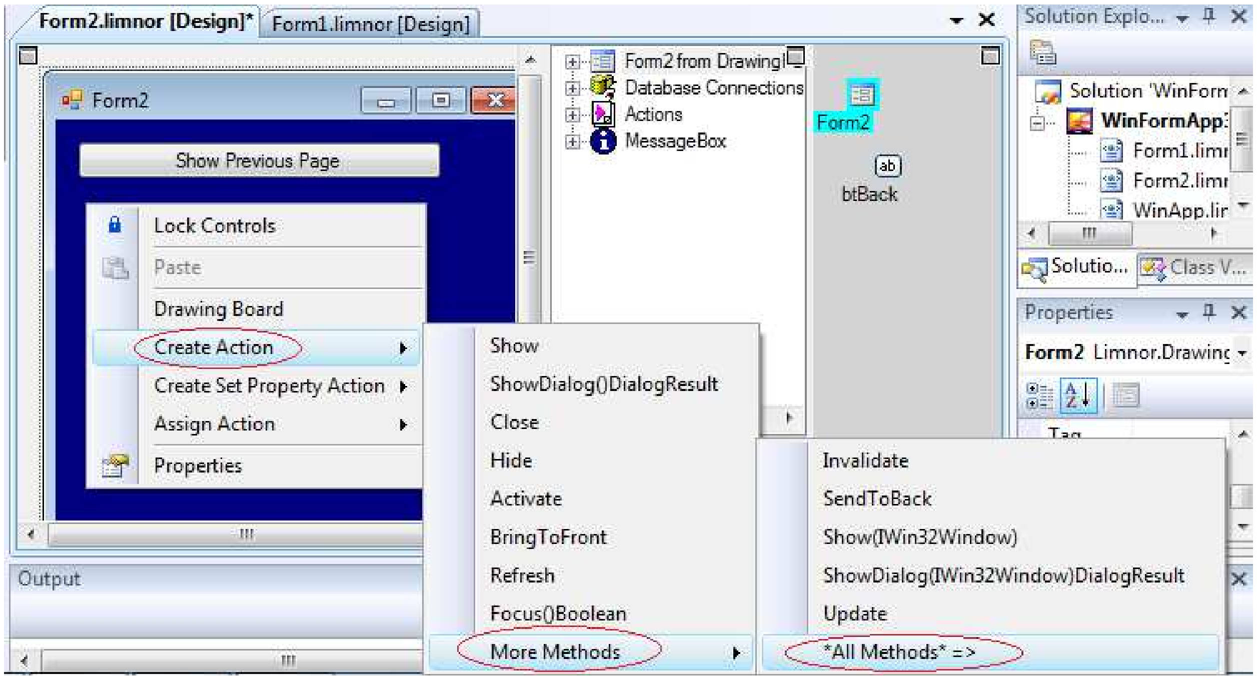
Click the button “Show Form21”. Form21, which is an instance of class Form2, appears:



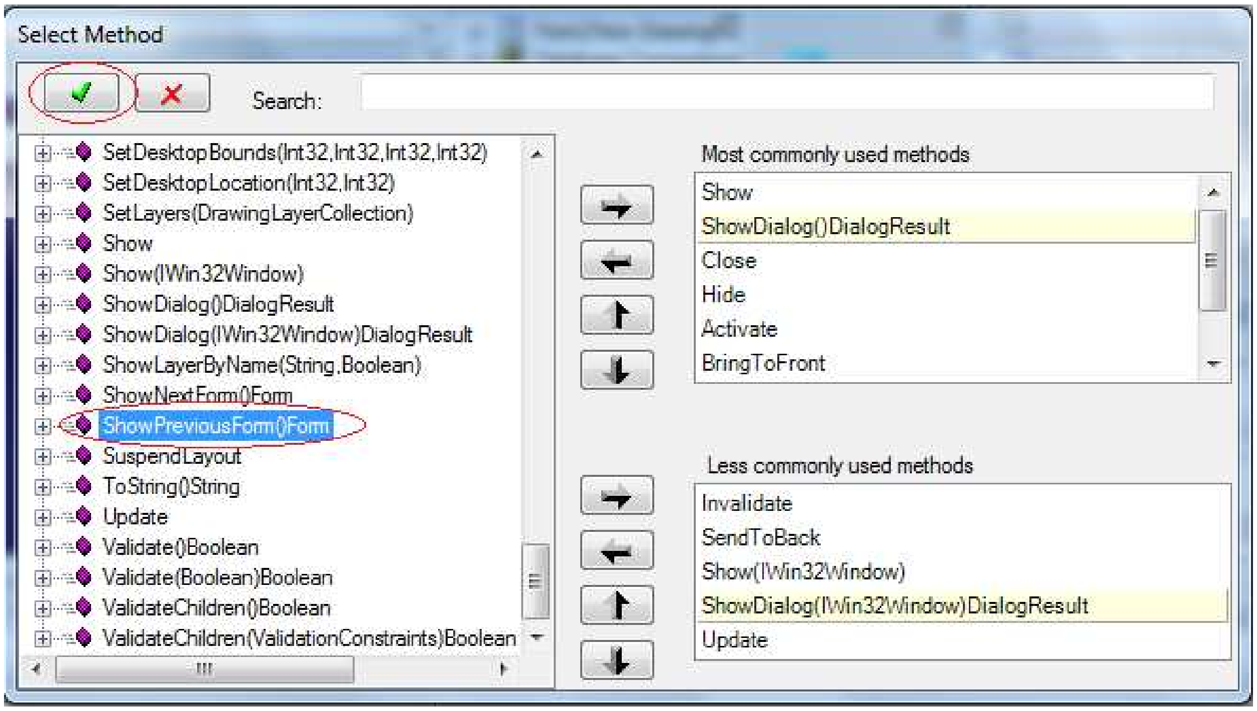
IV.3. Form Navigation

Form class has a ShowNextForm method and a ShowPreviousForm method for form navigation.

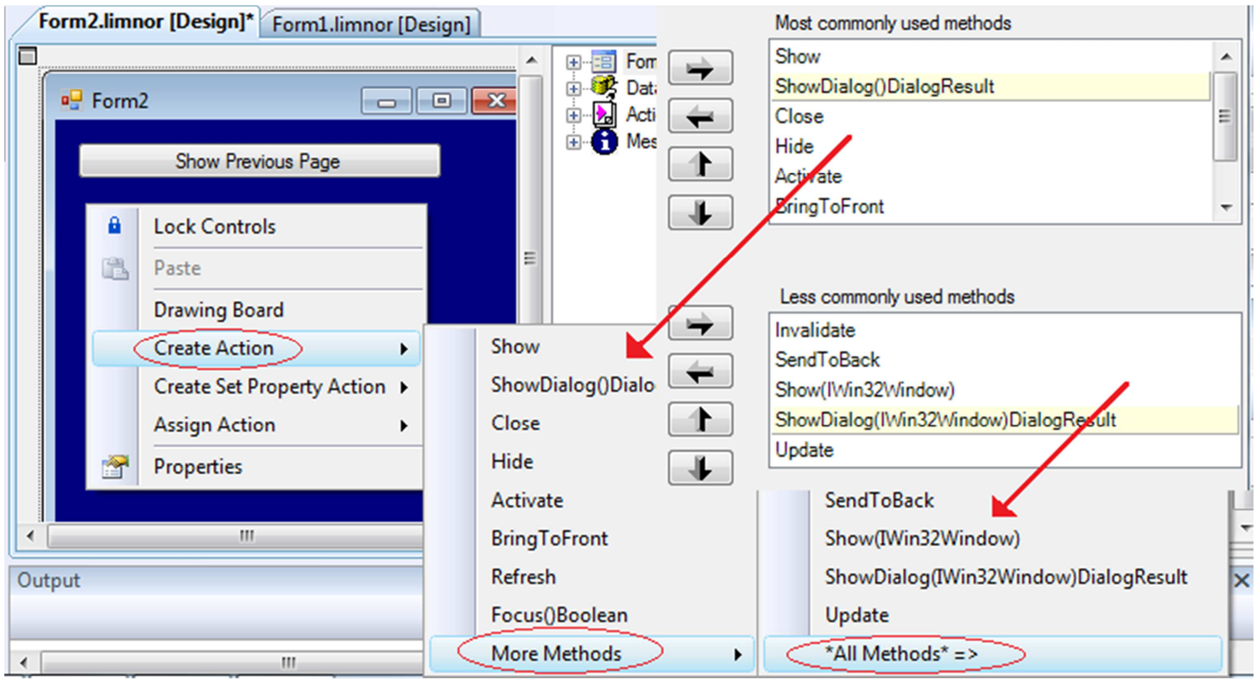
Add a button on class Form2 to execute ShowPreviousForm action. To create ShowPreviousForm action, right-click Form2, choose “Create Action”. Choose ShowPreviousForm method. ShowPreviousForm method may not be in the menu list. Choose “More Methods” and choose “*All Methods* =>” to locate the desired method:



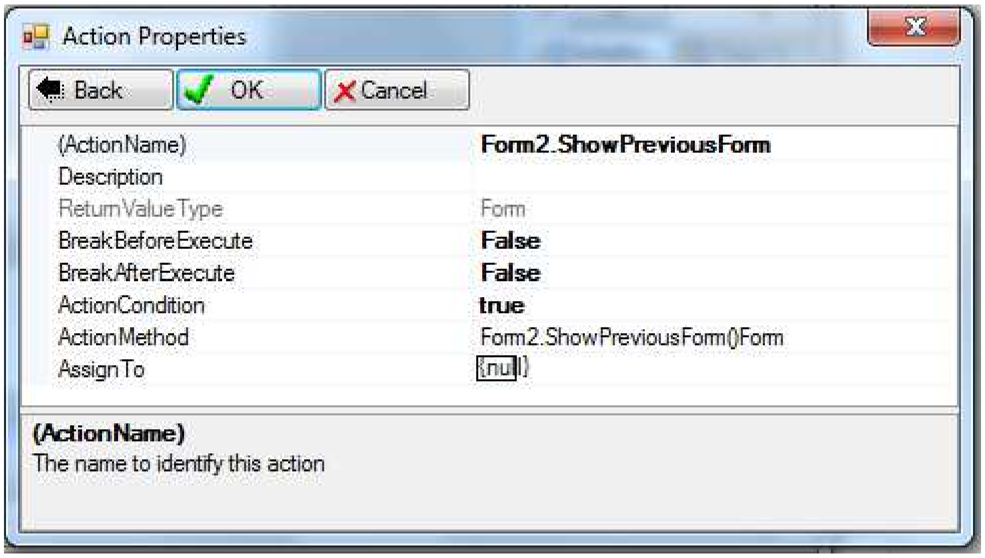
Select method ShowPreviousForm, click ✓:


Select Method
Search:
SetDesktopBounds(Int32,Int32,Int32,Int32)
SetDesktopLocation(Int32,Int32)
SetLayers(DrawingLayerCollection)
Show
Show(IWin32Window)
ShowDialog()DialogResult
ShowDialog(IWin32Window)DialogResult
ShowLayerByName(String,Boolean)
ShowNextForm()Form
ShowPreviousForm()Form
SuspendLayout
ToString()String
Update
Validate()Boolean
Validate(Boolean)Boolean
ValidateChildren()Boolean
ValidateChildren(ValidationConstraints)Boolean
Most commonly used methods
Show
ShowDialog()DialogResult
Close
Hide
Activate
BringToFront
Less commonly used methods
Invalidate
SendToBack
Show(IWin32Window)
ShowDialog(IWin32Window)DialogResult
Update

Note that you may change the lists of “Most commonly used methods” and “Less commonly used methods”. List “Most commonly used methods” is used to form the first level menu items. List “Less commonly used methods” forms the second level menu items.



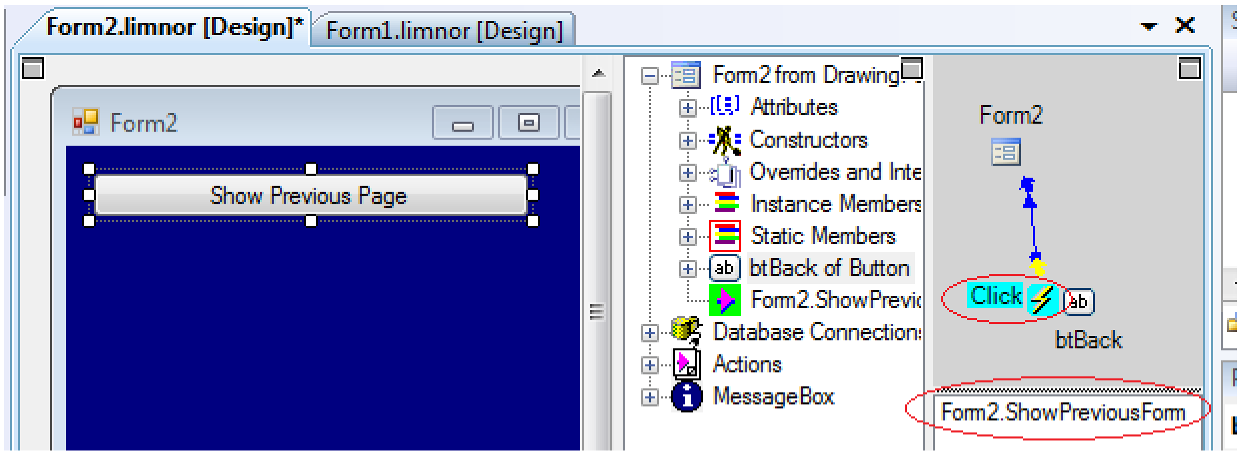
The ShowPreviousForm action does not require action parameters:



(ActionName)	Form2.ShowPreviousForm
Description	
ReturnValueType	Form
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	Form2.ShowPreviousForm()Form
AssignTo	(null)

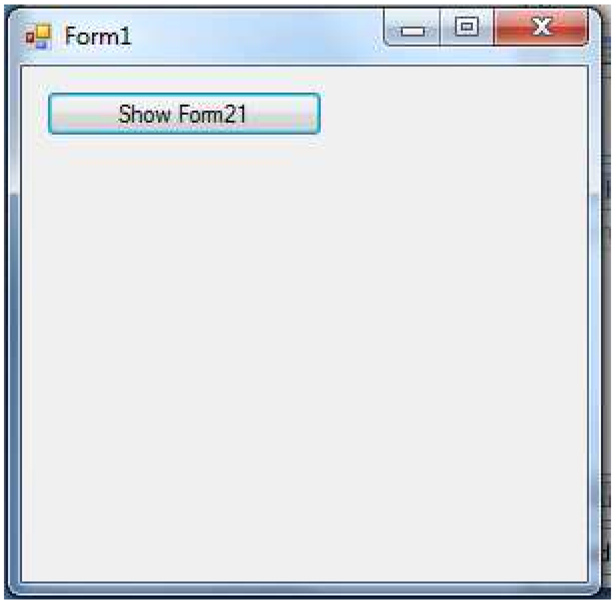
(ActionName)
The name to identify this action

Assign this action to the Click event of the button:

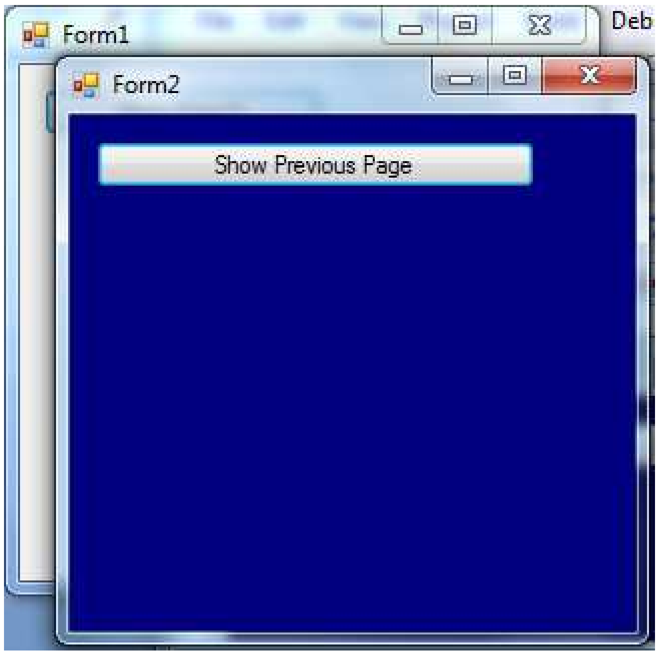


Click run button to compile and debug the project.

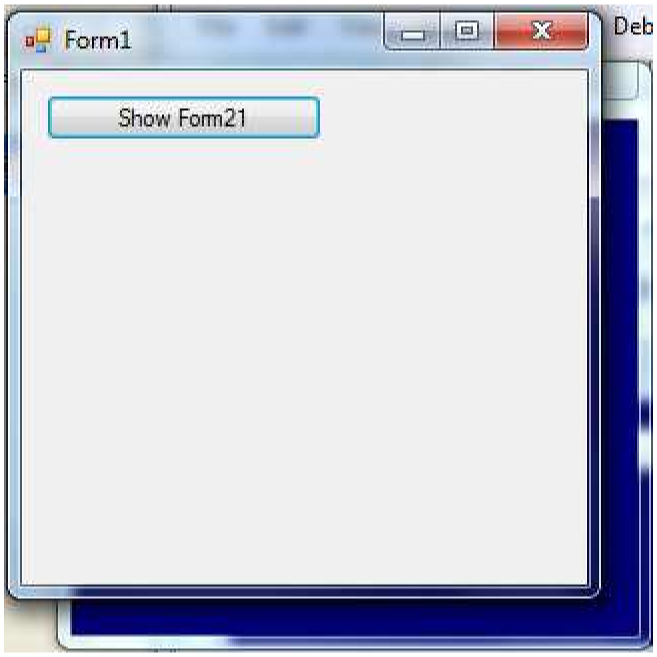
The application starts. Page1, which is an instance of class Form1, appears:



Click the button “Show Form21”. Form21, which is an instance of class Form2, appears:



Click button "Show Previous Page". Page1 appears:



IV.4. Use Default Form Instance

In the above sections, we create a new form instance, Form21, by dropping the form from the toolbox. This is for showing creation of class instances.

Form class is a special class. It is not recommended to use the above method to create form instance if we may avoid it. In most cases we can avoid it because Limnor Studio invented a “Default Form Instance” technique for simplifying the form application development. Using the default form instance is much simple and safe than creating your own form instances.

For details of using “Default Form Instance”, see
http://www.limnor.com/support/UseDefaultFormInstance.pdf