Use Default Form Instances

Contents
Introduction .................................................................................................................................................. 2
Add Form Classes .......................................................................................................................................... 3
Starting Form (Home Page). ........................................................................................................................... 5
Use Multiple Forms ....................................................................................................................................... 6
Different Ways of Showing Forms ............................................................................................................... 10
   Show form not as a dialogue ................................................................. 10
   Show form as a dialogue ........................................................................ 10
   Show/Hide form with fade-in and fade-out ........................................... 11
Accessing between forms – sample 1 ......................................................................................................... 11
Accessing between forms – Sample 2 ......................................................................................................... 20
   Modular Pattern .................................................................................... 20
   Method as interface ................................................................................ 21
Create GotoURL method on Form1 ........................................................................................................ 23
Show Form2 from Form1 ............................................................................................................................ 25
Execute GotoURL from Form2 ................................................................................................................ 27
Test .......................................................................................................................... 29
Create a dialogue box to collect user inputs ............................................................................................... 30
Close a dialogue box and get Dialogue Result ........................................................................................ 30
Use Properties for collecting user inputs ................................................................................................ 32
Use the dialogue box .............................................................................................................................. 36
   Show dialogue box .................................................................................... 36
   Get data from dialogue box ........................................................................ 37
   Use the dialogue actions ............................................................................. 51
Create Data Validation ............................................................................................................................ 54
   Insert data validation to dialogue box .................................................... 54
   Simple data validation ............................................................................... 55
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...”:

Choose “Form” and give the new class a name, for example, Form2, click Add:
A new Form class is added to the project:
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”:

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:
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:
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:
Select the first parameter:

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

![Form1.limnr [Design] Form2.limnr [Design]](image)
Give some testing data for the method parameters:

![Action Properties](image1)

Assign the action to a button:

![Form Design](image2)

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:
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:

Add a parameter and rename the parameter to webPageAddress:
Add a Navigate action:

Pass `webPageAddress` to the action:

Click OK:
Finish editing this method:

Show Form2 from Form1
Click OK:

The action is created and linked to the button:
Execute GotoURL from Form2:

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

The action is created and linked to the button:
Test

Compile and test the program:

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:

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

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:
Click the constant icon:

Select OK for the value:

This condition checks whether the dialogue box close reason is OK:
This is the action for getting Firstname from Form3:

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

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:
Click the constant icon:

Select OK for the value:

This condition checks whether the dialogue box close reason is OK:
This is the action for getting Lastname from Form3:

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”:

Give a message to be displayed:
Choose Form1 as the owner of the message box:
Set the ActionCondition to execute it only when the cancel button is clicked:

Like we did before, we create an expression to check DialogResult:
This action displays a message box when the user cancels the dialogue box:
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:

The data entered appear in Form1:

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

Set ActionCondition to verify data:
Click the Property icon:

Double-click the property element to select property:
Click the “greater than” icon:

Select the second element and 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:

Select the “greater than” element:

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:
Select the text length property of the Lastname text box:

The selected property appears in the expression:
Click the “greater than” icon:

A new “greater than” is added:

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:

This is the expression for data validation:

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:

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:

Click the “greater than” icon:
Select the second element and 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:

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”:

Give a message to the action:
While the new action is selected, click Next:

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:
Add another action to show another message box:
While the new action is selected, click Next:
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:
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.

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:
Feedback
Please send your feedback to support@limnor.com