Use Application Configurations

Contents

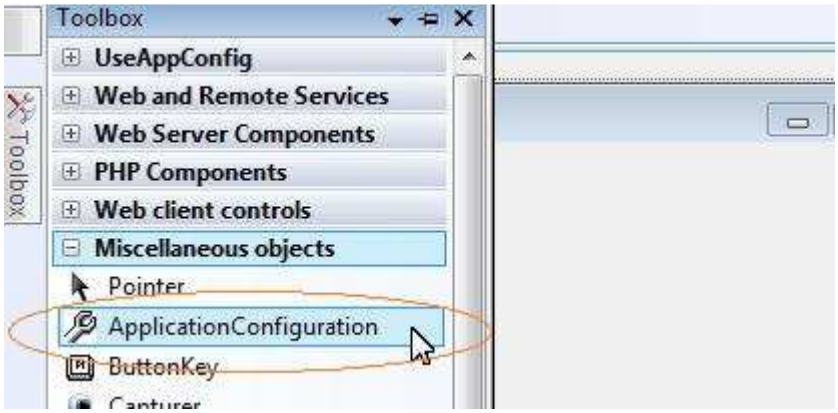
Introduction

The sample project created in this document can be downloaded from http://www.limnor.com/studio/UseAppConfig.zip.

Your application may allow users to choose their preferences. For example, you may want to allow the user to set background colors, change window sizes and locations, select images, etc. You may also want to remember the preferences the user selected so that the next time your application runs the user preferences will be reloaded.

A use preference is called a “configuration value”.

The ApplicationConfiguration component is used for saving and loading configuration values.



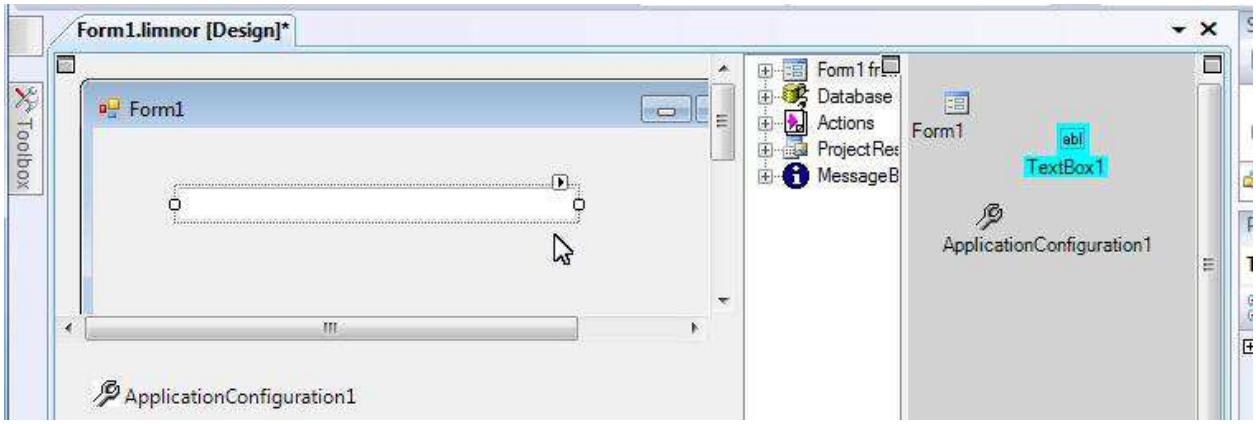
You may add ApplicationConfiguration to your forms or classes. All ApplicationConfiguration instances in one project use the same configuration files to remember configuration values; therefore all ApplicationConfiguration instances work as if there is only one instance.

Simple Configuration

To begin with, we use a most simple configuration sample to show the basic process of using ApplicationConfiguration.

Determine the value to be remembered

Let's add a text box to allow the user to enter text. When the application starts we want to let the text box display what the text box displayed the last time.

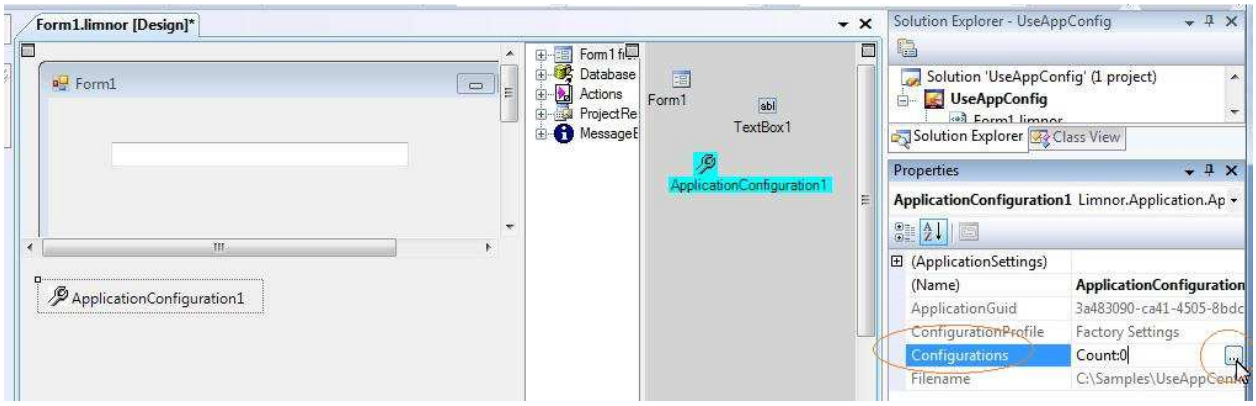


So, in our case the value to be remembered is the Text property of the text box.

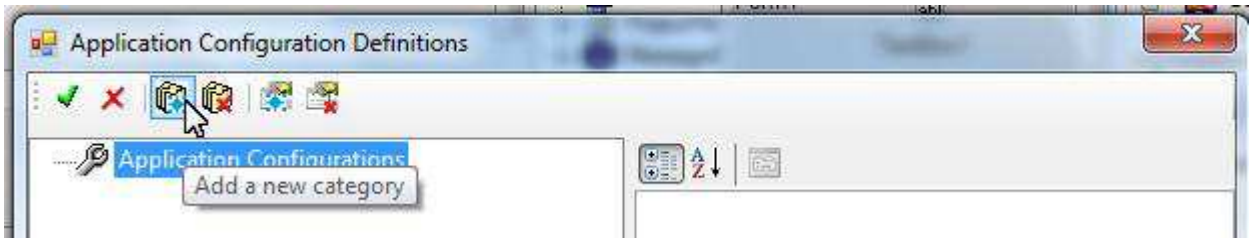
Create a configuration value

For every value we want to remember, we need to create a configuration value in the ApplicationConfiguration component. In our case, we need to create a configuration value for the Text property of the text box.

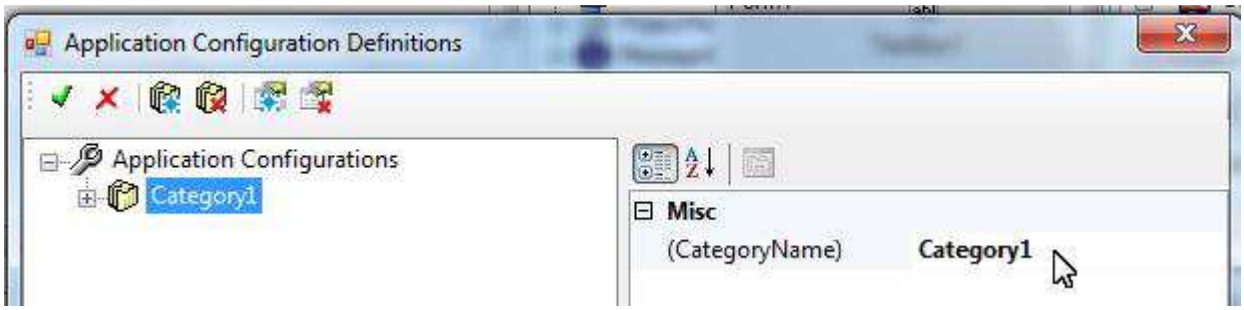
Set the Configurations property to create configuration values.



To help manage large number of configuration values, each configuration value belongs to a category. So, we need to first create a category:

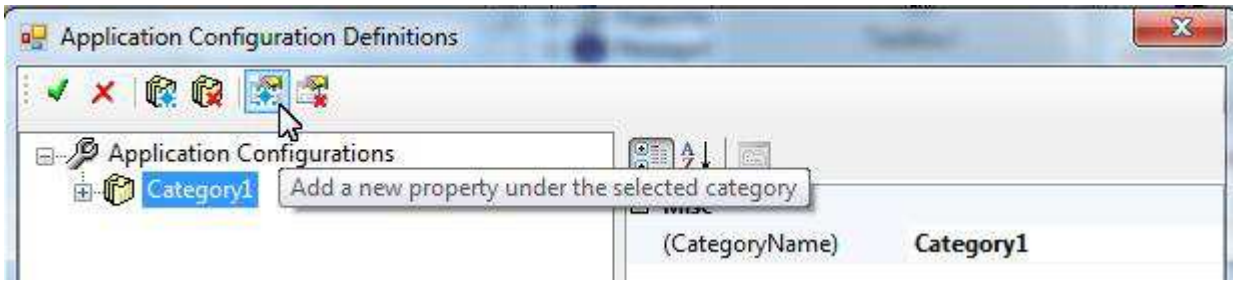


We may change the name of the category if we want:

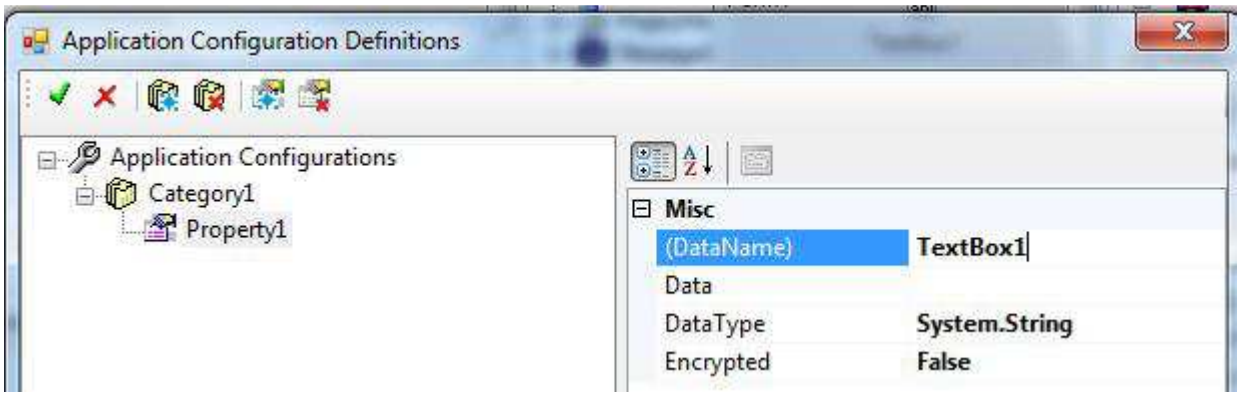


Do not use following words for the names of the categories: AppKey, ApplicationGuid, Configuration, Configurations,ConfigurationProfile,Filename,ProfileName, ProfileType

Now we may create a new configuration value:

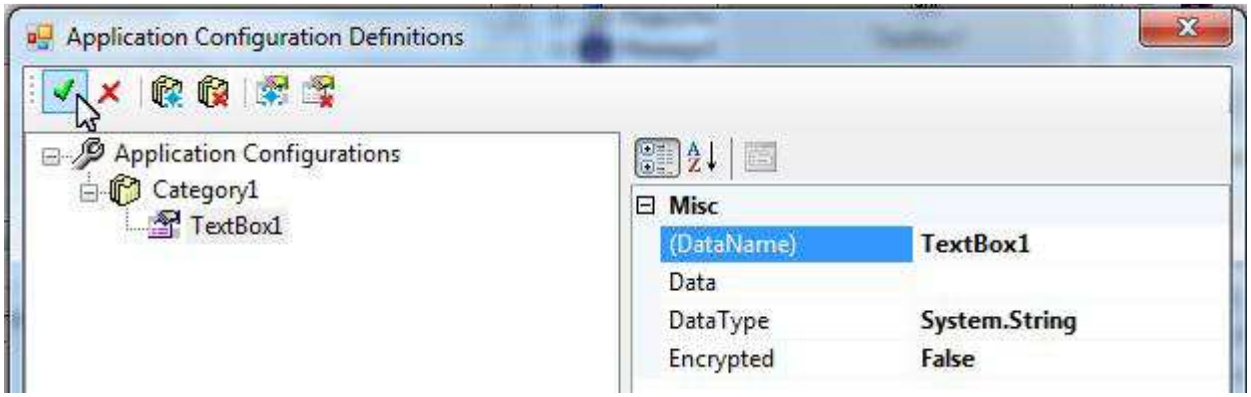


We may change the name of the configuration value. Let's name it "TextBox1":



Note that the DataType is String. That is what we want because we want to remember text value.

We may create many configuration values. In this sample, we just need one configuration value.

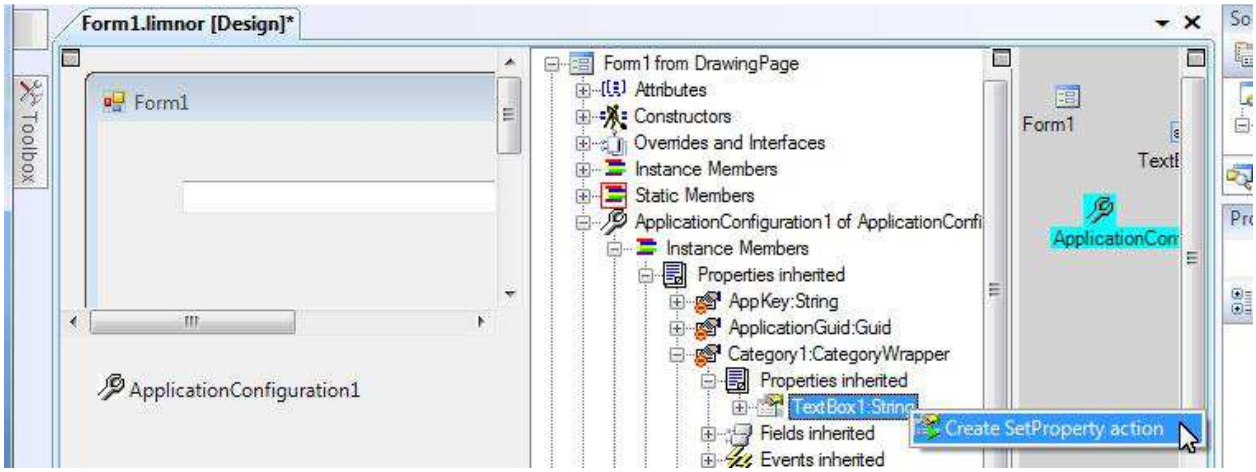


Remember Values

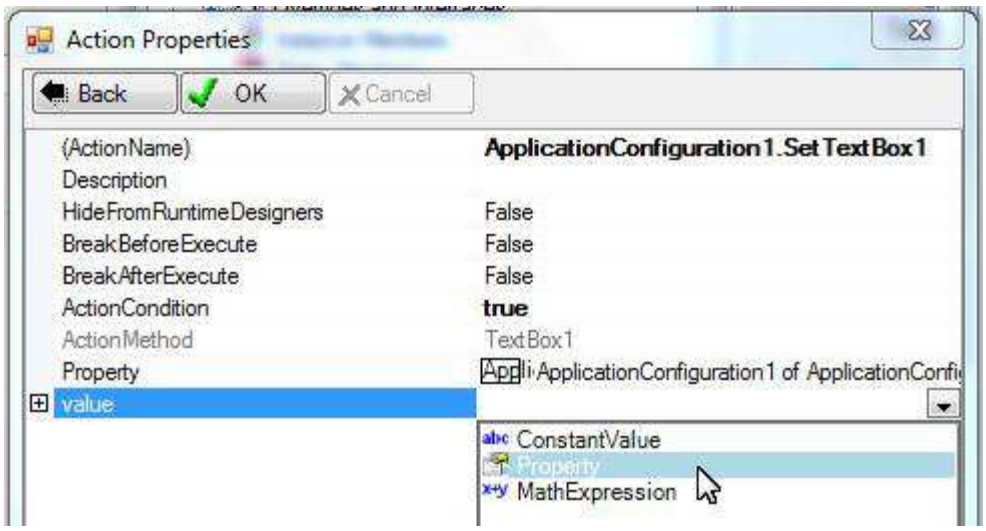
Pass value to configuration

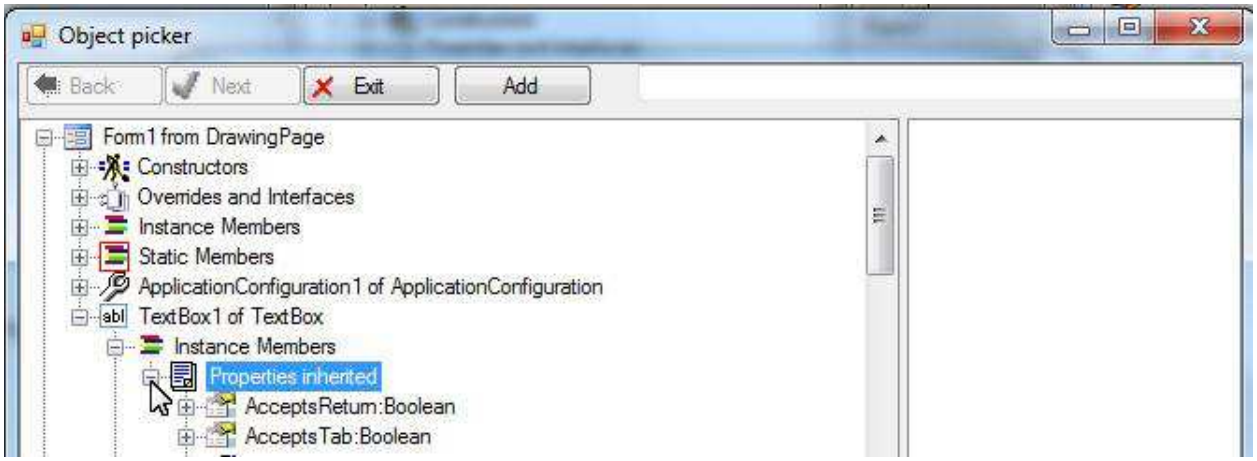
In this sample, we need to pass the Text property of the text box to the configuration value named "TextBox1". We need to create an action to do it.

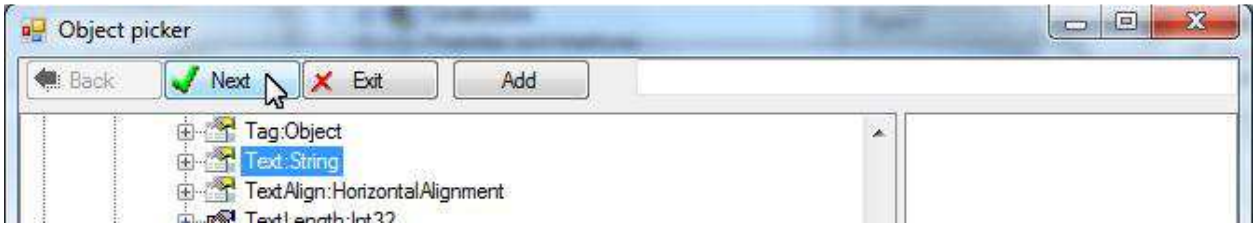
We can find categories and configuration values in the Object Explorer. Right-click configuration value "TextBox1" and choose "Create SetProperty action":



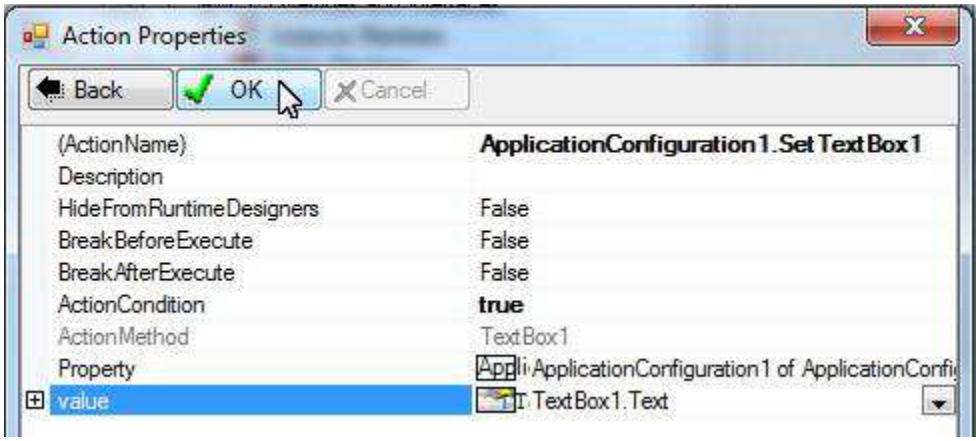
Pass the value we want to remember to the action:



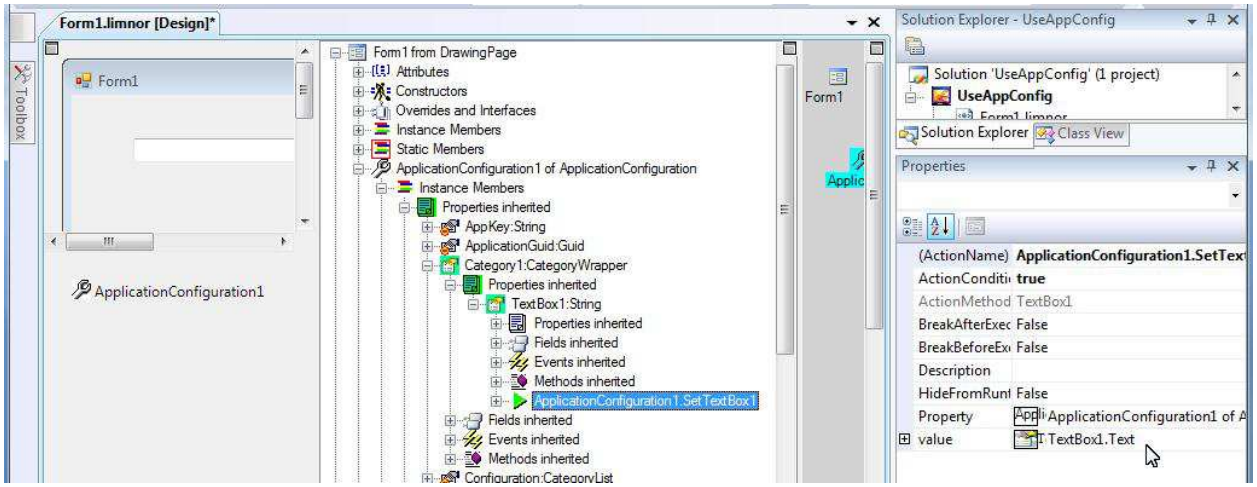




Click OK:



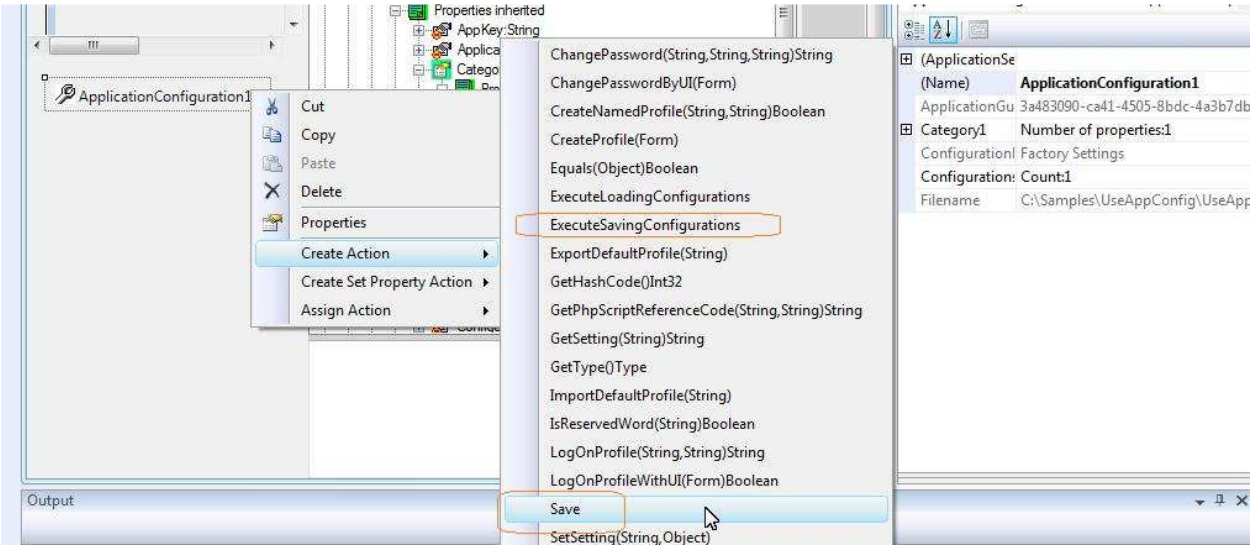
The action is created:



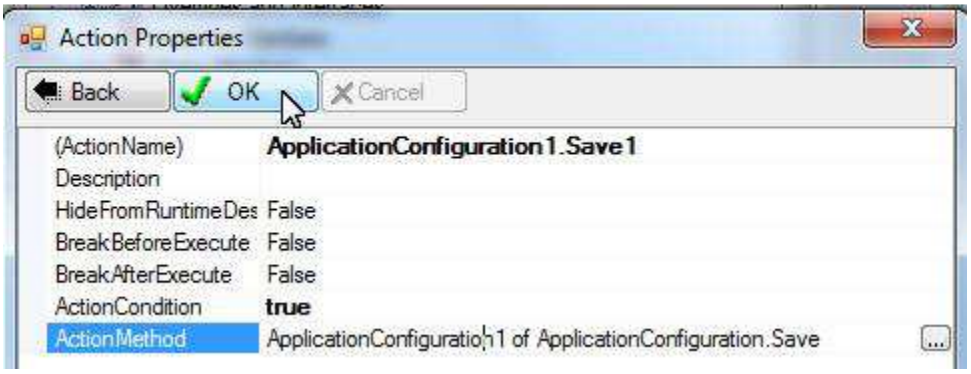
For every configuration value we need to create such an action to transfer data from your application to the ApplicationConfiguration. In this sample, we only have one configuration value.

Persist to disk

Save and ExecuteSavingConfigurations both can be used to save all configuration values to disk:



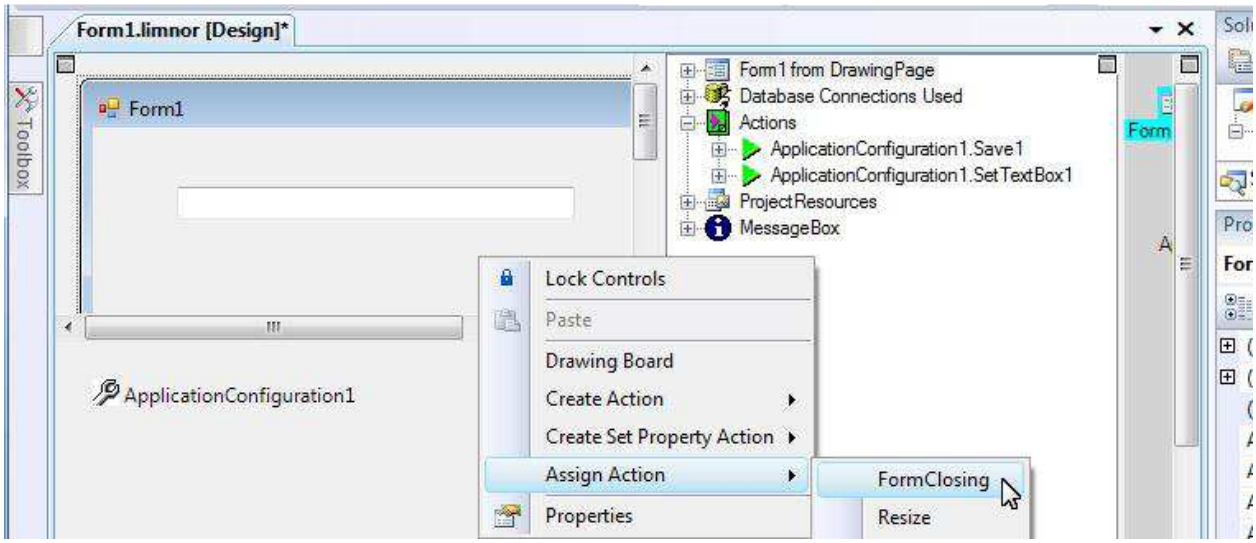
We use Save for this simple sample:



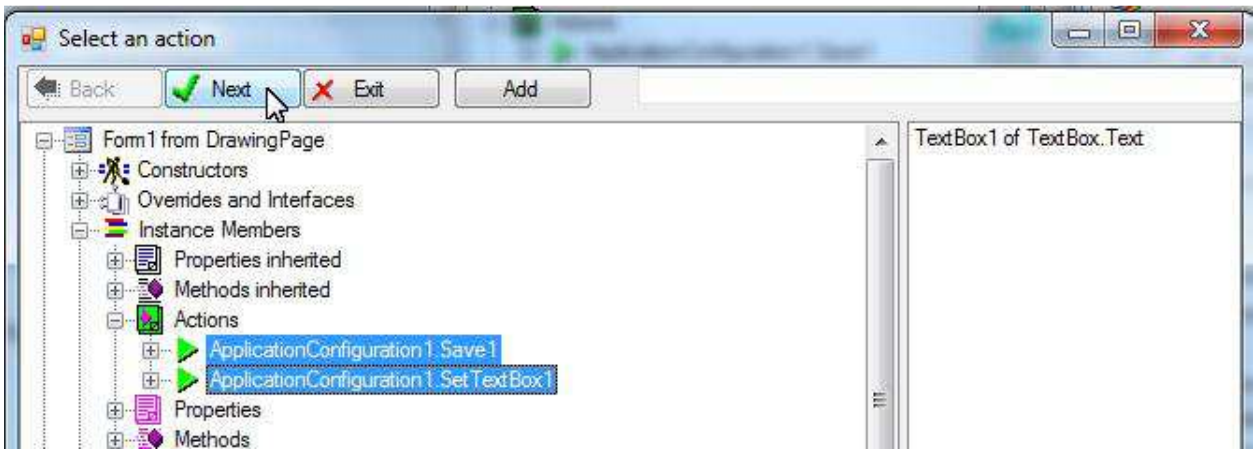
Determine when to execute value-saving actions

We need to determine at which events we want to execute value-saving actions. For this simple sample, we may execute value-saving actions at the time of closing the form:

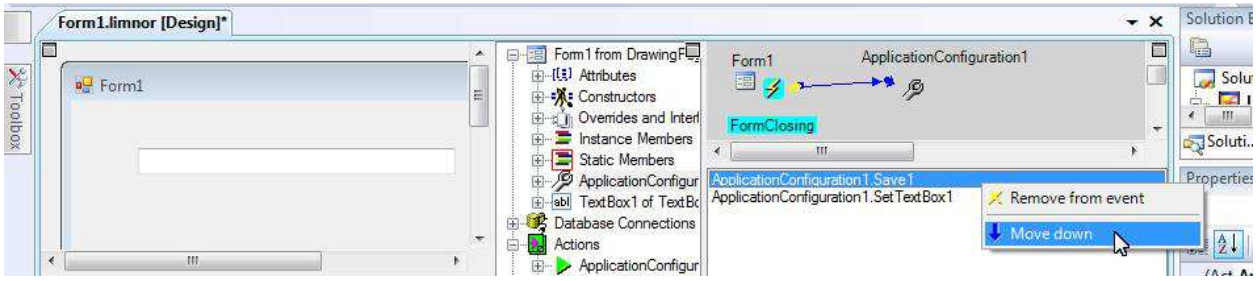
Right-click the form; choose “Assign Action”; choose “FormClosing” event:

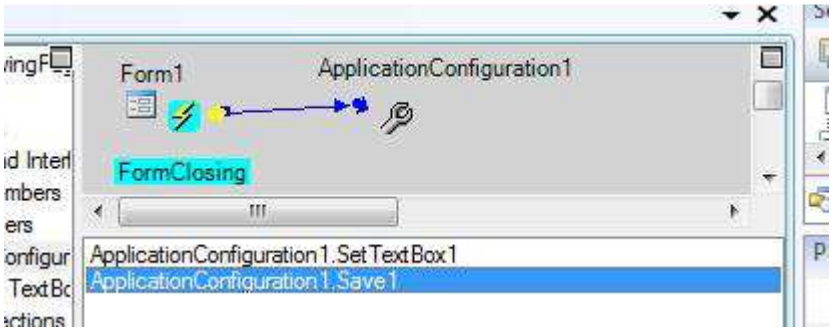


Select the value-saving actions:



The “Save” action must be the last action:

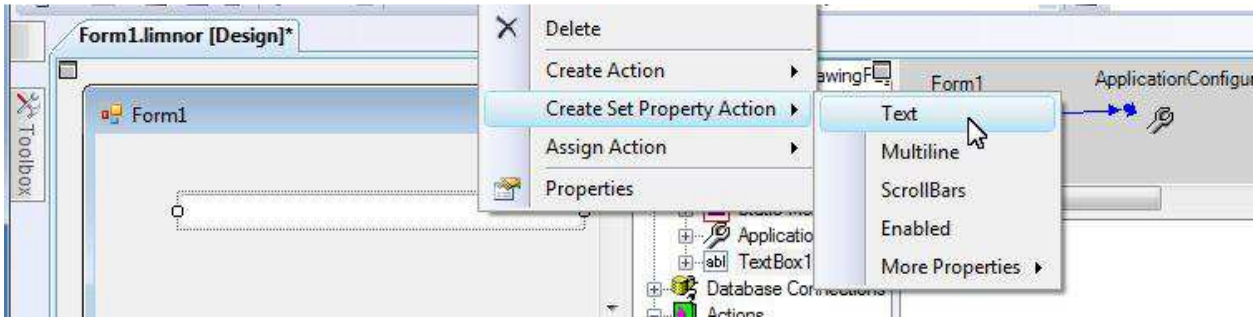




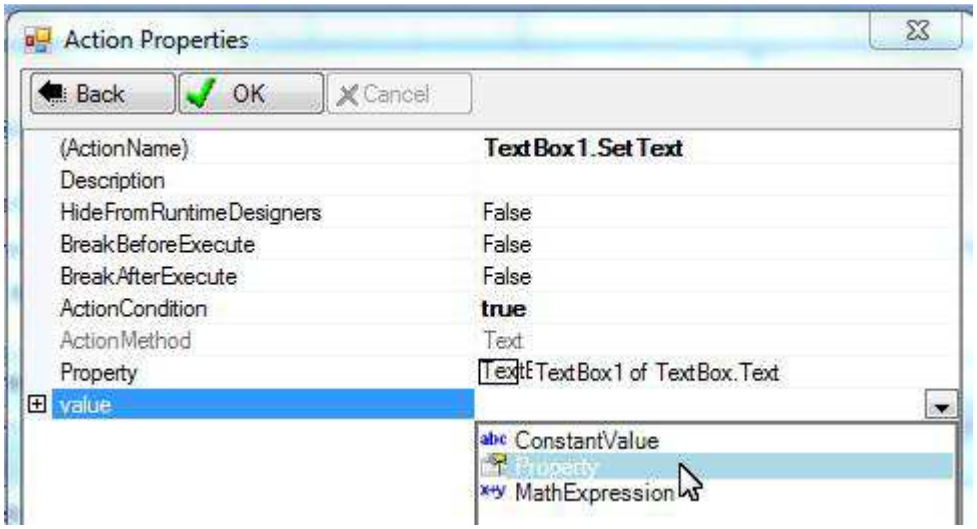
Apply saved values

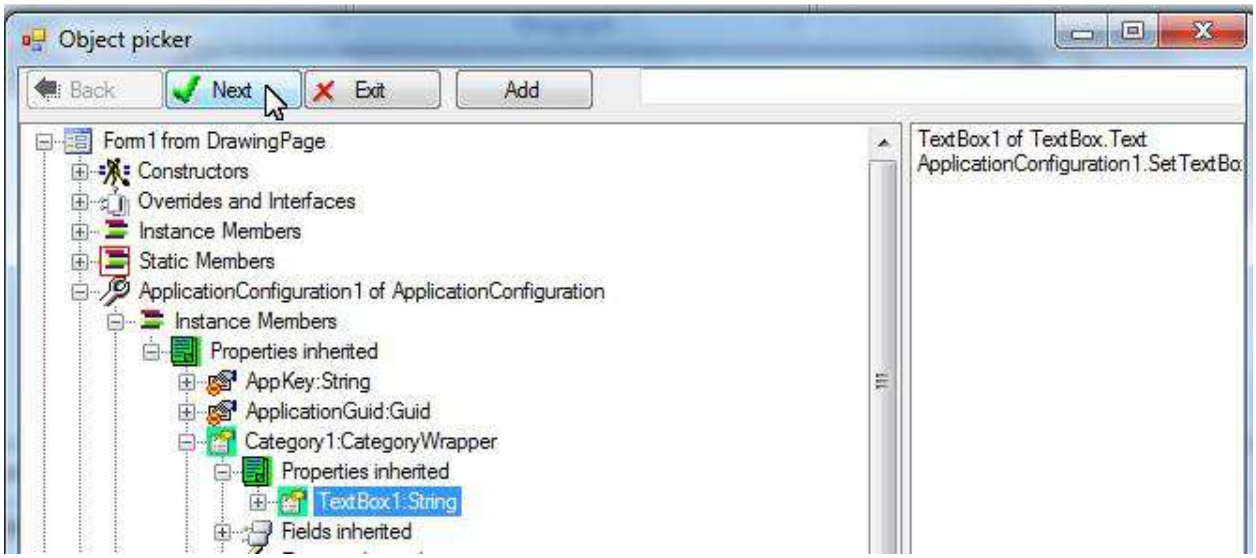
Pass configuration values to application

In our sample, we want to pass the configuration value named "TextBox1" to the Text property of the text box. We may create an action to do it:

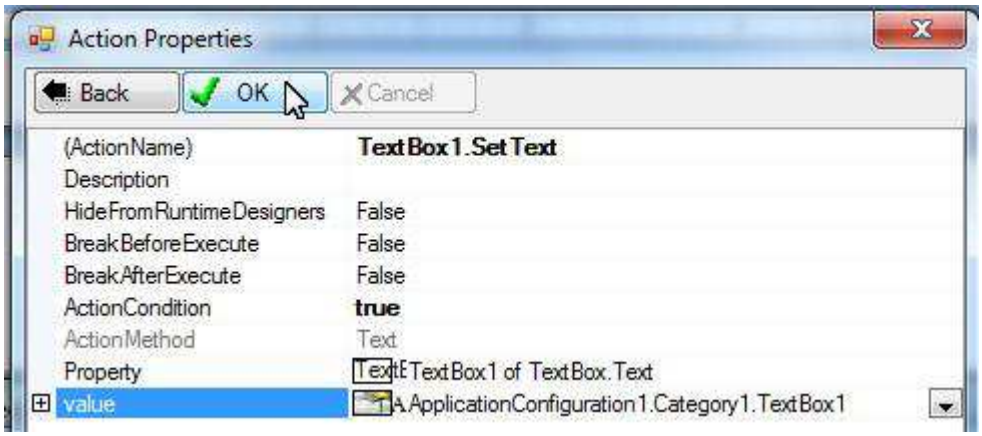


Use the configuration value in the action:

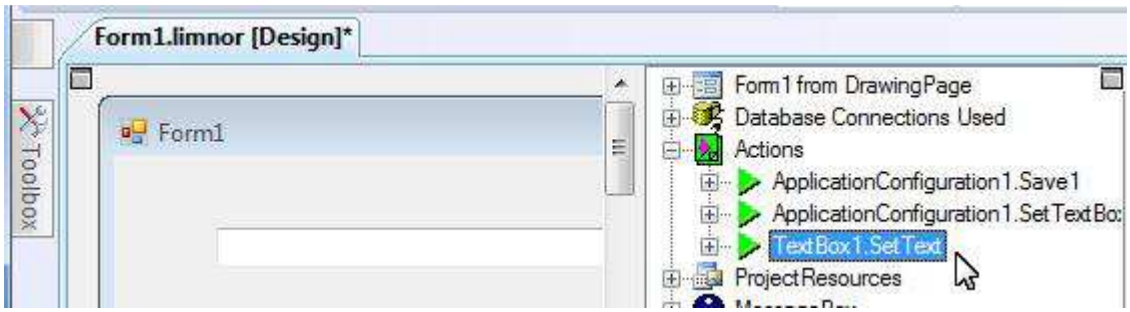




Click OK:

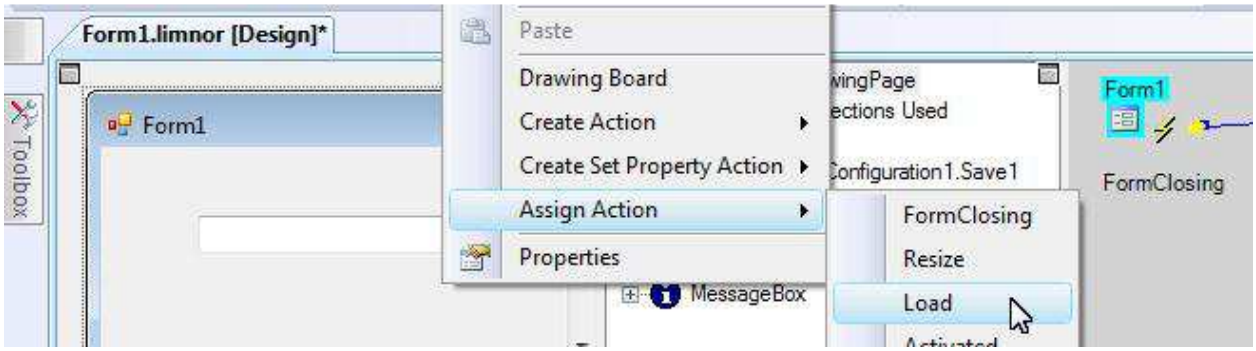


The action is created:

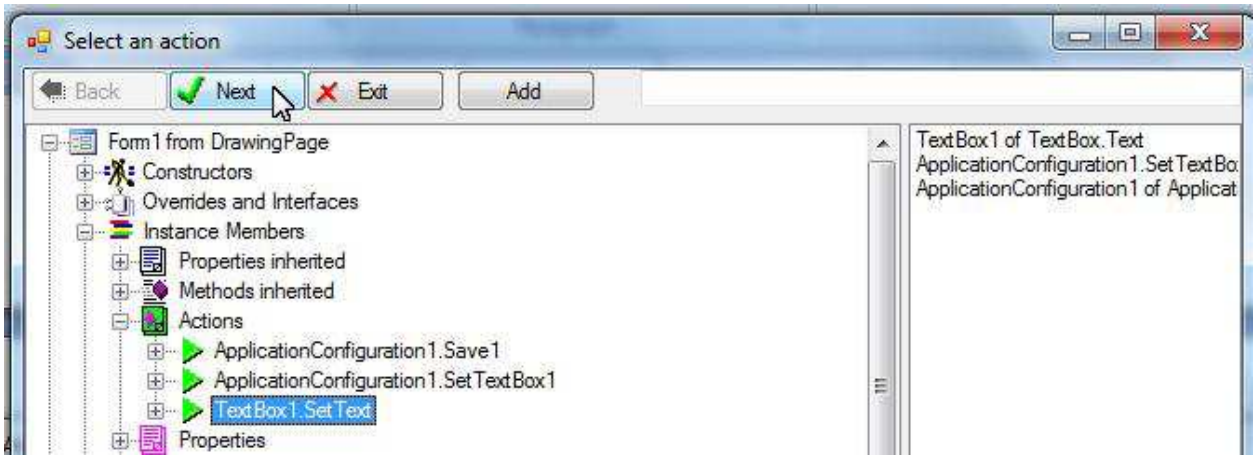


Determine when to apply configuration values

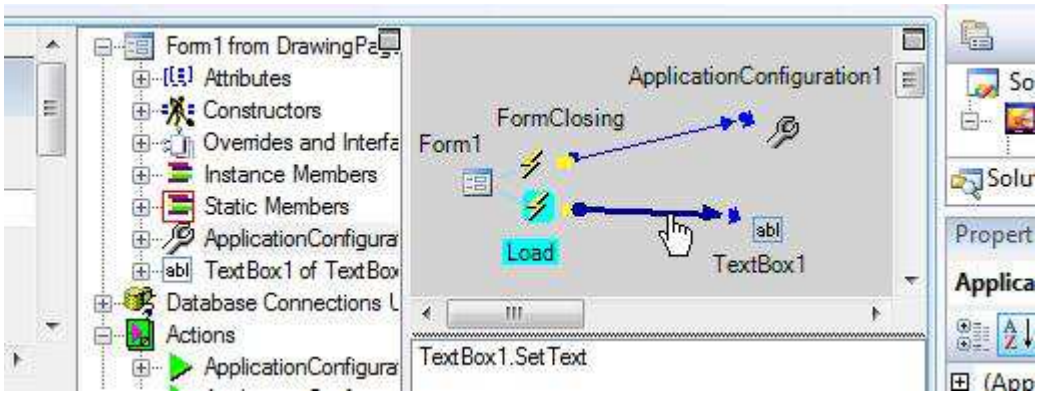
In this sample, we may apply the configuration values at the time when the form is loaded. Right-click the form; choose “Assign Action”; select event “Load”:



Select the value-applying action:

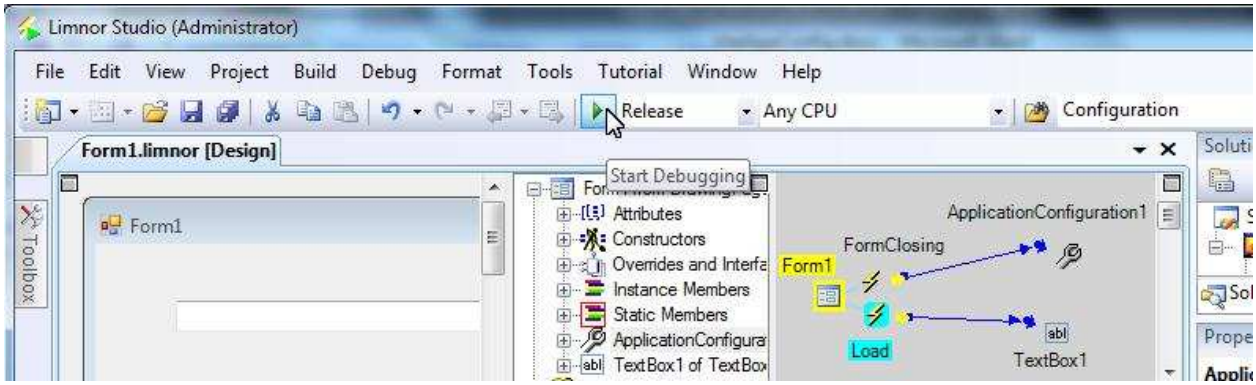


The action is assigned to event Load:

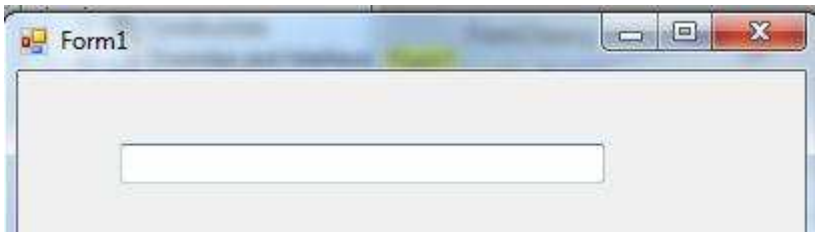


Test

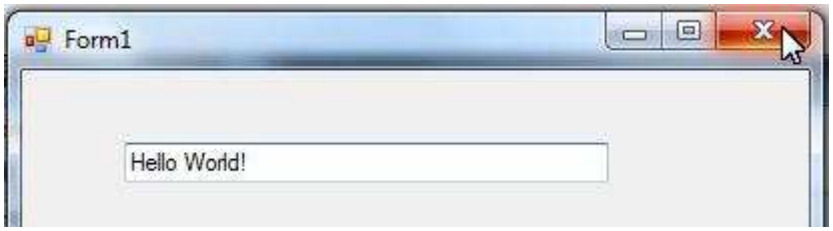
Click Run button to compile and run the application:



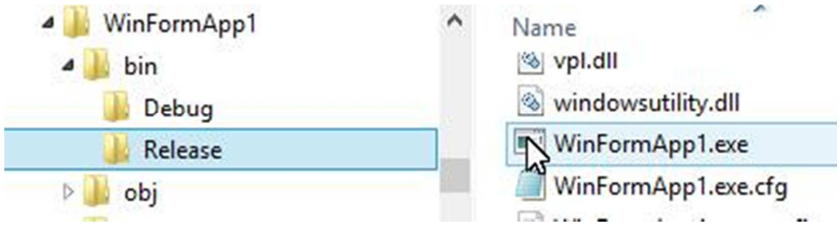
The form appears:



Enter some text in the text box and click close button:



Restart the application by double-click the EXE file:



The form appears. The text box restores the text when the application was closing:

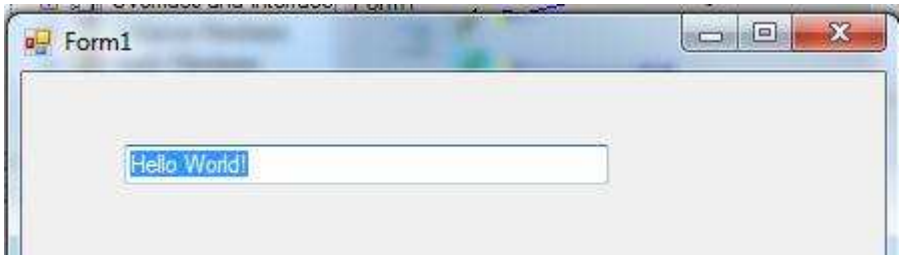
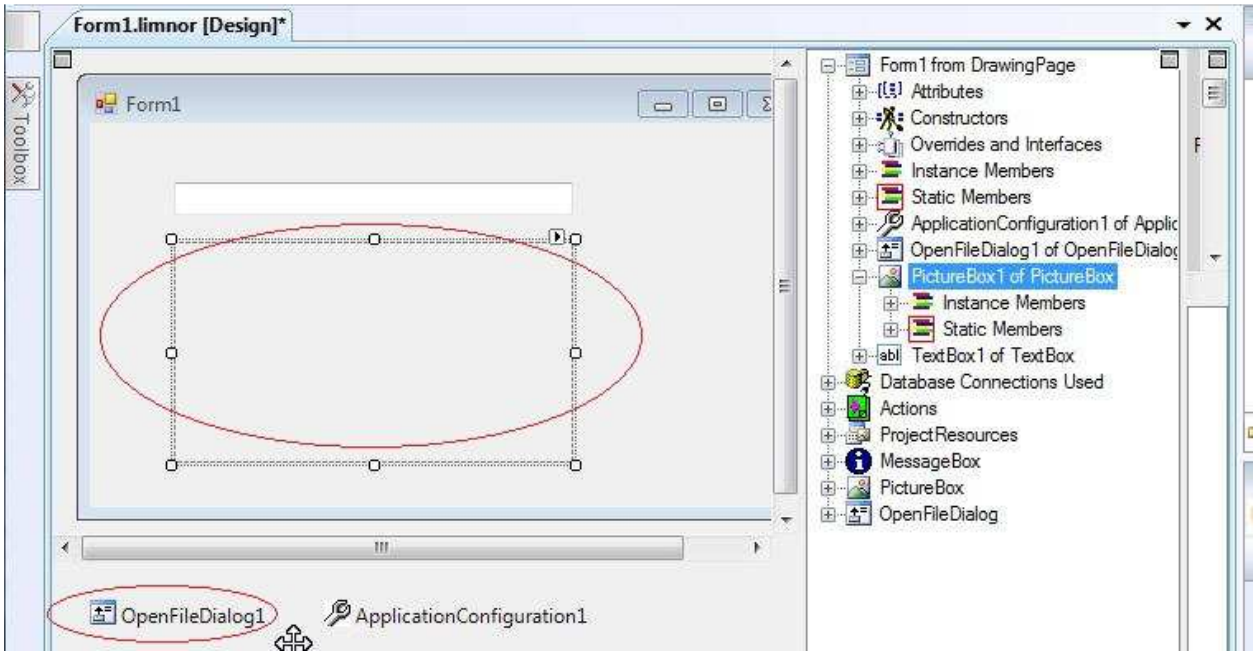


Image Configuration

Sample application

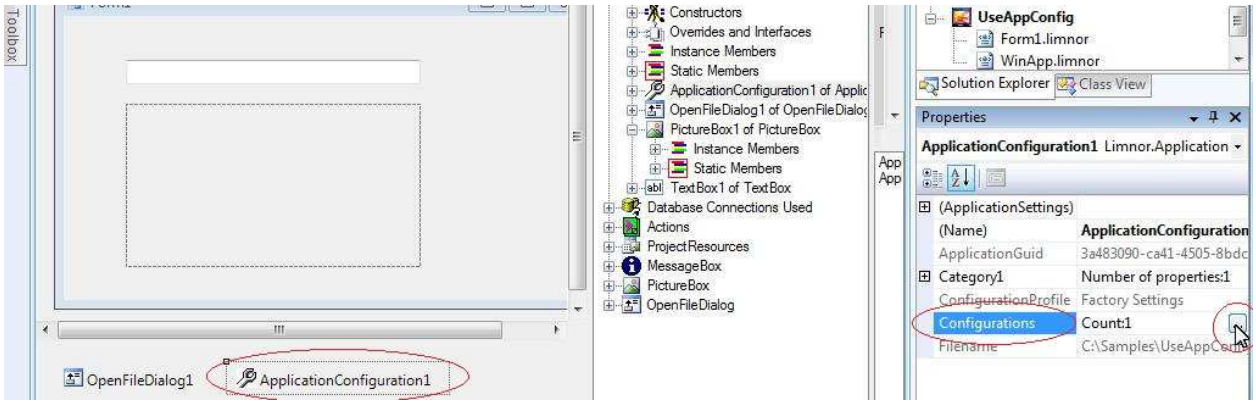
Let's show a sample of letting the user select image. Add a picture box to the form. Add an OpenFileDialog component. When the user clicks the picture box, the dialogue box is displayed for selecting an image file.

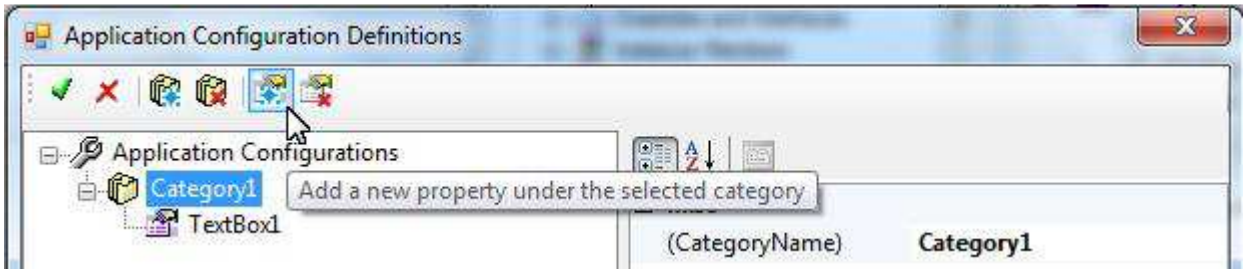


Suppose we want to make the image file path to be a value to be remembered.

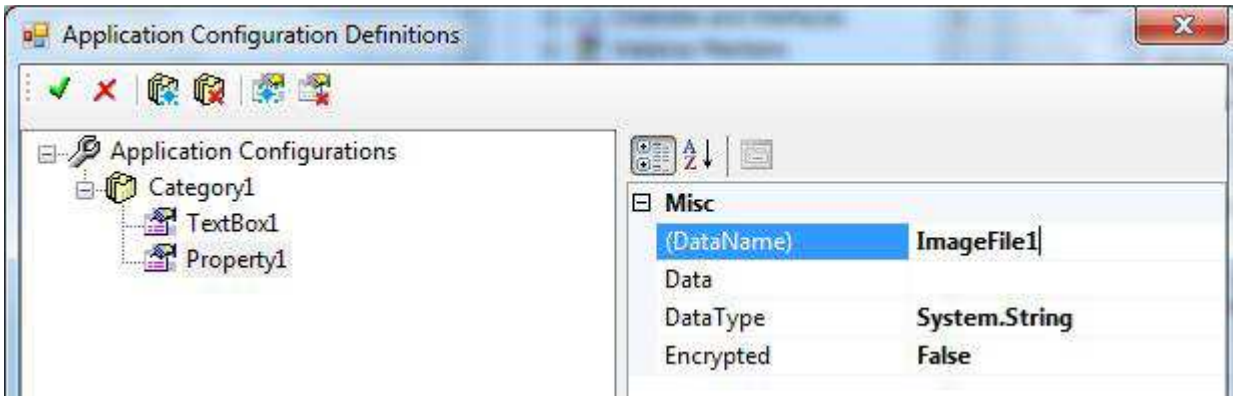
Create a new configuration value

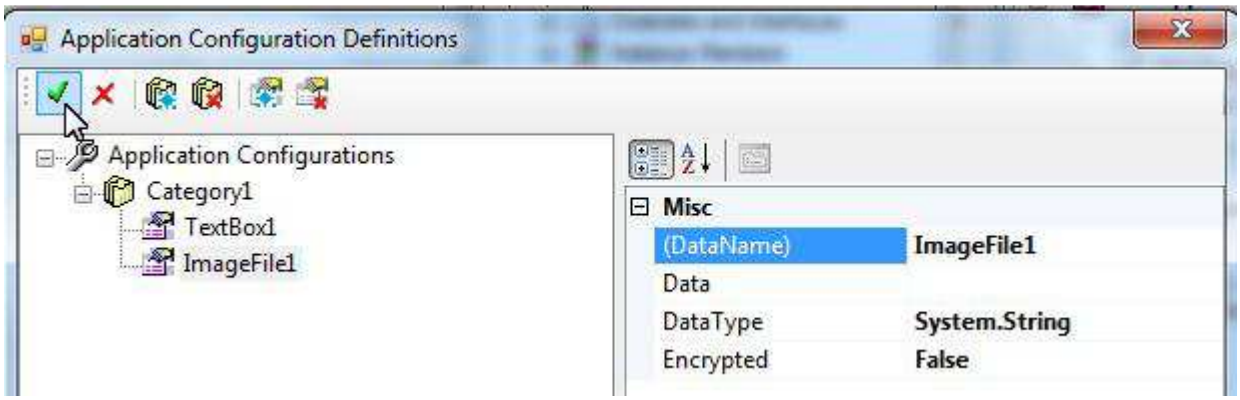
For remembering the image file path, we need to create a new configuration value for it:





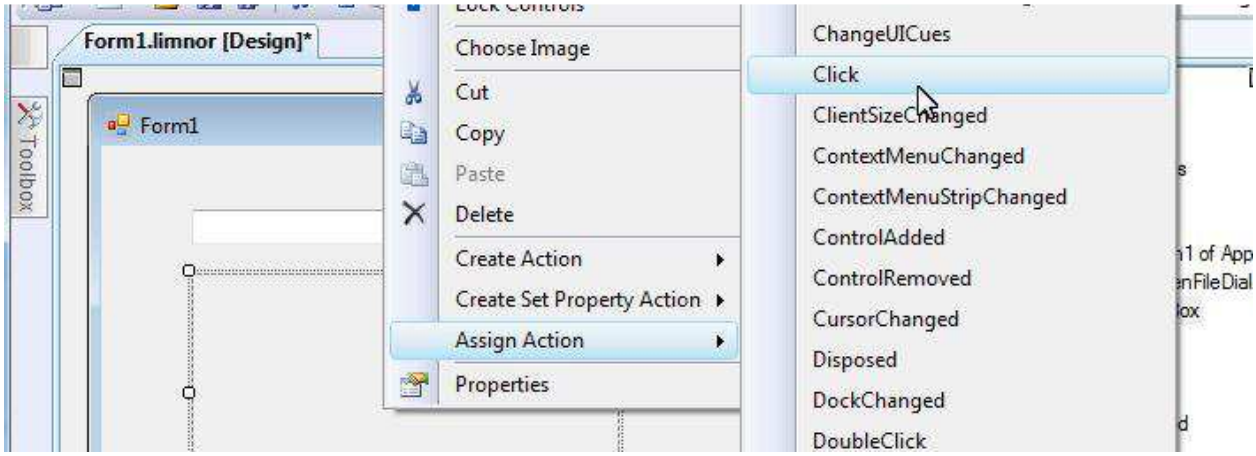
Name the configuration value “ImageFile1”:



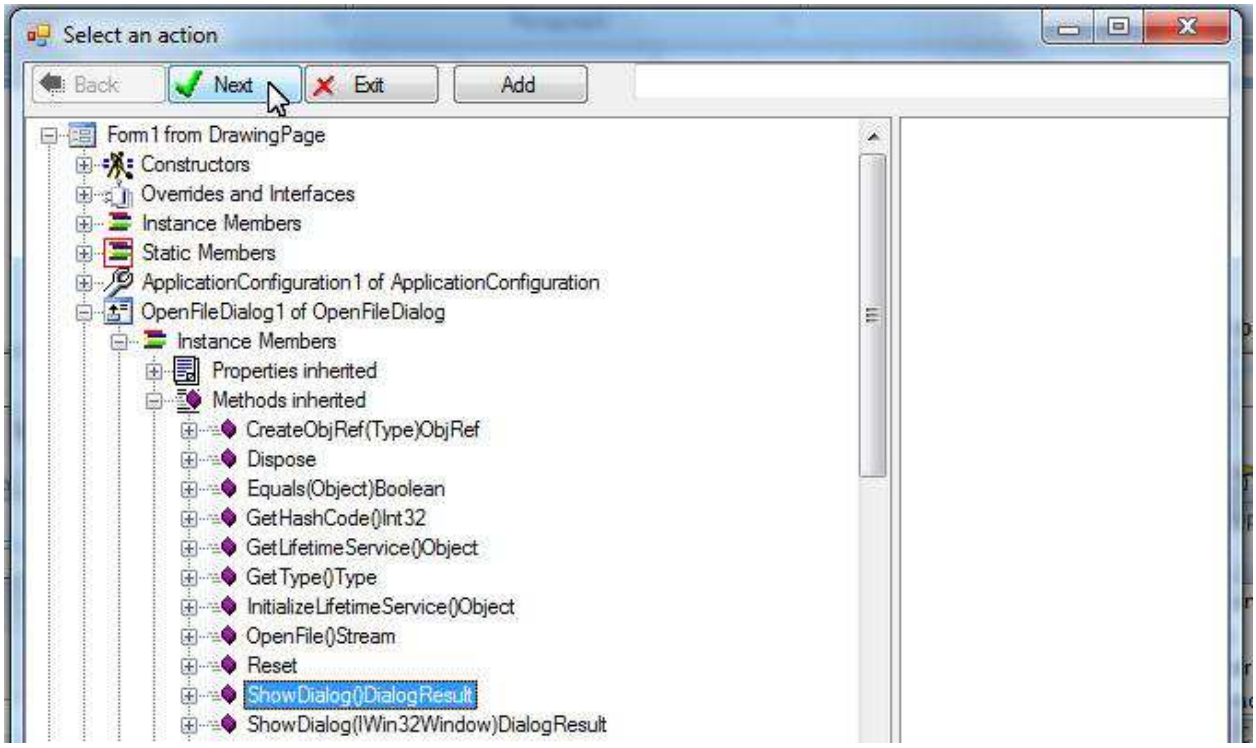


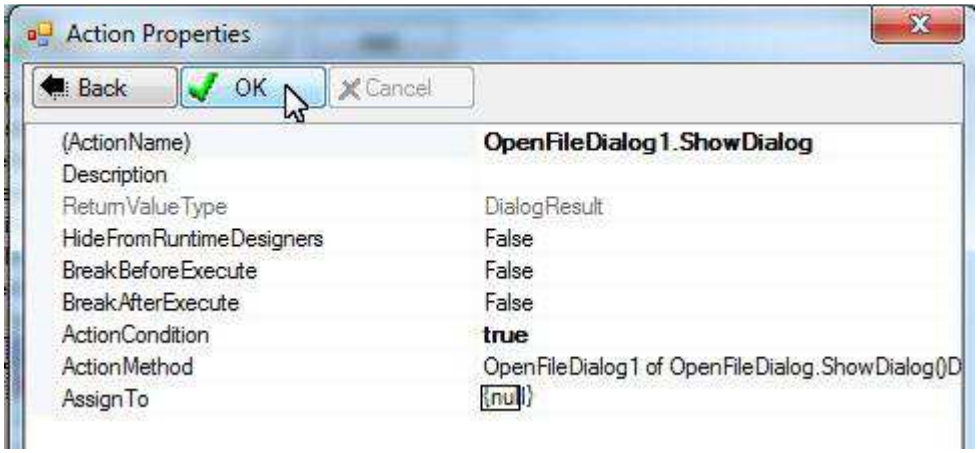
Remember the image file path

For this sample, we choose to remember the image file path when the user selects an image file path for the picture box. Let’s make programming to let the user select image file when the picture box is clicked:

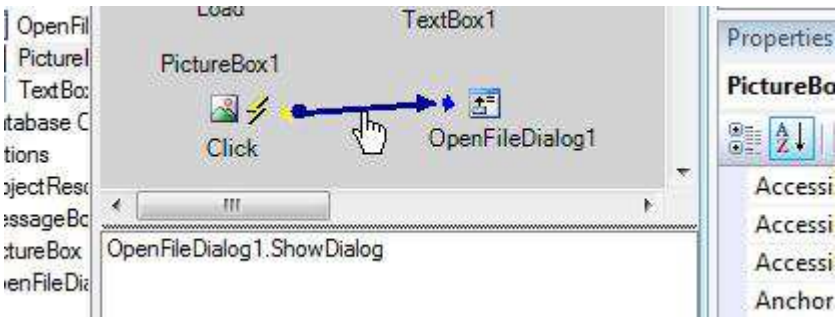


Show the OpenFileDialog for selecting image file:



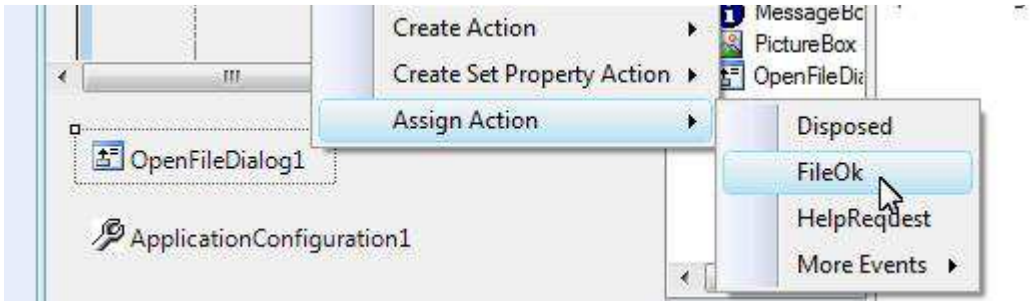


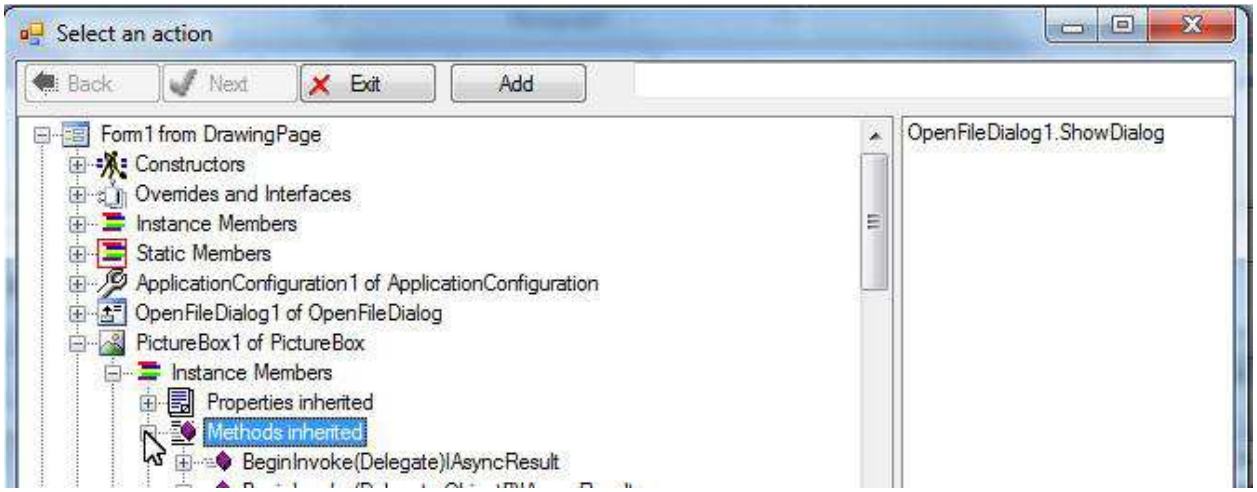
The action is created and assigned to the picture box:

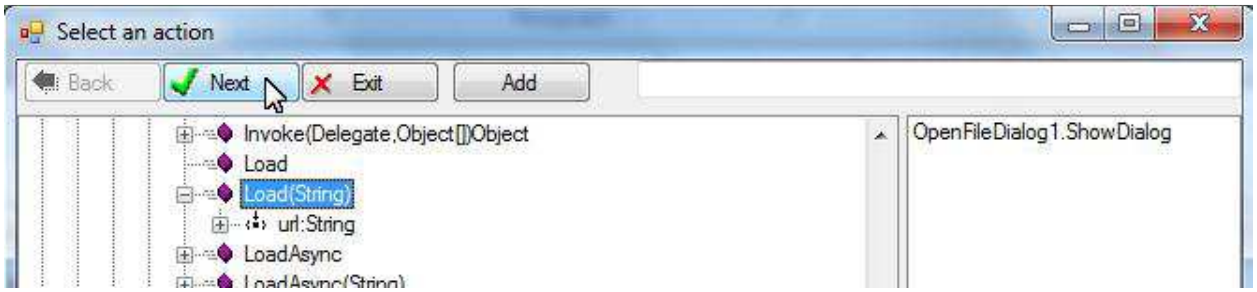


If the user selects a file path then the OpenFileDialog components fires a FileOK event. We may handle this event to load the image to the picture box and also pass the file path to the configuration to remember it.

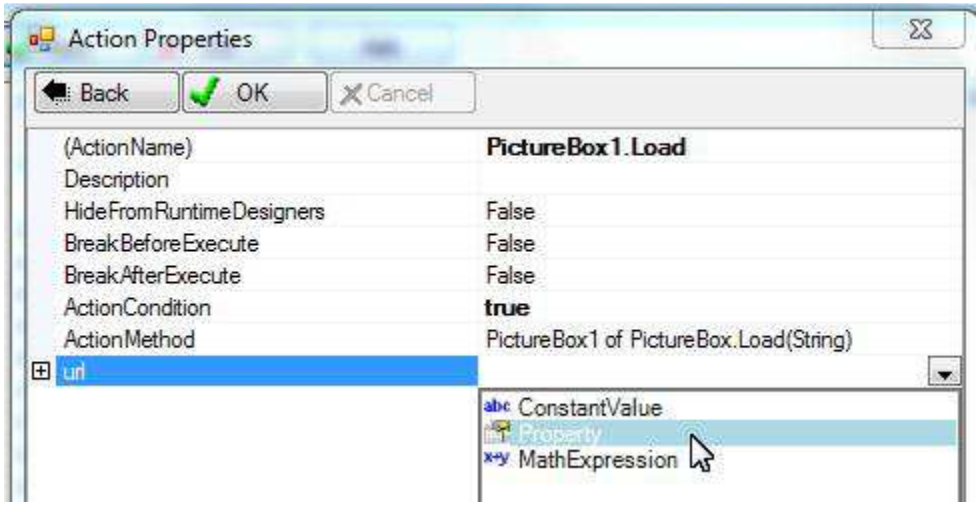
Use the image file path

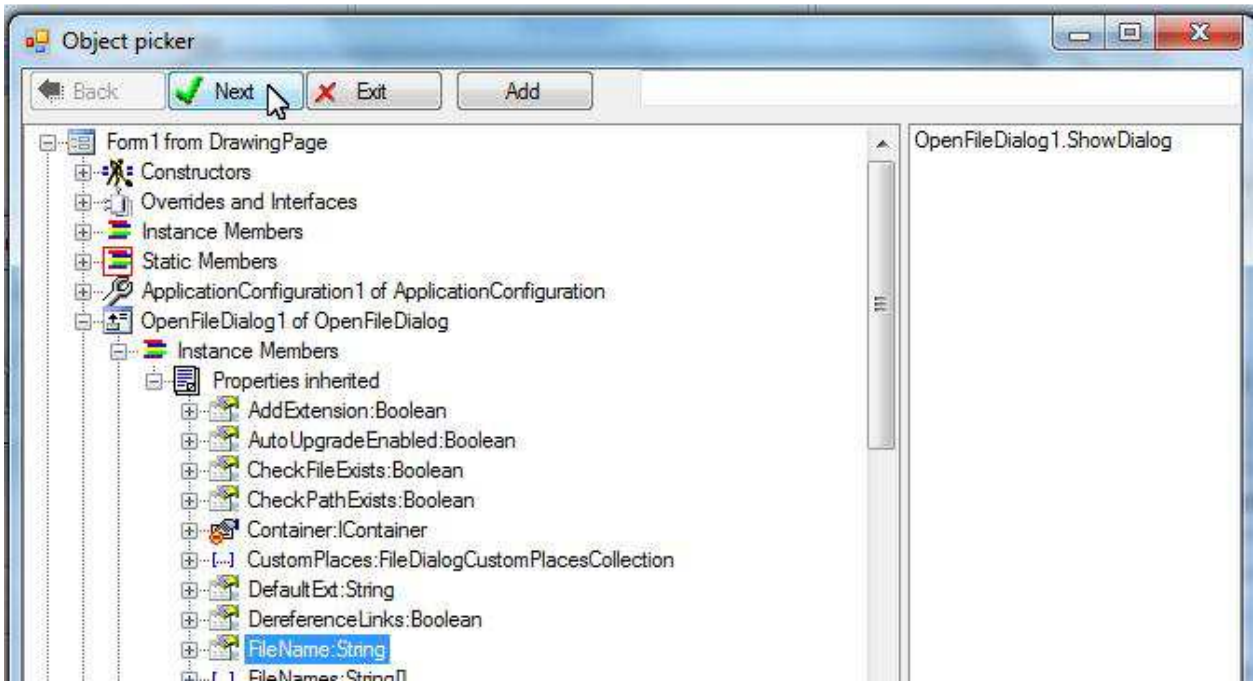




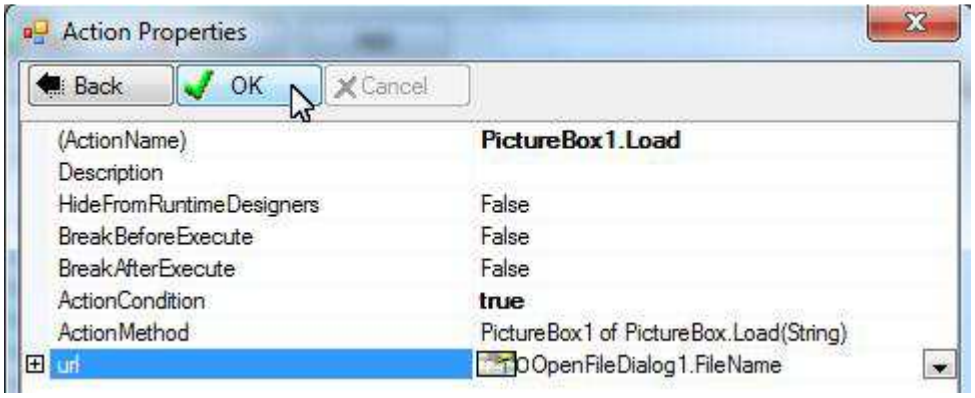


Pass the image file path to the action:

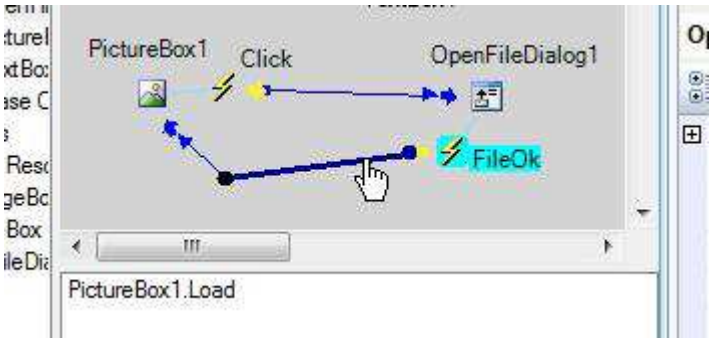




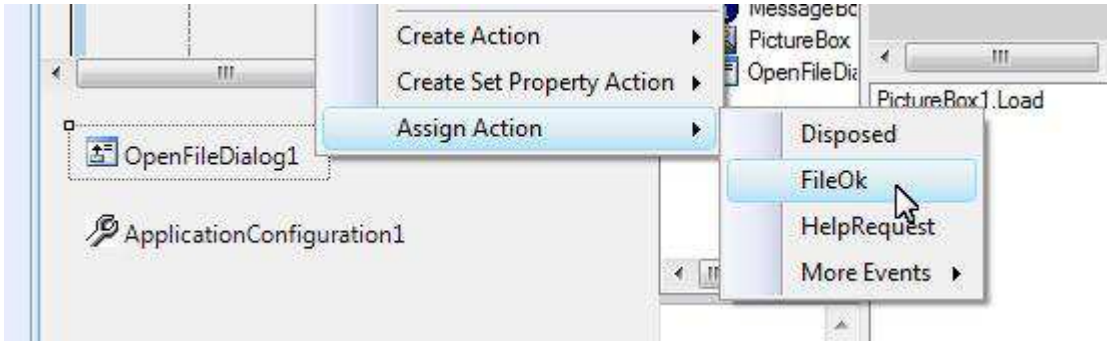
Click OK:

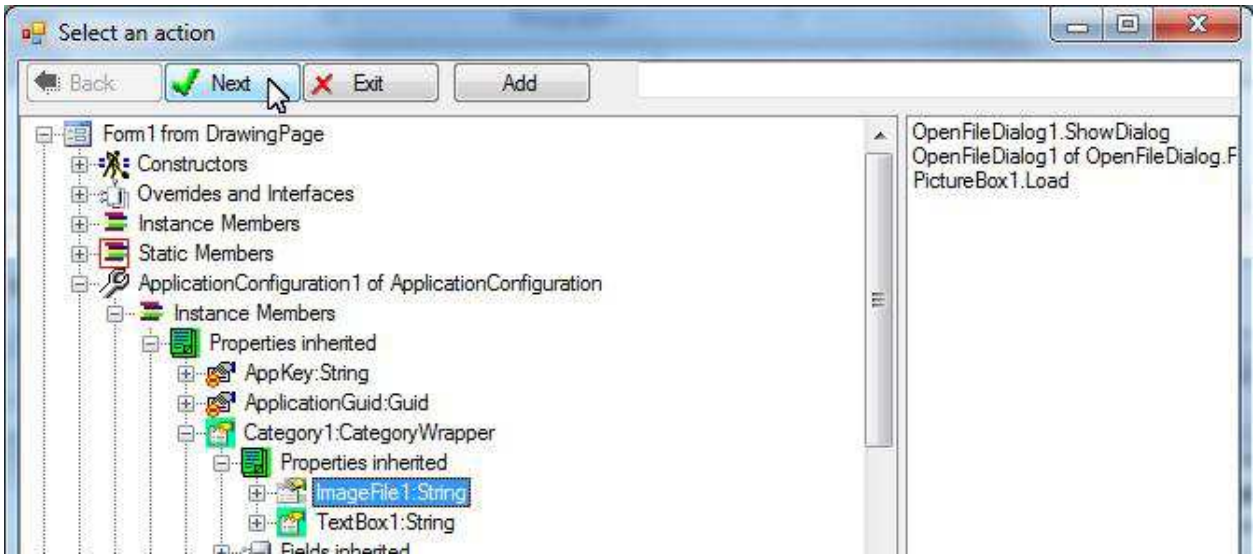


The action is created and assigned to the event:

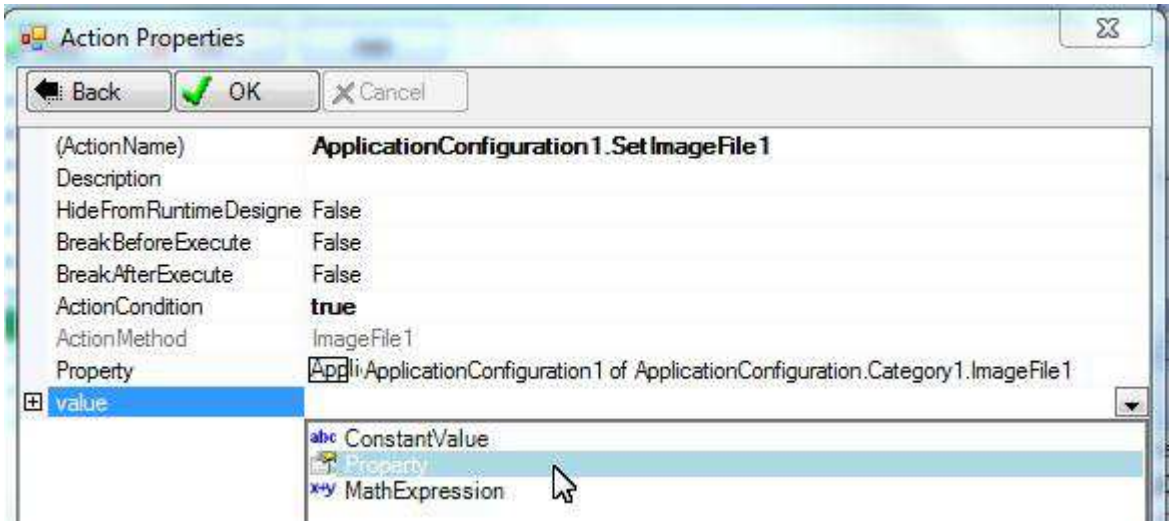


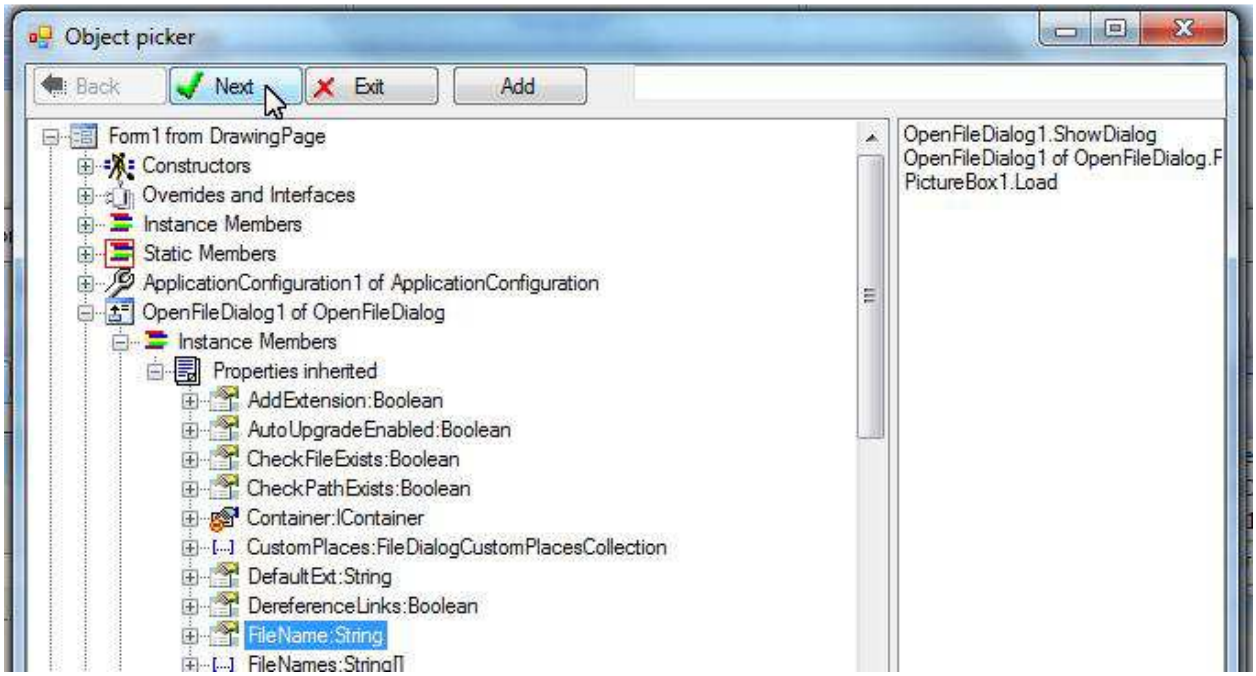
Remember the image file path



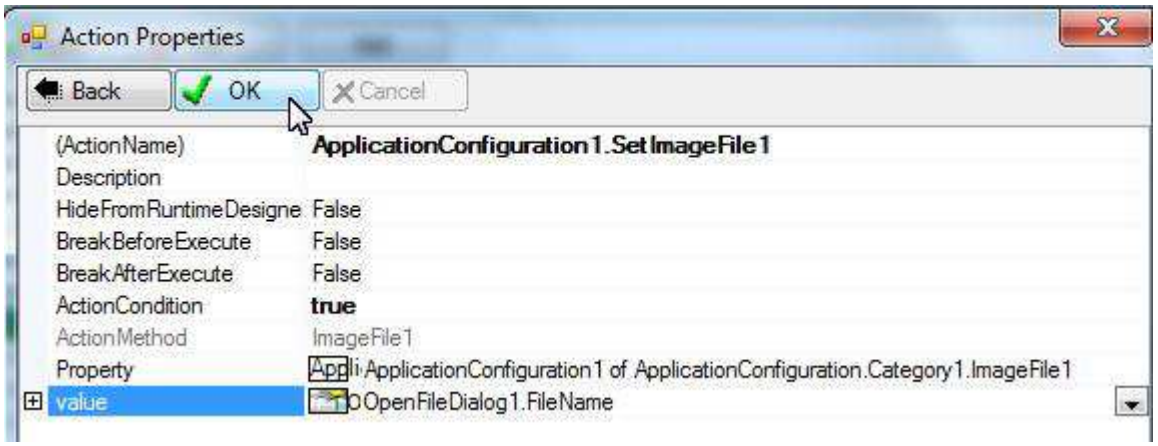


Pass the image file path to the action:

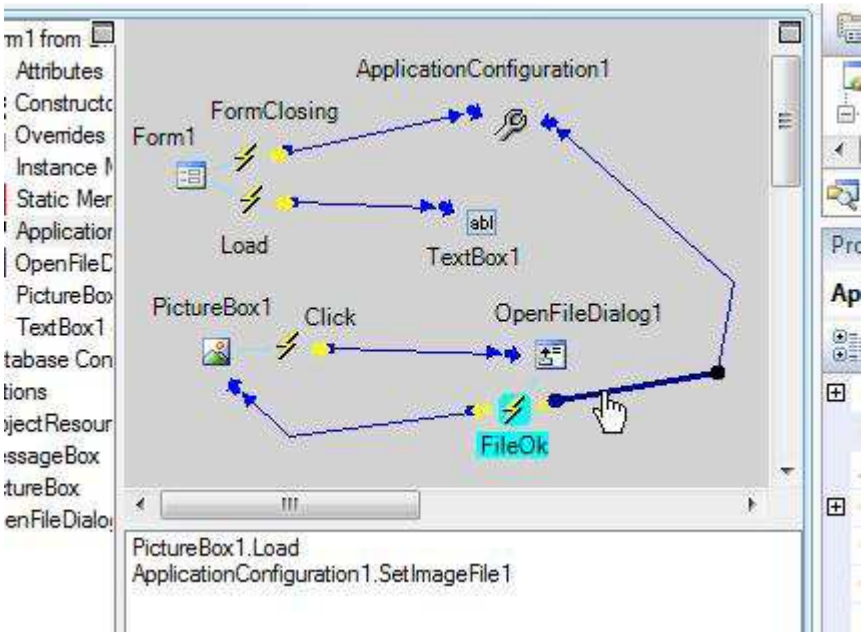




Click OK:

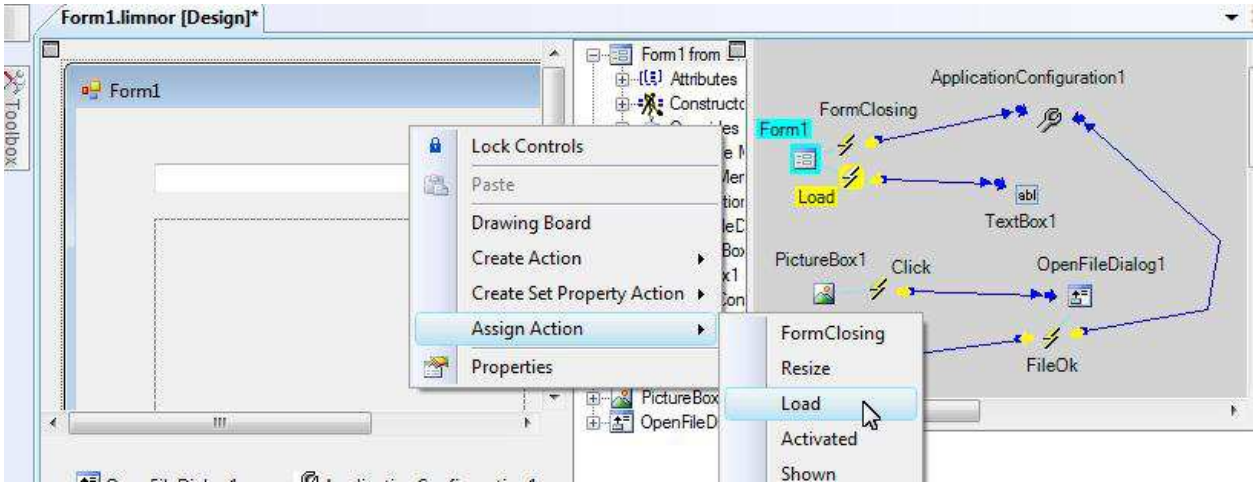


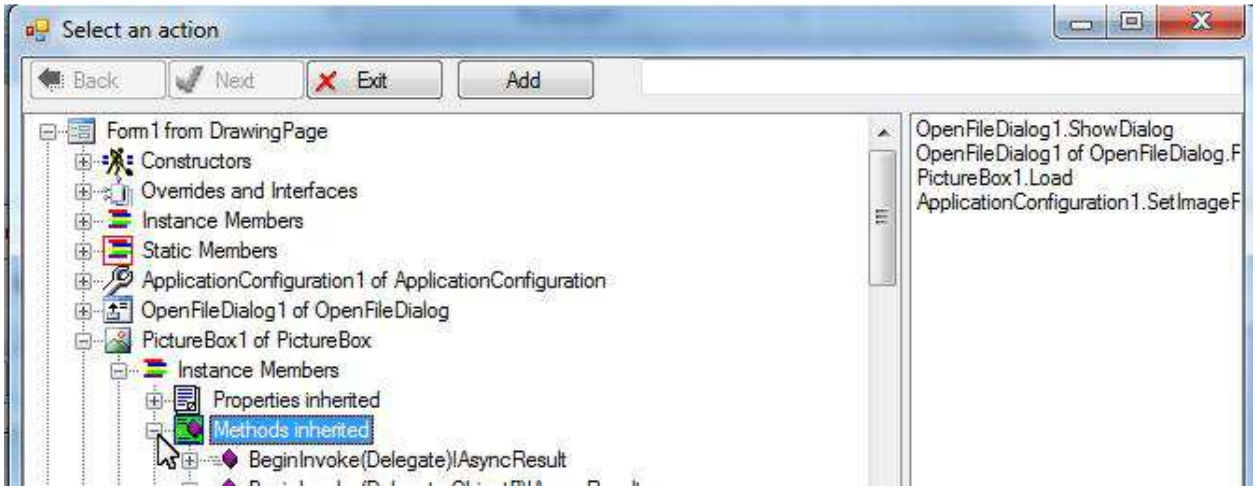
The action is created and assigned to the event:

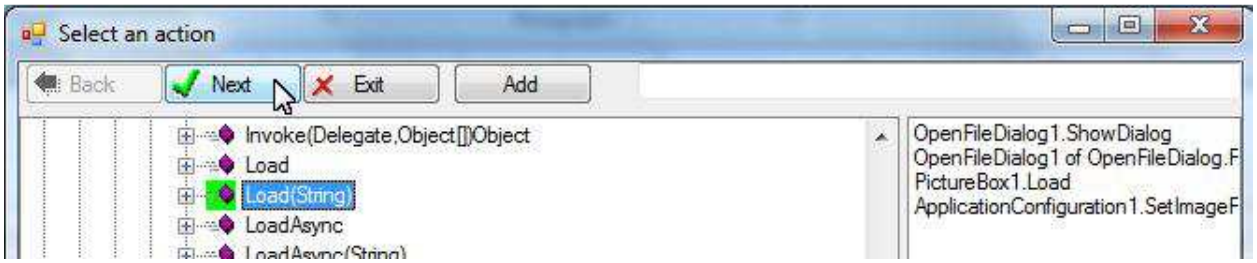


Apply configuration value

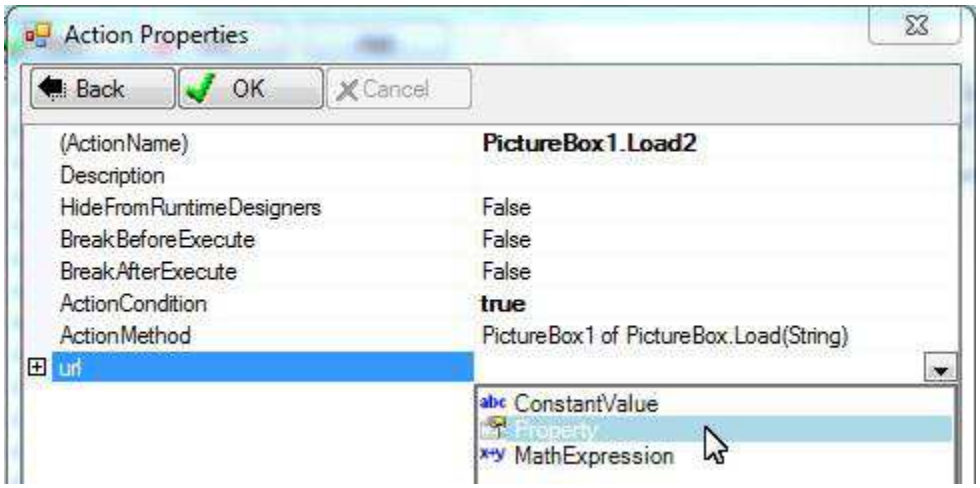
We choose to apply the image file path configuration at the time of Load event of the form:

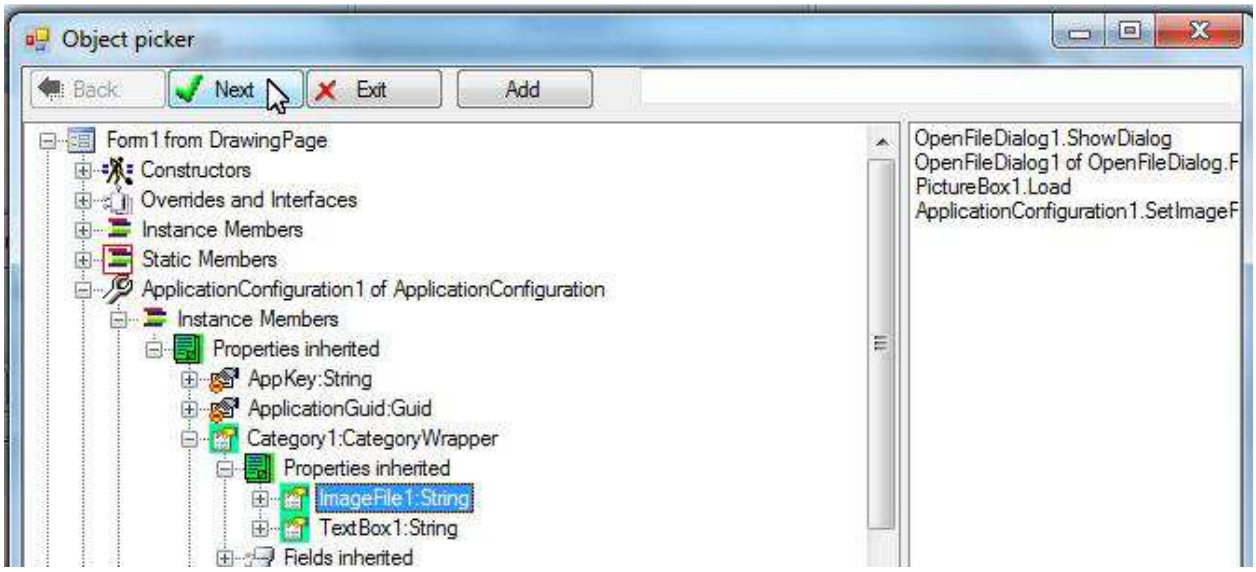




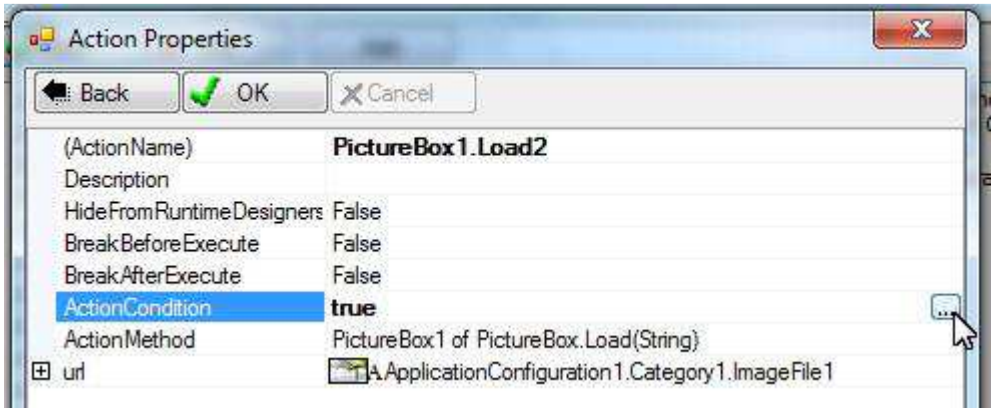


Path the configuration value to the action:

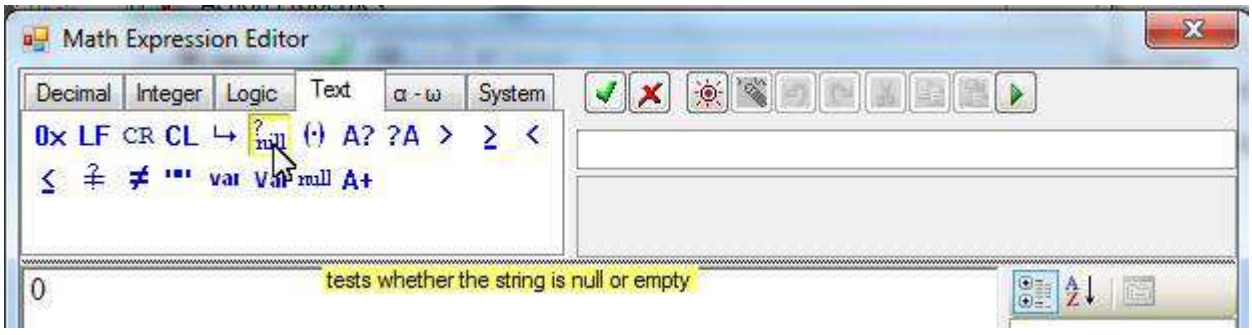


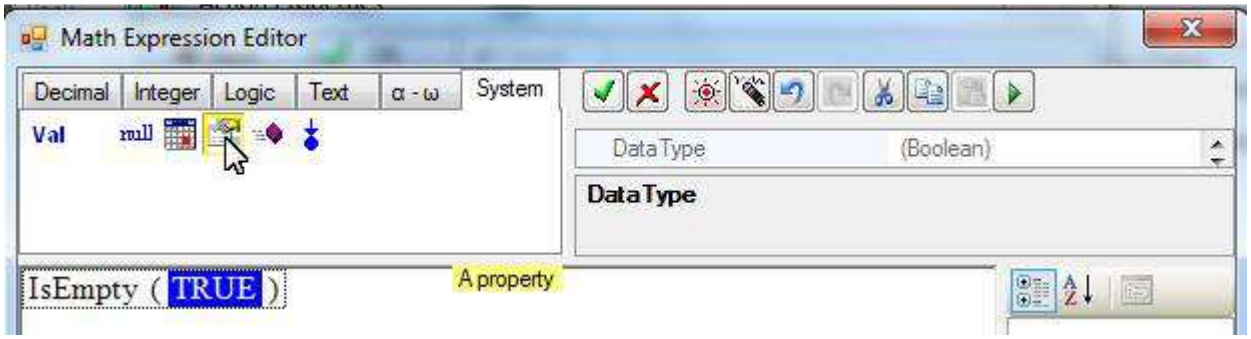


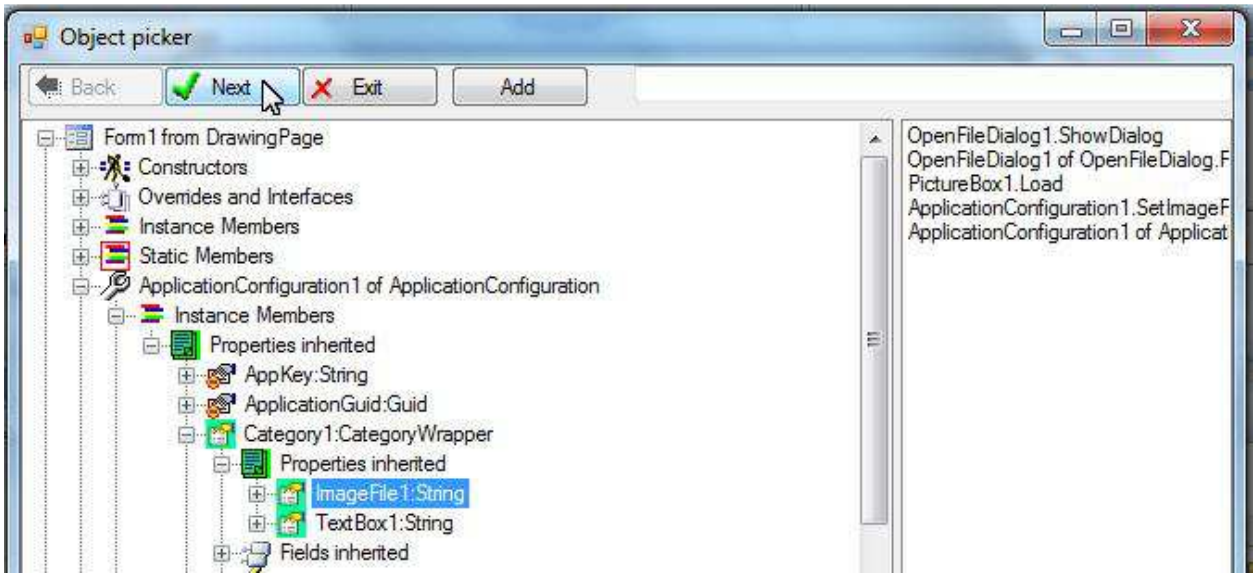
Set the ActionCondition such that the action executes only if the configuration value is a valid file path:



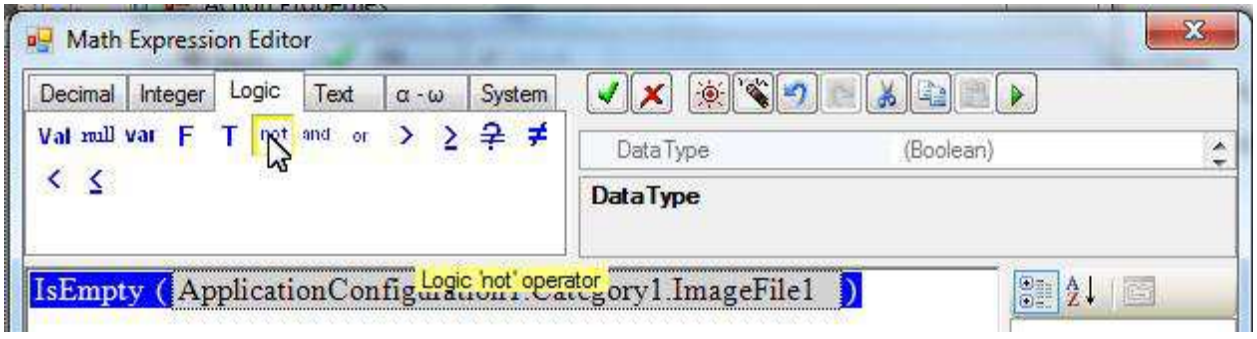
First, make sure the configuration value is not empty:



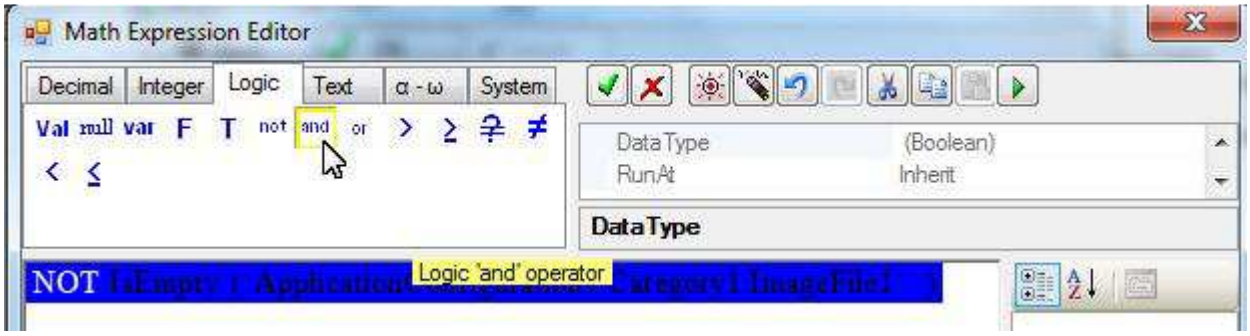




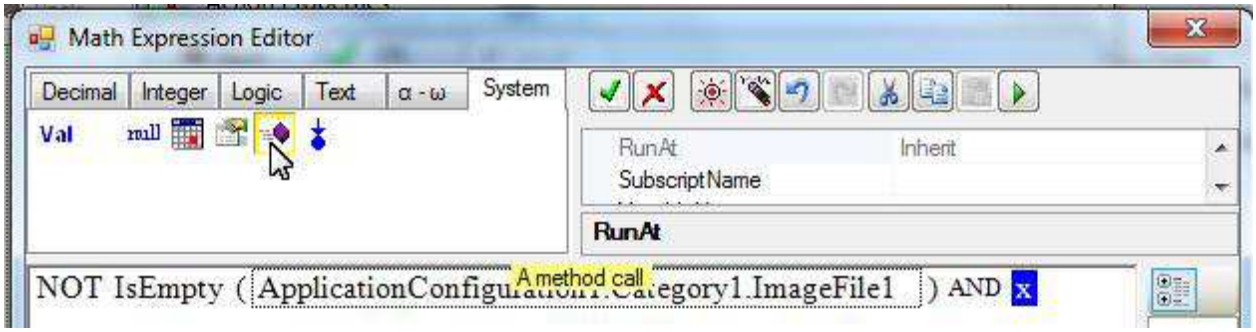
Apply “NOT” operator to the expression:

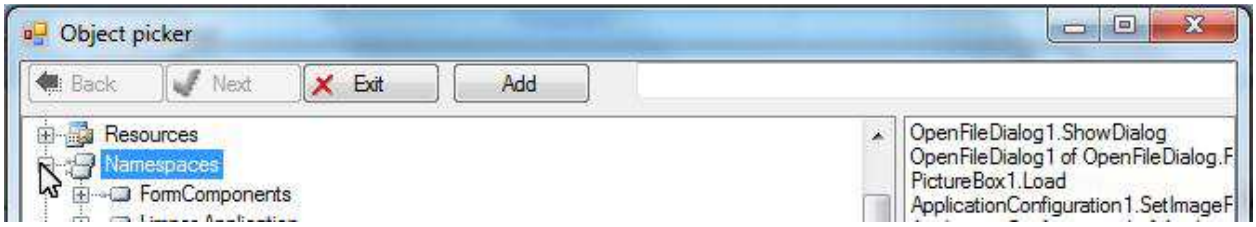


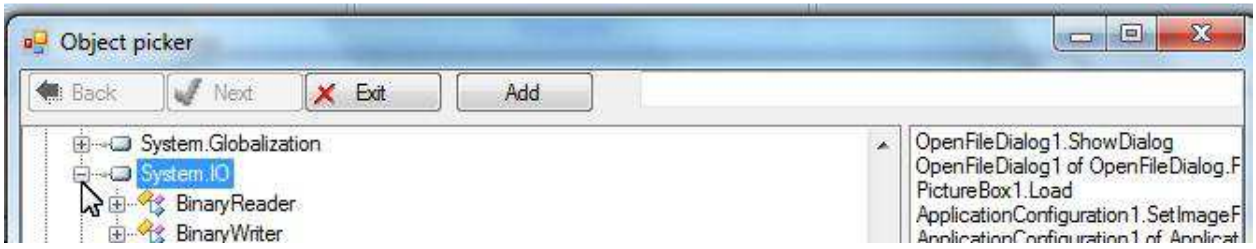
Add “AND” to further check file existence:

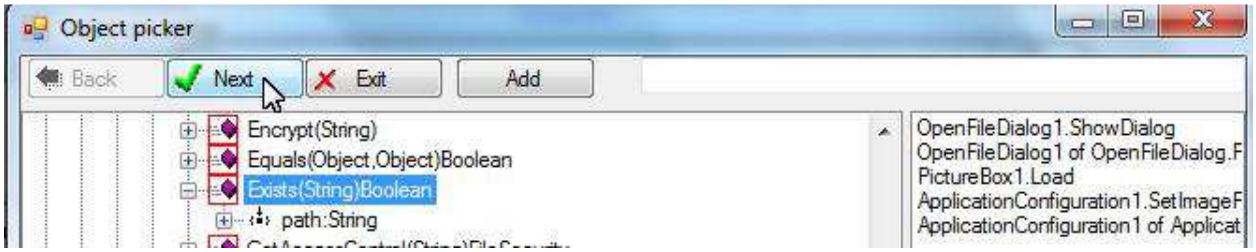


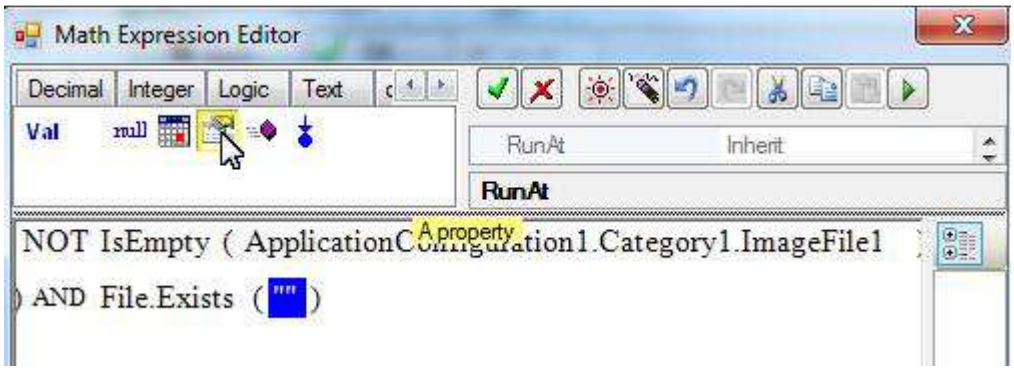
Select Exists method of File class:

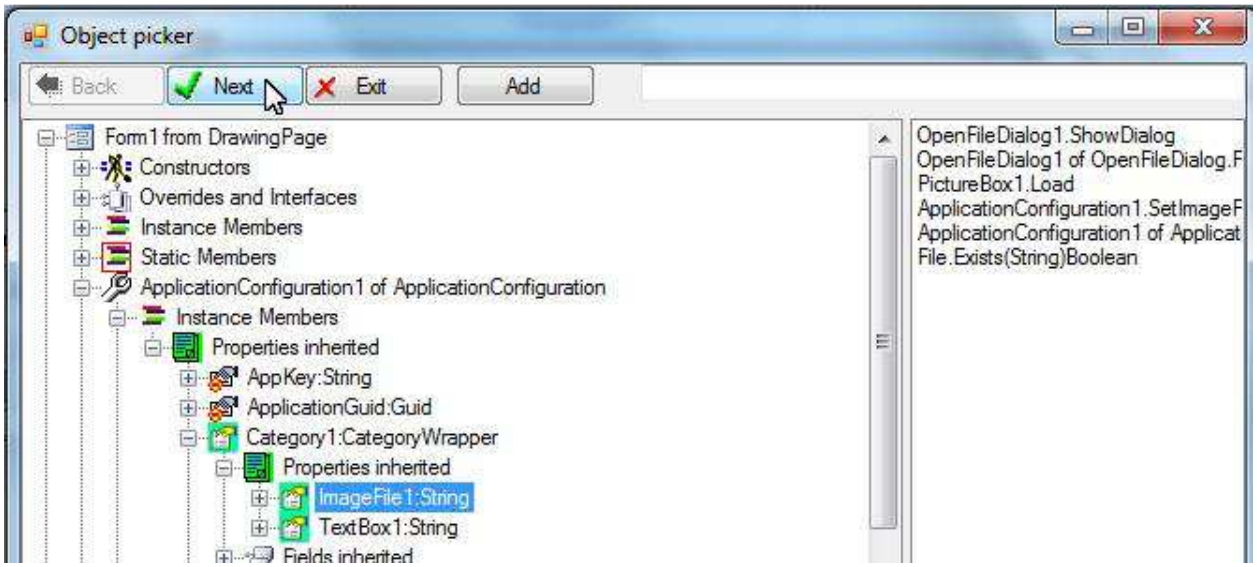


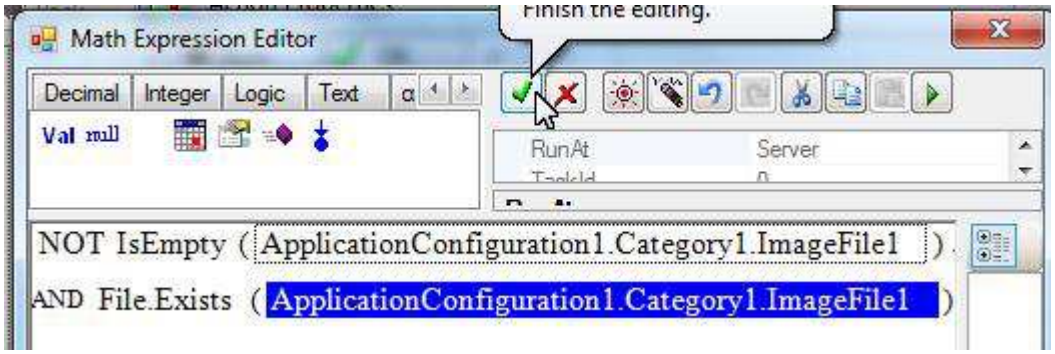


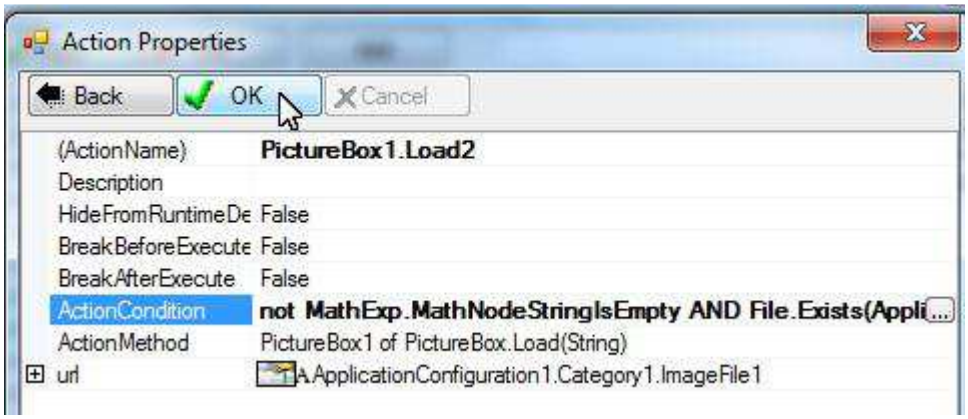



Object picker
Back
Next
Exit
Add
Encrypt(String)
Equals(Object,Object)Boolean
Exists(String)Boolean
path:String
OpenFileDialog1.ShowDialog
OpenFileDialog1 of OpenFileDialog.F
PictureBox1.Load
ApplicationConfiguration1.SetImageF
ApplicationConfiguration1 of Applicat

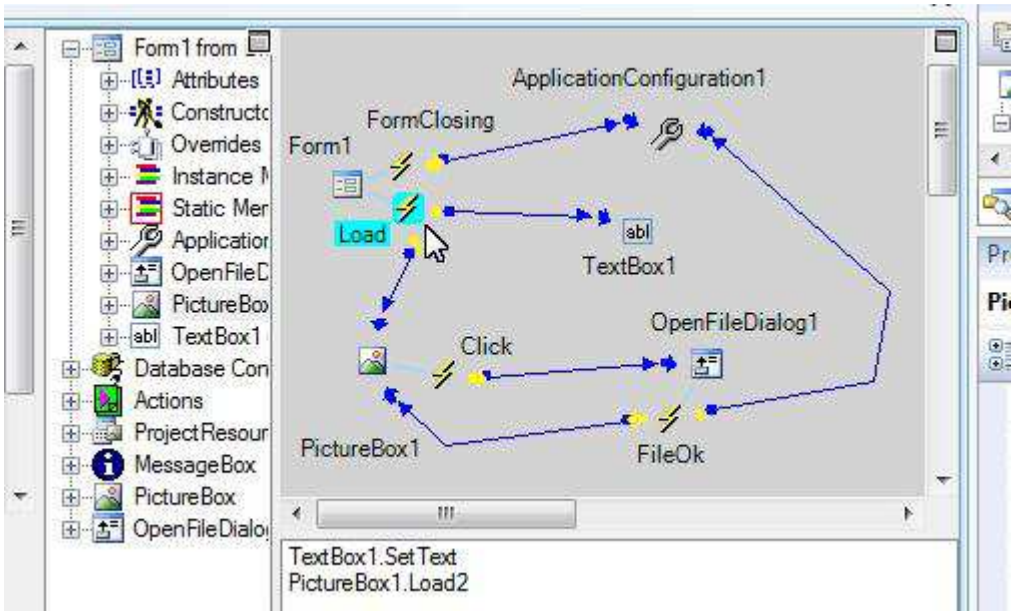

Math Expression Editor
Decimal
Integer
Logic
Text
Val
null
RunAt
Inherit
RunAt
A property
NOT IsEmpty (ApplicationConfiguration1.Category1.ImageFile1
) AND File.Exists ("")


Object picker
Back
Next
Exit
Add
Form1 from DrawingPage
Constructors
Overrides and Interfaces
Instance Members
Static Members
ApplicationConfiguration1 of ApplicationConfiguration
Instance Members
Properties inherited
AppKey:String
ApplicationGuid:Guid
Category1:CategoryWrapper
Properties inherited
ImageFile1:String
TextBox1:String
Fields inherited
OpenFileDialog1.ShowDialog
OpenFileDialog1 of OpenFileDialog.F
PictureBox1.Load
ApplicationConfiguration1.SetImageF
ApplicationConfiguration1 of Applicat
File.Exists(String)Boolean


Finish the editing.
Math Expression Editor
Decimal
Integer
Logic
Text
Val
null
RunAt
Server
NOT IsEmpty (ApplicationConfiguration1.Category1.ImageFile1)
AND File.Exists (ApplicationConfiguration1.Category1.ImageFile1)

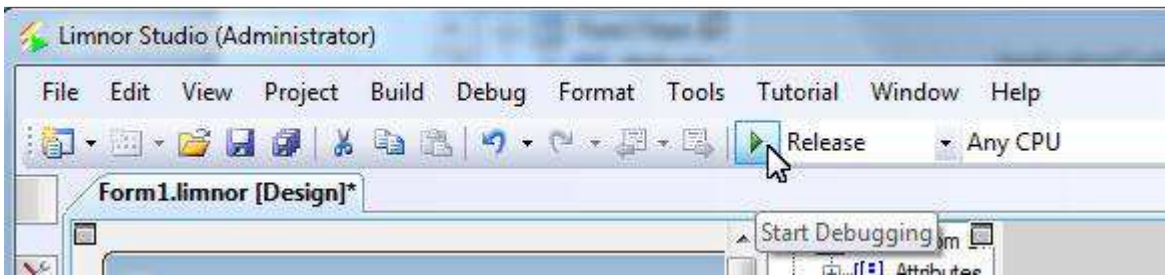


The action is created and assigned to the event:

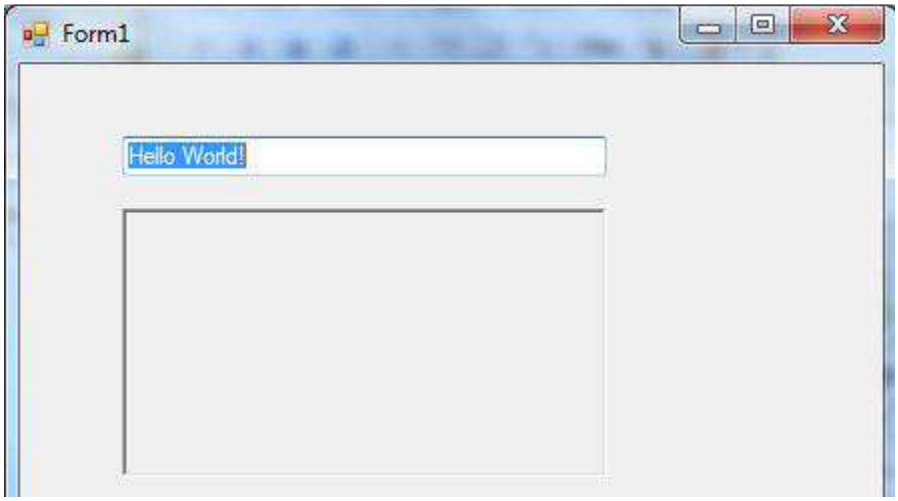


Test

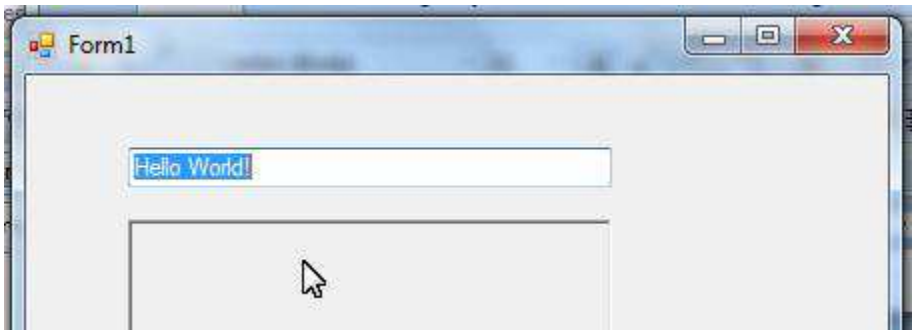
Click Run button to compile and run the application:



The form appears:

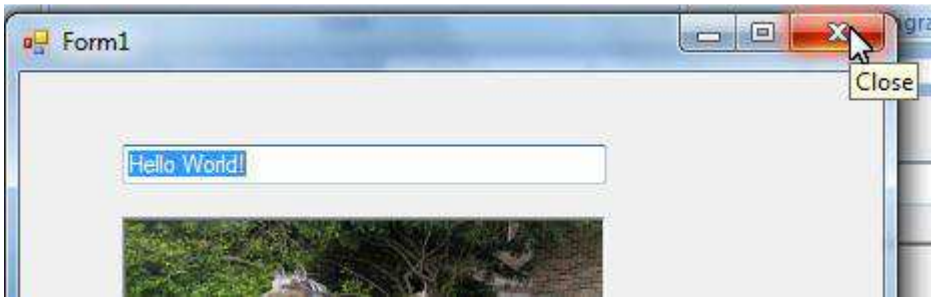


Click the picture box to select an image:

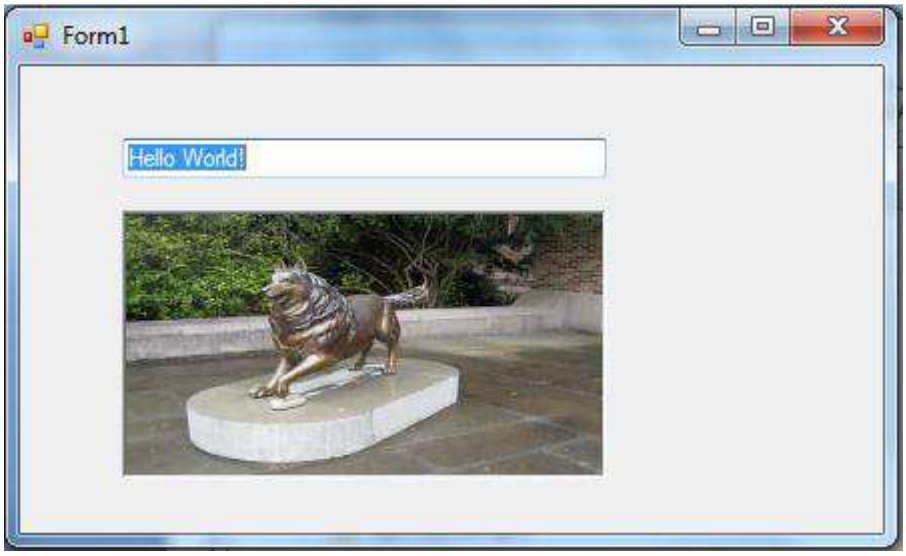


Select an image file. The image appears in the picture box:

Close the form:



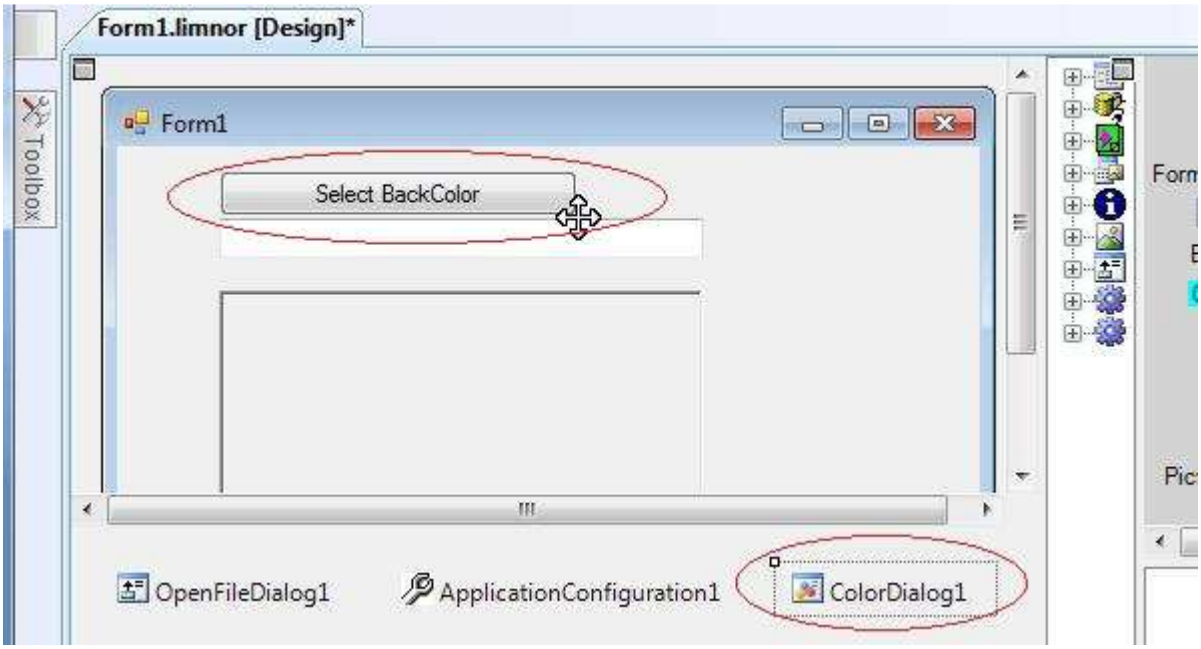
Start application again by double-clicking the EXE file. The form appears. The picture box is loaded with the image:



Color Configuration

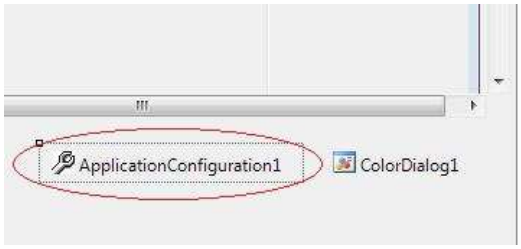
Sample application

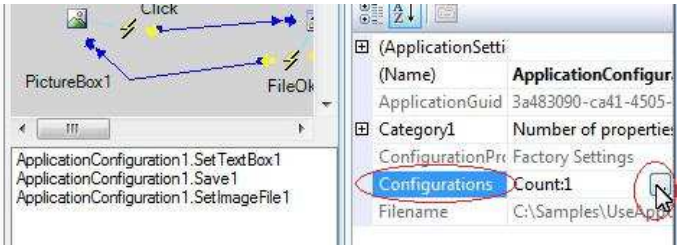
Suppose we want to let the user change the background color of the form. We add a button to launch a Color Dialog. We apply the color selected by the user to the BackColor property of the form. We also want to remember the color selection so that the next time the form is displayed the form uses the same background color.

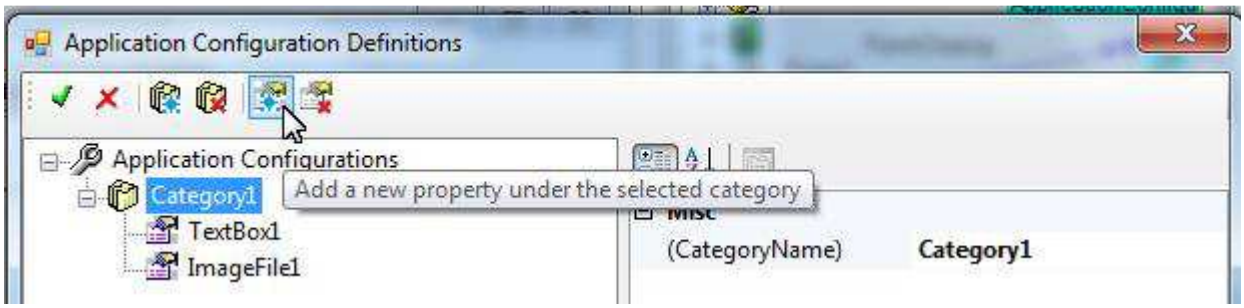


Create a new configuration value

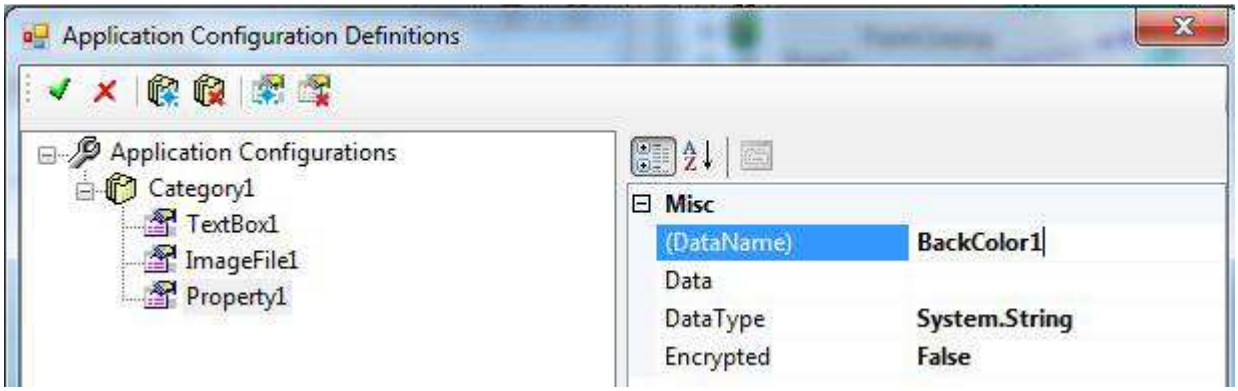
For remembering the BackColor property of the form, we need to create a new configuration value for it:





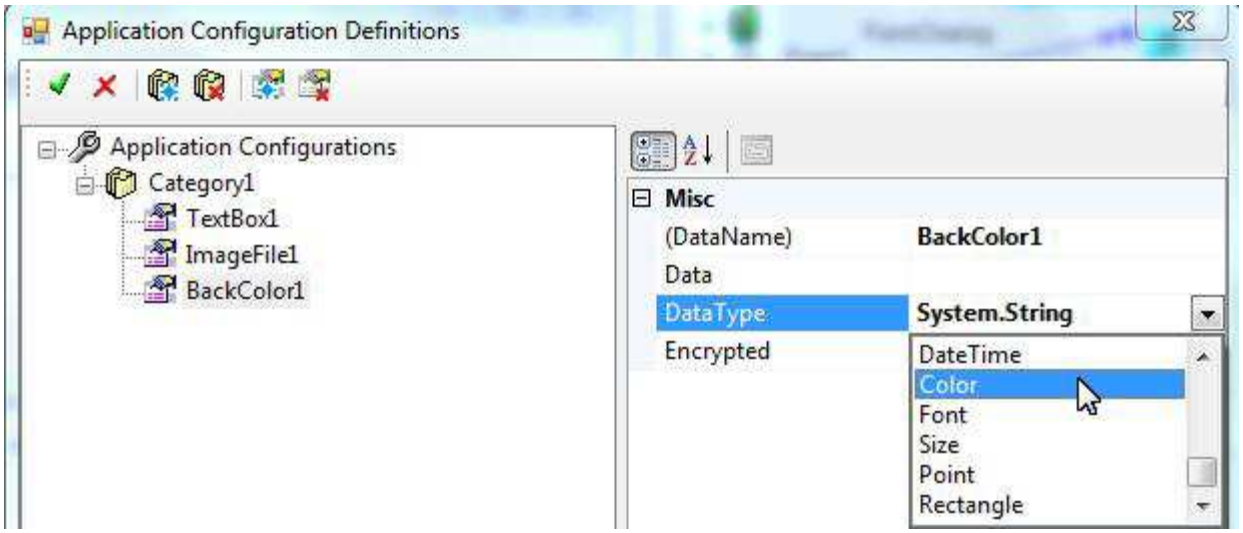


Name the new configuration value “BackColor1”:



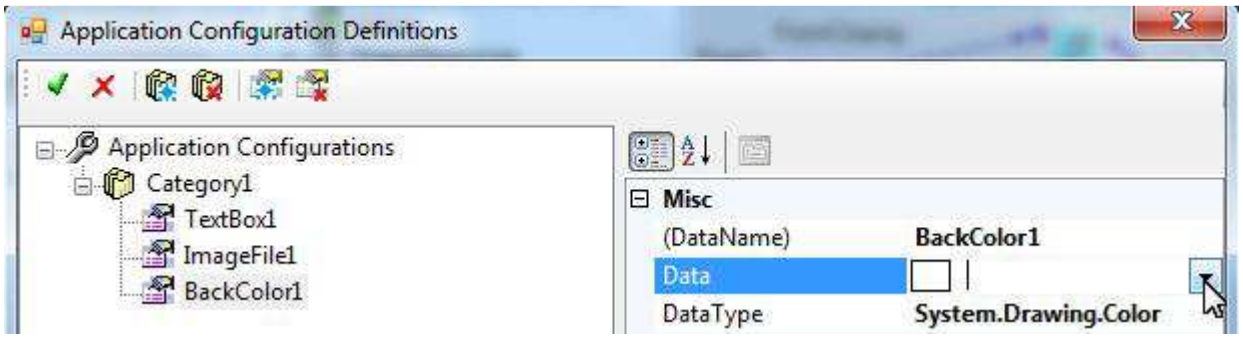
Set configuration value type

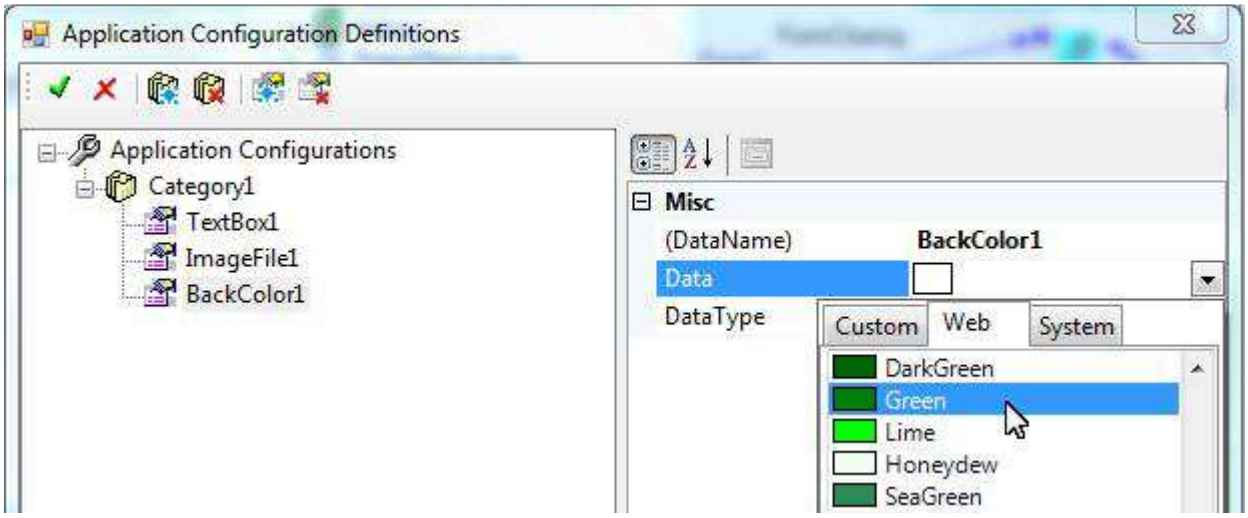
We want to make the configuration a color value:



Set default configuration value

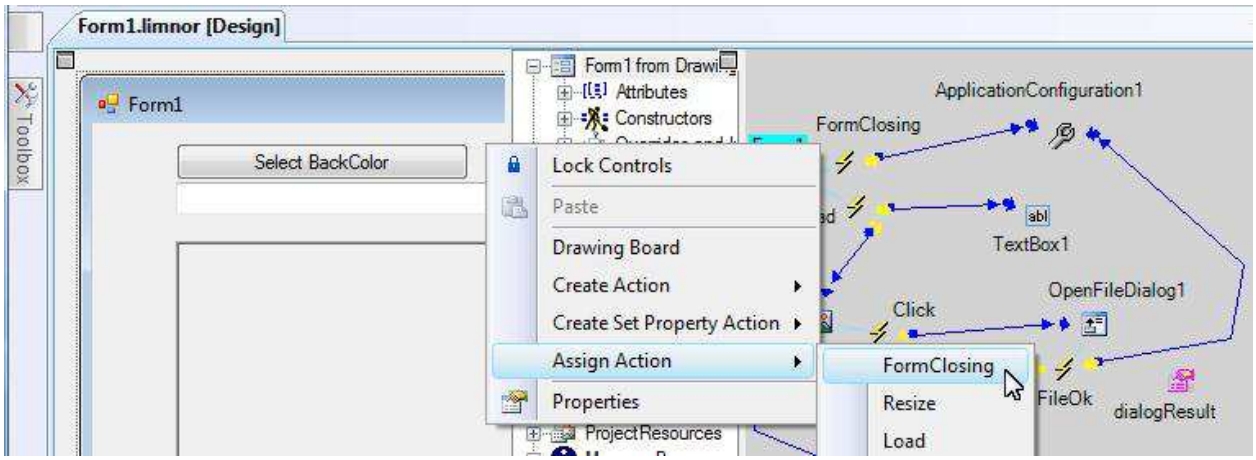
We may give a default value to a configuration value. For example, let's set the default value for "BackColor1" to green:

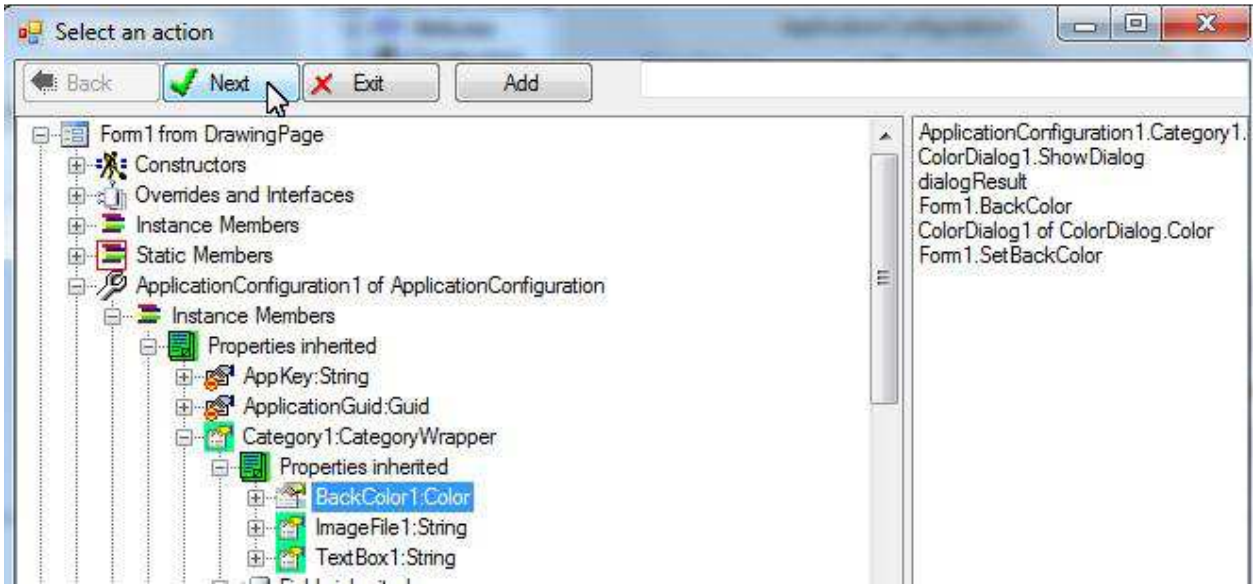




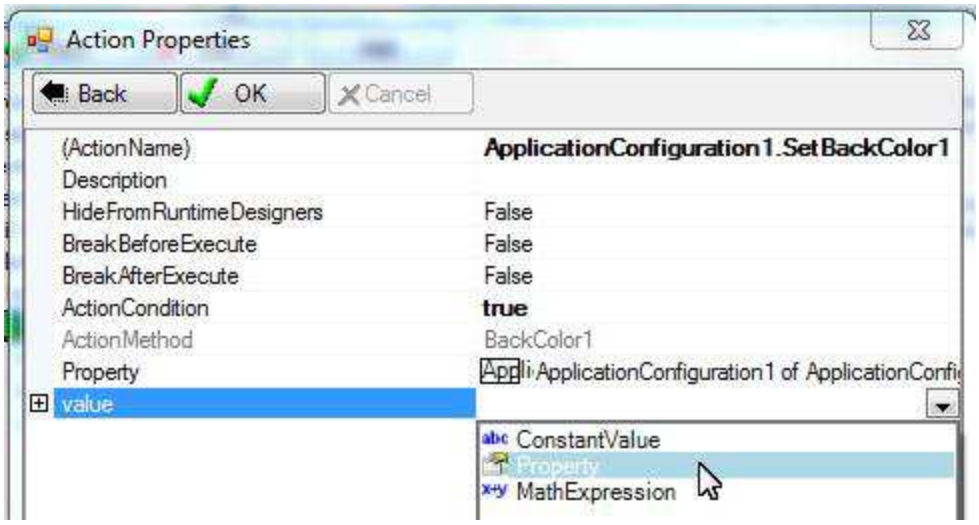
Remember the BackColor property

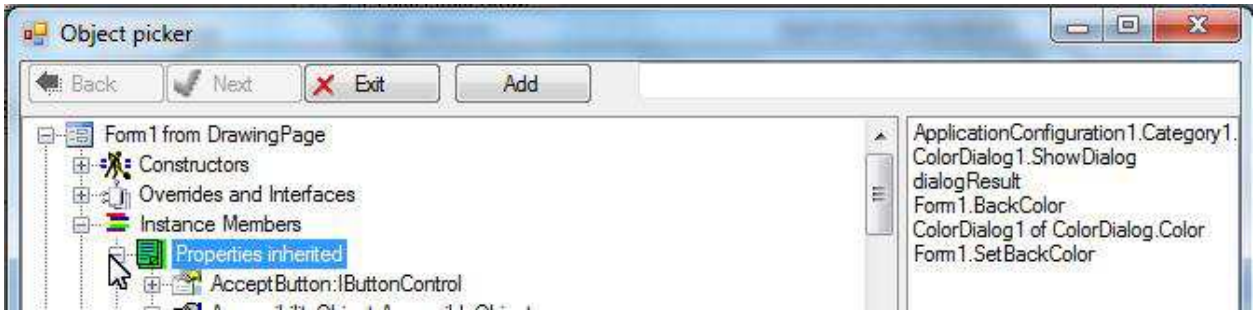
For this sample, we may choose to remember the BackColor when the user changes it, or we may remember it at the time when the form is closing. We choose to remember it at the time of form closing so that the action is not tied with color selection programming. So, we are not going to show the programming of selecting background color.

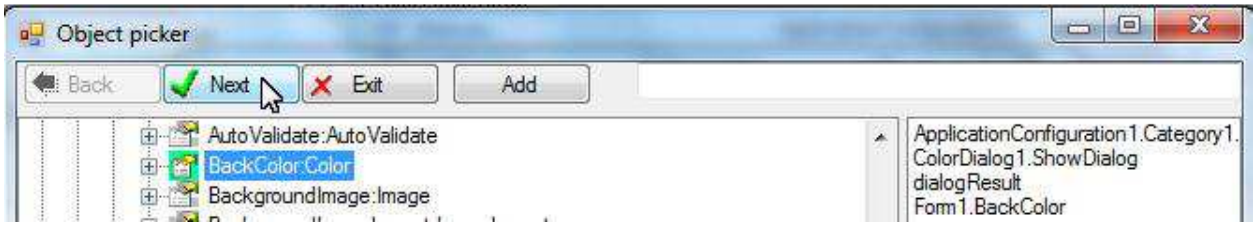


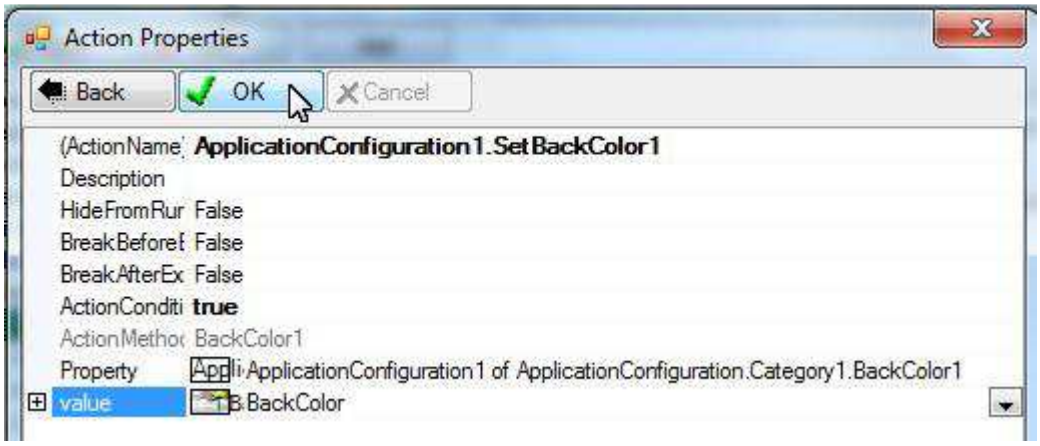


Pass the BackColor to the action:

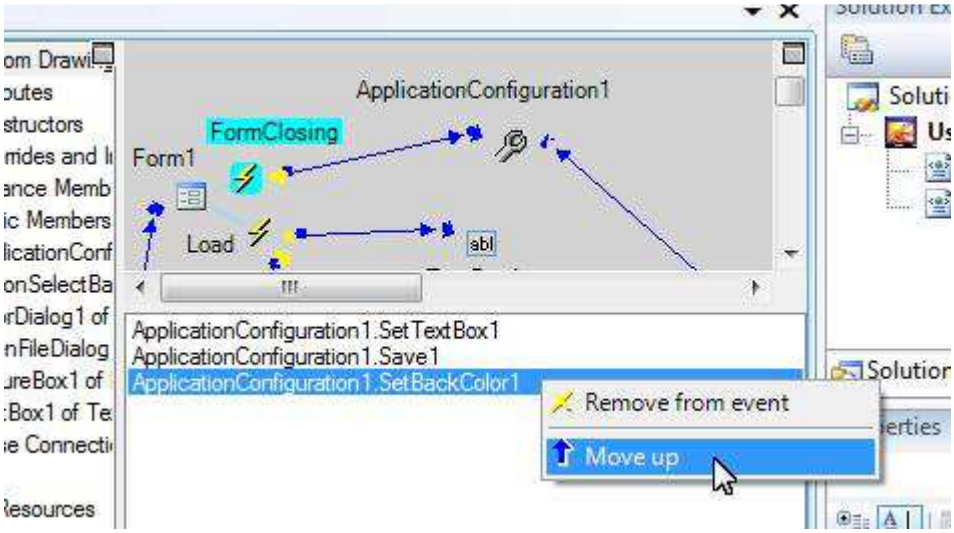


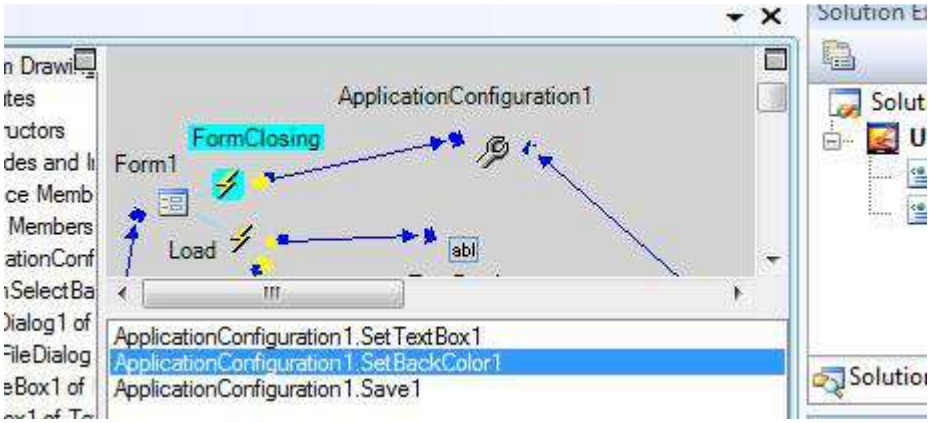






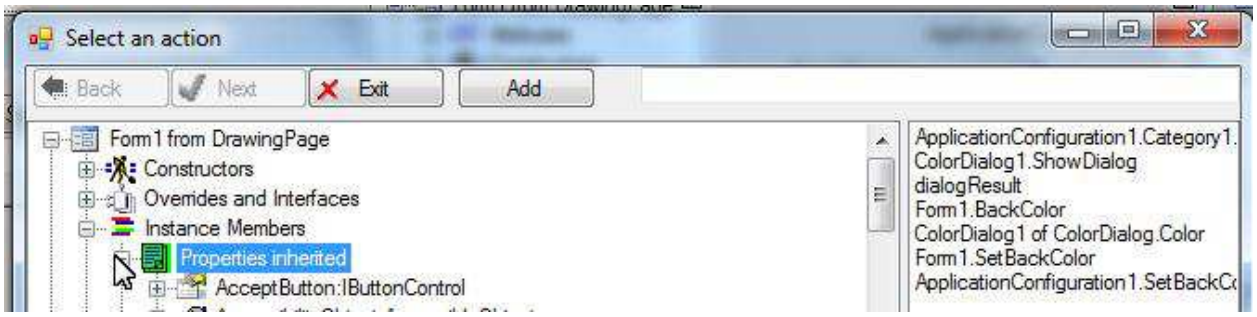
The action is created and assigned to the event. We need to move the action up before the Save action:

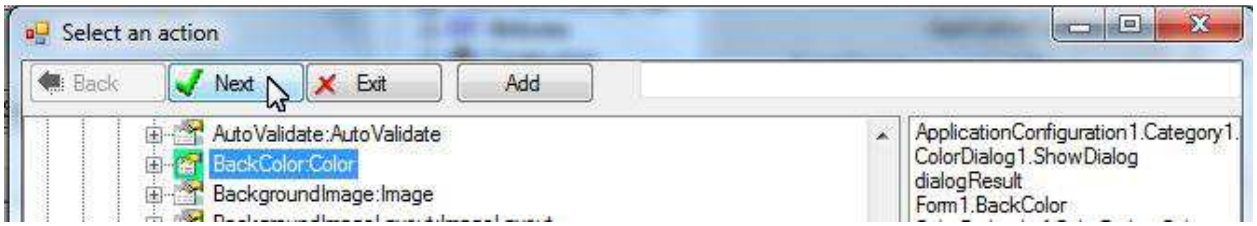




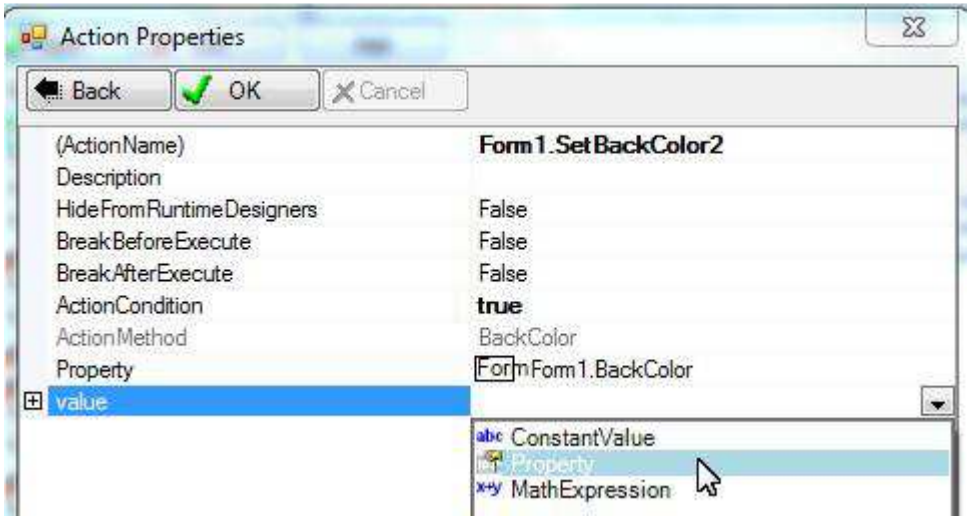
Apply configuration value

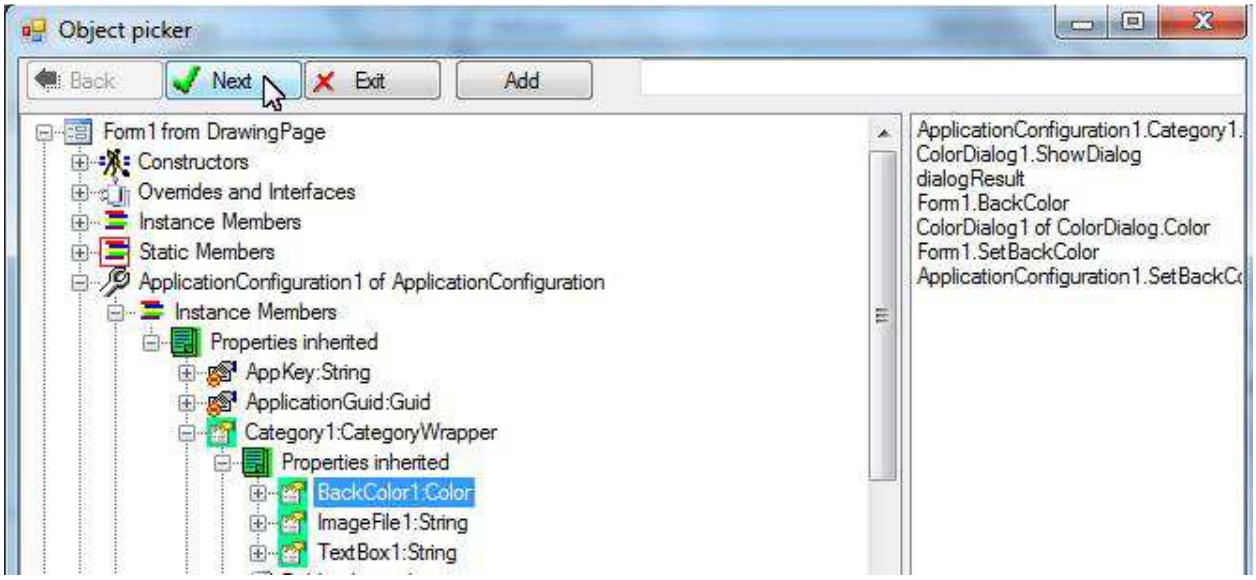
We choose to apply the configuration value “BackColor1” at the time of Load event of the form:

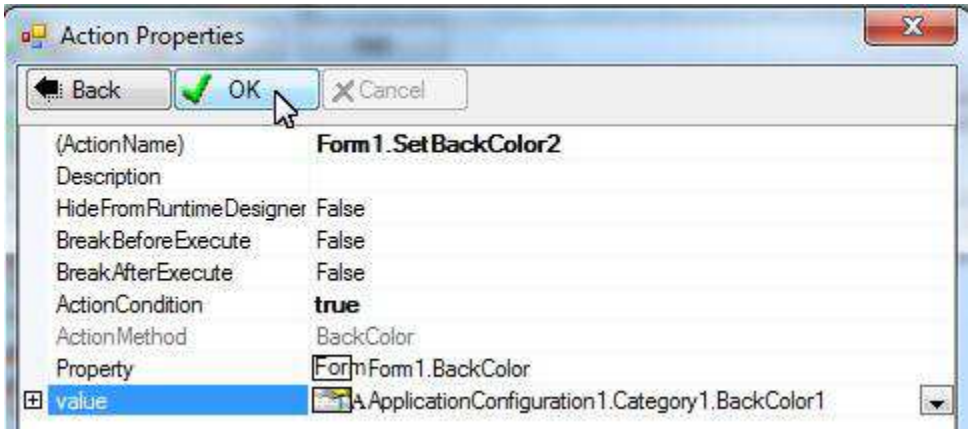




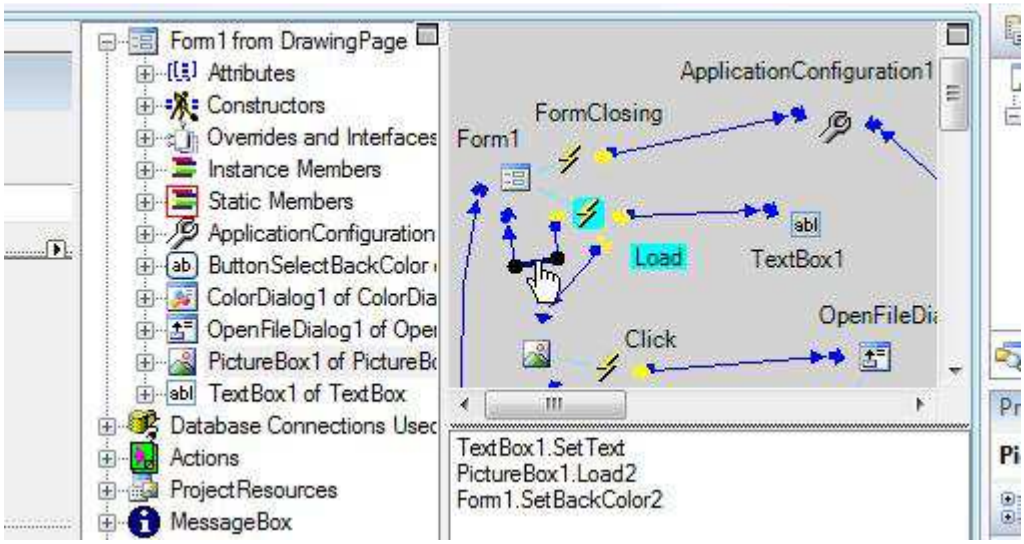
Pass the configuration value to the action:





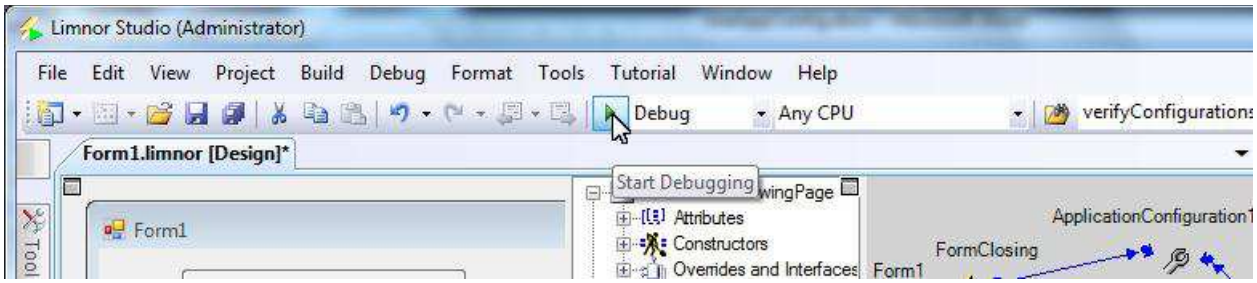


The action is created and assigned to the event:



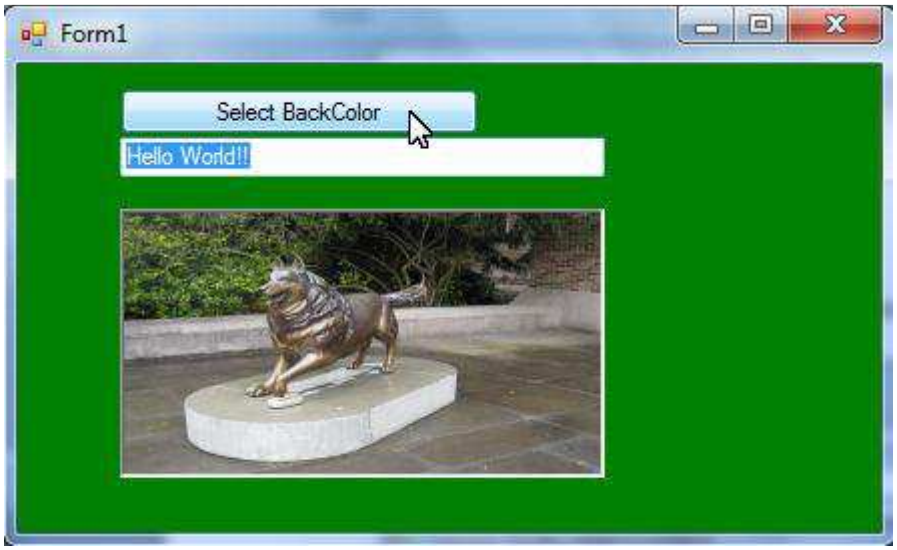
Test

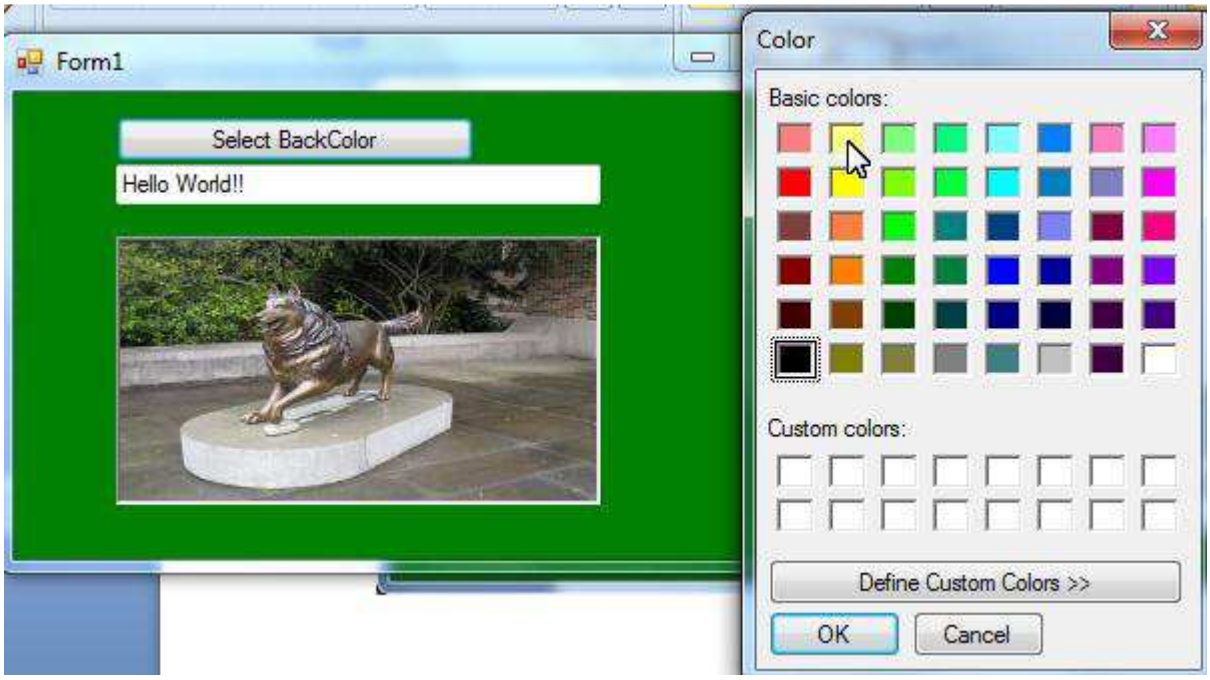
Click Run button to compile and run the application:



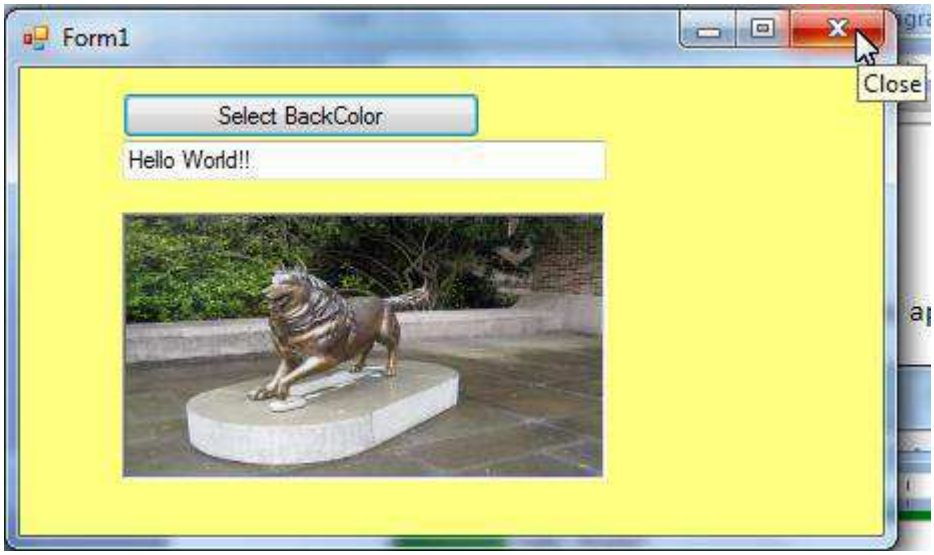
The form appears. Note that the background color of the form is green. This is the value set at design time to the configuration value “BackColor1”.

Let’s click the button to change background color:

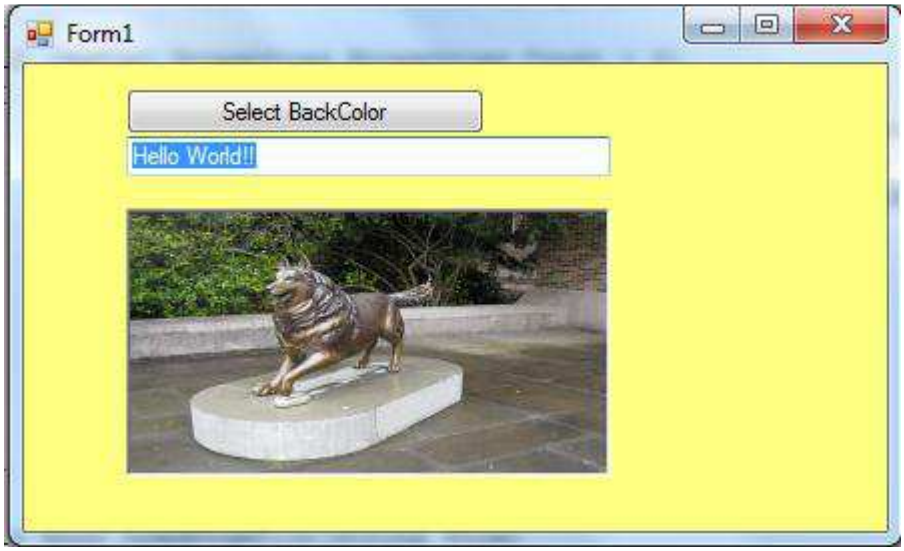




The background color is changed. Now close the form:



Restart the application by double-clicking the EXE file. The form appears using the same background color:

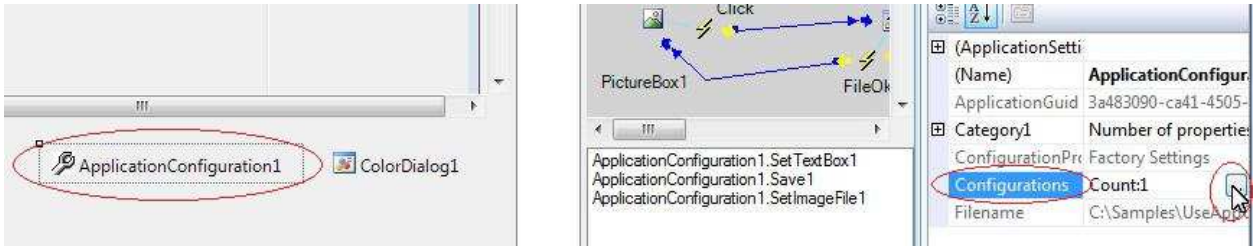


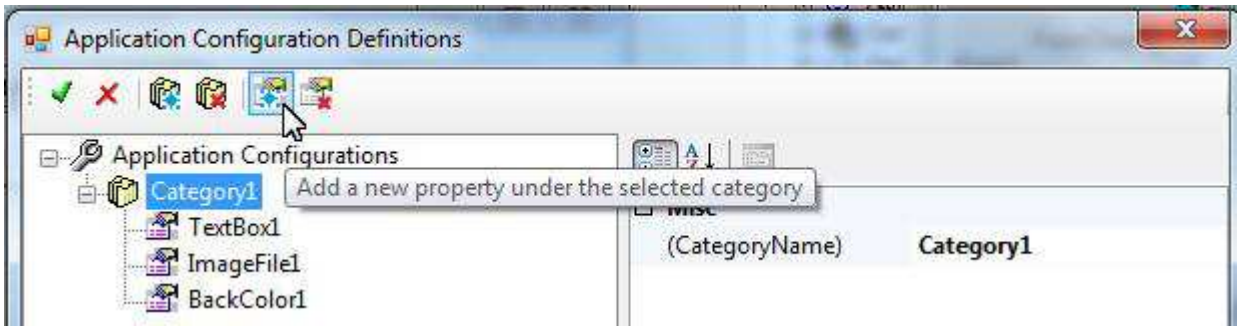
Configurations for window size and location

A form has a Bounds property which is a rectangle indicating its size and location. We may use a configuration value to restore the form to the last size and location.

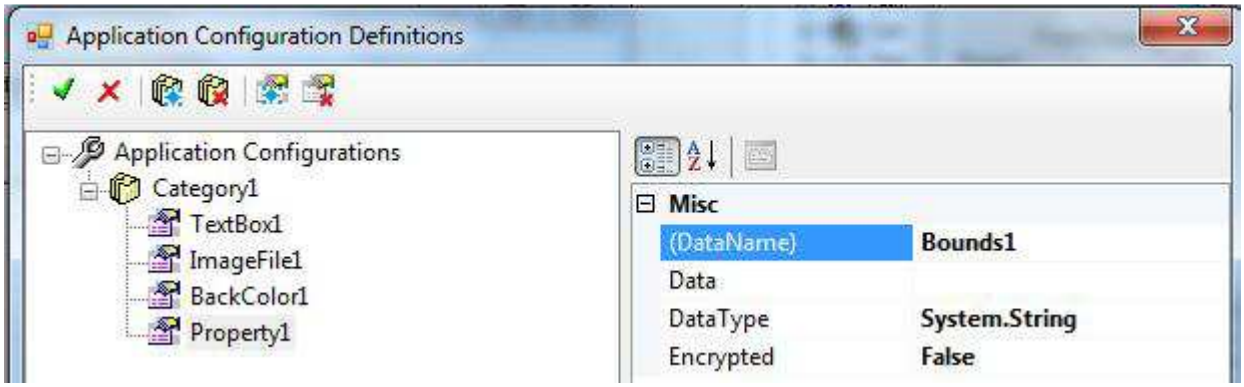
Create a new configuration value

For remembering the Bounds property of the form, we need to create a new configuration value for it:



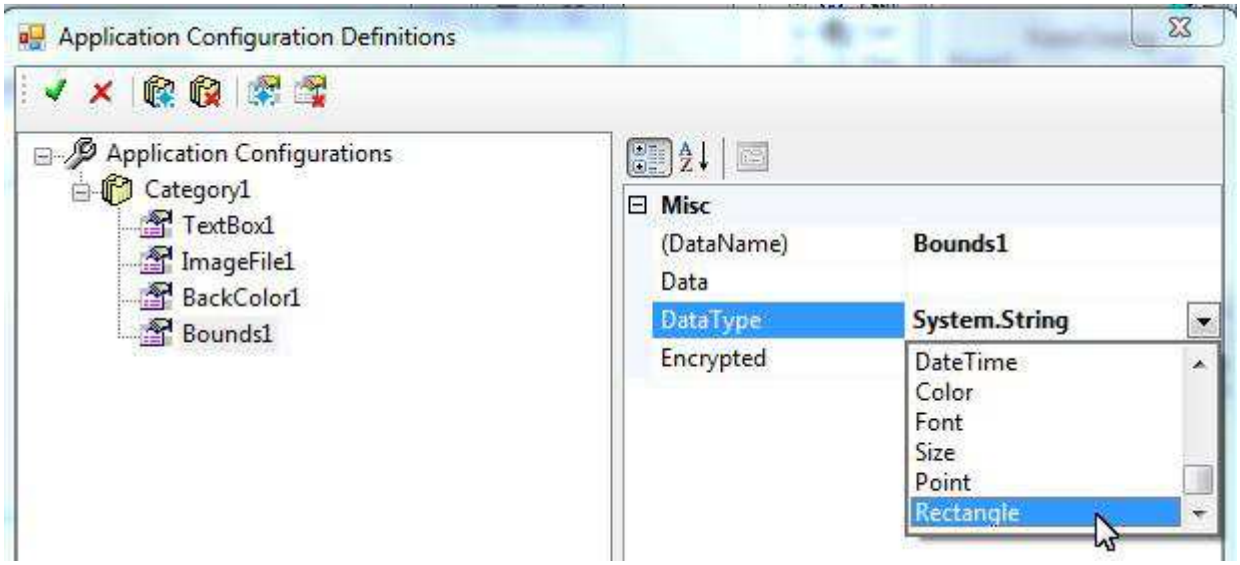


Name the new configuration value “Bounds1”:

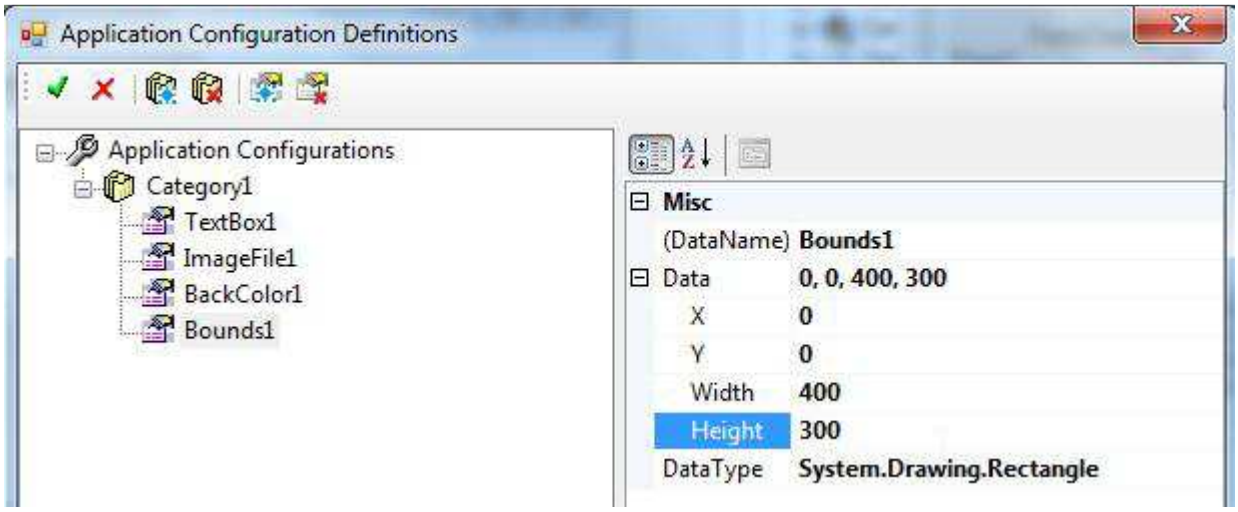


Set configuration value type

We want to make the new configuration a rectangle value:

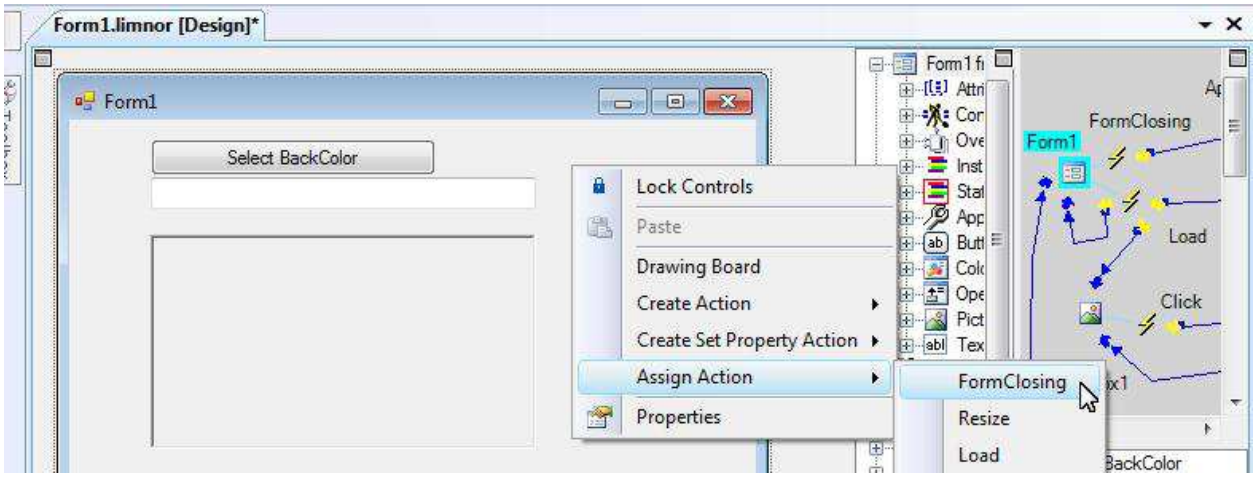


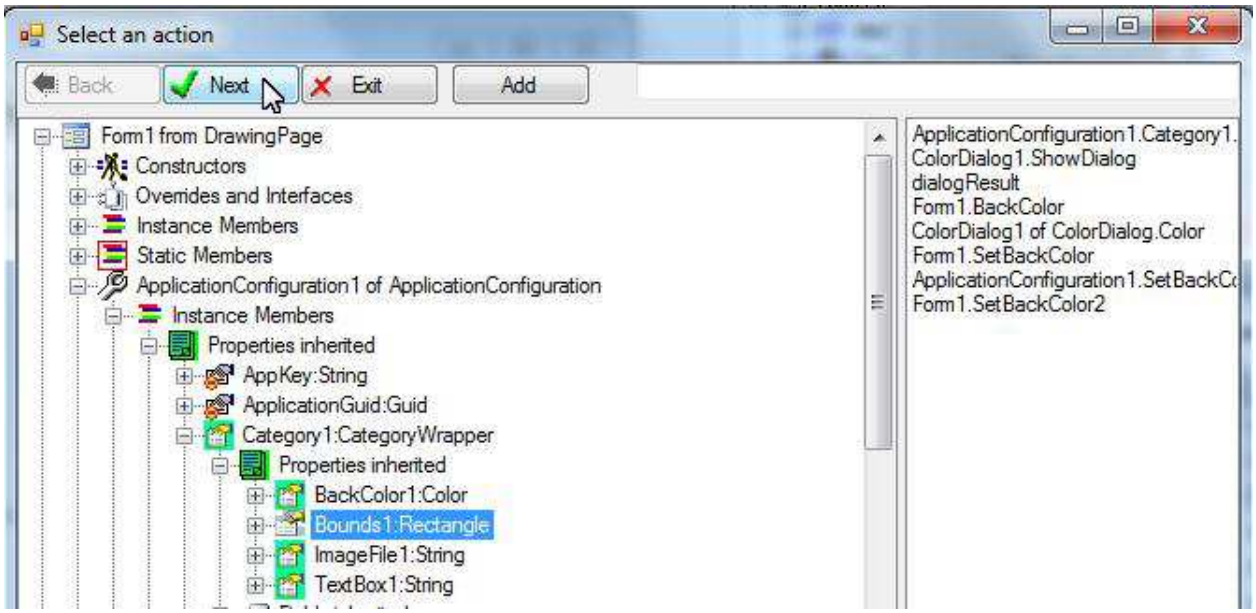
Set default configuration value



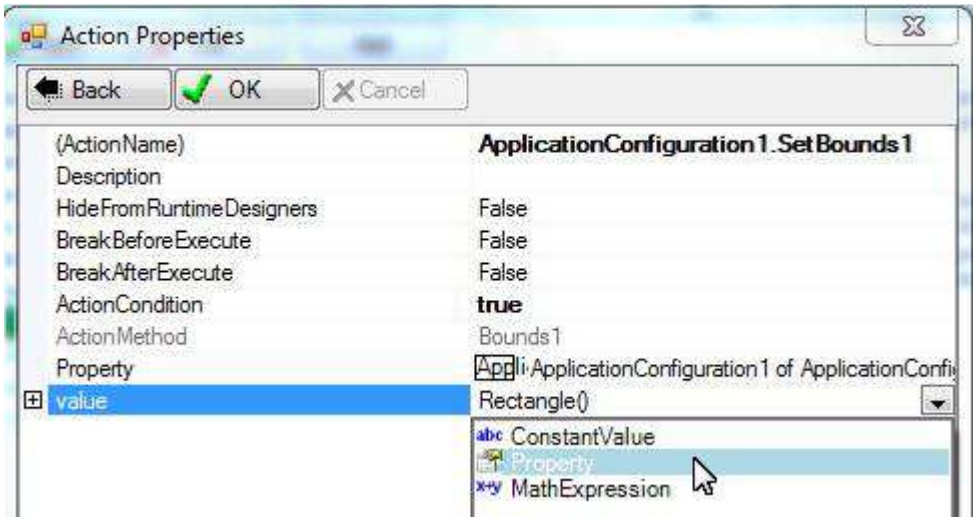
Remember form location and size

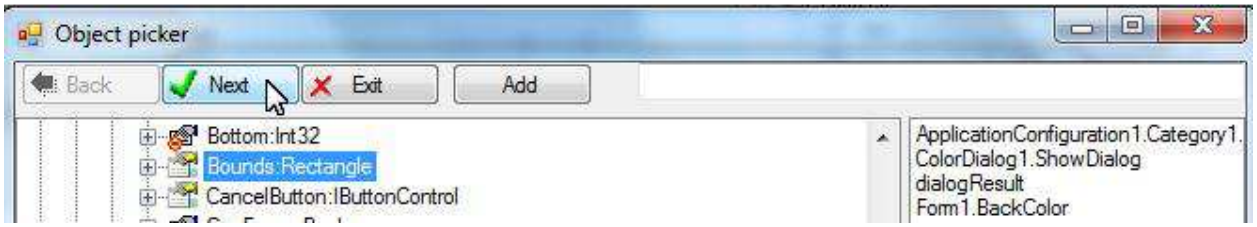
For this sample, we may remember the form location and size at the time of form closing:

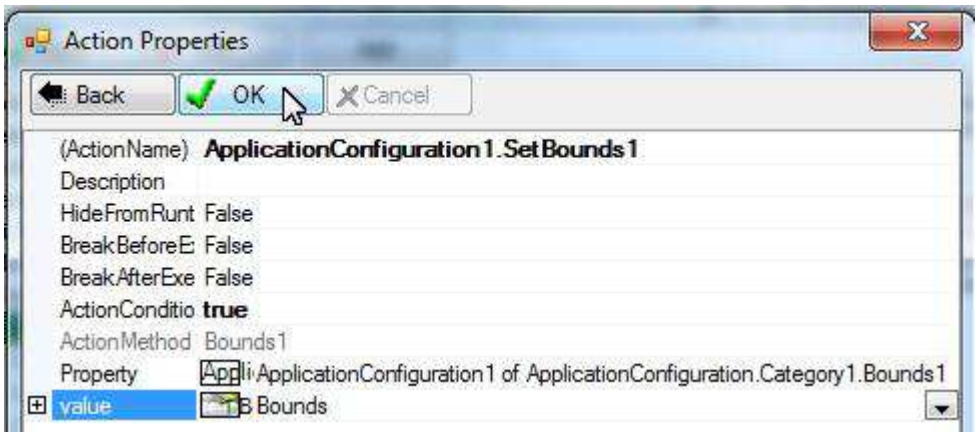




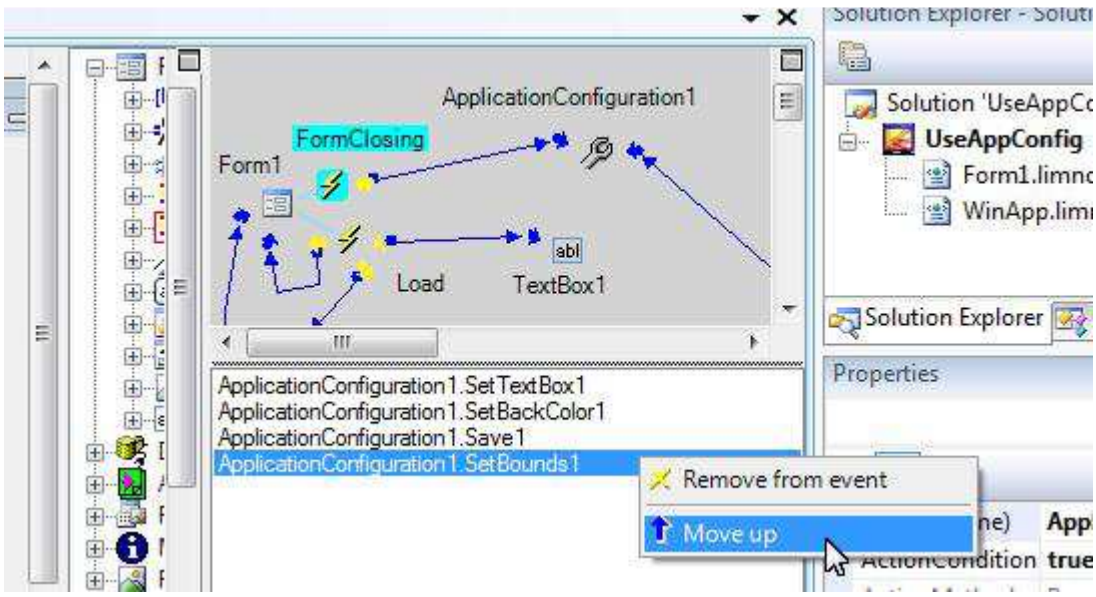
Pass the Bounds property of the form to the action:

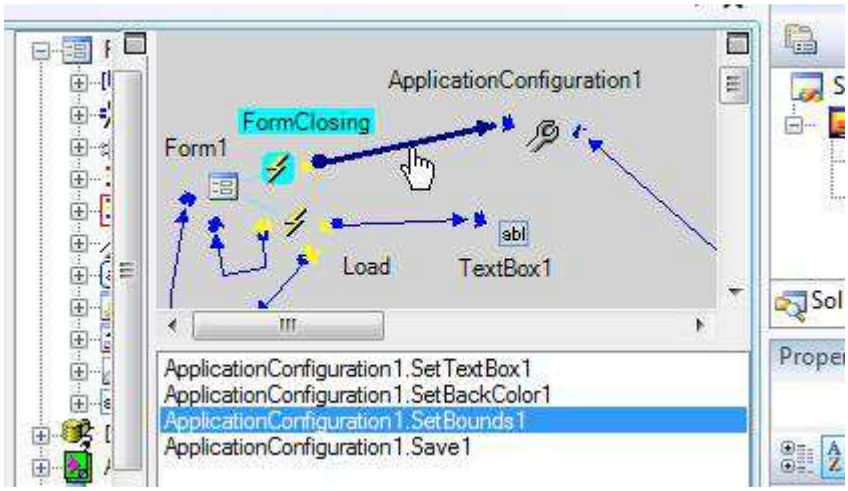






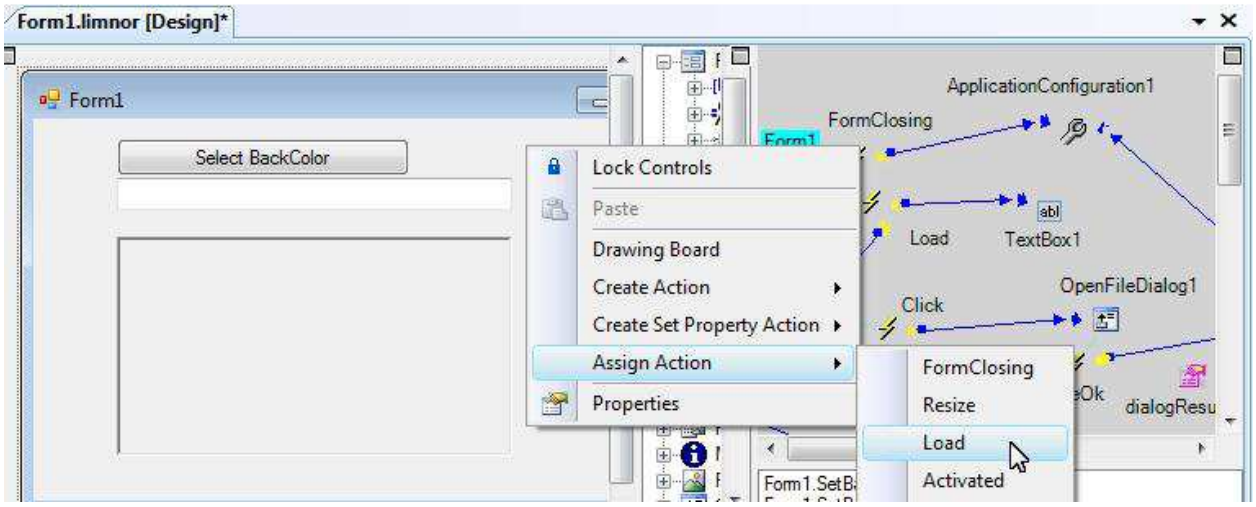
The action is created and assigned to the event. Move it up to before the Save action:

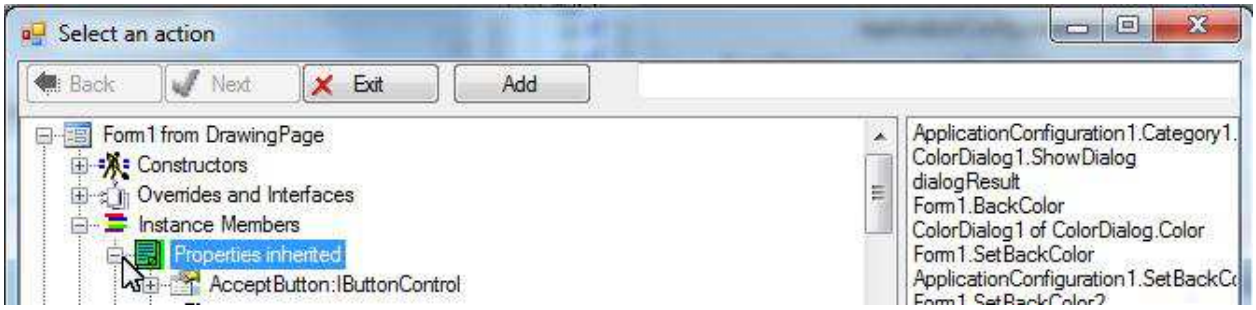


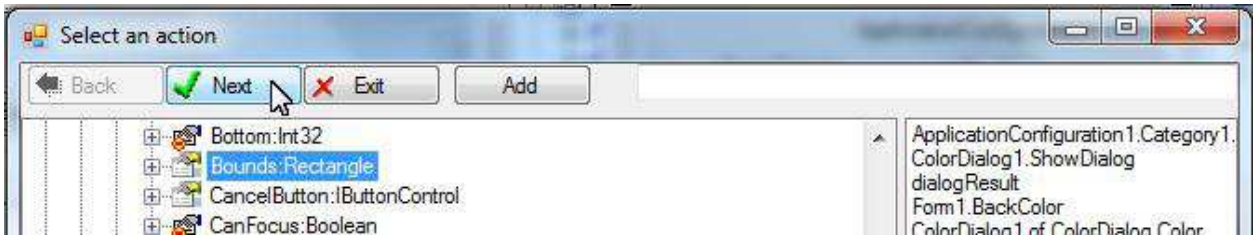


Apply configuration value

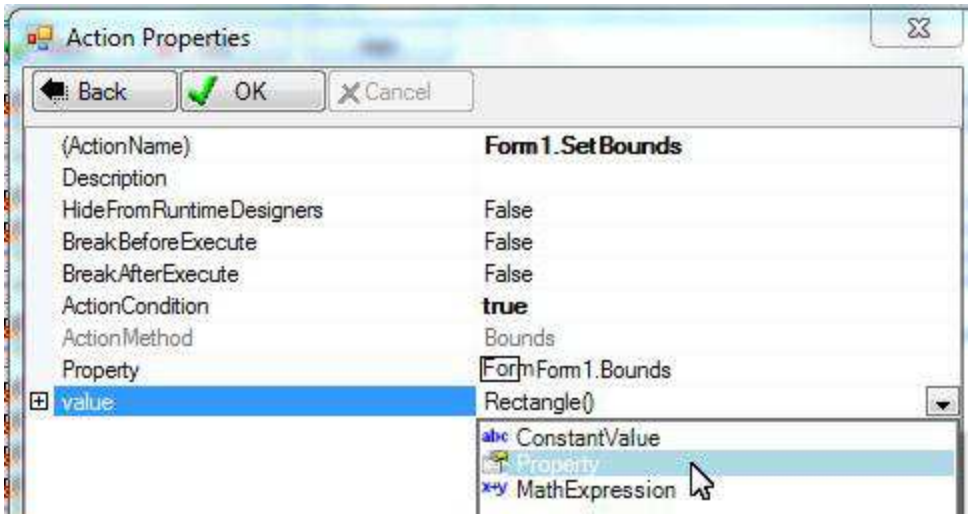
We may apply the configuration value “Bounds1” at the time of Load event of the form:

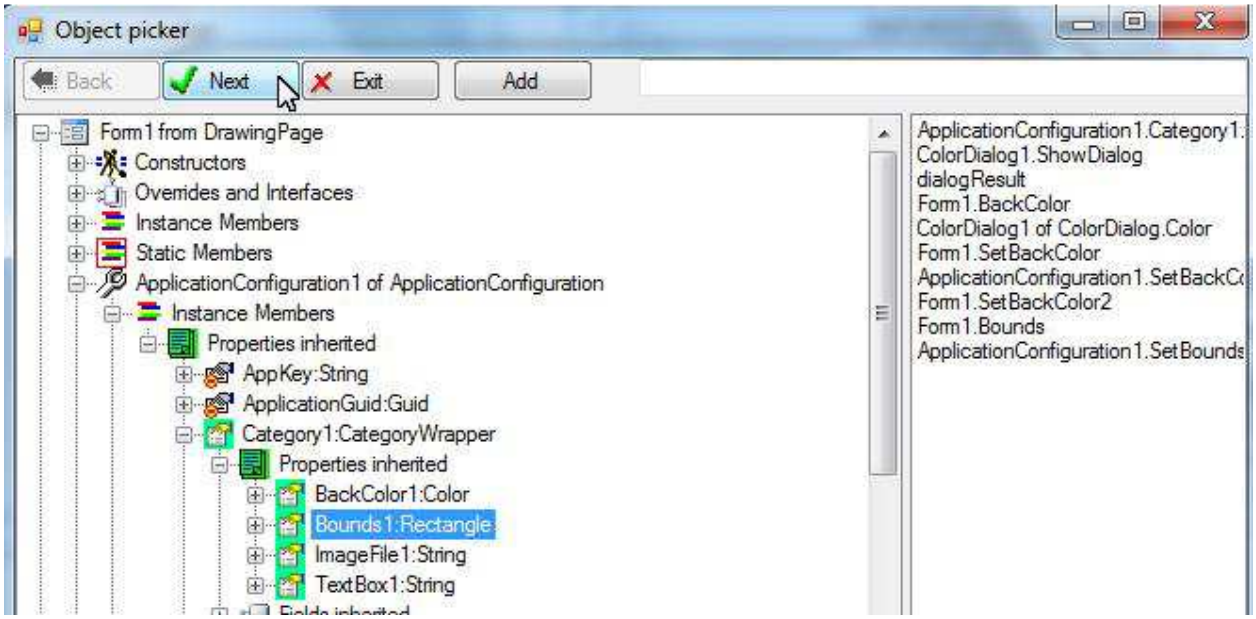


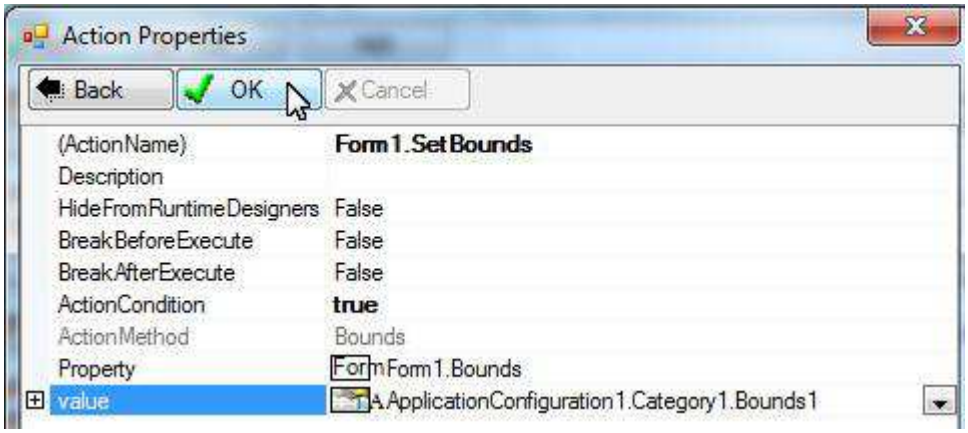




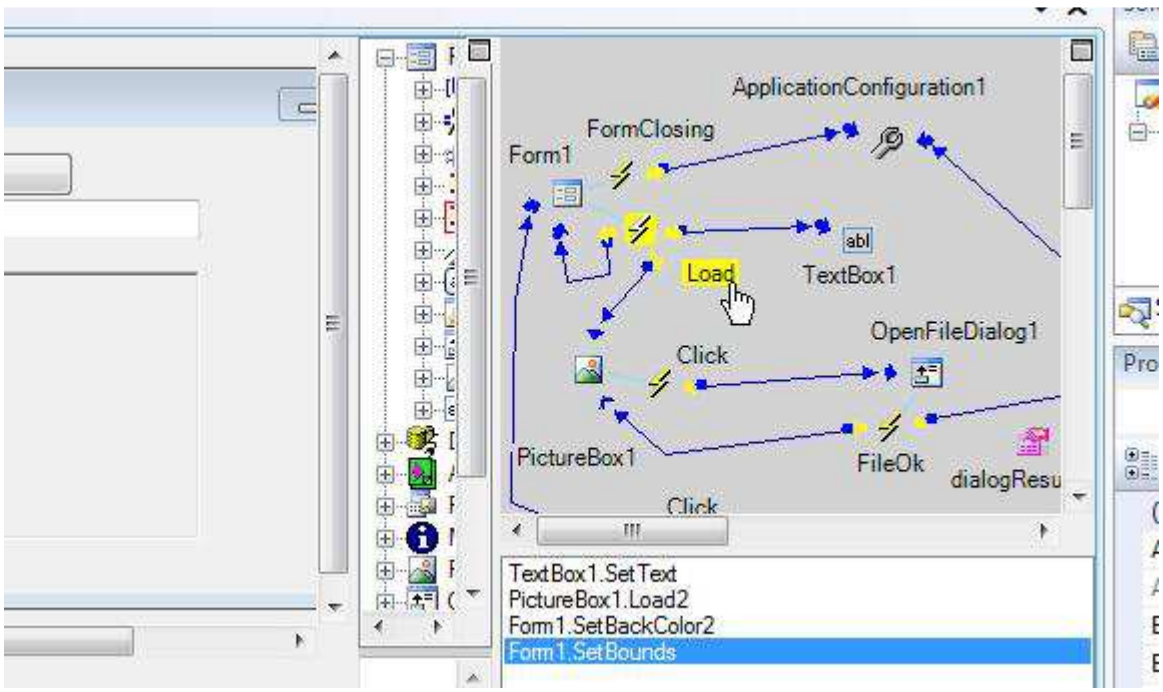
Pass the configuration value to the action:





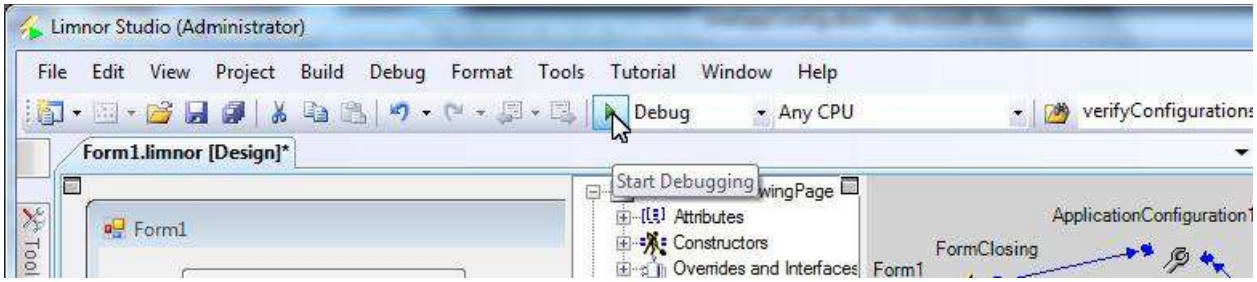


The action is created and assigned to the event:

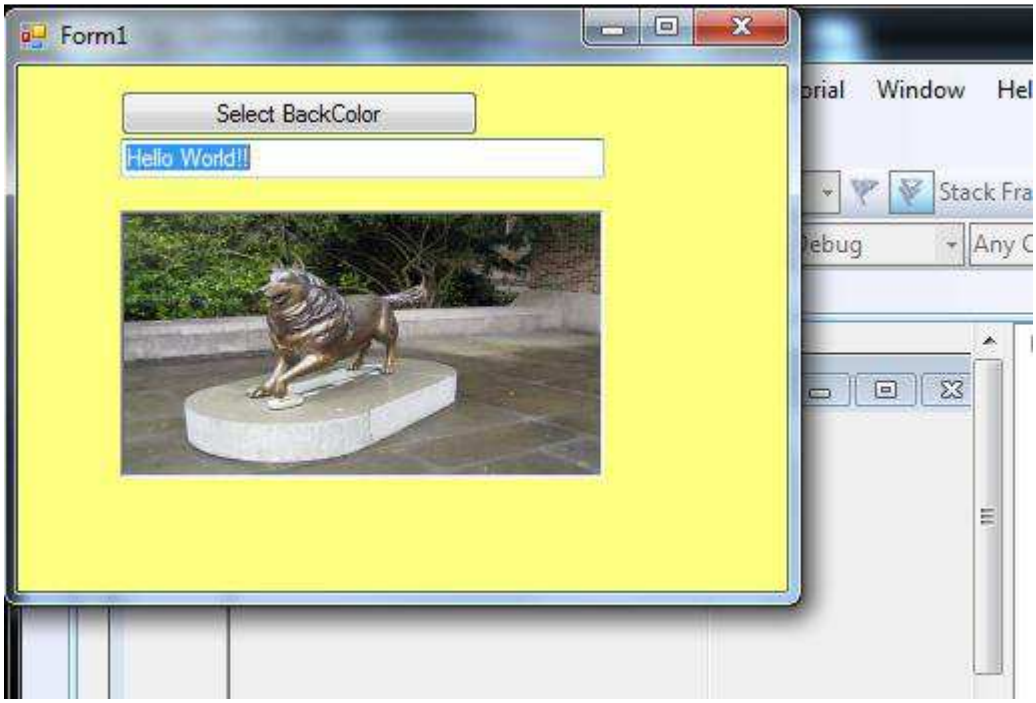


Test

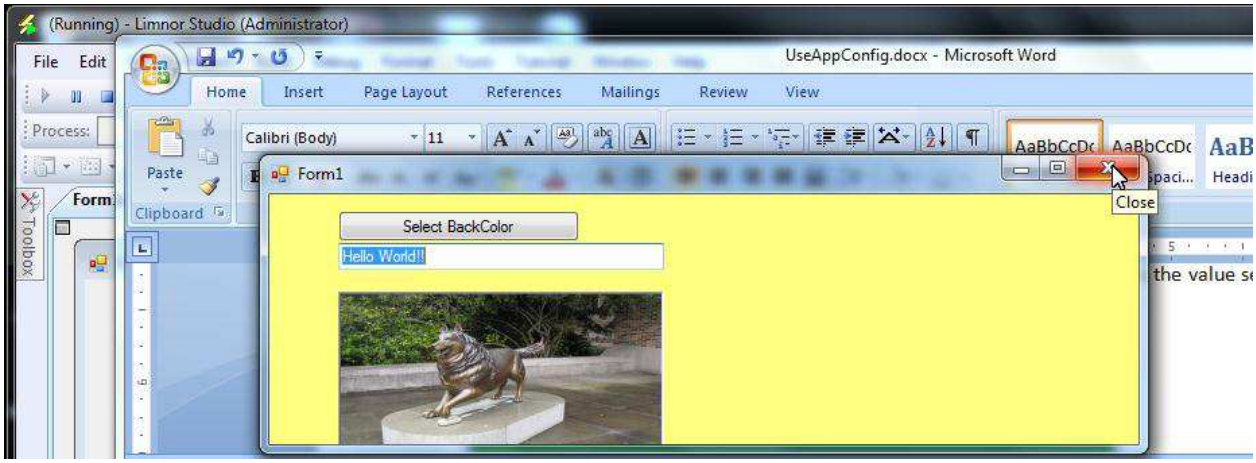
Click Run button to compile and run the application:



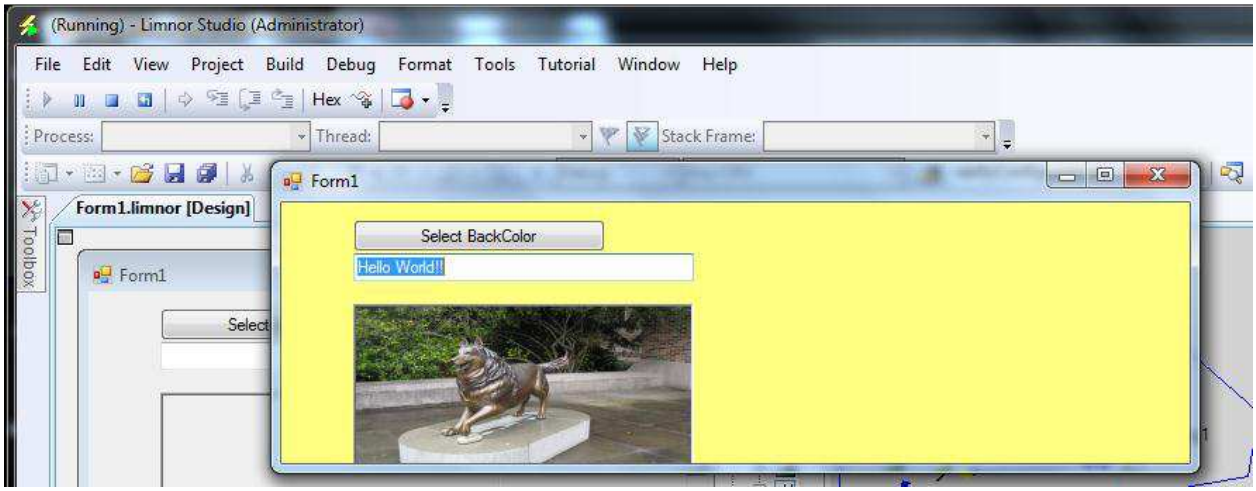
The form appears. It appears at the top left corner of the screen because we have set “Bounds1” to that location.



Now let's move the form and resize it. Then close it:



Restart the application by double-clicking the EXE file. We will see that the form appears at the location where it was closed last time. Its size also is the same as that when it was closed:



Use Image Resource as default Image Configuration

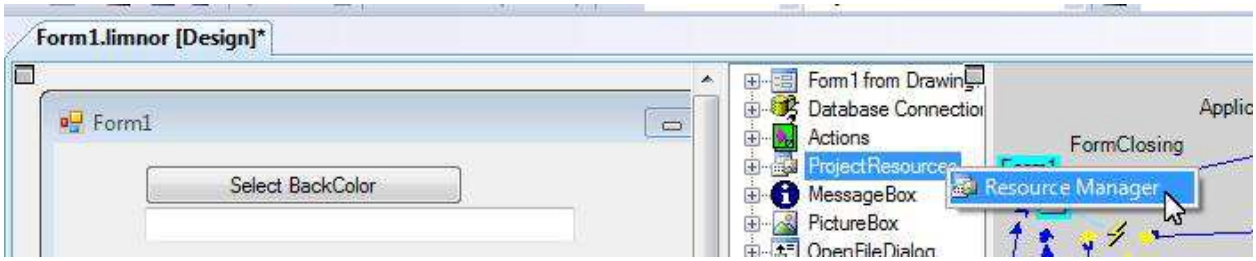
We have used image file configuration. Suppose we want to use a default image if the image file configuration is empty or not a valid file path. For more information on using resources, see

http://www.limnor.com/support/Limnor%20Studio%20-%20User%20Guide%20-%20Resource%20Manager.pdf

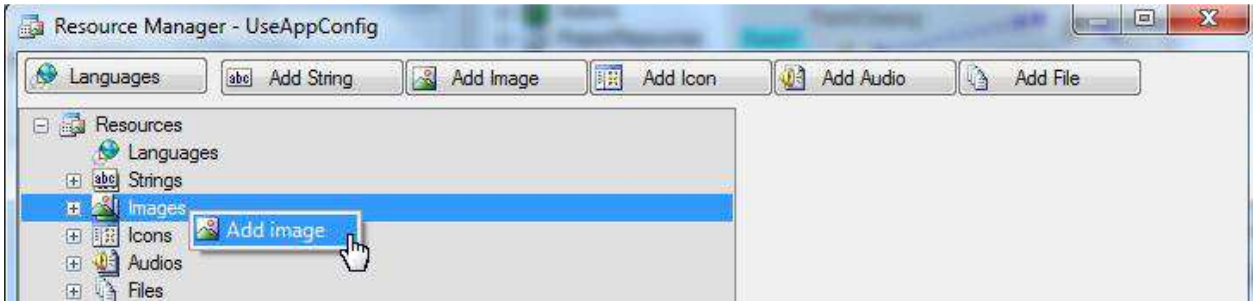
http://www.limnor.com/support/Limnor%20Studio%20-%20Non-Embedded%20Files.pdf

Create an image resource

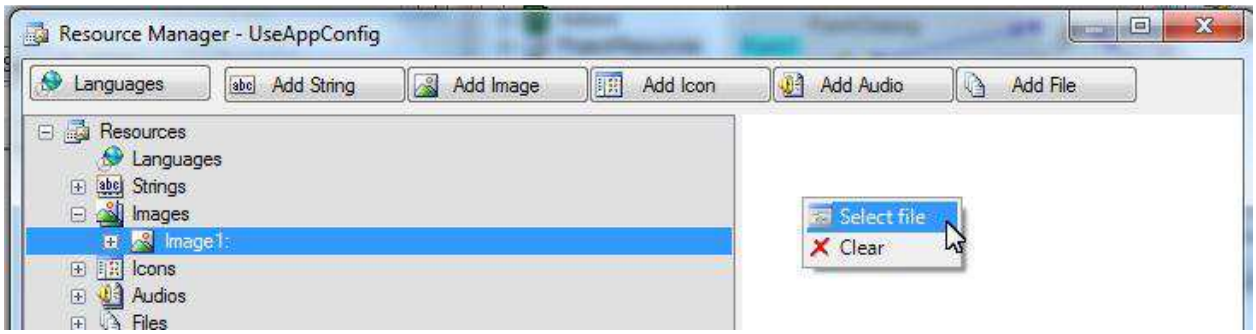
Launch a resource manager:



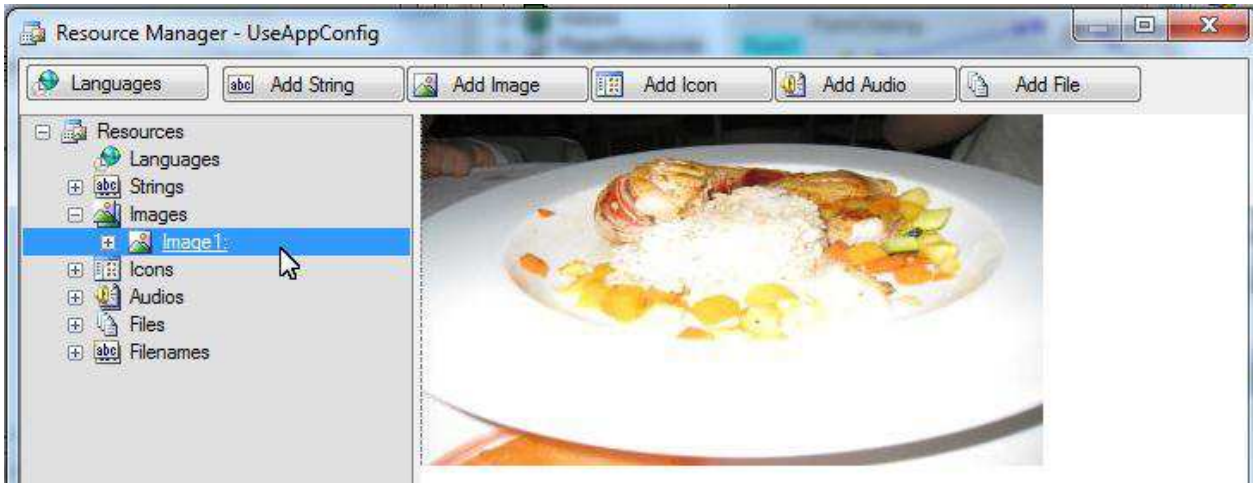
Create an image resource:



Select an image file:

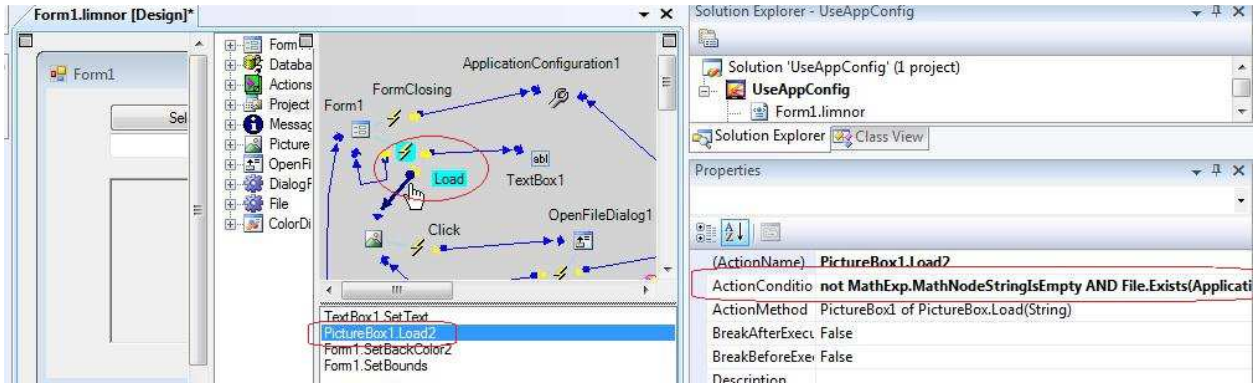


The image resource is created:



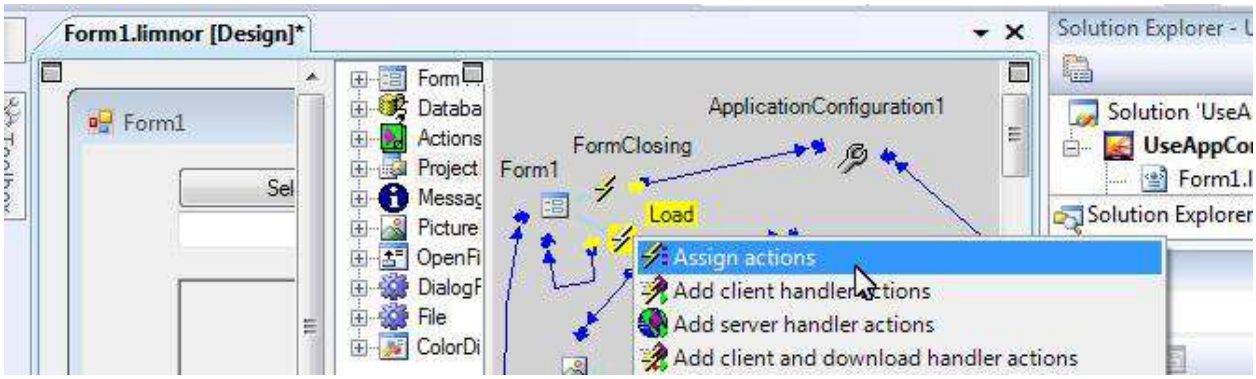
Use image resource

We already created an action to load image from configuration and its Action Condition is set to that the configuration value is not empty and is a valid file name. The action is assigned to the Load event:



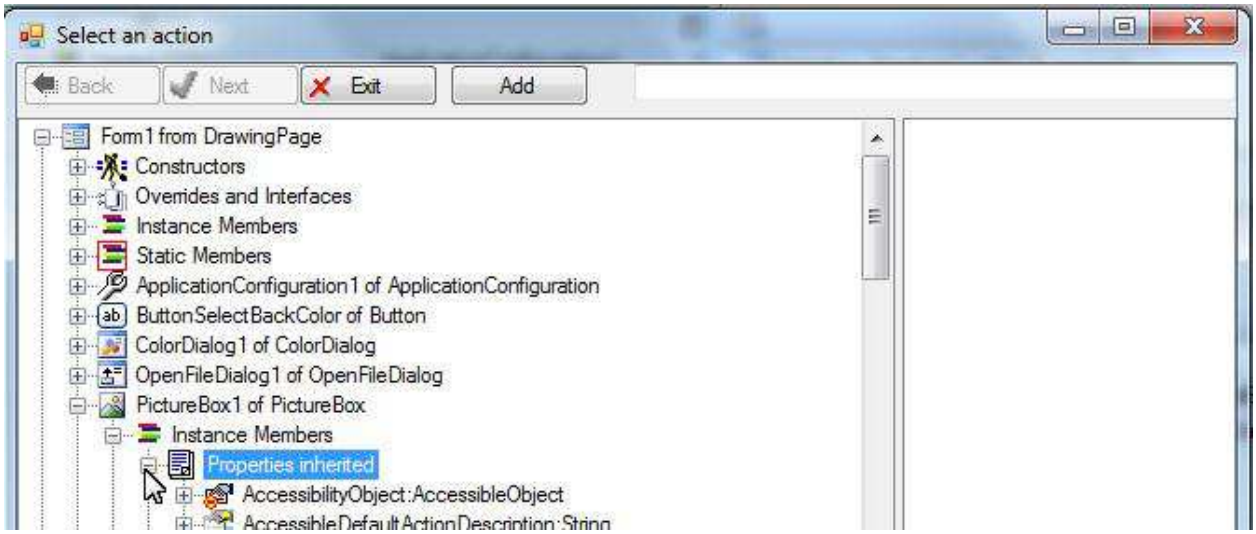
We may create another action to load image from the resource and assign the action to the Load event. The Action Condition for this new action will be opposite to the condition used for loading image from configuration.

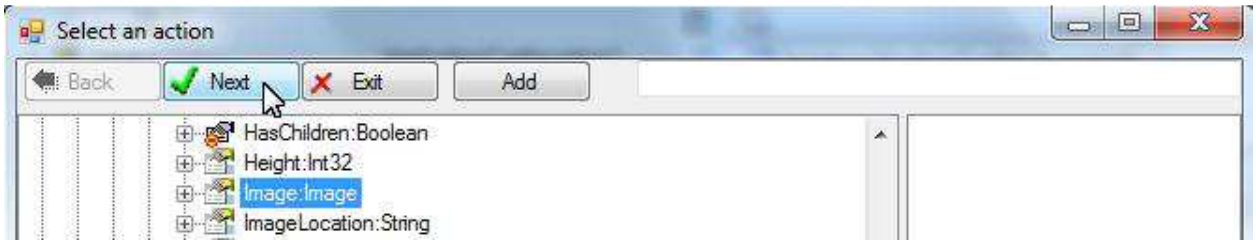
Assign a new action to Load event



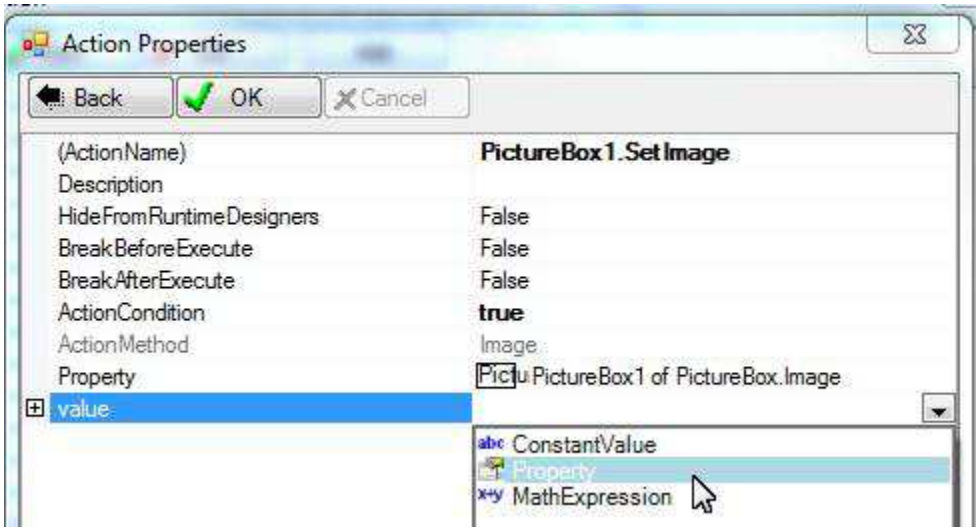
Create set image action

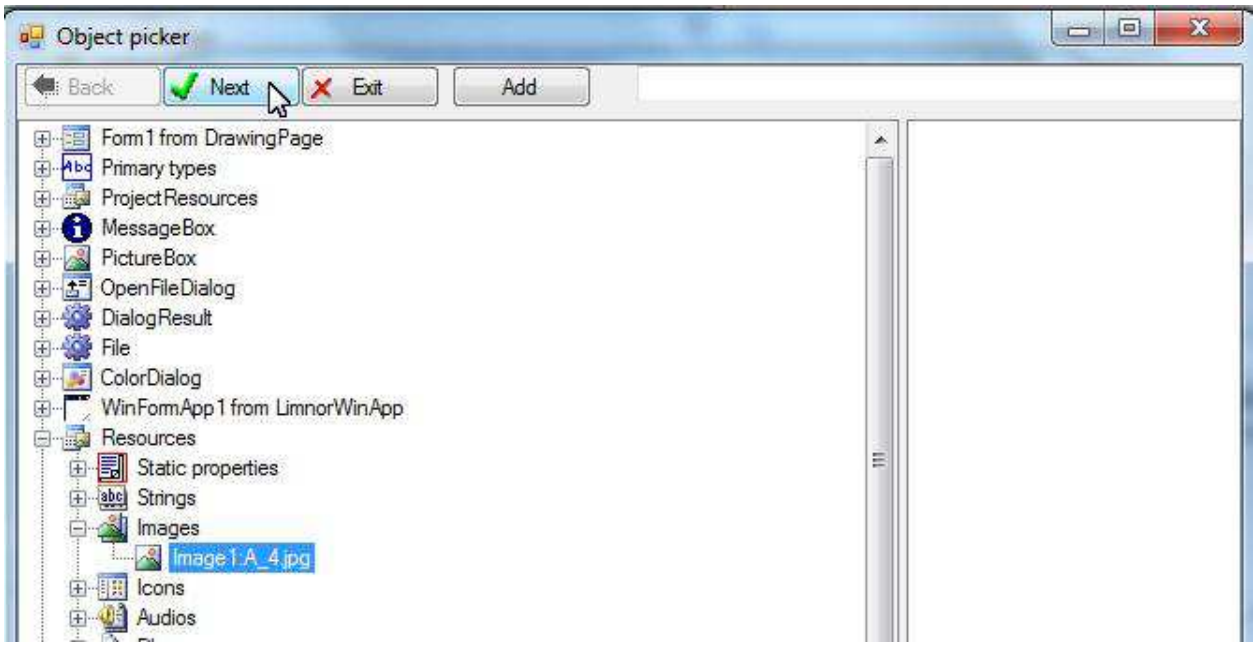
Select Image property of the picture box:



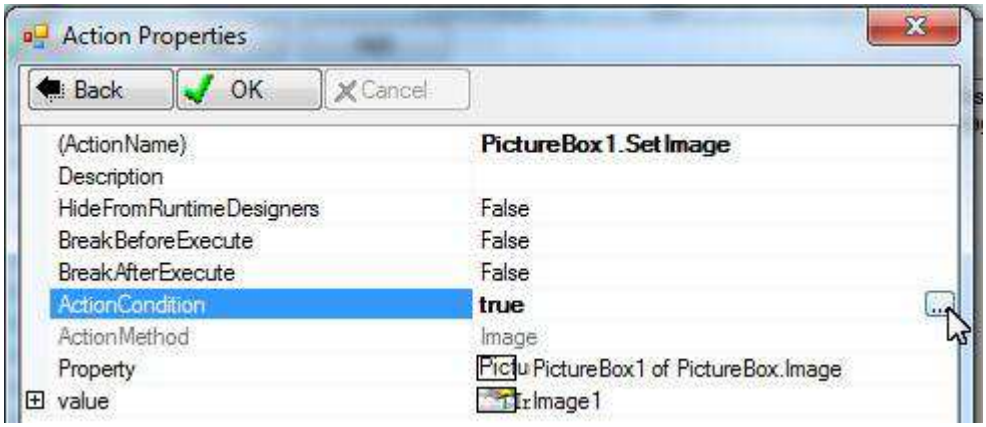


Select image resource

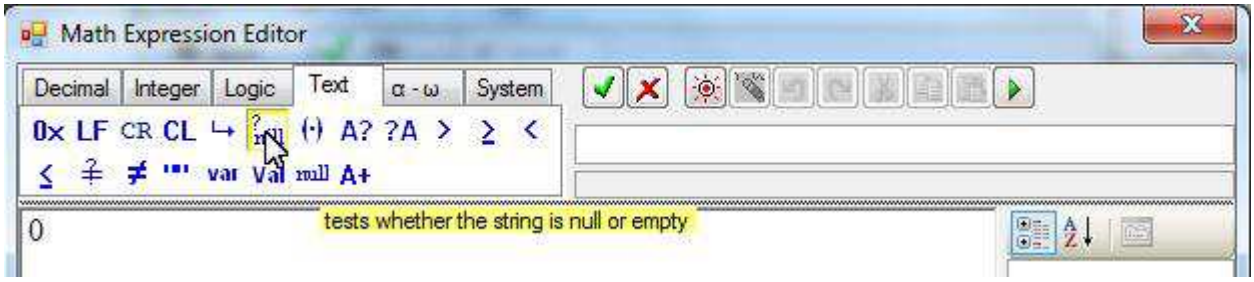


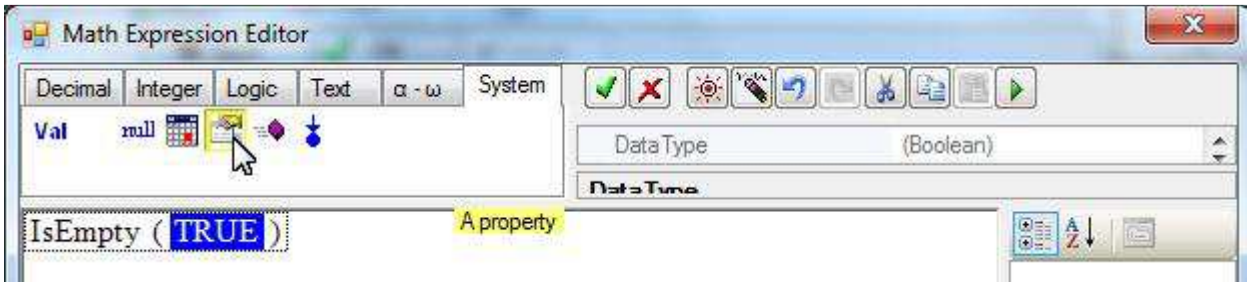


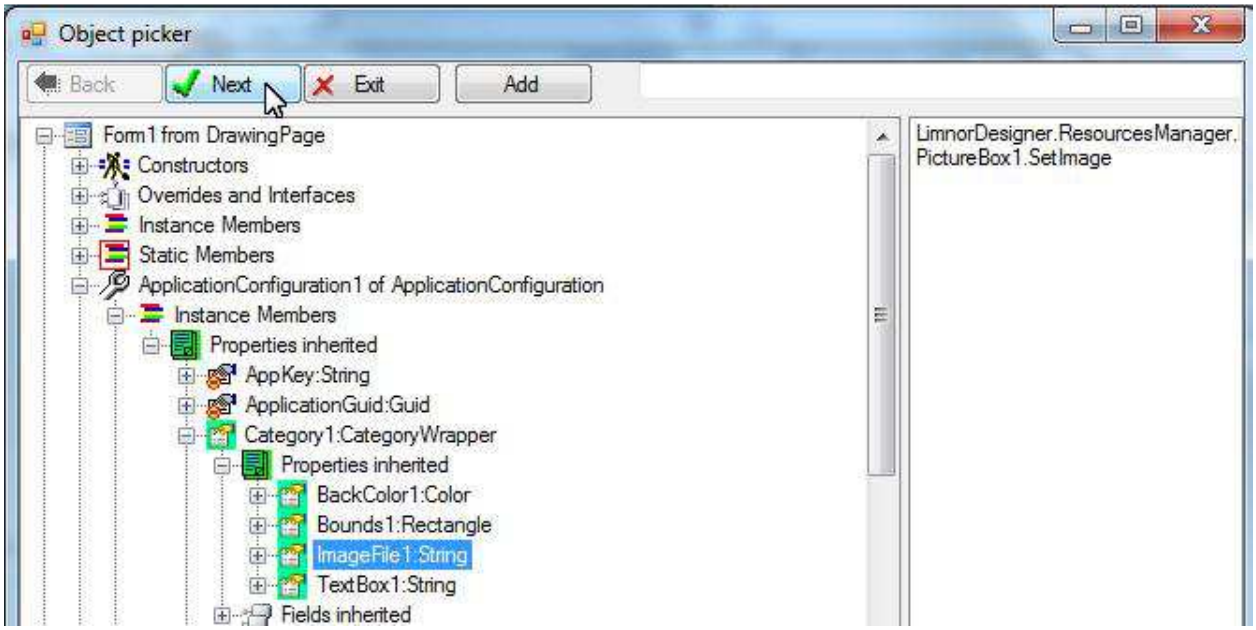
Set action condition



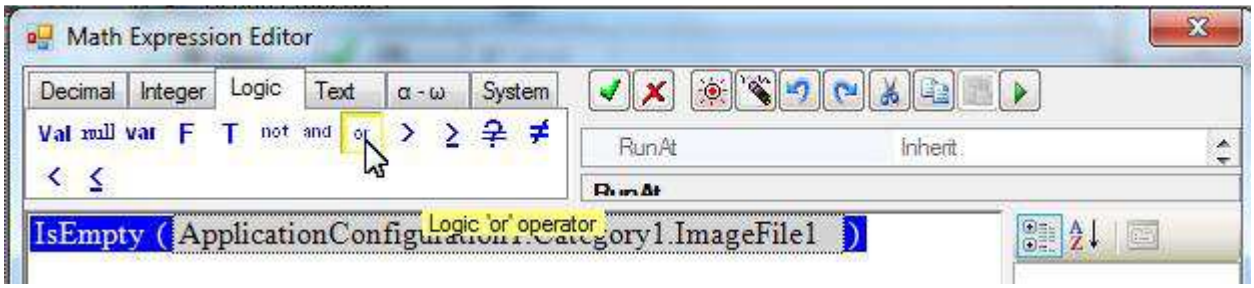
Check emptiness of the configuration value:

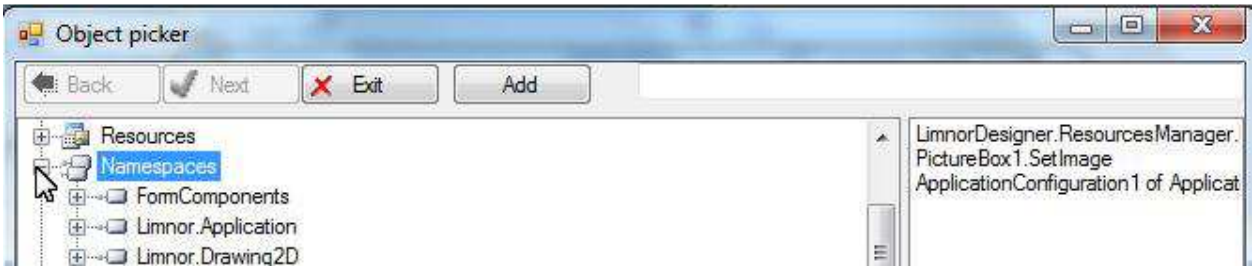


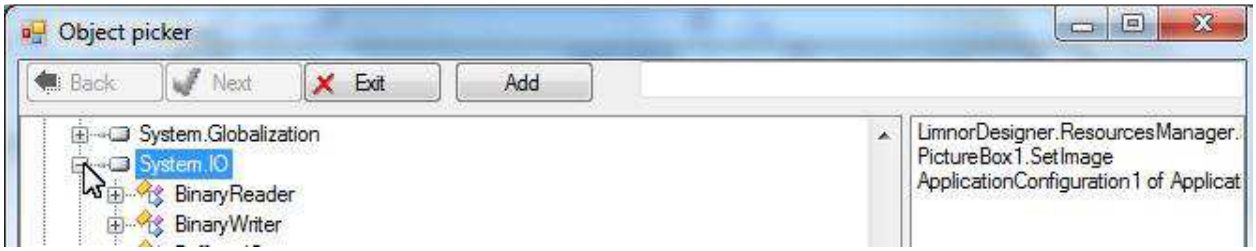


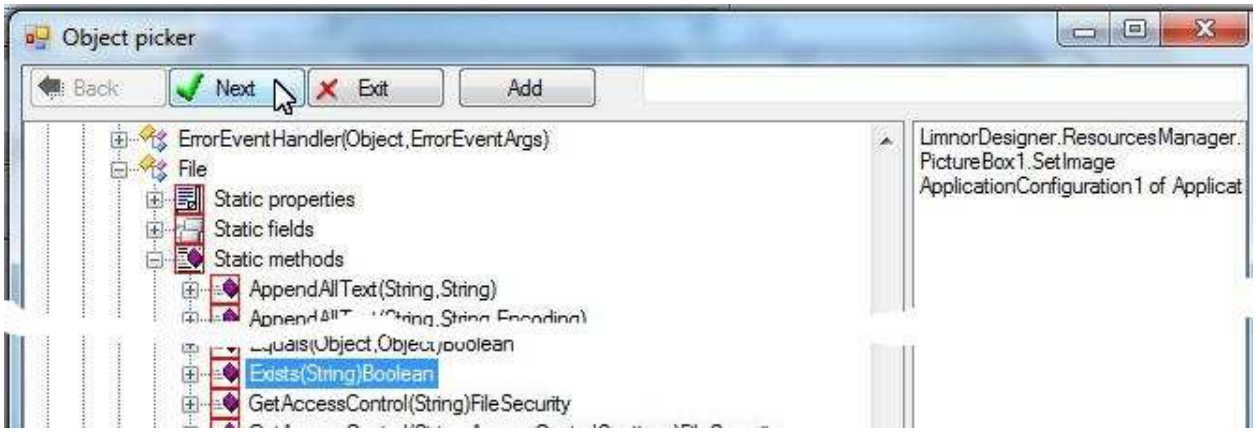


Further check file-existence:

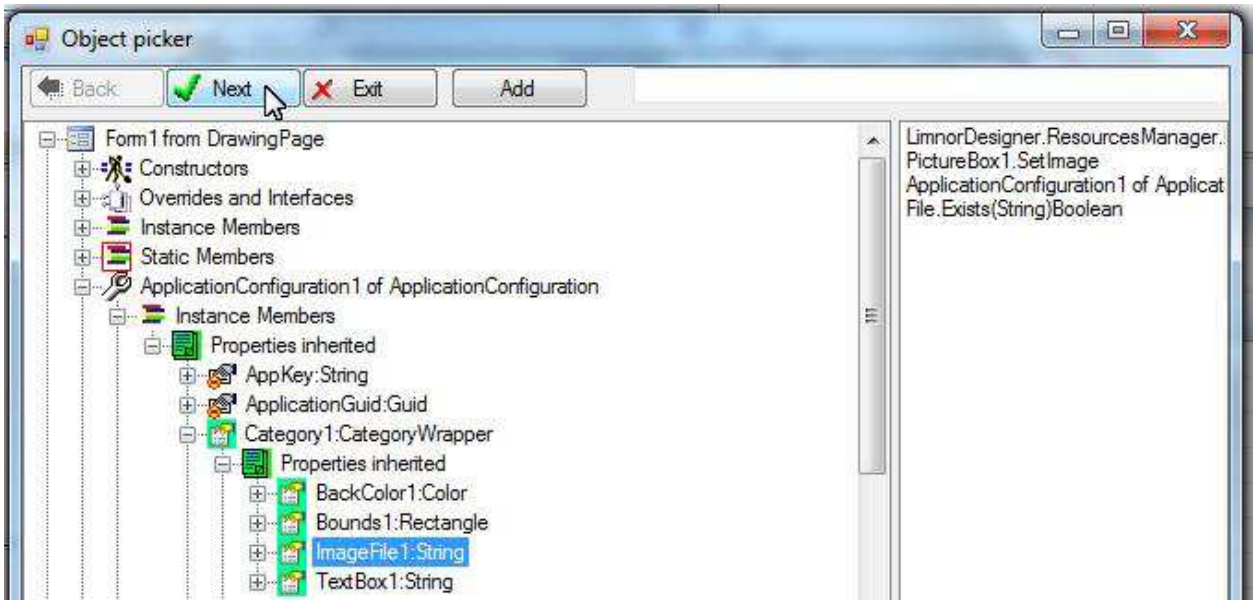




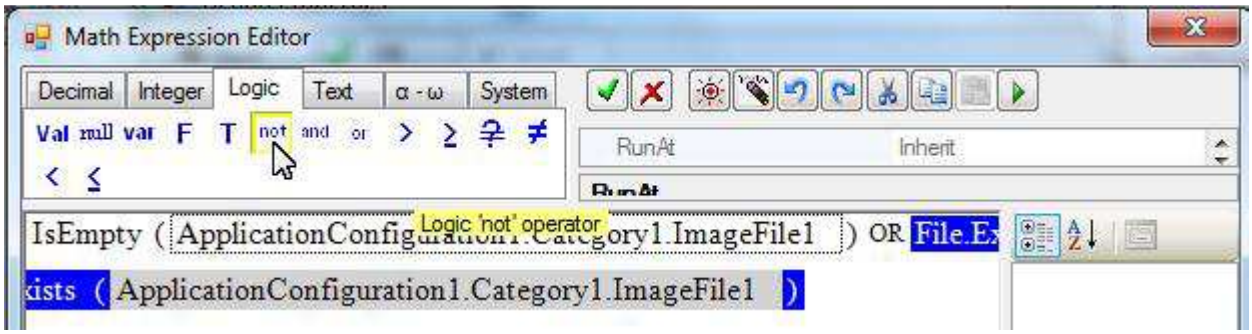

Object picker
Back
Next
Exit
Add
System.Globalization
System.IO
BinaryReader
BinaryWriter
LimnorDesigner.ResourcesManager.
PictureBox1.SetImage
ApplicationConfiguration1 of Applicat


Object picker
Back
Next
Exit
Add
ErrorEventHandler(Object,ErrorEventArgs)
File
Static properties
Static fields
Static methods
AppendAllText(String,String)
Exists(String)Boolean
GetAccessControl(String)FileSecurity
LimnorDesigner.ResourcesManager.
PictureBox1.SetImage
ApplicationConfiguration1 of Applicat

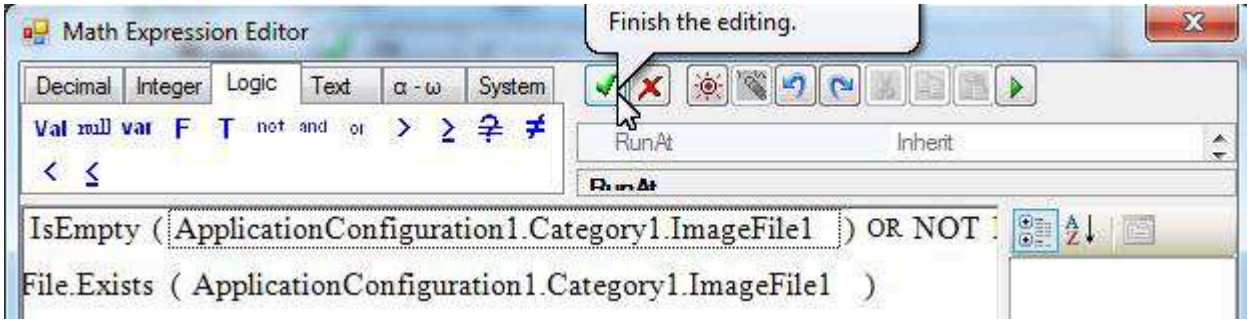

Math Expression Editor
Decimal
Integer
Logic
Text
α - ω
System
Val
null
RunAt
Inherit
IsEmpty (
A property
) OR File
Exists (
)


Object picker
Back
Next
Exit
Add
Form1 from DrawingPage
Constructors
Overrides and Interfaces
Instance Members
Static Members
ApplicationConfiguration1 of ApplicationConfiguration
Instance Members
Properties inherited
AppKey:String
ApplicationGuid:Guid
Category1:CategoryWrapper
Properties inherited
BackColor1:Color
Bounds1:Rectangle
ImageFile1:String
TextBox1:String
LimnorDesigner.ResourcesManager.
PictureBox1.SetImage
ApplicationConfiguration1 of Applicat
File.Exists(String)Boolean

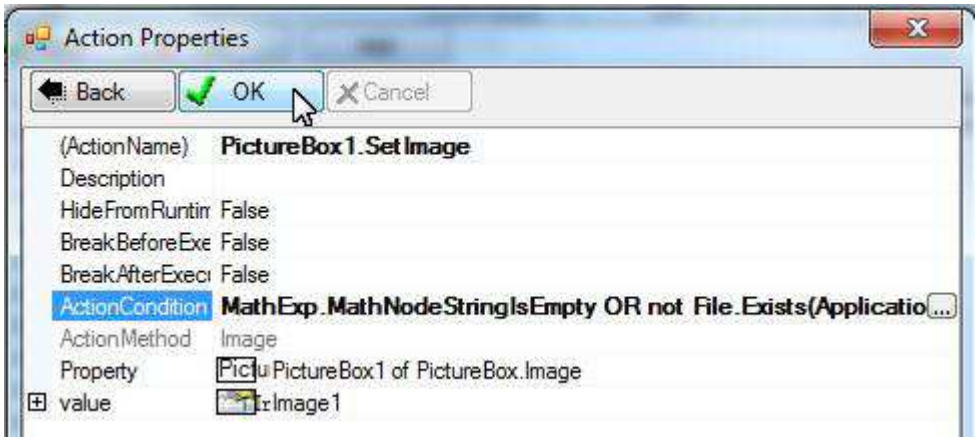
Apply “Not” operator to File Exist checking:



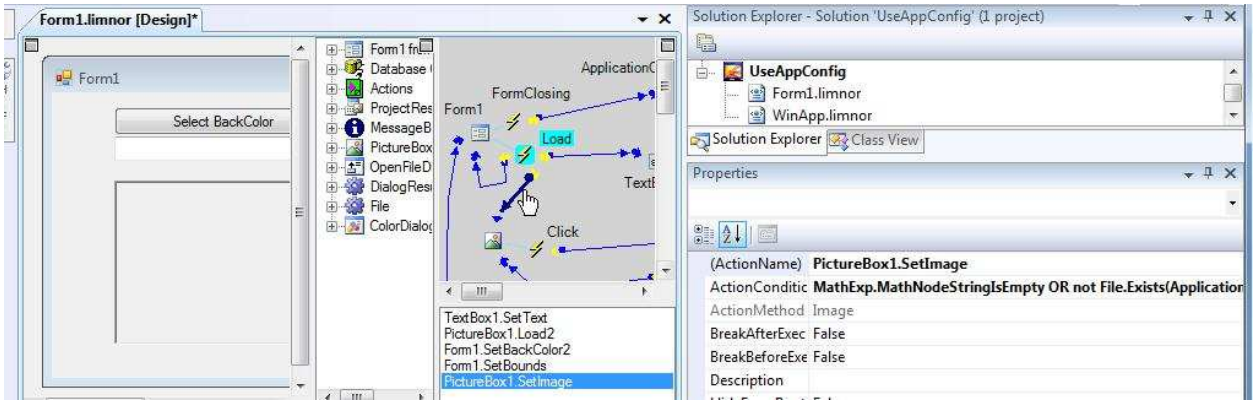
This is the action condition for using image resource:



Click OK.

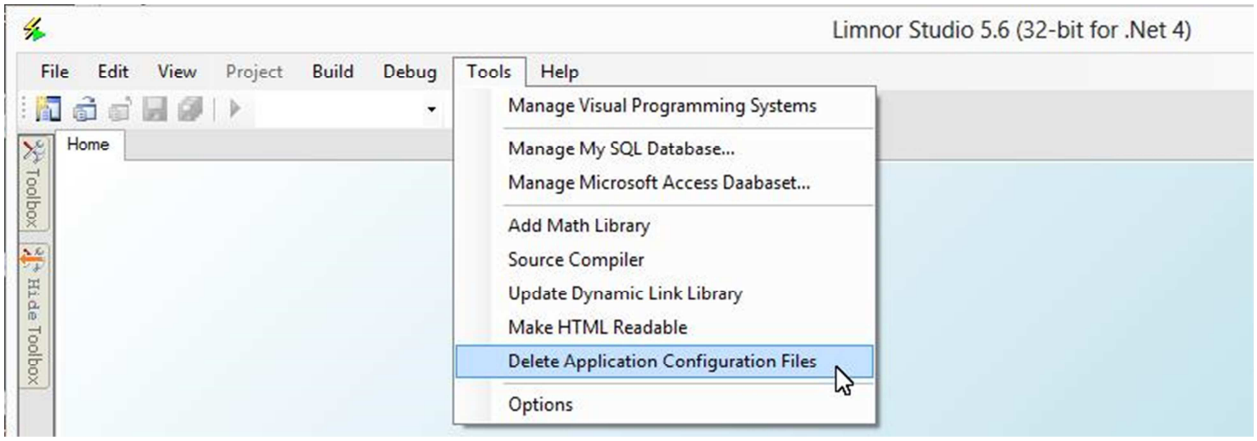


The action is created and assigned to the Load event:

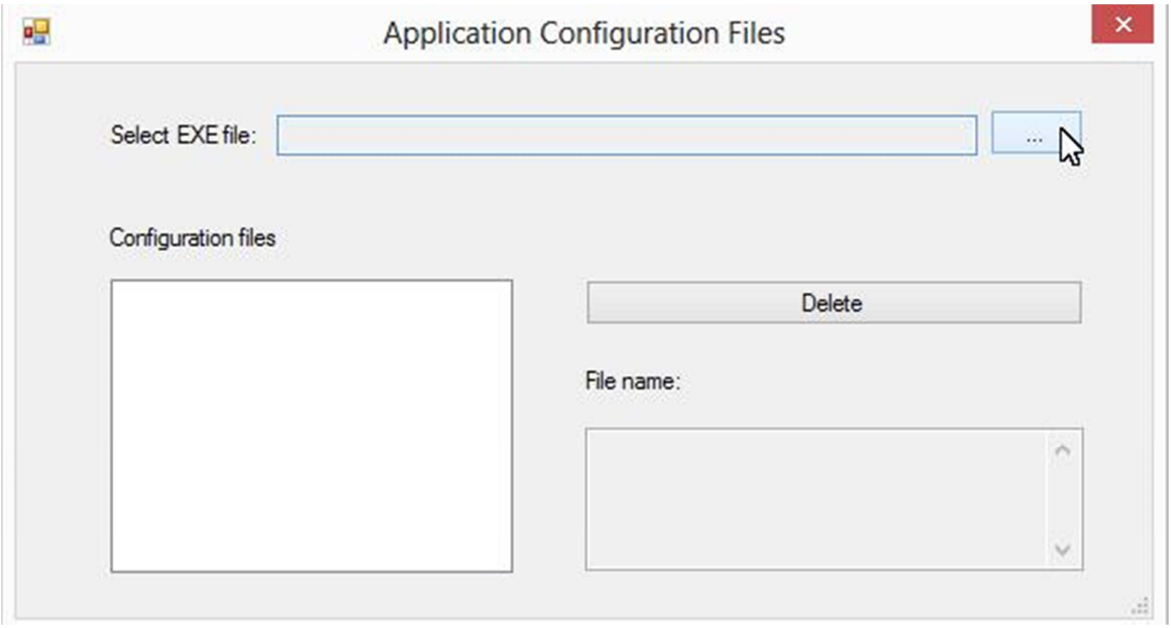


Reset configuration for test

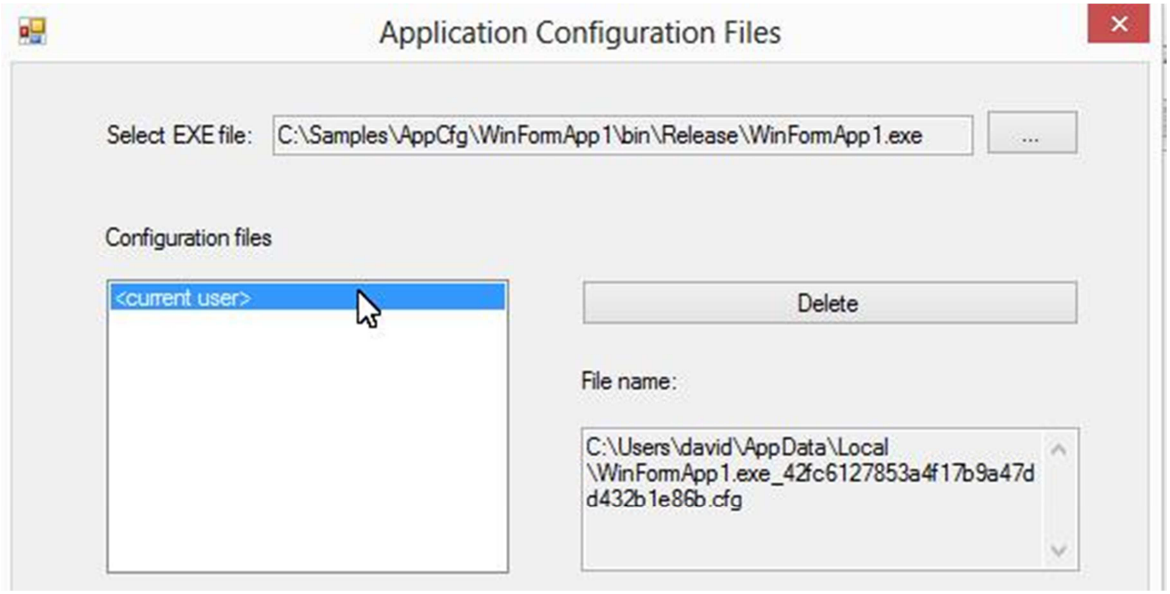
The configuration files already remember our testing data. We may delete those configuration files for this testing purpose. Limnor Studio provides a menu under Tools for this purpose:



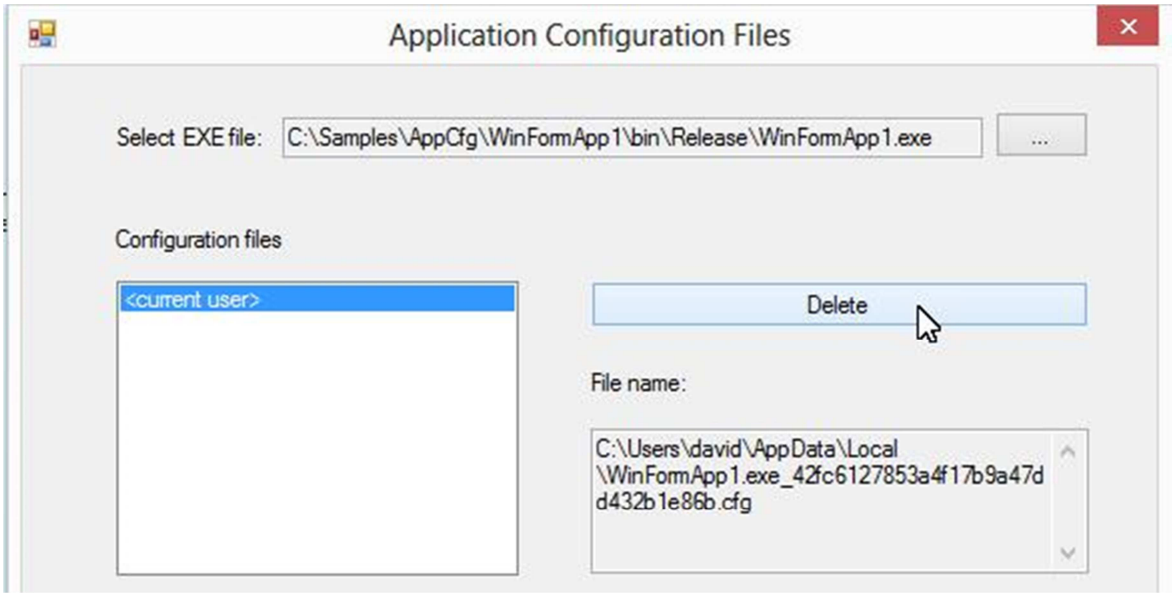
Click "…" to select the EXE file to remove its configurations:

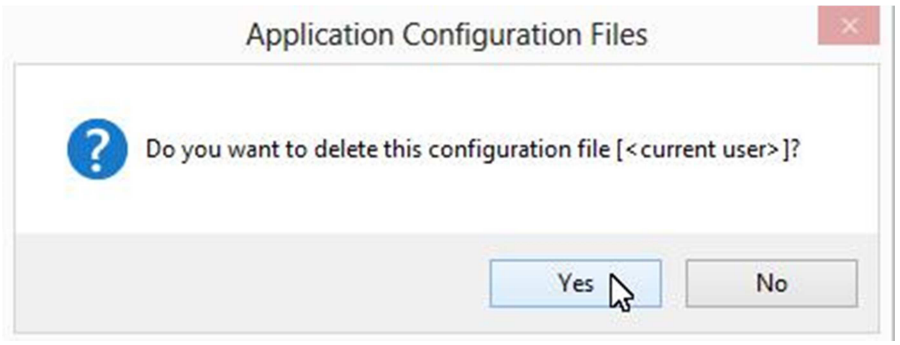


Once an EXE file is selected, all its configuration files will be listed:

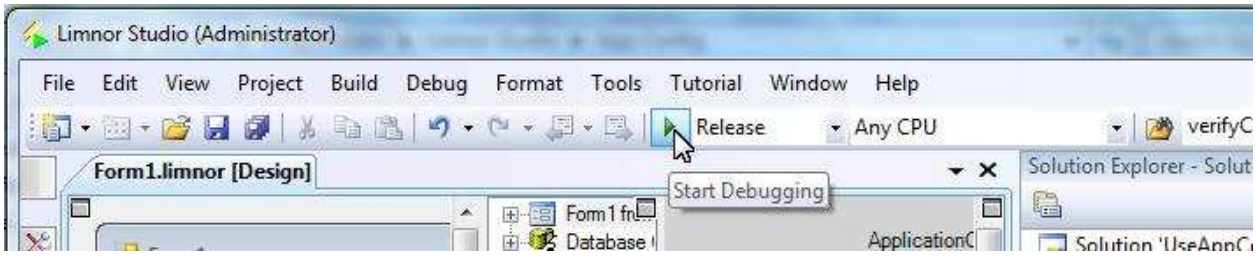


In the above example, there is just one configuration file for the current Windows logged on user. Click "Delete" button to delete a configuration file:

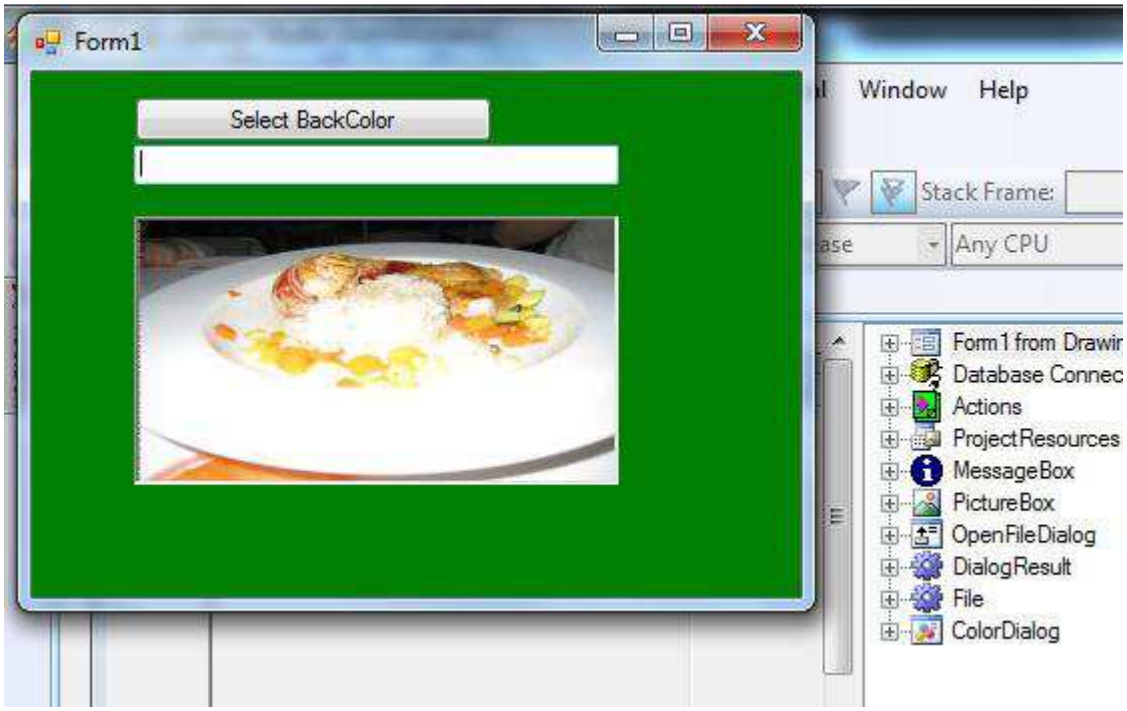




For this sample, we need to select UseAppConfig.exe and delete all its configuration files. Then compile and start the application:



The form starts with all the factory settings: at the top-left corner of the screen; background color is green. But the image box is not empty; it is using the image resource named Image1.



What Next

We can see that using application configurations we may save and restore user preferences. The process is simple and straight forward.

ApplicationConfiguration component also allows you to do more complex configurations.

- Named-profile – All configurations can be saved to and loaded from different sets of configurations. Each configuration set is identified by a unique name. Each set of configurations may be password-protected.
- User-profile – Each computer user may have his/her set of configurations. Depending on current computer user, different preferences are loaded and saved.
- Text configuration values can be encrypted so that such values can only be used by the application. Opening configuration files by other programs, such as a Notepad, will not reveal the real values of those configurations.

Read http://www.limnor.com/support/UseAppConfigForBranding.pdf to see how multiple configuration sets are used.

Read http://www.limnor.com/support/CreateAppConfigUtility.pdf to see how to separate configuration modification from the program being configured.

Feedback

Please send your feedback to support@limnor.com