University of Anbar
College of Computer Science
and Information Technology
Computer Network Systems Department

Visual Programming I

Lecture Thirteen

Third Stage

First Course 2021 - 2022

Seddiq Qais Abd Al-Rahman

MSc Computer Science

co.sedeikaldossary@uoanbar.edu.iq

Visual Programming

Windows Forms Message Boxes

Message boxes provide a simple, standardised method to display information, warnings and errors to the user. They can also be used to ask simple questions and to request confirmation of actions.

Message Boxes

When developing an application using Windows Forms, you will often wish to display a message to the user, or ask a simple yes or no question. In most cases, creating a custom dialog box or window for this purpose would be an inefficient use of time, and may lead to a non-standard dialog box that confuses users. To simplify and standardise displaying this type of dialog box, Windows Forms includes the *MessageBox* class.

The MessageBox class is used to display modal dialog boxes containing a basic message, a title and one or more buttons. You can also include an icon to show the purpose of the message. When you display a dialog box, the calling form stops responding until the user clicks a button, at which point the message box disappears. You can then detect which button was pressed and act accordingly.

Displaying a Simple Message

The simplest type of message box is displayed using a call to the MessageBox class's *Show* method, passing the message to display as a string.

We can demonstrate this with a simple example. Create a new Windows Forms project and add a button to the automatically generated form. Double-click the button to create a *Click* event handler. Add code to button's event, as follows:

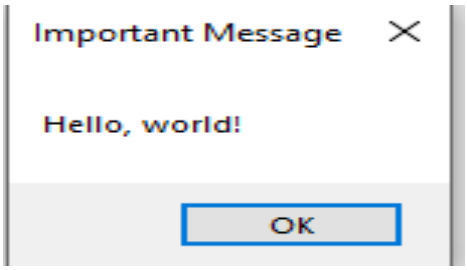
```
private void button1_Click(object sender, EventArgs e)
{
   MessageBox.Show("Hello, world!");
}
```

Run the program and click the button to see the results. When the message box appears, you should find that the original window no longer responds to input. Clicking the OK button closes the dialog box and returns control to the main window.

Adding a Title

As the above image shows, a message box does not include a caption on the title bar by default. You can add one with a second string argument in the call to Show, as in the following example:

```
private void button1_Click(object sender, EventArgs e)
{
   MessageBox.Show("Hello, world!", "Important Message");
}
```



Choosing a Button Set

If you only wish to display a message, the default OK button is generally sufficient. However, when you want to ask the user a question or confirm an action, you need to provide a different set of buttons. You can do this by using a third parameter, which accepts a value from the *MessageBoxButtons* enumeration. The enumeration contains one constant for each available button set. The options are:

- **OK**. The default option that displays the OK button alone.
- **OKCancel**. This option causes two buttons to be displayed. They show the text "OK" and "Cancel". This set is useful when you want the user to confirm an action that they have started but which may cause data loss.
- **YesNo**. The option shows "Yes" and "No" buttons. They are ideal for asking the user a simple question.
- **YesNoCancel**. This option adds a "Cancel" button alongside "Yes" and "No". This option is useful for asking the user how to proceed with an action whilst still giving the option to cancel. For example, when closing an application with an unsaved document you might ask the user if they want to save it. Clicking "Yes" would save

the document and close. Clicking "No" would close the application without saving. Clicking "Cancel" would leave the application open.

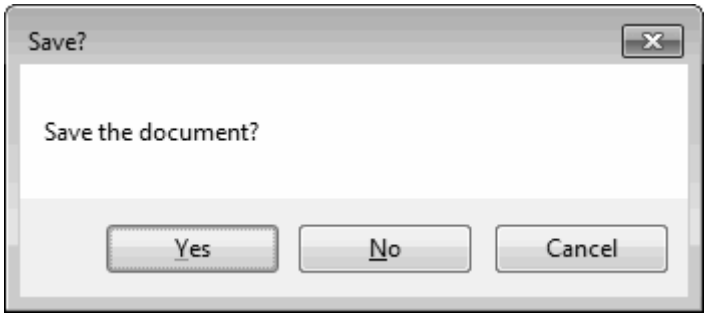
- **RetryCancel**. This value indicates that the dialog box will display "Retry" and "Cancel" buttons. It is useful for simple processes that may fail. On failing, you can display the problem and the user can elect to try again or give up.

- **AbortRetryCancel**. A more complex version of RetryCancel, this option adds an "Abort" button. Again, this can be useful for obtaining a user's response to a problem.

To demonstrate, try running the following, updated code:

```
private void button1_Click(object sender, EventArgs e)
{
   MessageBox.Show("Save the document?", "Save?", MessageBoxButtons.YesNoCancel);
}
```

The dialog box now appears as shown below:



To determine which button the user clicked, you use the return value. This will be a value from the *DialogResult* enumeration, which has one constant for each available button.

```
private void button1_Click(object sender, EventArgs e)
{
   DialogResult result = MessageBox.Show(
      "Save the document?", "Save?", MessageBoxButtons.YesNoCancel);

   Text = result.ToString();
}
```

Run the above code and click a button. The title bar of the main window will update to show which button was clicked.

Adding an Icon

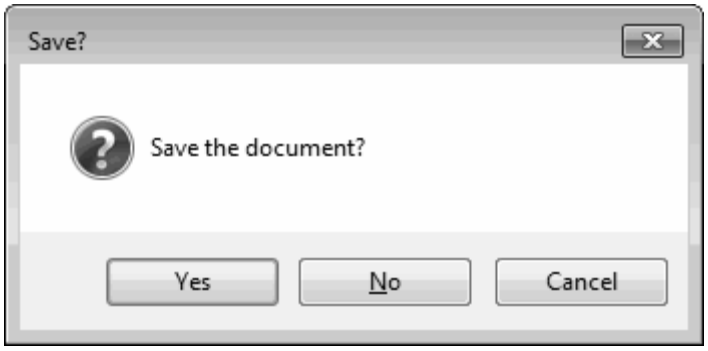
You can add an icon to a message box to give a visual cue to the severity of an error, or to indicate the type of action that you are requesting the user to take. To do so, add an extra argument to the call to Show, passing a value from the *MessageBoxIcon* enumeration. The enumeration includes a number of different icon options, though some show the same image.

Image	Constants	Description
	Error / Stop / Hand	Used when the message box indicates that an error occurred.
	Exclamation / Warning	Used to show a warning.
	Asterisk / Information	Used to indicate that the message box is showing only information.
	Question	Used when the message box includes a question.
	None	Shows no icon.

To demonstrate, let's add a question mark icon to the example message box.

```
private void button1_Click(object sender, EventArgs e)
{
   DialogResult result = MessageBox.Show(
      "Save the document?", "Save?", MessageBoxButtons.YesNoCancel,
      MessageBoxIcon.Question);
       Text = result.ToString();
}
```

The message box now appears as shown below:



Specifying a Default Button

If you look closely at the above image, you can see that the Yes button is the default option. If the user presses the Enter key before any other, the message box will close and return *Yes* as its result. You can change which button is the default by passing a further argument of the *MessageBoxDefaultButton* type. This has three constants, *Button1*, *Button2* and *Button3*. They map to the buttons in the order in which they appear in the message box, so to make "No" the default, you can use Button2:

```
private void button1_Click(object sender, EventArgs e)
{
   DialogResult result = MessageBox.Show(
      "Save the document?", "Save?", MessageBoxButtons.YesNoCancel,
      MessageBoxIcon.Question, MessageBoxDefaultButton.Button2);
```

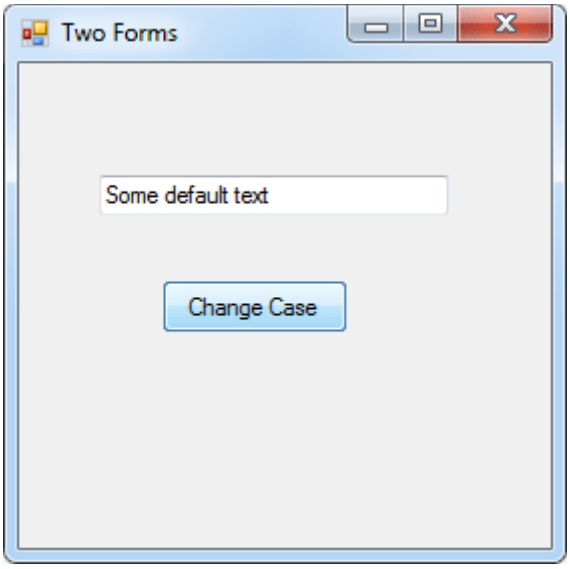
```
            Text = result.ToString();
        }
```

NB: If you use an invalid default button, such as Button3 when only two buttons are visible, the option is ignored.

Creating Multiple Forms

There aren't many programmes that have only one form. Most programmes have other forms that are accessible from the main one that loads at start up. In this section, you'll learn how to create programmes with more than form.

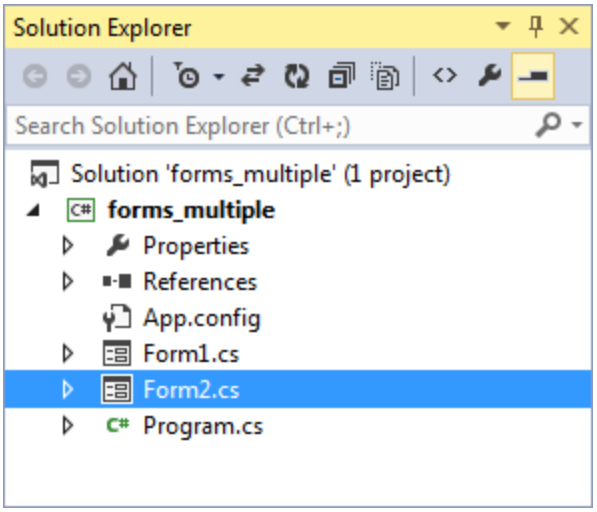
The programme we'll create is very simple one. It will have a main form with a text box and a button. When the button is clicked, it will launch a second form. On the second form, we'll allow a user to change the case of the text in the text box on form one. Here's what form one looks like:



Design the above form. Set the Name property of the text box to txtChangeCase. For the Text property, add some default text, but all in lowercase letters. Set the Name property of the button to btnFormTwo.

Adding a new form to the project is easy. Click Project from the menu bar at the top of the Visual C# software. From the Project menu, select Add New Windows Form. You'll see the Add New Item dialogue box appear. Make sure Windows Form is selected. For the Name, leave it on the default of Form2.cs. When you click OK, you should see a new blank form appear:

It will also be in the Solution Explorer on the right:



Adding the form to the project is the easy part - getting it to display is an entirely different matter!

To display the second form, you have to bear in mind that forms are classes. When the programme first runs, C# will create an object from your Form1 class. But it won't do anything with your Form2 class. You have to create the object yourself.

So double click the button on your Form1 to get at the coding window.

To create a Form2 object, declare a variable of Type Form2:

Form2 secondForm;

Now create a new object:

secondForm = new Form2();

Or if you prefer, put it all on one line:

Form2 secondForm = new Form2();

What we've done here is to create a new object from the Class called Form2. The name of our variable is secondForm.

To get this new form to appear, you use the Show( ) method of the object:

secondForm.Show();

Your code should now look like this:

```
private void btnFormTwo_Click(object sender, EventArgs e)
{
    Form2 secondForm = new Form2();

    secondForm.Show();
}
```

Run your programme and test it out. Click your button and a new form should appear - the blank second form.

However, there's a slight problem. Click the button again and a new form will appear. Keep clicking the button and your screen will be filled with blank forms!

To stop this from happening, move the code that creates the form outside of the button. Like this:

```
Form2 secondForm = new Form2();

private void btnFormTwo_Click(object sender, EventArgs e)
{
    secondForm.Show();
}
```

Try your programme again. Click the button and you won't get lots of forms filling the screen. Return to the code for your button. Instead of using the Show method, change it to this:

secondForm.ShowDialog*();*

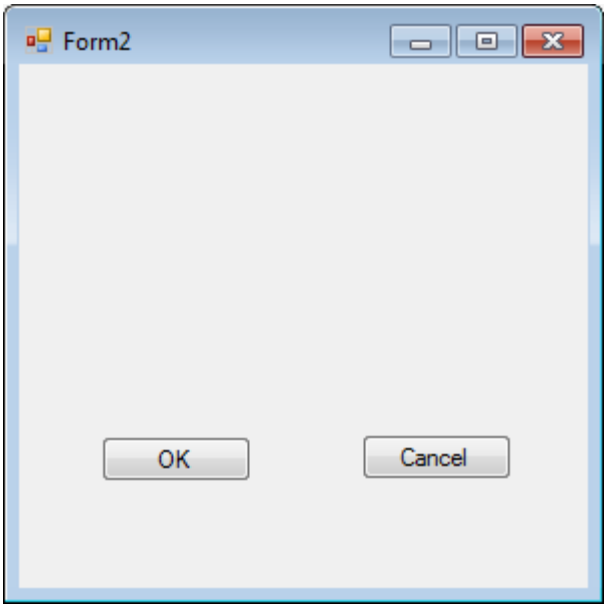
The method we're now using is ShowDialog. This creates what's known as a Modal form. A Modal form is one where you have to deal with it before you can continue. Run your programme to test it out. Click the button and a new form appears. Move it out of the way and try to click the button again. You won't be able to.

Modal forms have a neat trick up their sleeves. Add two buttons to your blank second form. Set the following properties for them:

Name:btnOK
Text: OK

Name:btnCancel
Text: Cancel

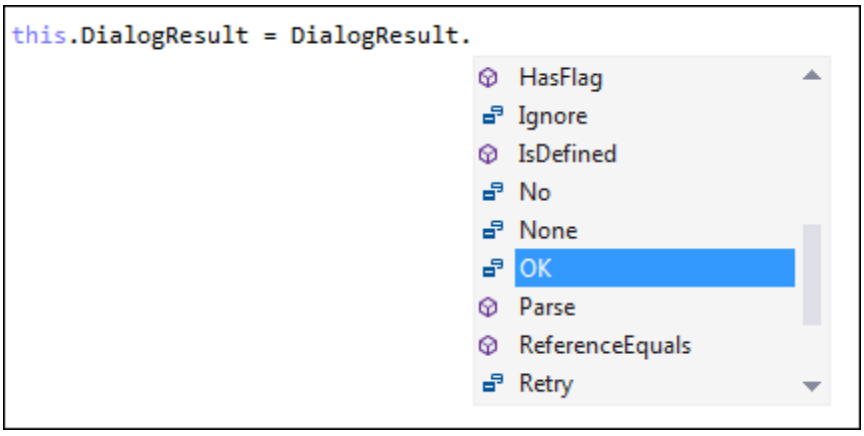
Your second form will then look like this:



Double click the OK button and add the following:

this.DialogResult = DialogResult.OK;

After you type the equals sign, the IntelliSense list will appear. Select DialogResult again, then a dot. The IntelliSense list will then show you this:



Select OK. What this does is to record the result of the button click, and set it to OK.

Double click your Cancel button and add the following code:

this.DialogResult = DialogResult.Cancel;

It's the same code, except we've chosen Cancel as the Result. Your coding window for form 2 should look like this:

```
public Form2()...

private void btnOK_Click(object sender, EventArgs e)
{
    this.DialogResult = DialogResult.OK;
}

private void btnCancel_Click(object sender, EventArgs e)
{
    this.DialogResult = DialogResult.Cancel;
}
```

You can use Form1 to get which of the buttons was clicked on Form2. Was it OK or was it Cancel?

Change the button code on Form1 to this:

```
private void btnFormTwo_Click(object sender, EventArgs e)
{

    if (secondForm.ShowDialog() == DialogResult.OK)
    {
        MessageBox.Show( "OK button clicked");
    }
}
```

The code checks to see if the OK button was clicked. If so, it displays a message. We'll get it to do something else in a moment. But you don't have to do anything with the Cancel button: C# will just unload the form for you.

Try it out. Click your Change Case button on Form1. When your new form appears, click the OK button. You should see the message. Try it again, and click the Cancel button. The form just unloads

References:

- Tony Gaddis, “Starting out with Visual C#.”, Fourth edition, Boston, Pearson Inc., 2017.
- Salvatore A. Buono, “C# and Game Programming: A Beginner's Guide.” Second Edition, Boca Raton, CRC Press Inc., 2019.
- Eric Butow and Tommy Ryan, "C#: Your Visual Blueprint for Building .NET Applications.", Hungry Minds Inc., New York, 2002.
- Faraz Rasheed, “Programmer's Heaven: C# School.”, First Edition, Fuengirola, Synchron Data, 2006.