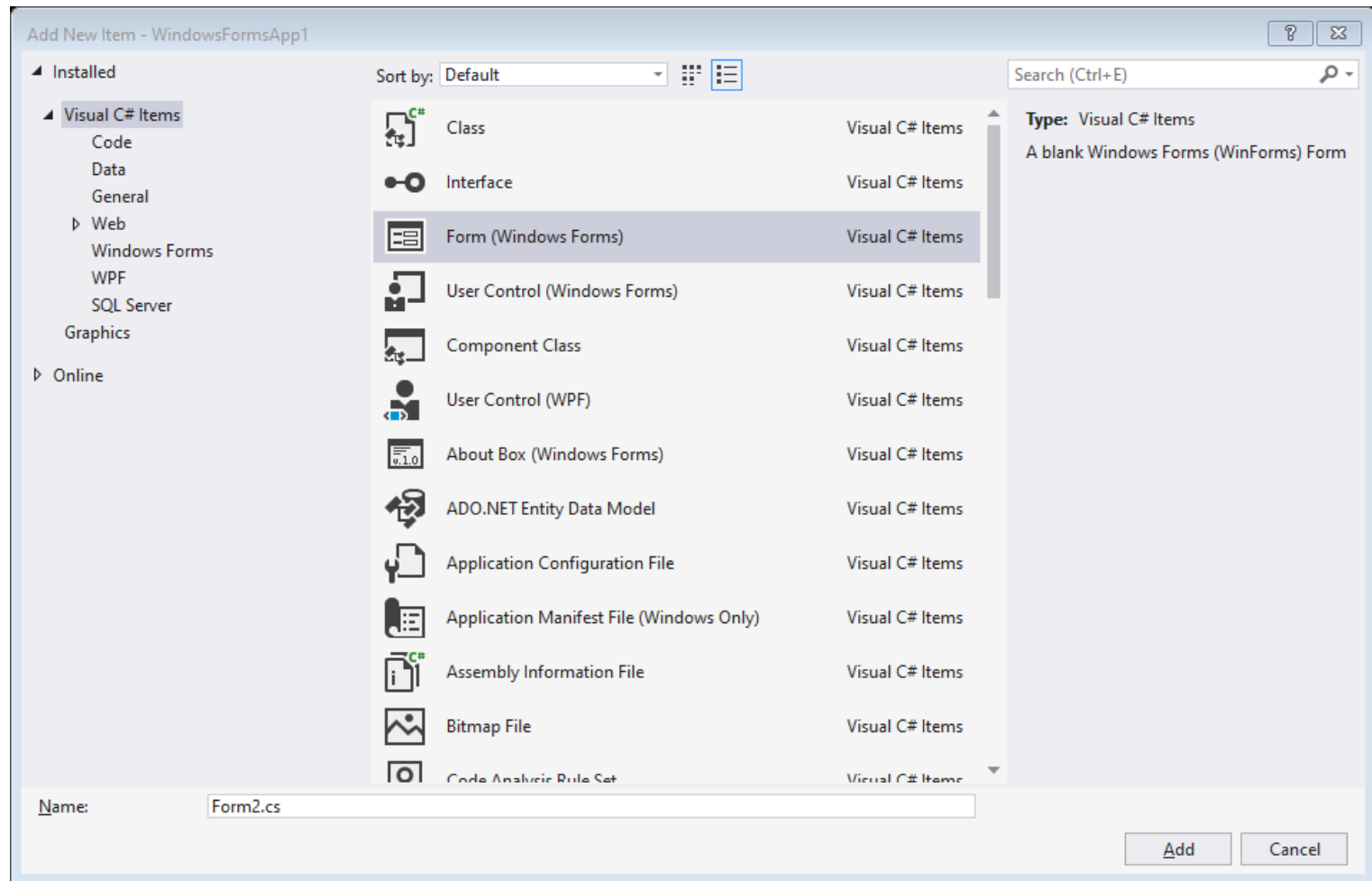


Dialogs

Basics

- A dialog is just a form with certain properties set.
- Right click the project name and then select Add|New Item.
- When the new item dialog appears select Windows Form and change the name to something appropriate, for example, “OptionsDialog.”

Adding the Dialog

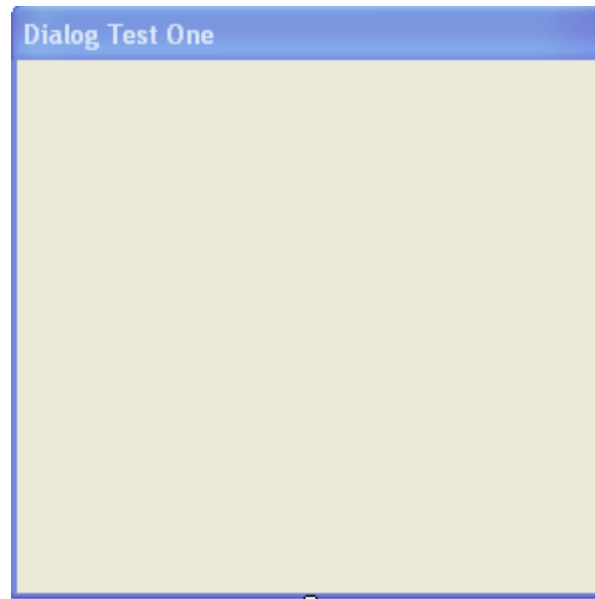


Properties

- Set these properties:

<i>Property</i>	<i>Value</i>
FormBorderStyle	FormBorderStyle.FixedDialog
ControlBox	false
MinimizeBox	false
MaximizeBox	false
ShowInTaskbar	false

The Empty Dialog



Adding Controls

- Use the designer to add controls just as you would do with the main form.
- Size the dialog appropriately.
- You will interact with the form just as you do with the main form except that you use the reference variable for the dialog.
- For example:
`Dialog1.TextBox1.Text="hello".`

Displaying the Dialog

- There are two main types of dialogs, the modal dialog and the modeless dialog.
- The modal dialog is a dialog that must be closed after use in order to continue using the main form.
- The modeless dialog stays on the screen and the user can switch back and forth between the main form and the dialog.
- Normally we provide a pair of buttons, OK and Cancel, for the user to indicate his or her intentions when changes are made in the dialog.

Displaying the Dialog – Contd.

- The *ShowDialog* method is called to display a modal dialog.
- The OK or Cancel button is normally used to close the dialog, but a dialog can be closed by typing Alt-F4.
- Here is the call to *ShowDialog*:
`myDialog.ShowDialog();`

Dialog Results

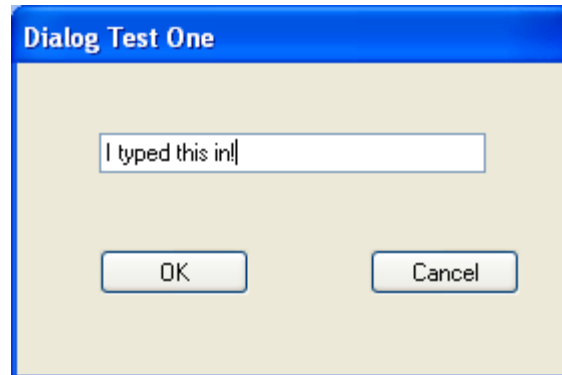
- The *ShowDialog* method returns one of these values:

<i>Member</i>	<i>Description</i>
Abort	The dialog box return value is Abort (usually sent from a button labeled Abort).
Cancel	The dialog box return value is Cancel (usually sent from a button labeled Cancel).
Ignore	The dialog box return value is Ignore (usually sent from a button labeled Ignore).
No	The dialog box return value is No (usually sent from a button labeled No).
None	Nothing is returned from the dialog box. This means that the modal dialog continues running.
OK	The dialog box return value is OK (usually sent from a button labeled OK).
Retry	The dialog box return value is Retry (usually sent from a button labeled Retry).
Yes	The dialog box return value is Yes (usually sent from a button labeled Yes).

Dialog Results – Contd.

- The *DialogResult* property is set to one of these values which are included in the *DialogResult* Enumeration.
- Notes that *DialogResult* is both a property and an enumeration. Again, the context is used to determine which.

Dialog1 Example



Dialog1 Example

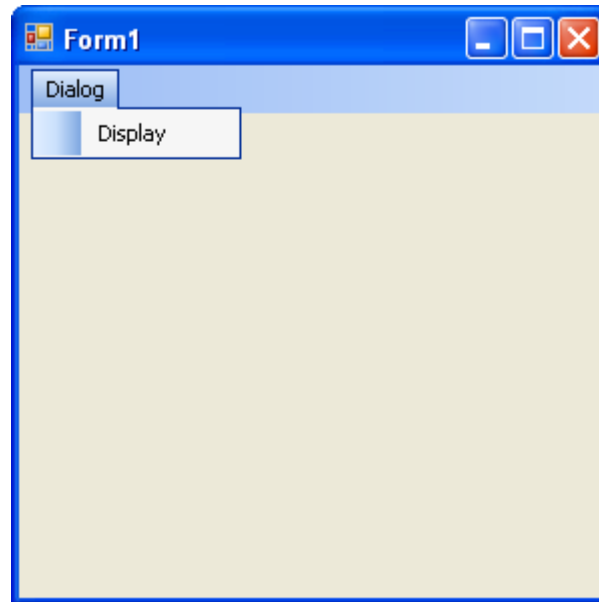
```
Dialog1 - Dialog1.cs
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
namespace Dialog1
{
    public partial class Dialog : Form
    {
        public Dialog()
```

Dialog1 Example – Contd.

```
{  
    InitializeComponent();  
}  
private void OKbutton_Click(object sender, EventArgs e)  
{  
    DialogResult = DialogResult.OK;  
}  
private void Cancelbutton_Click(object sender, EventArgs  
e)  
{  
    DialogResult = DialogResult.Cancel;  
}  
}  
}
```

Dialog1 Main Form

- The main form has a simple menu to invoke the dialog.



Dialog1 Main Form Code

```
Dialog1 - Form1.cs
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
namespace Dialog1
{
    public partial class Form1 : Form
    {
        private string boxtext = "";
    }
}
```

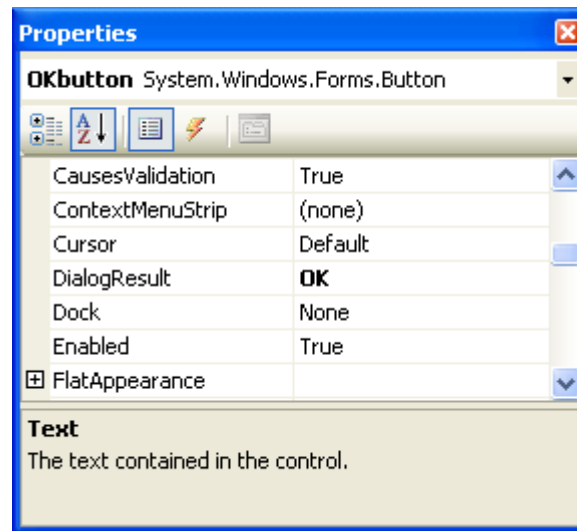
Dialog1 Main Form Code – Contd.

```
public Form1()  
{  
    InitializeComponent();  
}  
private void dIsplayToolStripMenuItem_Click(  
    object sender,           EventArgs e)  
{  
    Dialog myDialog = new Dialog();  
    myDialog.data.Text = boxtext;  
    if(myDialog.ShowDialog()==DialogResult.OK)  
        boxtext=myDialog.data.Text;  
}  
}
```


DialogResult Shortcut

Dialog2

- All we need to do is to set the *DialogResult* property for each of the buttons to the appropriate value.
- No code needs to be written. No event handlers for the buttons are needed now.



Design Issues

- Making the dialog fields *public* is not always the best approach.
- *Properties* are obviously the preferred solution. This way the dialog internals can be hidden.
- It is often better to instantiate the dialog only once for the life of the program rather than each time it is needed.
- The Dialog3 example demonstrates these techniques.

Dialog3 Example

```
Dialog3 - Form1.cs
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
namespace Dialog3
{
    public partial class Form1 : Form
    {
        private string boxtext = "";
    }
}
```

Dialog3 Example – Contd.

```
private Dialog myDialog = new Dialog();

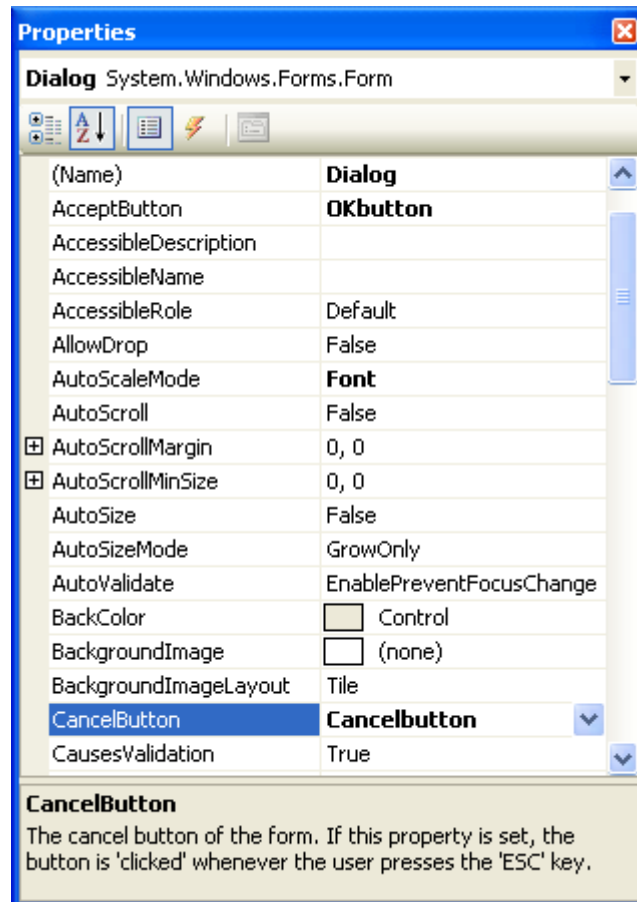
public Form1()
{
    InitializeComponent();
}

private void dIsplayToolStripMenuItem_Click(object
    sender, EventArgs e)
{
    myDialog.data.Text = boxtext;
    if (myDialog.ShowDialog() == DialogResult.OK)
        boxtext = myDialog.data.Text;
}
}
```

Accept and Cancel

- If a button control has the focus then pressing the return key or typing a space is identical to clicking on it with your mouse.
- If a non-button control has the focus then hitting return has a different effect depending on the control.
- It is very simple to associate the default action for the return key, if it is not processed by the control (e.g. by a text box).
- The *AcceptButton* and *CancelButton* properties of the form can be set to the buttons of your choice.
- Return associates with accept and ESC associates with cancel.

Setting the Properties



Screen Position

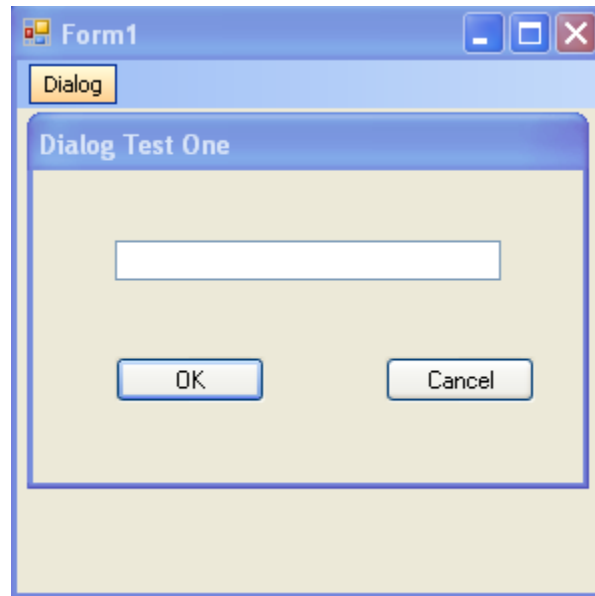
- Use the following as appropriate:

<i>Member</i>	<i>Description</i>
CenterParent	The form is centered within the bounds of its parent form.
CenterScreen	The form is centered on the current display, and has the dimensions specified in the form's size.
Manual	The position of the form is determined by the Location property.
WindowsDefaultBounds	The form is positioned at the Windows default location and has the bounds determined by Windows default.
WindowsDefaultLocation	The form is positioned at the Windows default location and has the dimensions specified in the form's size.

The Parent Container

- The default container for a new form is the desktop.
- If you want the dialog centered in the main form you need to set the *Owner* property:
`myDialog.Owner = this;`
- Normally this property is set when you instantiate the dialog object.

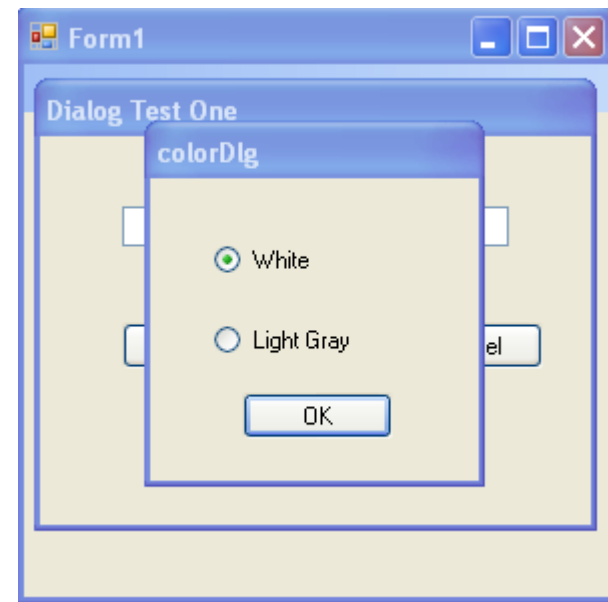
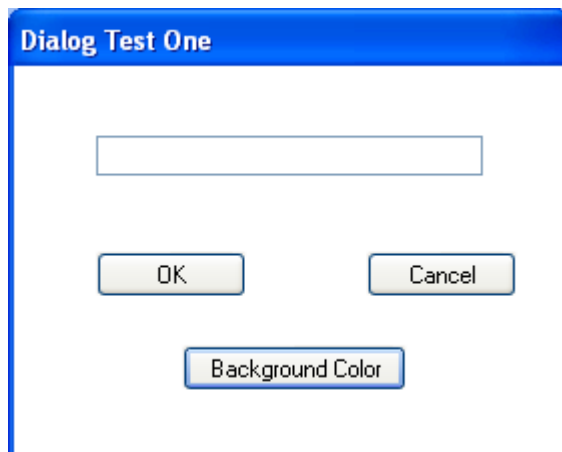
Using CenterParent



Handling Events in a Dialog

- You can add event handlers in a dialog that are processed by the dialog code rather than code in the main form.
- The designer automatically adds event handlers for a form in that form itself. This is usually what you want.
- For example, you might want to reset all the controls in the form to a *default* state.
- Dialog4 is a simple example that changes the background color.
- In the color dialog rather than make the radio buttons public I added a property that returns a *Color* object.

Dialog4 Example



Modeless Dialogs

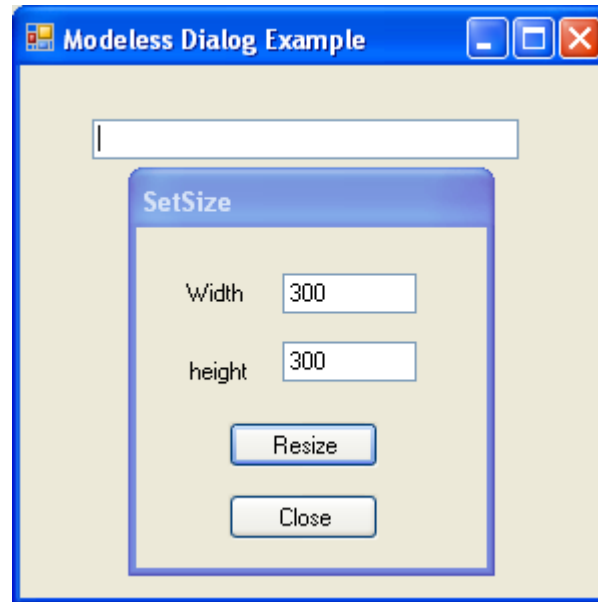
- A modeless dialog is designed to stay on the screen until the user closes it.
- Work can be done using the main form while the modeless dialog is displayed.
- Normally an OK or Cancel button is not appropriate for a modeless dialog.
- A Close button or other appropriate button is provided instead.
- Use also normally need at least one action button that applies the information in the dialog to your application.
- Use the *Show* method instead of *ShowDialog*.

Modeless Dialog Techniques

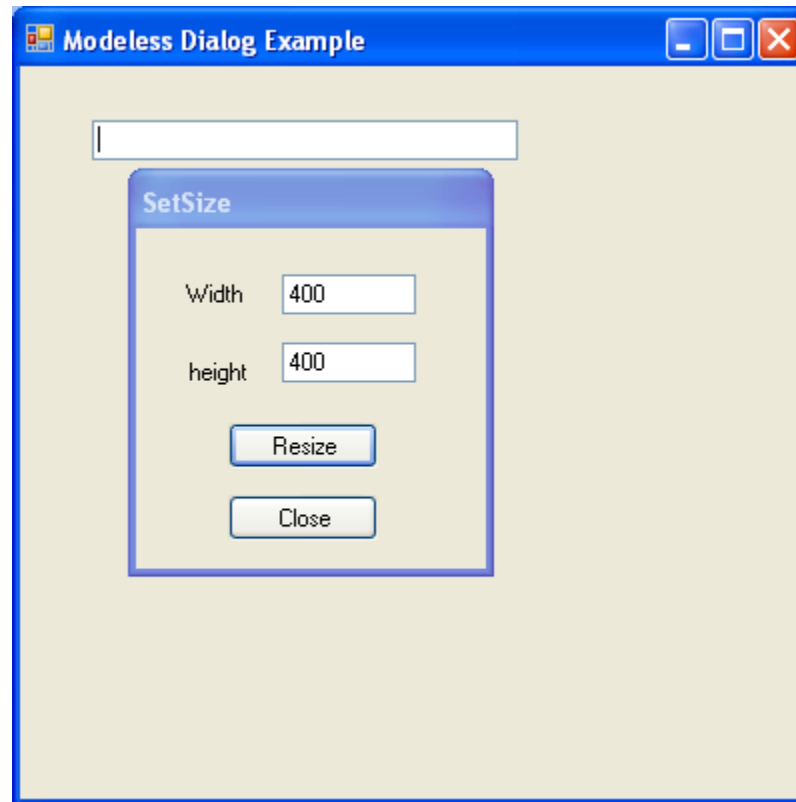
- In the following example I instantiate the modeless dialog only once, in the main form's constructor.
- A right mouse click displays the modeless dialog.
- The Close button does not actually close the form which would require re-instantiating the class.
- I used the *Hide* method to remove the form from the screen. It's still there though.

Example

Resize the Form



After Changing the Size



Dialog Code

```
ModelessDialog - SetSize.cs
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
namespace ModelessDialog
{
    public partial class SetSize : Form
    {
        public SetSize()
        {
            InitializeComponent();
        }
        private void close_Click(object sender, EventArgs e)
        {
            Hide();
        }
    }
}
```


Dialog Code – Contd.

```
}  
private void resizeButton_Click(object sender, EventArgs e)  
{  
    int w, h;  
    try  
    {  
        w = Convert.ToInt32(width.Text);  
        h = Convert.ToInt32(height.Text);  
    }  
    catch  
    {  
        MessageBox.Show("Invalid size!");  
        return;  
    }  
    Owner.Width = w;  
    Owner.Height = h;  
}  
}
```

Main Form Code

```
ModelessDialog - Form1.cs
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
namespace ModelessDialog
{
    public partial class Form1 : Form
    {
        SetSize sizeDialog = new SetSize();
    }
}
```

```
public Form1()  
{  
    InitializeComponent();  
}  
private void Form1_MouseDown(object sender,  
    MouseEventArgs e)  
{  
    if (e.Button == MouseButton.Right)  
    {  
        sizeDialog.Show(this);  
    }  
}  
}
```

Common Dialog Classes

- Windows has always included several standard dialogs internal to the operating system.
- These are wrapped by .NET classes and are simple to use.
- The following table lists the common dialogs available under .NET.
- I will show you the font and color dialogs.

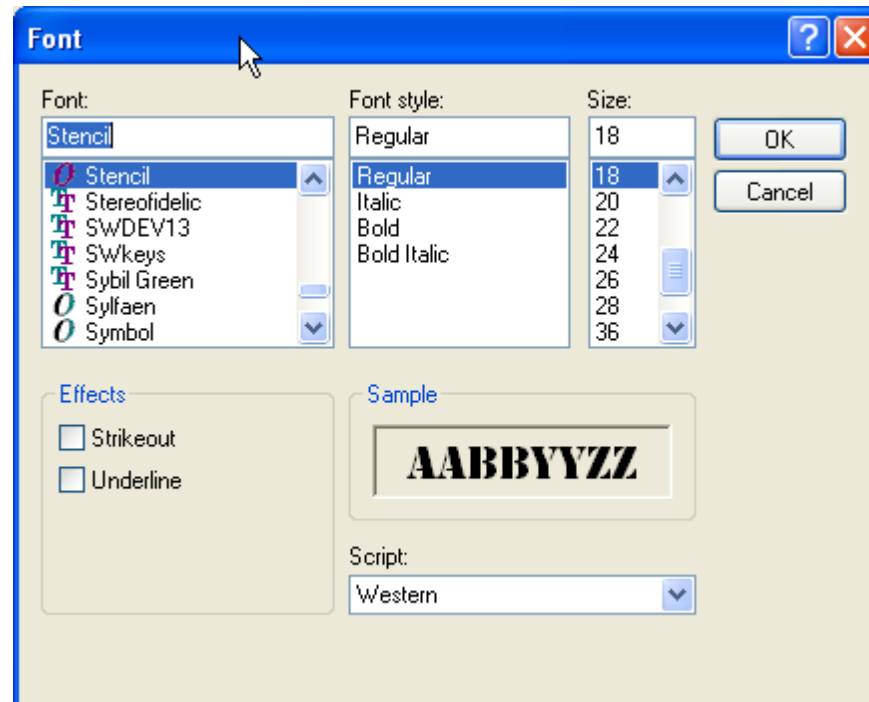
Available Common Dialogs

<i>Class</i>	<i>Description</i>
System.Windows.Forms.ColorDialog	Represents a common dialog box that displays available colors along with controls that allow the user to define custom colors.
System.Windows.Forms.FileDialog	Displays a dialog box from which the user can select a file.
System.Windows.Forms.FolderBrowserDialog	Prompts the user to select a folder. . This class cannot be inherited.
System.Windows.Forms.FontDialog	Prompts the user to choose a font from among those installed on the local computer.
System.Windows.Forms.PageSetupDialog	Enables users to change page-related print settings, including margins and paper orientation.
System.Windows.Forms.PrintDialog	Allows users to select a printer and choose which portions of the document to print.

The Font Dialog

- Selecting a font is very common in applications.
- The user wants to select at least the font, size, and style.
- It is also useful to see what the font looks like in a sample.
- The font dialog does all this and a bit more.
- To use a common dialog all we do is to instantiate it and use the properties and methods that are appropriate.
- In some cases it is even possible to inherit from a common dialog.

The Font Dialog



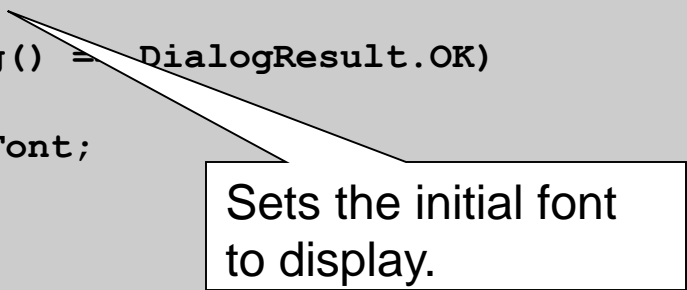
Font Example

CommDlg1

```
CommDlg - Form1.cs
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
namespace CommDlg1
{
    public partial class Form1 : Form
    {
        private Font myFont;
        public Form1()
        {
            InitializeComponent();
            myFont = Font; //default font
        }
        protected override void OnPaint(PaintEventArgs e)
```


Font Example – Contd.

```
{
    Graphics g = e.Graphics;
    g.DrawString("Right click to change the font.",
        myFont, Brushes.Black, 10, 10);
}
private void Form1_MouseClick(object sender, MouseEventArgs e)
{
    if (e.Button == MouseButton.Right)
    {
        FontDialog fontDialog = new FontDialog();
        fontDialog.Font = myFont;
        if (fontDialog.ShowDialog() == DialogResult.OK)
        {
            myFont = fontDialog.Font;
            Invalidate();
        }
    }
}
}
```



Sets the initial font to display.

Result with a Font Change



The Color Dialog

- The color dialog allows the user to select a color using several different methods including RGB.
- Initially the dialog opens in a simple selection mode.
- Custom colors can be defined which displays more extensive selection methods.
- The following example allows changing the background color using a color dialog.

Color Dialog Example

CommDlg2

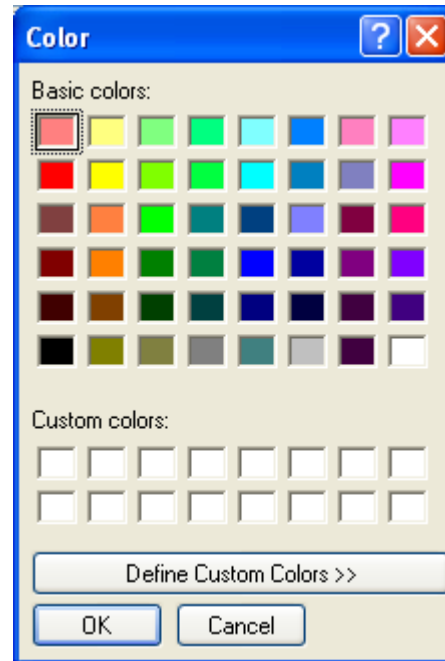
```
CommDlg2 - Form1.cs
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
namespace CommDlg2
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        protected override void OnPaint(PaintEventArgs e)
        {

```

Color Dialog Example – Contd.

```
        Graphics g = e.Graphics;
        g.DrawString("Right click to change the form's backgroun
color.",
                    Font, Brushes.Black, 10, 10);
    }
    private void Form1_MouseClick(object sender, MouseEventArgs e)
    {
        ColorDialog colorDialog = new ColorDialog();
        colorDialog.Color = BackColor;
        if (e.Button == MouseButtons.Right)
        {
            if (colorDialog.ShowDialog() == DialogResult.OK)
            {
                BackColor = colorDialog.Color;
                Invalidate();
            }
        }
    }
}
```

The Simple Color Dialog



Color Dialog Expanded

