FULL SAIL UNIVERSITY

Modal Dialogs

Description

There are several steps involved in converting a default Form window into a properly functioning modal dialog box. Once this is done the OK and Cancel buttons should work properly. Properties must then be added to the dialog in order to exchange its data. Finally a custom event needs to implement for the Apply button.

Contents

Form to Modal Dialog

Add a new Form to your project

Using the *Add New Item…* templates add a new *Windows Form* to your project. Let's call it ModalDialog.

FormBorderStyle

Set the **FormBorderStyle** property to **FixedDialog**. This will remove the system menu and make the dialog unsizable.

MinimizeBox and MaximizeBox

Set the **MinimizeBox** and **MaximizeBox** properties to **false**. This will remove them from the title bar.

OK and Cancel Buttons

Add an OK and Cancel button to the dialog box. You won't need to handle the Click events for either button but you will need to set their **DialogResult** properties to the correct values. For the OK button that is **DialogResult.OK**. For the Cancel button it's **DialogResult.Cancel**. Setting these properties will cause both buttons to close the dialog when they are clicked and the correct **DialogResult** value will be returned from the **ShowDialog** method.

Enter and Escape Keys

Set the dialogs box's **AcceptButton** property to the OK button and its **CancelButton** property to the Cancel button. This will cause the dialog to treat the *enter key* as if the user clicked on the OK button and the *escape key* as if the Cancel button was clicked.

Centering the Dialog

Set the dialog's **StartPosition** property to **CenterScreen**.

Running the Modal Dialog

To display a dialog modally call the **ShowDialog** method. You will what to know whether the dialog was closed with the OK or Cancel button. This is determined by the return value of **ShowDialog**.

```
ModalDialog dlg = new ModalDialog();

if (DialogResult.OK == dlg.ShowDialog())
{
    // You only want to retrieve information
    // from the dialog if it was closed with
    // the OK button.
}
```

Properties are for Data Exchange

The purpose of a modal dialog is to present information to the user and allow them to edit that information. You will be passing data into the dialog before you call **ShowDialog** so the user can see the value of what they are editing. You will be retrieving the edited information when the user closes the dialog with the OK button. Since the information needs to be passed in and out of the dialog we will use properties for data exchange.

```
ModalDialog dlg = new ModalDialog();

// Set the properties
dlg.MyInteger = 10;
dlg.MyString = "Hello World!";

if (DialogResult.OK == dlg.ShowDialog())
{
    // Get the properties
    int x = dlg.MyInteger;
    string s = dlg.MyString;
}
```

Apply Button

The Apply button will need to publish a custom event that passes the information contained in the dialog's properties through the event argument class.

AppyEventArgs Class

Add a class to your project derived from the **EventArgs** class. Give it the same data exchange properties as your modal dialog

```
public class ApplyEventArgs : EventArgs
{
    int myInteger;
    string myString;

    public int MyInteger
    {
        get { return myInteger; }
        set { myInteger = value; }
    }

    public string MyString
    {
        get { return myString; }
        set { myString = value; }
    }

    public ApplyEventArgs(int myInteger, string myString)
    {
        this.myInteger = myInteger;
        this.myString = myString;
    }
}
```

ApplyEventHandler Delegate

In order to pass the **ApplyEventArgs** class with the event a custom delegate must be declared.

```
public delegate void ApplyEventHandler(object sender, ApplyEventArgs e);
```

The Apply Event

Now we must add a new event to the dialog box based on the **ApplyEventHandler**. Then publish the event when the Apply button is clicked and pass it an instance of the **ApplyEventArgs** class.

```
public partial class ModalDialog : Form
{
    // Declare the event
    public event ApplyEventHandler Apply;

    public ModalDialog()
    {
        InitializeComponent();
    }

    public int MyInteger { get; set; }
    public string MyString { get; set; }

    private void buttonApply_Click(object sender, EventArgs e)
    {
        // Publish the event if it is not null
        // and pass the information with the custom
        // event arguements class.
        if (Apply != null) Apply(this, new ApplyEventArgs(this.MyInteger, this.MyString));
    }
}
```

Subscribe to the Apply Event

Subscribe to the Apply event in the main Form and retrieve the information through the event arguments.

```
private void modalToolStripMenuItem_Click(object sender, EventArgs e)
{
    ModalDialog dlg = new ModalDialog();

    // Set the properties

    // Suscribe to the Apply event
    dlg.Apply += new ApplyEventHandler(dlg_Apply);

    if (DialogResult.OK == dlg.ShowDialog())
    {
        // Get the properties
    }
}

void dlg_Apply(object sender, ApplyEventArgs e)
{
    // Retrieve the event arguements
    int x = e.MyInteger;
    string s = e.MyString;
}
```