Windows and Dialogs

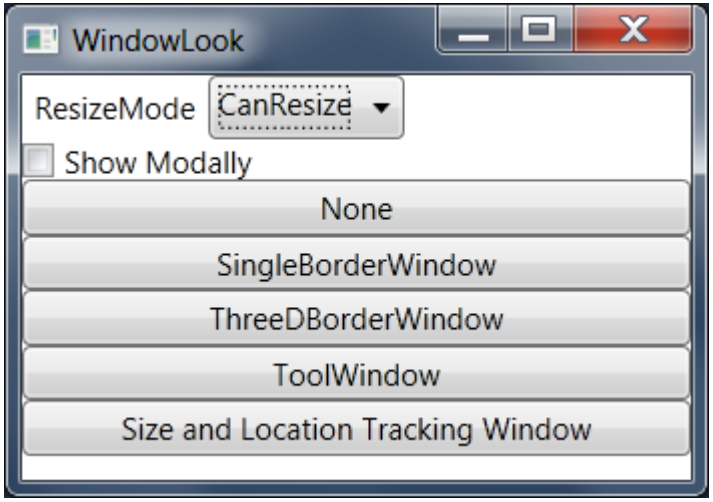
Study demos in 12WindowsAndDialogsDemos For further information

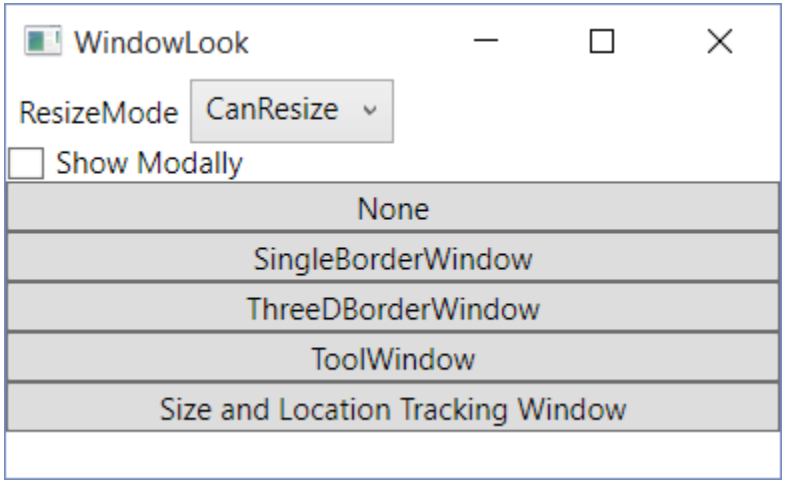
Agenda

- Windows
- Modale Dialogs
- Data Validation
- Modeless Dialogs
- Common Dialogs

The Window class

- The Window class derives from the ContentControl
  - and adds the chrome around the edges that contains:
    - the title
    - the minimize, maximize, and close buttons
- The content can look however you want
- But the chrome itself has more limited options

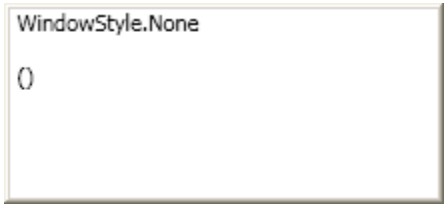


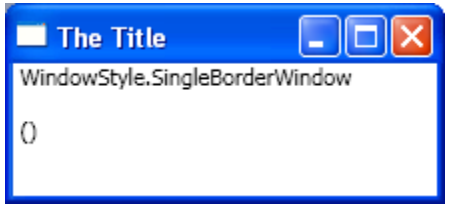


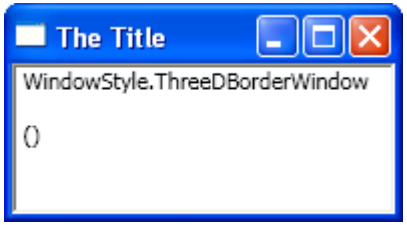
Window Look and Feel

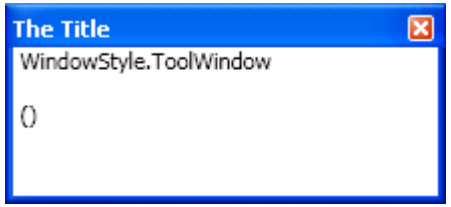
- The look and feel of the frame of a window is largely determined by the Icon, Title, and WindowStyle properties

On XP:

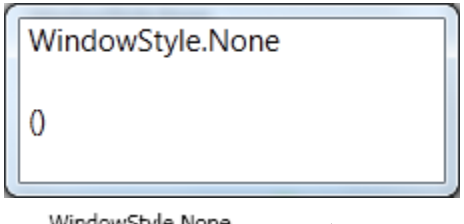


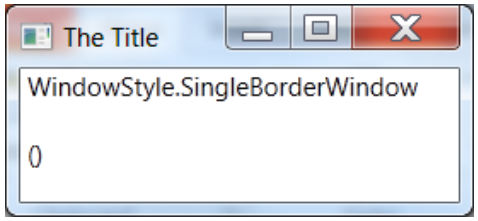


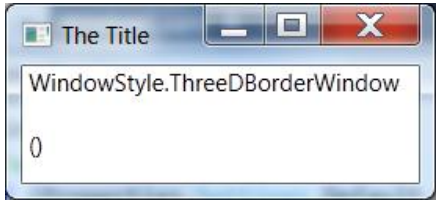


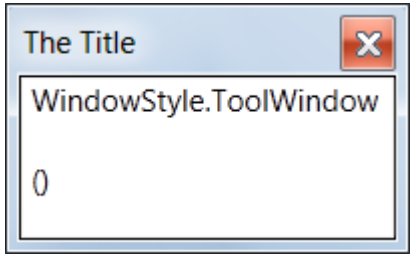


On Vista and Windows7:



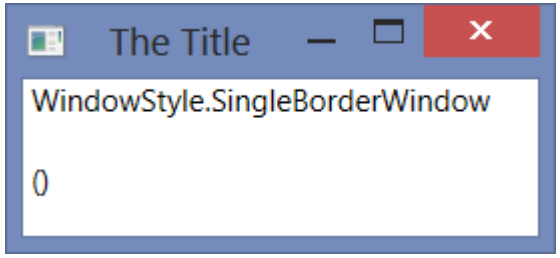






NoResize
gives no border

On Windows 8:

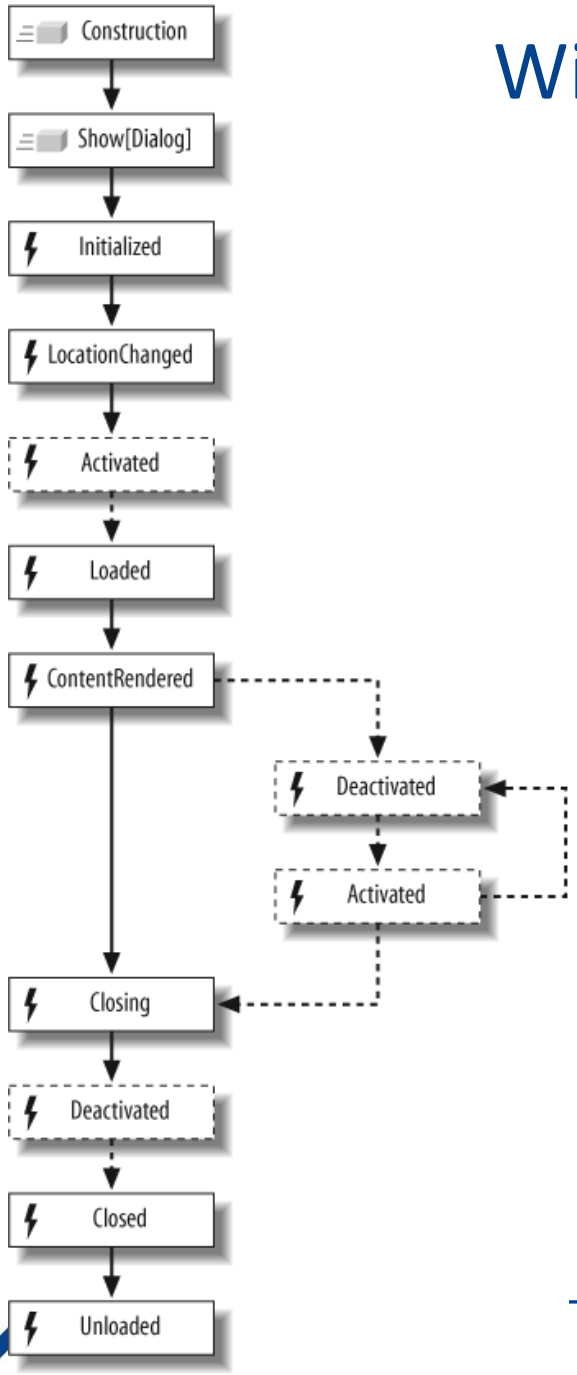


Window Lifetime

- Creating a window is as easy as calling new on the Window class, or more often, on a class derived from Window
- When you create a new WPF project in Visual Studio the Wizard will create a derived Windows class for you in XAML, and there will be a partial class MainWindow in C#
- The constructor for MainWindow must call the InitializeComponent() method
  - InitializeComponent will load the window's XAML and use it to initialize the properties that make up the window object and hook up the events defined in XAML
- **To show the Window you must call either**:
  - **Show**()
    - This is the method to use to create a new top-level window (~modeless dialog)
  - **ShowDialog**()
    - Is used to show the window as a modal dialog

Window Lifetime Events

- Once a window is created, it exposes a series of events to let you monitor and affect its lifetime



Window Startup Location

- There are 2 main principles to determine the startup location for a Window:
  - You can set the WindowStartupLocation property to:
    - **CenterScreen**

```
Window1 window = new Window1( );
window.WindowStartupLocation = WindowStartupLocation.CenterScreen;
window.Show( );              // will be centered on the screen
```

    - **CenterOwner**
    - Or **Manual** without setting the Top and Left properties – this will let the Windows OS decide the location
  - Or you can set the upper left corner of the window to a specific location with the Top and Left Properties before showing the window
- You can influence the Z order with the TopMost property
- Across the entire desktop, all windows with the TopMost property set to true appear above all of the windows with the TopMost property set to false
- The Z order of the windows within their layer is determined by user interaction
  - clicking on a non-topmost window will bring it to the top of the non-topmost layer of windows
  - but will not bring it in front of any topmost windows

Window Size

- You can get the size of the window from the **ActualWidth** and **ActualHeight** properties of the Window class
- The actual size properties are expressed (like all Window-related sizes) in device-independent pixels measuring 1/96th of an inch
- The ActualWidth and ActualHeight properties are read-only
- Use the **Width** and **Height** properties to set/influence the width and height of the window

```
Window1 window = new Window1( );
window.Show( ); // render window so ActualWidth is calculated
window.MinWidth = 200;
window.MaxWidth = 400;
window.Width = 100;
Debug.Assert(window.Width == 100); // regardless of min/max settings
Debug.Assert(window.ActualWidth == 200); // bound by min/max settings
```

Window Owners

- An owned window always shows in front of its owner unless the owned window is minimized
- When an owner window is minimized, all of the owned windows are minimized (and likewise for restoration)
- When the owner window is closed, so are all of the windows the owner owns
- When an owner window is activated, all of the owned windows are brought to the foreground with it

- **In practice, the chief visual use of an owned window is to create a floating tool window or modeless dialog**

MODAL DIALOGS

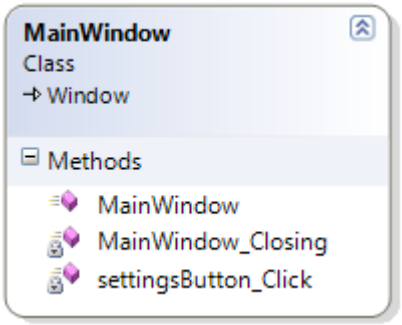
Modal Dialogs

- To create a modal dialog:
  1. Construct an ordinary Window subclass
  2. Show the window as a modal dialog:

```
void settingsButton_Click(object sender, RoutedEventArgs e) {
// Create dialog and show it modally, centered on the owner
SettingsDialog dlg = new SettingsDialog( );
dlg.Owner = this;
if (dlg.ShowDialog( ) == true) {
   // Do something with the dialog properties
   ...
 } }
```

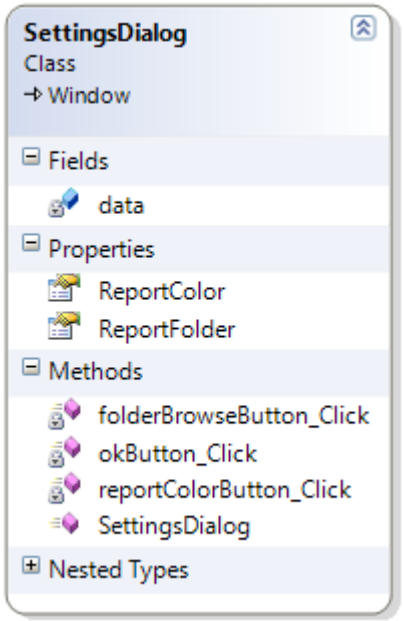
- Dialogs need more to fit into a Windows world:
  - The initial focus is set on the correct element
  - The dialog doesn't show in the taskbar
  - Data is passed in and out of the dialog
  - The OK button is shown as the default button (and is activated when the Enter key is pressed)
  - Cancel is activated when the Esc key is pressed
  - Data is validated before "OK" is really "OK"

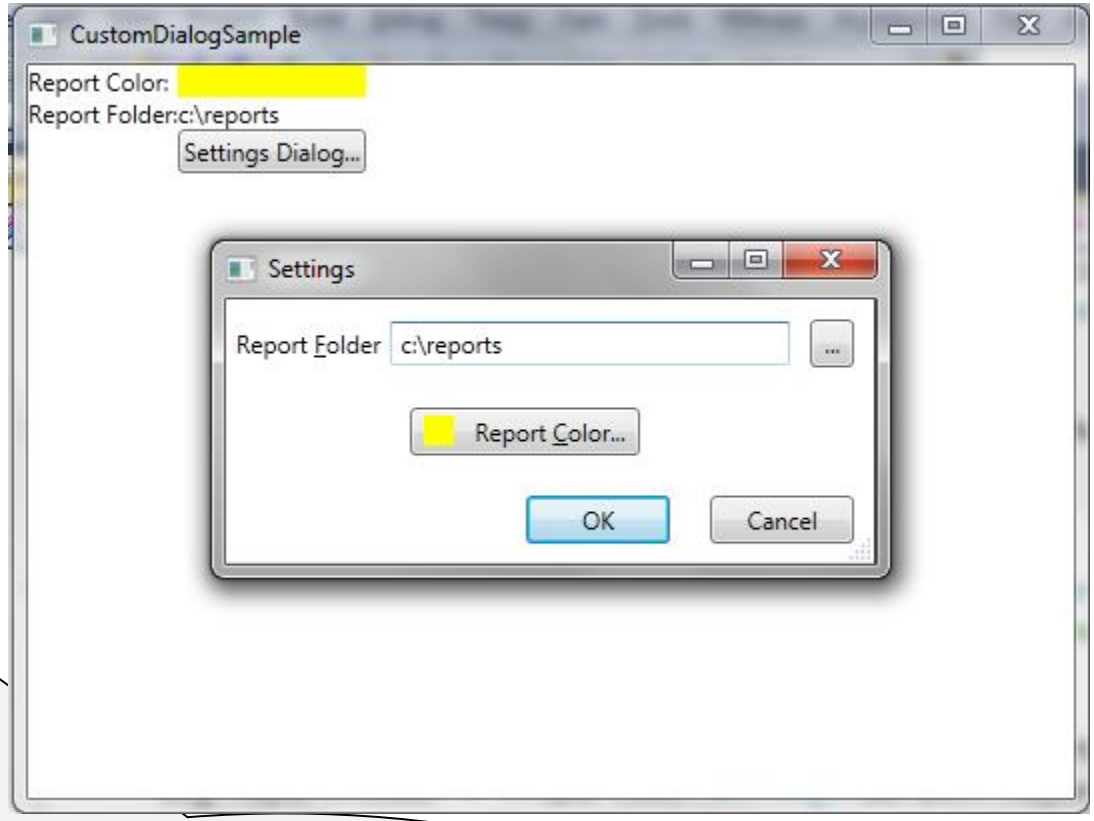
Communication between a modal dialog and its parent Window



Properties and methods calls

(Events)





Very often there are no communication from the dialog to the parent window. The parent fetches the data from the dialog

Dialog Data Exchange

```
class MainWindow : Window {
  ...
  Color reportColor;
  string reportFolder;
  void settingsButton_Click(object sender, RoutedEventArgs e)
  {
     // 1. Create and initialize the dialog with initial values
     SettingsDialog dlg = new SettingsDialog( );
     dlg.ReportColor = reportColor;
     dlg.ReportFolder = reportFolder;
     // 2. Show the dialog, letting the user choose new,
     // validated values
     if (dlg.ShowDialog( ) == true) {
        // 3. Harvest the values for use in your application
        reportColor = dlg.ReportColor;
        reportFolder = dlg.ReportFolder;
      }
   }
}
```

You'll notice that we're using a degree of good old-fashioned object-oriented encapsulation here, passing in and harvesting the values using .NET properties, but having no say about how the dialog shows those values to the user or how they're changed

Data Binding and Dialogs

- You'll want to avoid binding directly to reference type objects passed into a dialog
- The problem is that you could well be in a situation where the user makes changes to data and then clicks the Cancel button
- In that case, **WPF** provides no facilities for rolling back changes made via data binding
- That's why our example settings dialog keeps its own copies of the color and folder information

Handling OK and Cancel

- Closing a modal dialog automatically by changing DialogResult

```
void okButton_Click(object sender, RoutedEventArgs e) {
   // The return from ShowDialog will be true
   DialogResult = true;
   // No need to explicitly call the Close method
   // when DialogResult transitions to non-null
   // Close( );
}
```

- Cancel buttons transition DialogResult automatically

```
<!-- no need to handle the Click event to close dialog -->
<Button Name="cancelButton" IsCancel="True">Cancel</Button>
```

DATA VALIDATION

Data Validation

- Just because the user clicks the OK button doesn't mean that everything is OK
  - The data entered should be validated - if possible
- WPF provides per-control validation as part of the binding engine. E.g.:

```
<TextBox Grid.Row="0" Grid.Column="1"
         x:Name="reportFolderTextBox"
         ToolTip="{Binding RelativeSource={RelativeSource Self},
                           Path=(Validation.Errors)[0].ErrorContent}">
   <TextBox.Text>
      <Binding Path="ReportFolder">
         <Binding.ValidationRules>
            <local:FolderMustExist />
         </Binding.ValidationRules>
      </Binding>
   </TextBox.Text>
</TextBox>
```

How to Enforce a Complete Validation 1

- WPF miss functionality to manually initiate a validation of all controls in a window
  - Without this the user may skip some controls in a dialog, and they will not be validated.
- But you can use this method to do the job:

```
public static bool ValidateBindings(DependencyObject parent)
{
  bool valid = true;
  LocalValueEnumerator localValues = parent.GetLocalValueEnumerator();
  while (localValues.MoveNext()) {
    LocalValueEntry entry = localValues.Current;
    if (BindingOperations.IsDataBound(parent, entry.Property)) {
       Binding binding = BindingOperations.GetBinding(parent,
                                                      entry.Property);
       foreach (ValidationRule rule in binding.ValidationRules) {
         ValidationResult result = rule.Validate(
                                    parent.GetValue(entry.Property), null);
         if (!result.IsValid) {
  // Se the rest of the method in the demo 03DialogValidation
}
```

How to Enforce a Complete Validation 2

- Call your utility method `ValidateBindings` in your OkButton_Click handler:
  - (or in the ApplyButton handler for modeless dialogs)

```
void OkButton_Click(object sender, RoutedEventArgs e)
        {
            // Validate all controls
            if (ValidateBindings(this))
            {
                DialogResult = true;
            }
        }
```

MODELESS DIALOGS

Modeless Dialogs

- To create a modeless dialog:
  1. Construct an ordinary Window subclass
  2. Show the window as a modeless dialog:
  3. Style it to make it look like a dialog
  4. Pass the data into and out of the dialog with properties

```
public partial class SettingsDialog : System.Windows.Window
{
    DialogData data = new DialogData();  // Custom type

    public Color ReportColor {
      get { return data.ReportColor; }
      set { data.ReportColor = value; }
    }

    public string ReportFolder {
      get { return data.ReportFolder; }
      set { data.ReportFolder = value; }
    }

    public string Reporter {
      get { return data.Reporter; }
      set { data.Reporter = value; }
    }
```

Use Events in Modeless Dialogs

```
public partial class SettingsDialog : System.Windows.Window
{
 public SettingsDialog() {
      InitializeComponent();
      applyButton.Click += applyButton_Click;
      closeButton.Click += closeButton_Click;
    }

    // Fired when the Apply button is pressed
    public event EventHandler Apply;

    void applyButton_Click(object sender, RoutedEventArgs e) {
      // Validate all controls
      if( ValidateBindings(this) )
      {
        // Let modeless clients know they should read the data back
        if( Apply != null )
          { Apply(this, EventArgs.Empty); }

        // Don't close the dialog until Close is pressed
      }
    }
```

Create and Show a Modeless Dialog

```
void settingsButton_Click(object sender, RoutedEventArgs e)
{
    // Initialize the dialog
    if (dlg != null)
        dlg.Focus();
    else
    {
        dlg = new SettingsDialog();
        dlg.Owner = this;
        dlg.Reporter = Properties.Settings.Default.Reporter;
        dlg.ReportColor = Properties.Settings.Default.ReportColor;
        dlg.ReportFolder = Properties.Settings.Default.ReportFolder;

        // Listen for the Apply button and show the dialog modelessly
        dlg.Apply += new EventHandler(Dlg_Apply);
        dlg.Closed += new EventHandler(Dlg_Closed);
        dlg.Show();
    }
}
```

Implement Event Handlers in Parent Window

```
void Dlg_Closed(object sender, EventArgs e)
{
    dlg.Apply -= new EventHandler(Dlg_Apply);
    dlg.Closed -= new EventHandler(Dlg_Closed);
    dlg = null;
    Focus();
}

void Dlg_Apply(object sender, EventArgs e)
{
   // Pull the dialog out of the event args and fetch data
    SettingsDialog dlg = (SettingsDialog)sender;
    this.Reporter = dlg.Reporter;
    this.ReportColor = dlg.ReportColor;
    this.ReportFolder = dlg.ReportFolder;
}
```

It's possible to send the data with the event if you use a custom EventArg class

COMMON DIALOGS

WPF Common Dialogs

- Since Windows 3.1, the operating system itself has provided common dialogs for use by applications to keep a consistent look and feel
- **WPF** has three intrinsic dialogs that map to those provided by Windows:
  - OpenFileDialog,
  - SaveFileDialog,
  - PrintDialog

```
using Microsoft.Win32; // home of OpenFileDialog and SaveFileDialog
using System.Windows.Controls; // home of PrintDialog ...

string filename;
void openFileDialogButton_Click(...) {
  OpenFileDialog dlg = new OpenFileDialog( );
  dlg.FileName = filename;
  if (dlg.ShowDialog( ) == true) {
   filename = dlg.FileName; // open the file... } }
```

If you'd like access to other common Windows Forms dialogs, like the folder browser dialog, you can bring in the System.Windows.Forms assembly and use them without much trouble.