- **Programming with windows (WPF part 1)**
- Introduction to WPF
- WPF Controllers
    - Data Binding
    - Dependency Properties

WPF

- New desktop API since .NET 3.5
- Earlier many solutions were needed

Table 27-1. *Pre-WPF Solutions to Desired Functionalities*

Desired Functionality	Technology
Building windows with controls	Windows Forms
2D graphics support	GDI+ (System.Drawing.dll)
3D graphics support	DirectX APIs
Support for streaming video	Windows Media Player APIs
Support for flow-style documents	Programmatic manipulation of PDF files

- Now: WPF

Table 27-2. *.NET 3.0 Solutions to Desired Functionalities*

Desired Functionality	Technology
Building forms with controls	WPF
2D graphics support	WPF
3D graphics support	WPF
Support for streaming video	WPF
Support for flow-style documents	WPF

XAML (Extensible Application Markup Language)

- WPF uses XAML
  - Defines UI elements
  - Divide UI elements from logic
- XAML can be used by Non-UI applications and mobile apps

Note XAML is not limited to WPF applications. Any application can use XAML to describe a tree of .NET objects, even if they have nothing to do with a visible user interface. For example, the Windows Workflow Foundation API uses a XAML-based grammar to define business processes and custom activities. As well, other .NET GUI frameworks such as Silverlight, Windows Phone 7, and Windows 8 applications all make use of XAML.

GDI (Graphical Device Interface) is not used anymore

- All rendering is done using DirectX
- WPF uses hardware better

Advantages: WPF

- Layout manager
- Better data binding
- Styles and themes
- Vector graphics
- Support for 2D, 3D, Animations and video
- Support for XAML Paper Specification (XPS)
- Can still be used with other technologies (ex.: Windows Forms, ActiveX)

Different types of WPF applications

- Traditional desktops

Figure 27-1. *This WPF desktop application makes use of several WPF APIs*

- Transformation

Figure 27-2. Transformations and animations are very simple under WPF

- Navigation based (WEB like)
  - ”forward” and ”back” functionality

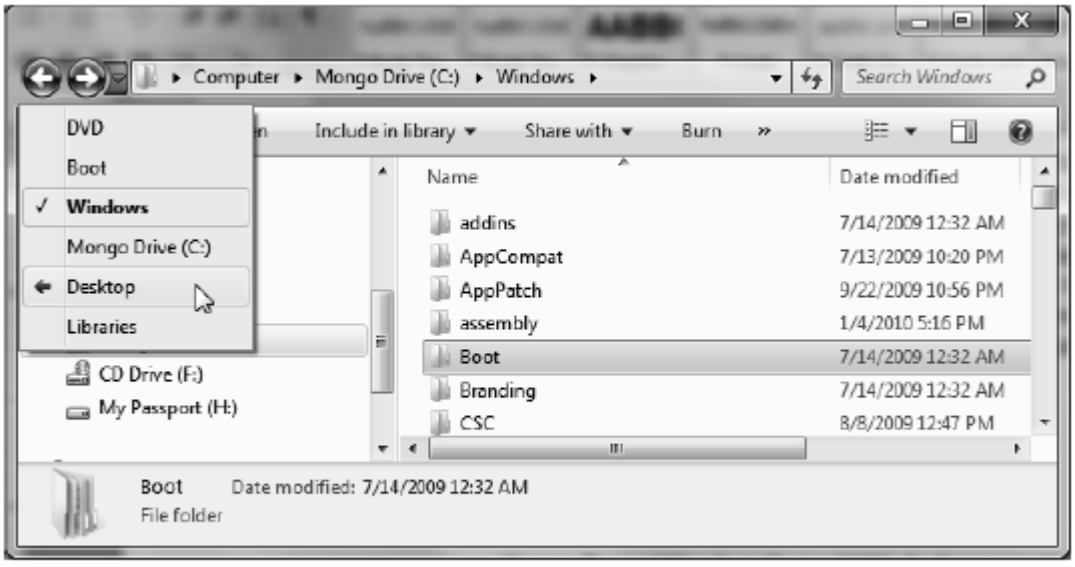


Figure 27-3. A navigation-based desktop program

- Use of WEB browser (XBAP applications)

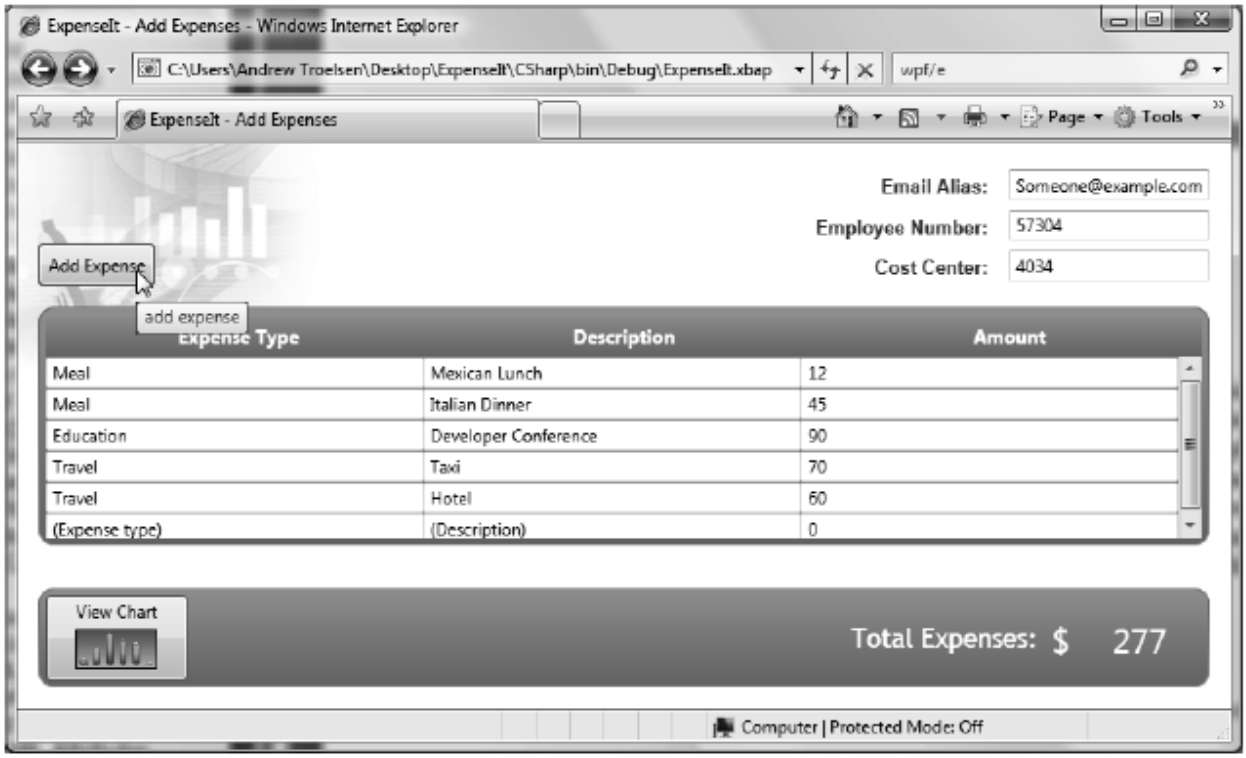


Figure 27-4. *XBAP programs are downloaded to a local machine and hosted within a web browser*

XBAP applications are running in a sandbox (”internet zone”)

WPF assemblies

Assembly	Meaning in Life
PresentationCore.dll	This assembly defines numerous namespaces that constitute the foundation of the WPF GUI layer. For example, this assembly contains support for the WPF Ink API (for programming against stylus input for Pocket PCs and Tablet PCs), animation primitives, and numerous graphical rendering types.
PresentationFramework.dll	This assembly contains a majority of the WPF controls, the Application and Window classes, support for interactive 2D graphics and numerous types used in data binding.
System.Xaml.dll	This assembly provides namespaces that allow you to program against a XAML document at runtime. By and large, this library is only useful if you are authoring WPF support tools or need absolute control over XAML at runtime.
WindowsBase.dll	This assembly defines types that constitute the infrastructure of the WPF API, including those representing WPF threading types, security types, various type converters, and support for *dependency properties* and *routed events* (described in Chapter 31).

WPF namespaces

Table 27-4. *Core WPF Namespaces*

Namespace	Meaning in Life
System.Windows	This is the root namespace of WPF. Here, you will find core classes (such as Application and Window) that are required by any WPF desktop project.
System.Windows.Controls	Contains all of the expected WPF widgets, including types to build menu systems, tool tips, and numerous layout managers.

System.Windows.Data	Contains types to work with the WPF data-binding engine, as well as support for data-binding templates.
System.Windows.Documents	Contains types to work with the documents API, which allows you to integrate PDF-style functionality into your WPF applications, via the XML Paper Specification (XPS) protocol.
System.Windows.Ink	Provides support for the Ink API, which allows yo to capture input from a stylus or mouse, respond to input gestures, and so forth. Very useful for Tablet PC programming; however, any WPF can make use of this API.
System.Windows.Markup	This namespace defines a number of types that allow XAML markup (and the equivalent binary format, BAML) to be parsed and processed programmatically.
System.Windows.Media	This is the root namespace to several media-centric namespaces. Within these namespaces you will find types to work with animations, 3D rendering, text rendering, and other multimedia primitives.
System.Windows.Navigation	This namespace provides types to account for the navigation logic employed by XAML browser applications (XBAPs) as well as standard desktop applications that require a navigational page model.
System.Windows.Shapes	Defines classes that allow you to render interactive 2D graphics that automatically respond to mouse input.

Application class

- An instance of a WPF application

```
// Define the global application object
// for this WPF program.
class MyApp : Application
{
[STAThread]
static void Main(string[] args)
{
// Create the application object.
MyApp app = new MyApp();
// Register the Startup/Exit events.
app.Startup += (s, e) => { /* Start up the app */ };
```

```
app.Exit += (s, e) => { /* Exit the app */ };
}
}
```

Properties in Application class

Table 27-5. *Key Properties of the Application Type*

Property	Meaning in Life
Current	This static property allows you to gain access to the running **Application** object from anywhere in your code. This can be very helpful when a window or dialog box needs to gain access to the **Application** object that created it, typically to access application-wide variables and functionality.
MainWindow	This property allows you to programmatically get or set the main window of the application.
Properties	This property allows you to establish and obtain data that is accessible throughout all aspects of a WPF application (windows, dialog boxes, etc.).
StartupUri	This property gets or sets a URI that specifies a window or page to open automatically when the application starts.
Windows	This property returns a **WindowCollection** type, which provides access to each window created from the thread that created the **Application** object. This can be very helpful when you want to iterate over each open window of an application and alter its state (such as minimizing all windows).

The System.Windows.Window class

- Represents a window with dialog boxes etc.

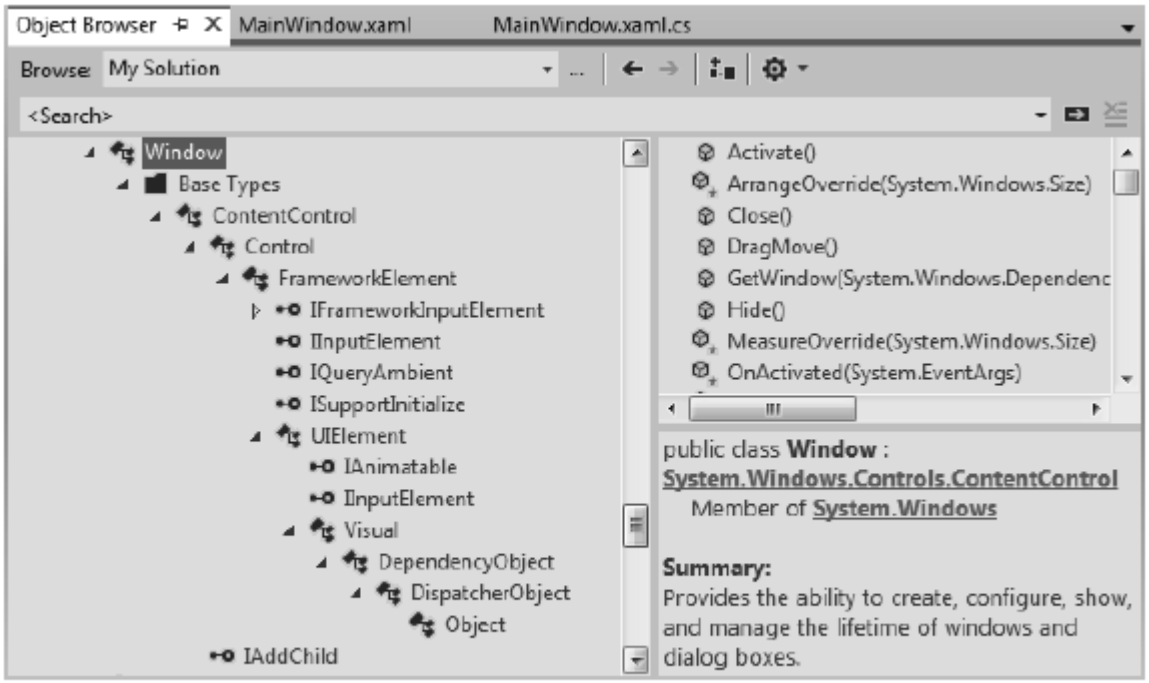


Figure 27-5. The hierarchy of the Window class

The System.Windows.Controls.ContentControl class

- A ContentControl contains a number of UI elements
- XAML example
  - (a <Button> contains a <StackPanel> with an <Ellipse>)

```
<!-- Implicitly setting the Content property with complex data -->
<Button Height="80" Width="100">
<StackPanel>
<Ellipse Fill="Red" Width="25" Height="25"/>
<Label Content ="OK!"/>
</StackPanel>
</Button
```

The System.Windows.Controls.Control class

- All WPF controls share a common Control base class
- Defines properties of a control

Table 27-6. Key Members of the Control Type

Members	Meaning in Life
Background, Foreground, BorderBrush, BorderThickness, Padding, HorizontalContentAlignment, VerticalContentAlignment	These properties allow you to set basic settings regarding how the control will be rendered and positioned.
FontFamily, FontSize, FontStretch, FontWeight	These properties control various font-centric settings.
IsTabStop, TabIndex	These properties are used to establish tab order among controls on a window.
MouseDoubleClick, PreviewMouseDoubleClick	These events handle the act of double-clicking a widget.
Template	This property allows you to get and set the control's template, which can be used to change the rendering output of the widget.

The System.Windows.FrameworkElement class

- Low-level work
  - ex. story boards (animations)

Table 27-7. Key Members of the FrameworkElement Type

Members	Meaning in Life
ActualHeight, ActualWidth, MaxHeight, MaxWidth, MinHeight, MinWidth, Height, Width	These properties control the size of the derived type.
ContextMenu	Gets or sets the pop-up menu associated with the derived type.
Cursor	Gets or sets the mouse cursor associated with the derived type.
HorizontalAlignment, VerticalAlignment	Gets or sets how the type is positioned within a container (such as a panel or list box).
Name	Allows to you assign a name to the type, in order to access its functionality in a code file.
Resources	Provides access to any resources defined by the type (see Chapter 30 for an examination of the WPF resource system).
ToolTip	Gets or sets the tool tip associated with the derived type.

System.Windows.UIElement

- Connects a component with an event

Members	Meaning in Life
Focusable, IsFocused	These properties allow you to set focus on a given derived type.
IsEnabled	This property allows you to control whether a given derived type is enabled or disabled.
IsMouseDirectlyOver, IsMouseOver	These properties provide a simple way to perform hit-testing logic.
IsVisible, Visibility	These properties allow you to work with the visibility setting of a derived type.
RenderTransform	This property allows you to establish a transformation that will be used to render the derived type.

Dispatcher object

- System.Windows.Threading.DispatcherObject
  - Entry point for event handling
  - Thread handling model

Dependency object

- Gives extra functionality to properties (for instance Binding, styles, animation)

KaXaml

- XAML editor/parser

XAML keywords

Table 24-9. *XAML Keywords*

XAML Keyword	Meaning in Life
`x:Array`	Represents a .NET array type in XAML.
`x:ClassModifier`	Allows you to define the visibility of the C# class (internal or public) denoted by the `Class` keyword.
`x:FieldModifier`	Allows you to define the visibility of a type member (internal, public, private, or protected) for any named subelement of the root (e.g., a `<Button>` within a `<Window>` element). A *named element* is defined using the `Name` XAML keyword.
`x:Key`	Allows you to establish a key value for a XAML item that will be placed into a dictionary element.
`x:Name`	Allows you to specify the generated C# name of a given XAML element.
`x:Null`	Represents a `null` reference.
`x:Static`	Allows you to make reference to a static member of a type.
`x:Type`	The XAML equivalent of the C# `typeof` operator (it will yield a `System.Type` based on the supplied name).
`x:TypeArguments`	Allows you to establish an element as a generic type with a specific type parameter (e.g., `List<int>` vs. `List<bool>`).

Building WPF applications without XAML

1. Inheritance from Application class

- See example app without XAML

```
// A simple WPF application, written without XAML.
using System;
using System.Windows;
using System.Windows.Controls;
namespace WpfAppAllCode
{
// In this first example, you are defining a single class type to
// represent the application itself and the main window.
class Program : Application
{
[STAThread]
static void Main(string[] args)
{
// Handle the Startup and Exit events, and then run the application.
Program app = new Program();
app.Startup += AppStartUp;
app.Exit += AppExit;
app.Run(); // Fires the Startup event.
}
static void AppExit(object sender, ExitEventArgs e)
{
MessageBox.Show("App has exited");
}
static void AppStartUp(object sender, StartupEventArgs e)
{
// Create a Window object and set some basic properties.
Window mainWindow = new Window();
mainWindow.Title = "My First WPF App!";
mainWindow.Height = 200;
mainWindow.Width = 300;
mainWindow.WindowStartupLocation = WindowStartupLocation.CenterScreen;
mainWindow.Show();
}
}
}
```

- Notice the use of events
- AppStartUp creates the window (Window object)
- Inherits from Application

Notice: A WPF application must use the attribute [STAThread] on Main

2. Inheritance from Window class

- Another way: inherit the Window class

Example:

```
class MainWindow : Window
{
public MainWindow(string windowTitle, int height, int width)
{
this.Title = windowTitle;
this.WindowStartupLocation = WindowStartupLocation.CenterScreen;
this.Height = height;
this.Width = width;
}
}
```

```
static void AppStartUp(object sender, StartupEventArgs e)
{
// Create a MainWindow object.
MainWindow wnd = new MainWindow("My better WPF App!", 200, 300);
wnd.Show();
}
```

The use of UI elements

1. Define a member variable to represent the control
2. Configuration of this control
3. Set up Content

- Example
  - Notice: this.Content

```
class MainWindow : Window
{
// Our UI element.
private Button btnExitApp = new Button();
public MainWindow(string windowTitle, int height, int width)
{
// Configure button and set the child control.
btnExitApp.Click += new RoutedEventHandler(btnExitApp_Clicked);
btnExitApp.Content = "Exit Application";
btnExitApp.Height = 25;
btnExitApp.Width = 100;
// Set the content of this window to a single button.
this.Content = btnExitApp;
// Configure the window.
this.Title = windowTitle;
this.WindowStartupLocation = WindowStartupLocation.CenterScreen;
this.Height = height;
this.Width = width;
this.Show();
}
private void btnExitApp_Clicked(object sender, RoutedEventArgs e)
{
// Close the window.
this.Close();
}
}
```

Application data

- Command line args: flag GODMODE

```
private static void AppStartUp(object sender, StartupEventArgs e)
{
// Check the incoming command-line arguments and see if they
// specified a flag for /GODMODE.
Application.Current.Properties["GodMode"] = false;
foreach(string arg in e.Args)
{
if (arg.ToLower() == "/godmode")
{
Application.Current.Properties["GodMode"] = true;
break;
```

```
}
}
```

- Closing

```
private void btnExitApp_Clicked(object sender, RoutedEventArgs e)
{
// Did user enable /godmode?
if((bool)Application.Current.Properties["GodMode"])
MessageBox.Show("Cheater!");
this.Close();
}
```

Mouse / keyboard events

Mouse events

```
{
...
public Point GetPosition(IInputElement relativeTo);
public MouseButtonState LeftButton { get; }
public MouseButtonState MiddleButton { get; }
public MouseDevice MouseDevice { get; }
public MouseButtonState RightButton { get; }
public StylusDevice StylusDevice { get; }
public MouseButtonState XButton1 { get; }
public MouseButtonState XButton2 { get; }
}
```

- Mouse event example

```
public MainWindow(string windowTitle, int height, int width)
{
...
this.MouseMove += MainWindow_MouseMove;
}
…
/*
Here is an event handler for MouseMove that will display the location of
the mouse in the window's title area (notice you are translating the returned
Point type into a text value via ToString()):
```

```
*/
private void MainWindow_MouseMove(object sender,
System.Windows.Input.MouseEventArgs e)
{
// Set the title of the window to the current (x,y) of the mouse.
this.Title = e.GetPosition(this).ToString();
}
```

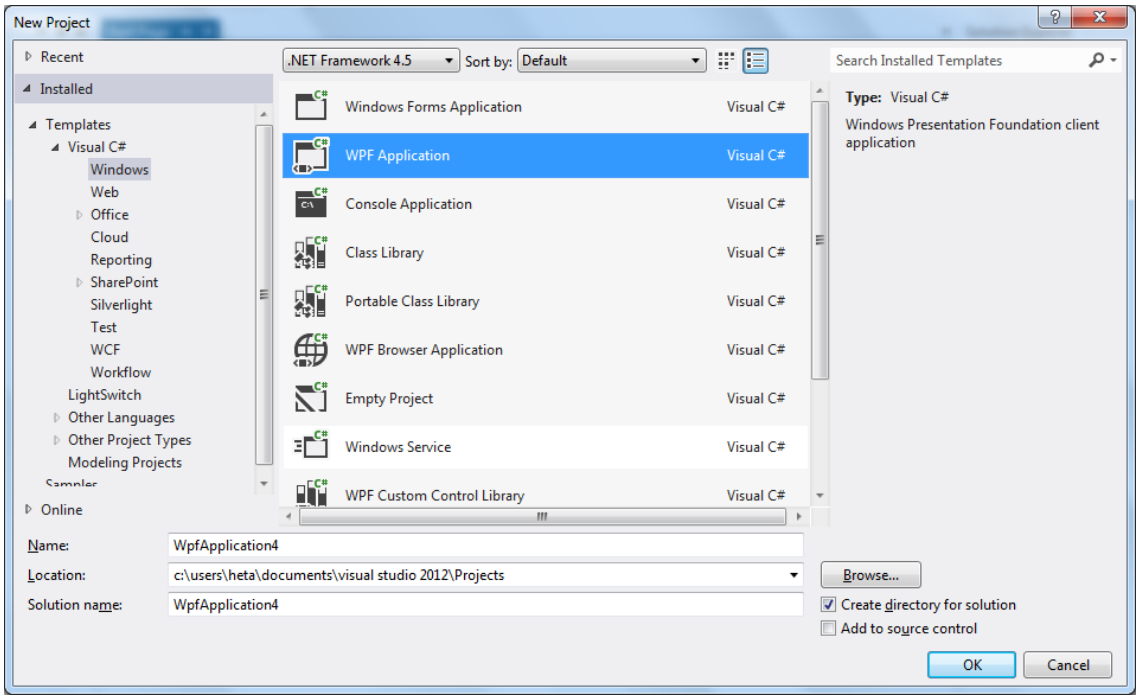
Keyboard events

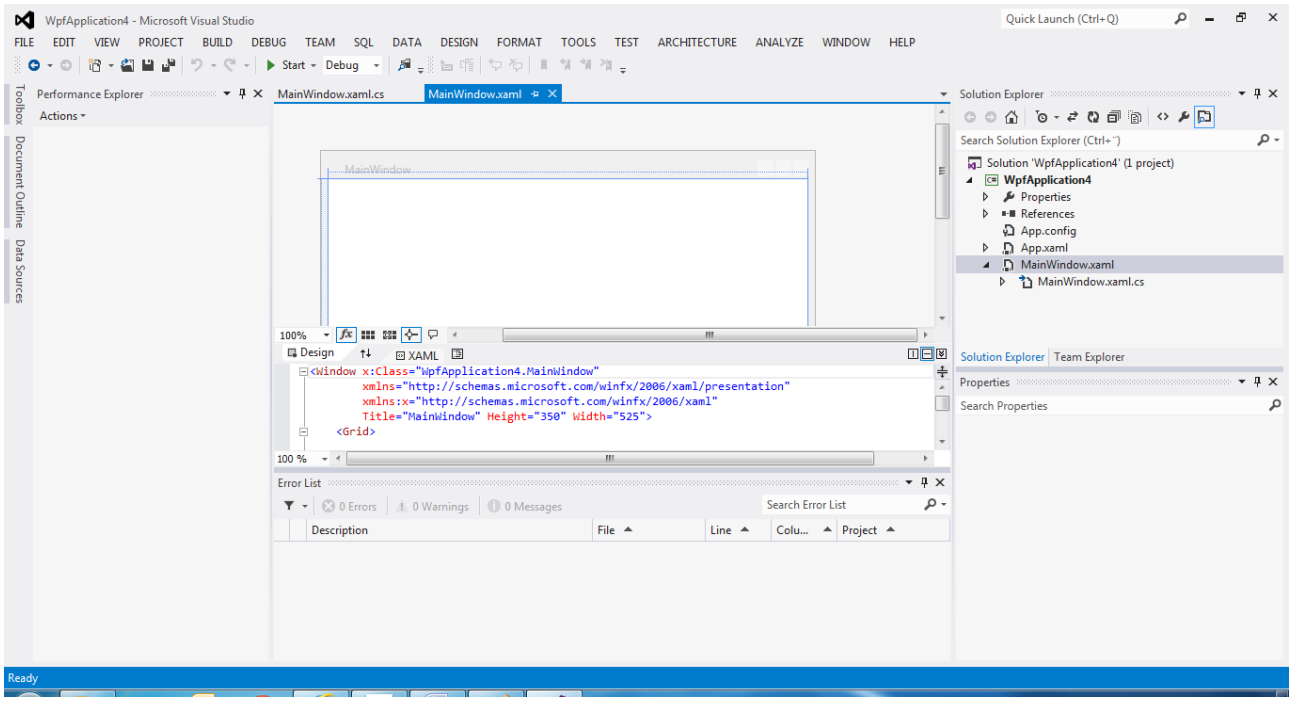
```
{
...
public bool IsDown { get; }
public bool IsRepeat { get; }
public bool IsToggled { get; }
public bool IsUp { get; }
public Key Key { get; }
public KeyStates KeyStates { get; }
public Key SystemKey { get; }
}
```

- Keyboard event example

```
{
// Display key press on the button.
btnExitApp.Content = e.Key.ToString();
}
```

Building WPF applications WITH XAML – practical exmple

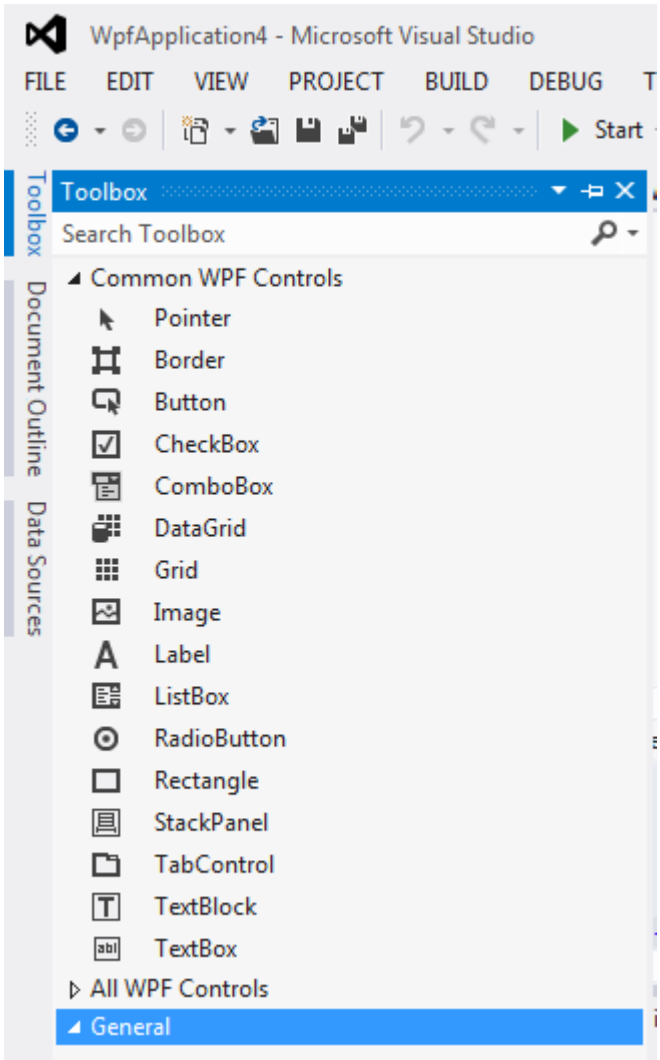

New Project
Recent
Installed
Templates
Visual C#
Windows
Web
Office
Cloud
Reporting
SharePoint
Silverlight
Test
WCF
Workflow
LightSwitch
Other Languages
Other Project Types
Modeling Projects
Online
.NET Framework 4.5
Sort by: Default
Search Installed Templates
Windows Forms Application Visual C#
WPF Application Visual C#
Console Application Visual C#
Class Library Visual C#
Portable Class Library Visual C#
WPF Browser Application Visual C#
Empty Project Visual C#
Windows Service Visual C#
WPF Custom Control Library Visual C#
Type: Visual C#
Windows Presentation Foundation client application
Name: WpfApplication4
Location: c:\users\heta\documents\visual studio 2012\Projects
Solution name: WpfApplication4
Browse...
Create directory for solution
Add to source control
OK
Cancel

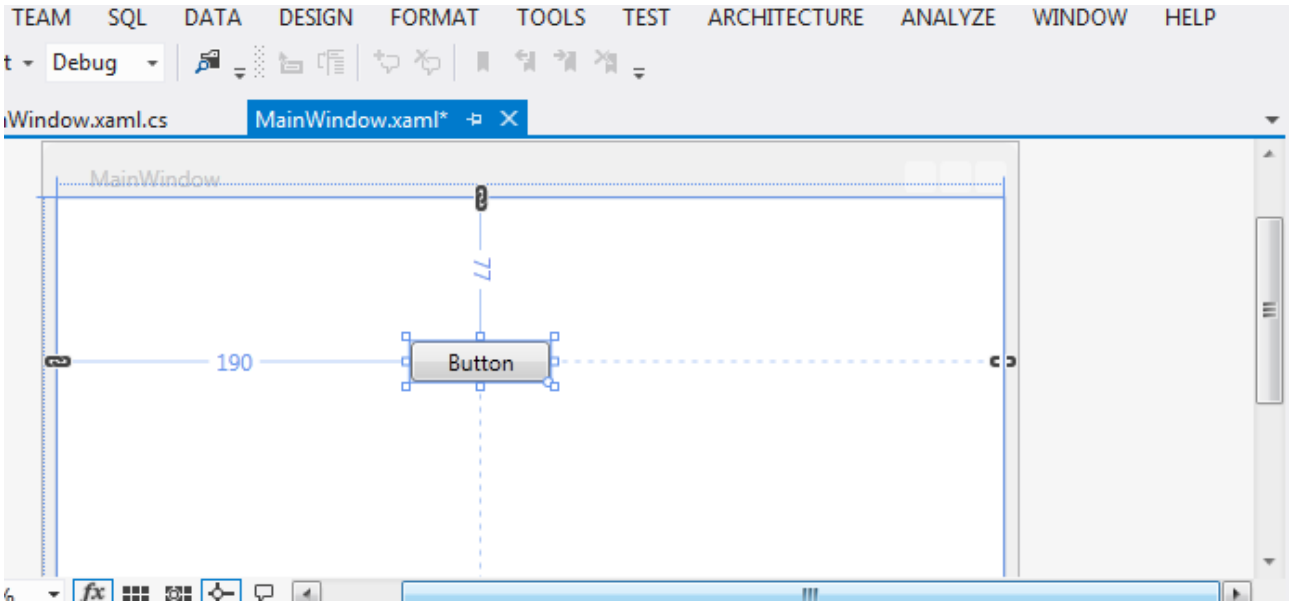

WpfApplication4 - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM SQL DATA DESIGN FORMAT TOOLS TEST ARCHITECTURE ANALYZE WINDOW HELP
Start Debug
Performance Explorer
Actions
MainWindow.xaml.cs
MainWindow.xaml
Design XAML
<Window x:Class="WpfApplication4.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="MainWindow" Height="350" Width="525">
<Grid>
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'WpfApplication4' (1 project)
WpfApplication4
Properties
References
App.config
App.xaml
MainWindow.xaml
MainWindow.xaml.cs
Solution Explorer Team Explorer
Properties
Search Properties
Error List
0 Errors 0 Warnings 0 Messages
Search Error List
Description File Line Colu... Project
Ready

XAML

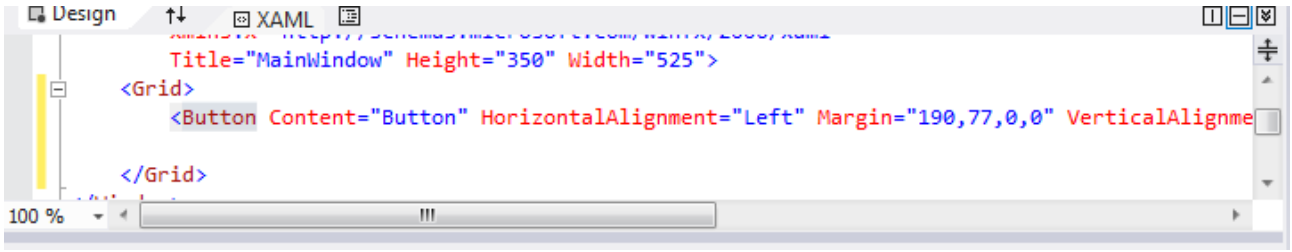
```
<Window x:Class="WpfApplication4.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">
    <Grid>

    </Grid>
</Window>
```

Normal use of ToolBox

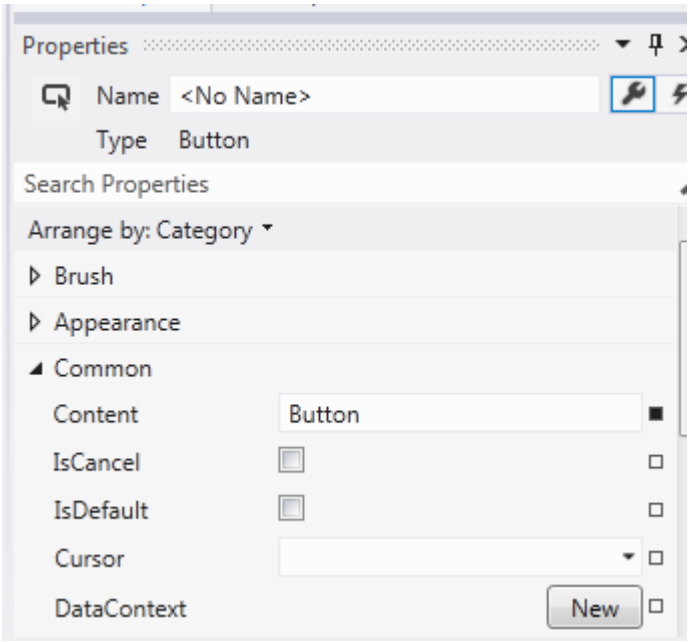




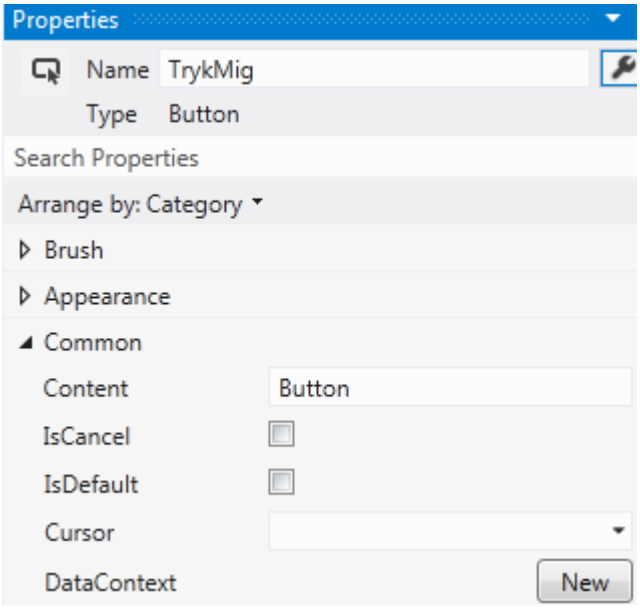
XAML is automatically updated:



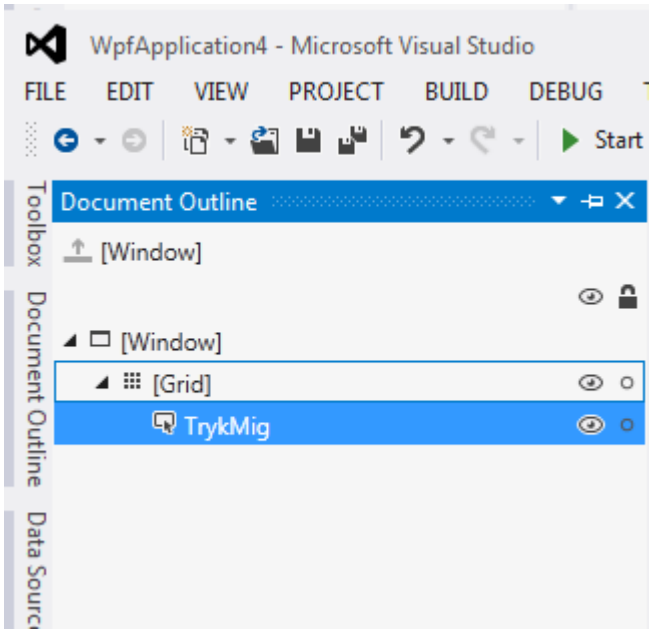
Properties (Here for a button)



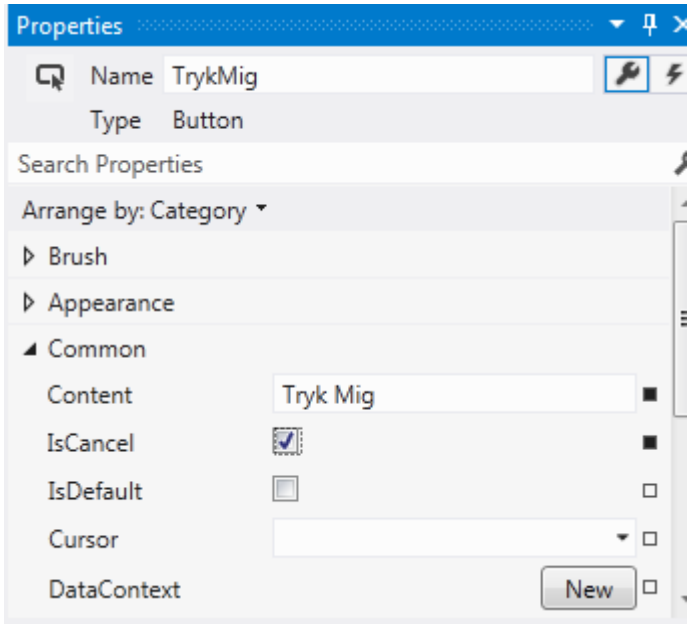
Control Name and Content

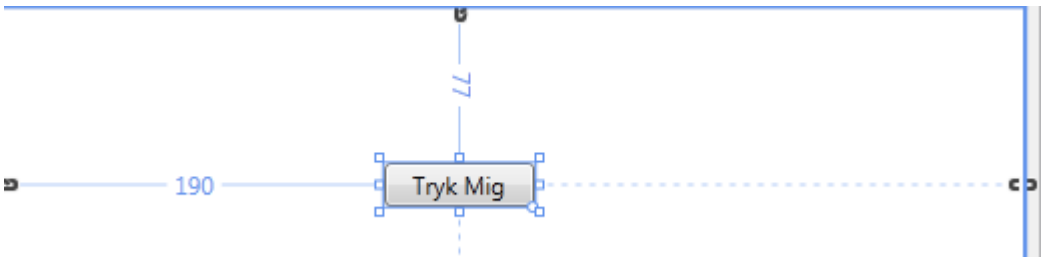


Outline (design)



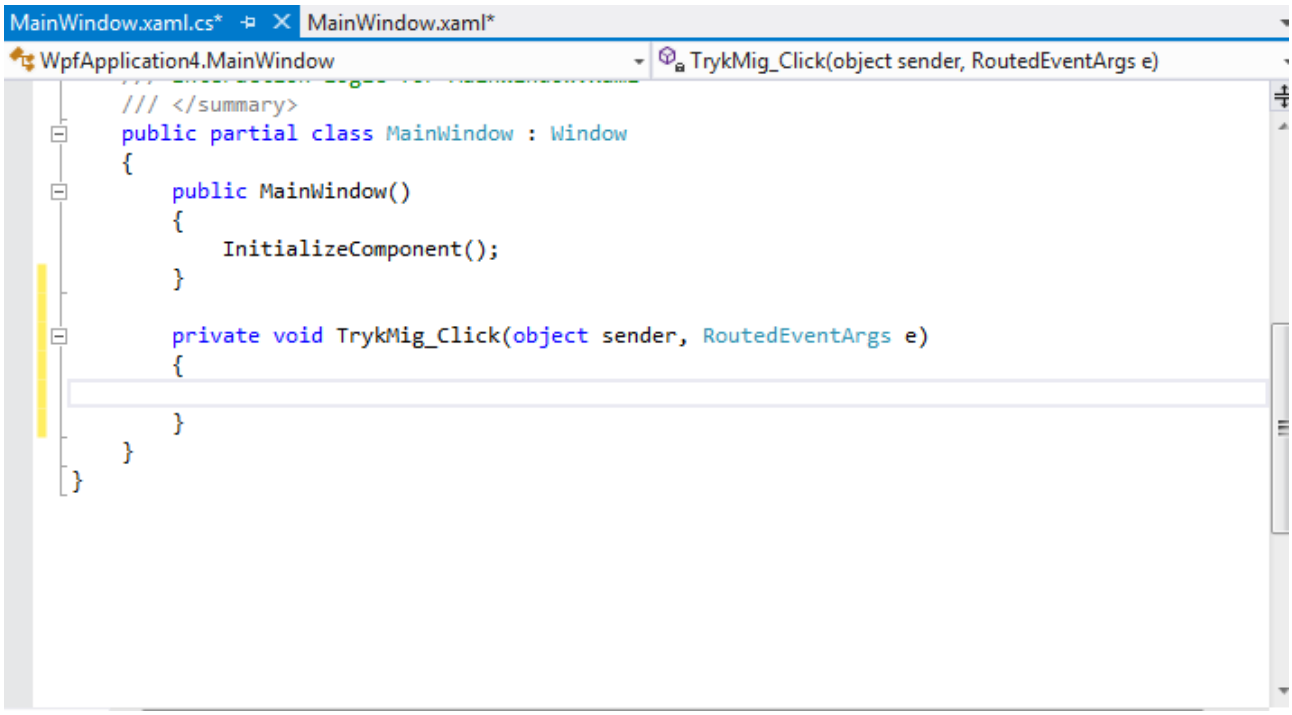
- Text is now Content





Adding an event to a button

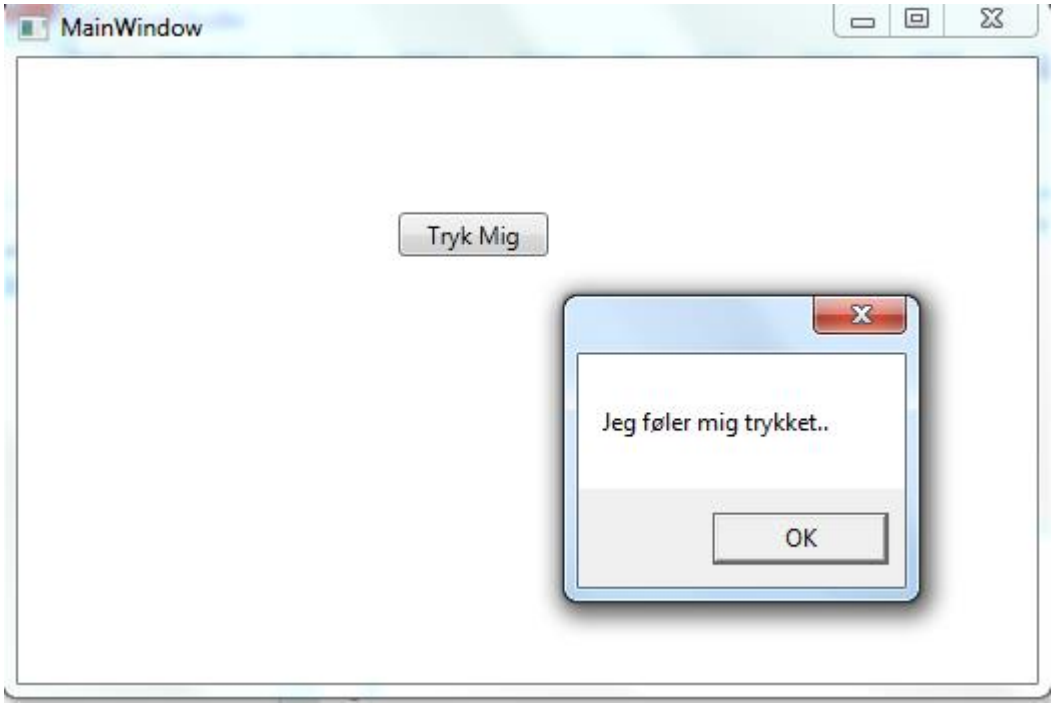
- Still: just Double-click on the control in the designer
- Jumps to fill out the event in the .cs file




```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace WpfApplication4
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
```

```
            InitializeComponent();
        }

        private void TrykMig_Click(object sender, RoutedEventArgs e)
        {
            MessageBox.Show("Jeg føler mig trykket..");
        }
    }
}
```

Output



Files in a WPF application

- XAML (look above)
- App.xaml.cs
- Window1.xaml.cs

App.xaml.cs

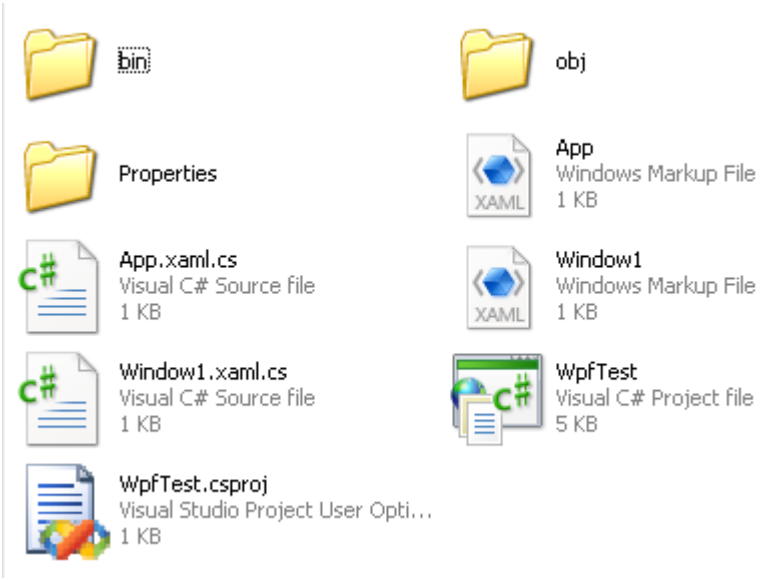
```
using System;
using System.Collections.Generic;
```

```
using System.Configuration;
using System.Data;
using System.Linq;
using System.Windows;

namespace WpfTest
{
    /// <summary>
    /// Interaction logic for App.xaml
    /// </summary>
    public partial class App : Application
    {
    }
}
```

Window1.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace WpfTest
{
    /// <summary>
    /// Interaction logic for Window1.xaml
    /// </summary>
    public partial class Window1 : Window
    {
        public Window1()
        {
            InitializeComponent();
        }

        private void TrykMig_Click(object sender, RoutedEventArgs e)
        {
            MessageBox.Show("Jeg føler mig trykket");
        }
    }
}
```

Files



- Notice the XAML files

Compiling –1

- C# compiler cannot be used directly
- MSBUILD transforms XAML to C# code
  - reads the .csproj file generated by Visual Studio
  - From the prompt msbuild.exe can be called directly

Mapping of XAML to C# code

- Produces some *.g.cs files (g = autogenerated)

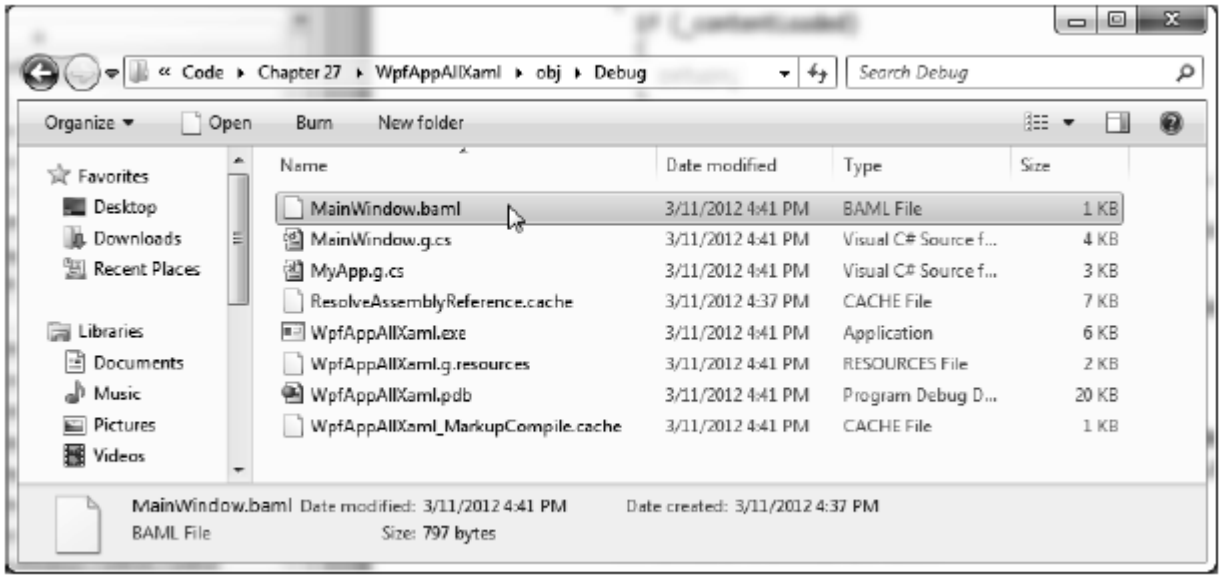
```
...
namespace WpfTest {


    /// <summary>
    /// MainWindow
    /// </summary>
    [System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]
    public partial class MainWindow : System.Windows.Window,
System.Windows.Markup.IComponentConnector {


        #line 6 "..\..\..\MainWindow.xaml"
```

```
        [System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1823:AvoidUnusedPrivateFields")]
        internal System.Windows.Controls.Button button1;

        #line default
...
```

(part of g.cs fil)

About msbuild.exe

- Generates a .baml file (Binary Application Markup Language)

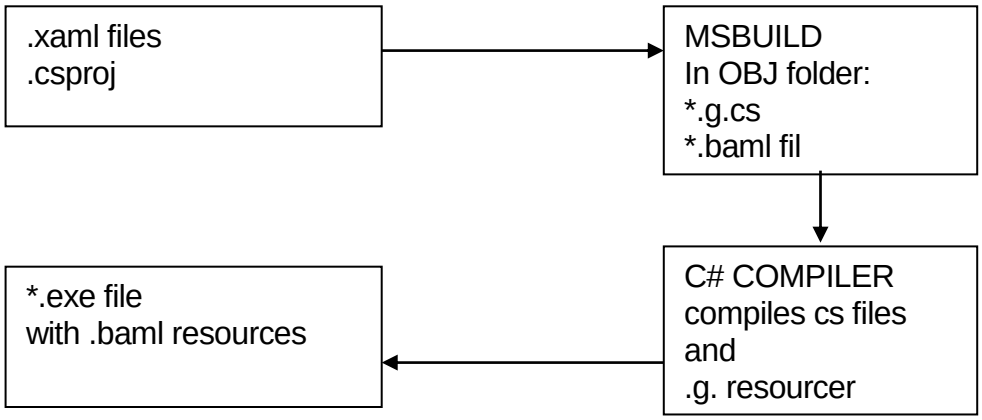


Mapping XAML to C# code

```
public partial class MyApp : System.Windows.Application
{
void AppExit(object sender, ExitEventArgs e)
{
MessageBox.Show("App has exited");
}
[System.Diagnostics.DebuggerNonUserCodeAttribute()]
public void InitializeComponent()
{
this.Exit += new System.Windows.ExitEventHandler(this.AppExit);
this.StartupUri = new System.Uri("MainWindow.xaml", System.UriKind.Relative);
}
[System.STAThreadAttribute()]
[System.Diagnostics.DebuggerNonUserCodeAttribute()]
```

```
public static void Main() {
SimpleXamlApp.MyApp app = new SimpleXamlApp.MyApp();
app.InitializeComponent();
app.Run();
}
}
}
```

Compiling



BAML (Binary Application Markup Language)

- Binary representation of XAML
- Becomes part of an assembly as a resource

WPF XAML syntax

XAML elements and properties

- Root elements can have child elements
    - A control can be expanded

```
…
<Button Height="50" Width="100" Content="OK!"
FontSize="20" Foreground="Yellow">
<Button.Background>
<LinearGradientBrush>
<GradientStop Color="DarkGreen" Offset="0"/>
<GradientStop Color="LightGreen" Offset="1"/>
</LinearGradientBrush>
</Button.Background>
</Button>
…
```

- Pattern

```
<DefiningClass>
<DefiningClass.PropertyOnDefiningClass>
<!-- Value for Property here! -->
</DefiningClass.PropertyOnDefiningClass>
</DefiningClass>
```

XAML Attached Properties

- Child elements can specify unique values in the parent element
- Has the following form

```
<ParentElement>
 <ChildElement ParentElement.PropertyOnParent = ”value” />
</ParentElement>
```

Example:

```
<Page
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
<Canvas Height="200" Width="200" Background="LightBlue">
<Ellipse Canvas.Top="40" Canvas.Left="40" Height="20" Width="20" Fill="DarkBlue"/>
</Canvas>
</Page>
```

- Canvas has an Eliipse
- Ellipse can inform Canvas about top/left

Markup Extensions

```
<Element PropertyToSet = "{MarkUpExtension}"/>
```

Understanding XAML Markup Extensions

As explained, property values are most often represented using a simple string or via property-element syntax. There is, however, another way to specify the value of a XAML attribute, using *markup extensions*. Markup extensions allow a XAML parser to obtain the value for a property from a dedicated, external class. This can be beneficial given that some property values require a number of code statements to execute to figure out the value.

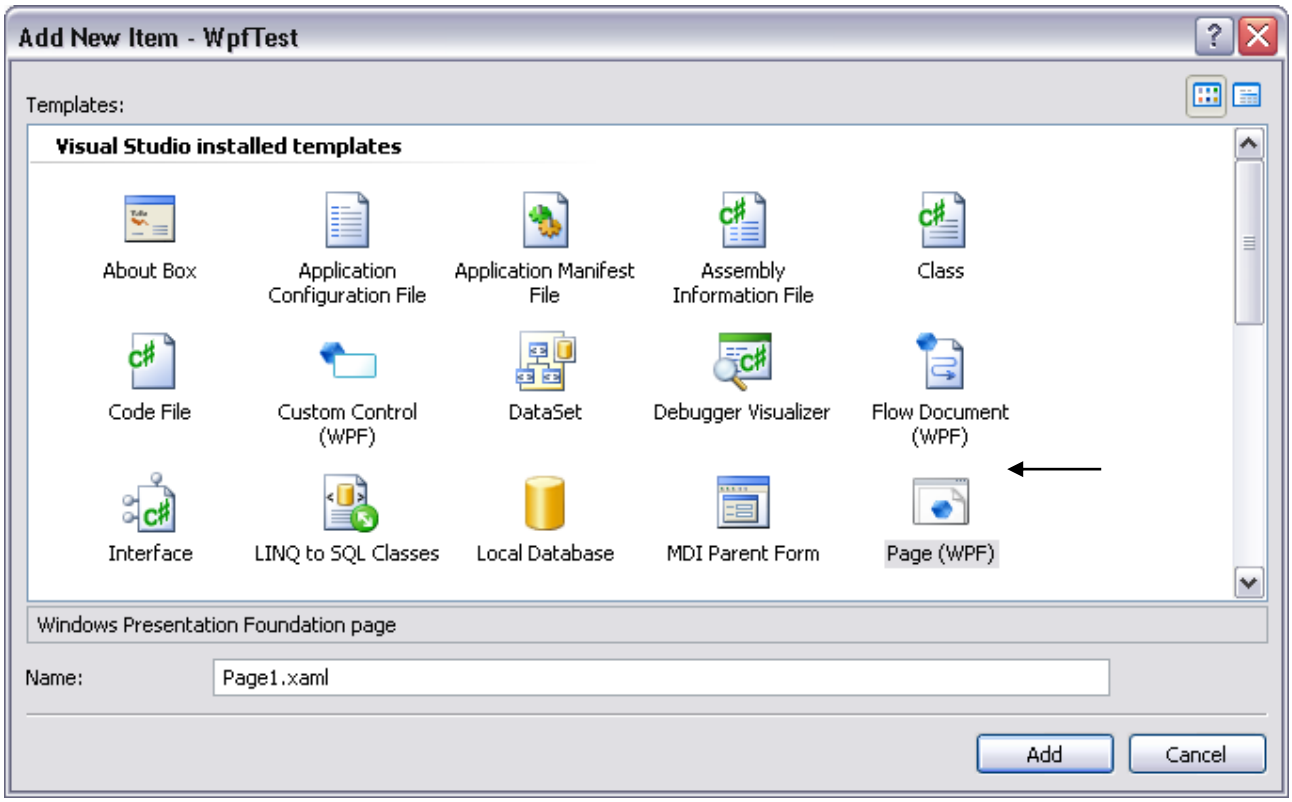
Markup extensions provide a way to cleanly extend the grammar of XAML with new functionality. A markup extension is represented internally as a class that derives from `MarkupExtension`. Note that the chances of you ever needing to build a custom markup extension will be slim to none. However, a subset of XAML keywords (such as `x:Array`, `x:Null`, `x:Static`, and `x:Type`) are markup extensions in disguise!

A markup extension is sandwiched between curly brackets, like so:

```
<Element PropertyToSet = "{MarkUpExtension}"/>
```

Static Databinding: A simple example

- Insert a new Page (Use ”Add New Item”)



Content in page1.xaml (here an array is created)

```
<Page x:Class="WpfTest.Page1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:Basic="clr-namespace:System;assembly=mscorlib"

      Title="Page1">
    <x:Array Type="Basic:String" x:Name="Andeby">

        <Basic:String>Rip</Basic:String>
        <Basic:String>Rap</Basic:String>
        <Basic:String>Rup</Basic:String>
    </x:Array>

</Page>
```

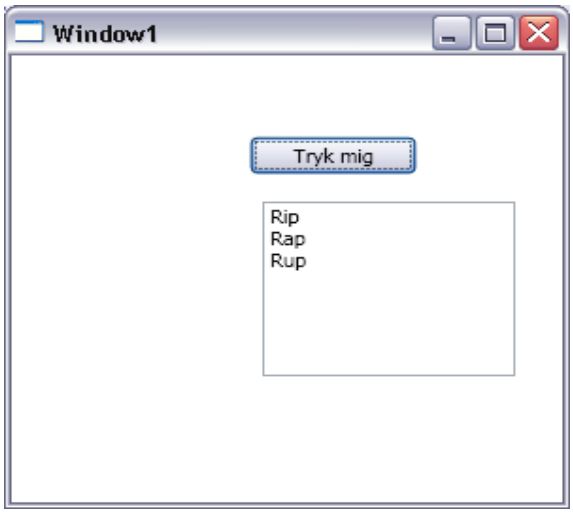
- Notice the reference to the System namespace

Inserting a ListBox in the UI

The functionality is changed to obtain data from the page...

```
...

    private void TrykMig_Click(object sender, RoutedEventArgs e)
        {
            Page1 page1 = new Page1();
            int antal = page1.Andeby.Items.Count;
            for( int i = 0; i<antal;i++)
                  listBox1.Items.Add(page1.Andeby.Items[i]);

        }
...
```

Output



Handling events (property window)

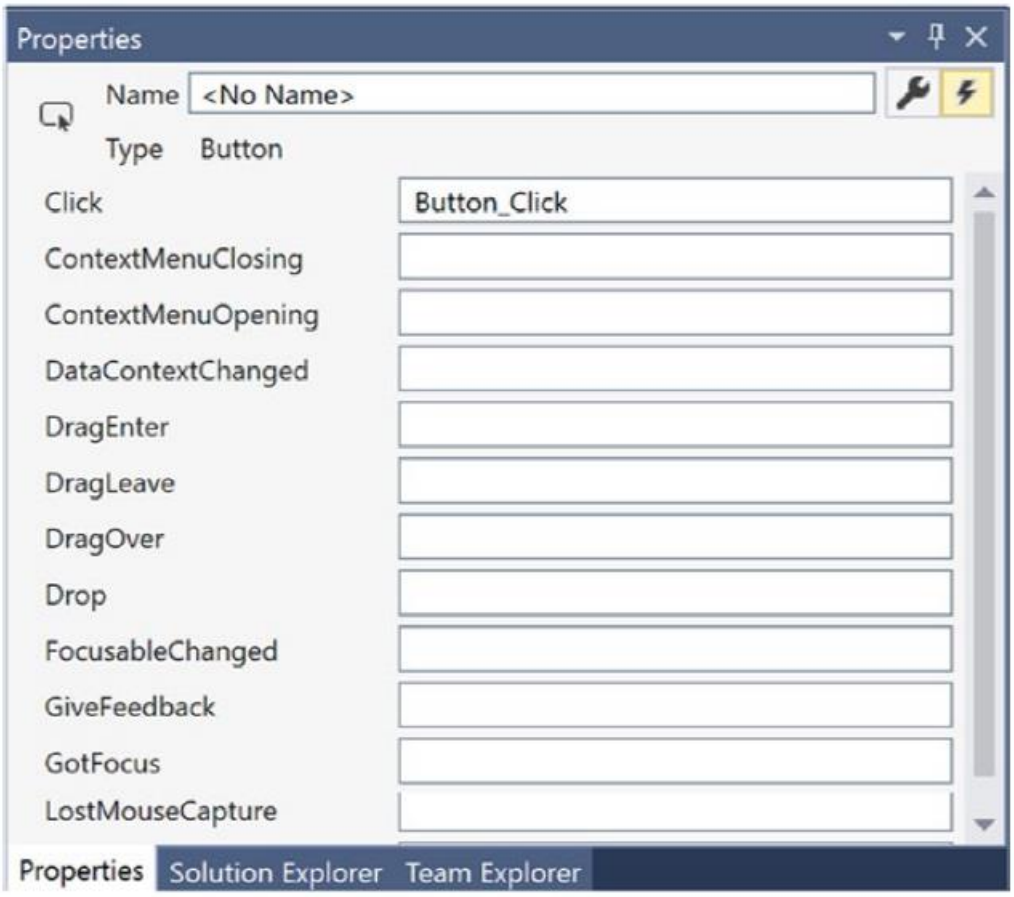
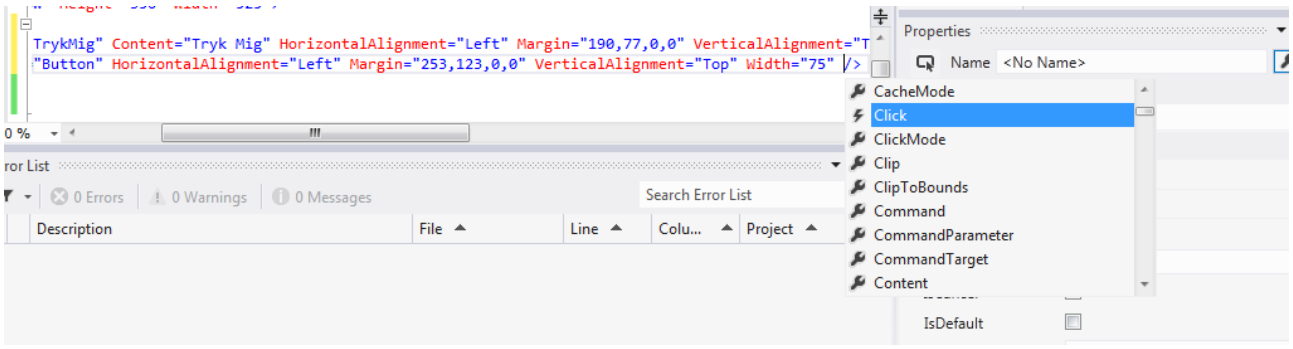
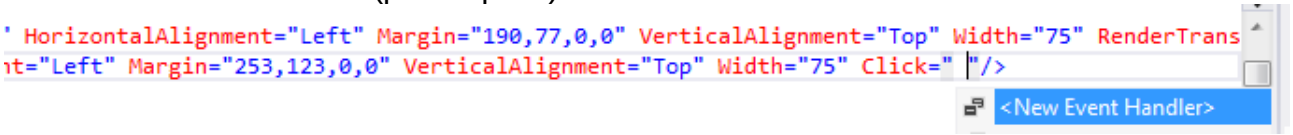


Figure 24-8. *Handling events using the Properties window*

Handling events via XAML editor



- Click Event
- New Event Handler (press space)



```
...
  private void Button_Click_1(object sender, RoutedEventArgs e)
  {

  }
...
```

Visualizing XAML

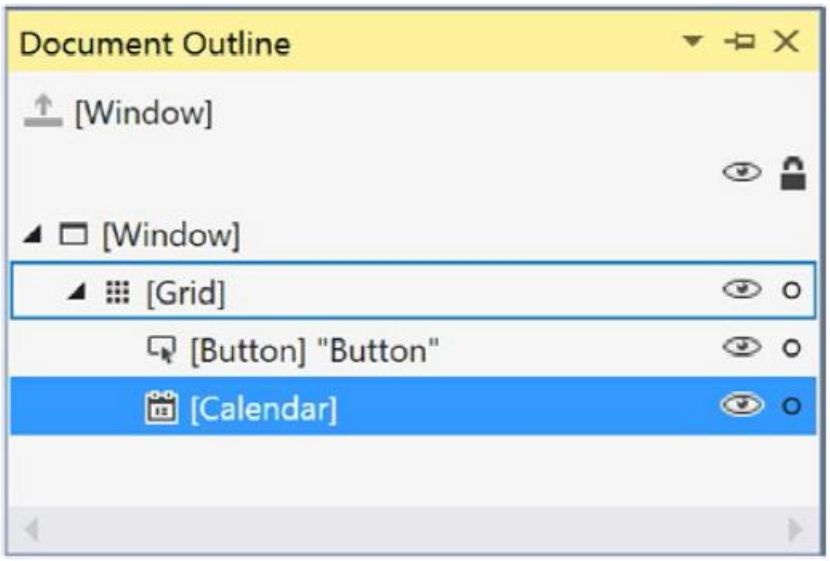


Figure 24-10. *Visualizing your XAML via the Document Outline window*

App.xaml

Examining the App.xaml File

How did the project know what window to launch? Even more intriguing, if you examine the code files in your application, you will also see that there isn't a `Main()` method anywhere to be found. You've learned throughout this book that applications must have an entry point, so how does .NET know how to launch your app? Fortunately, both of these plumbing items are handled for you through the Visual Studio templates and the WPF framework.

To solve the riddle of which window to launch, the `App.xaml` file defines an application class through markup. In addition to the namespace definitions, it defines application properties such as the `StartupUri`, application-wide resources (covered in Chapter 27), and specific handlers for application events such as `Startup` and `Exit`. The `StartupUri` indicates which window to load on startup. Open the `App.xaml` file and examine the markup, shown here:

```
<Application x:Class="WpfTesterApp.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:local="clr-namespace:WpfTesterApp"
             StartupUri="MainWindow.xaml">
    <Application.Resources>
    </Application.Resources>
</Application>
```

Mapping the Window XAML Markup to C# Code

When `msbuild.exe` processed your `*.csproj` file, it produced three files for each XAML file in your project with the form of `*.g.cs` (where `g` denotes auto*generated*), `*.g.i.cs` (where *i* denotes IntelliSense), and `*.baml` (for Binary Application Markup Language). These are saved into the `\obj\Debug` directory (and can be viewed in Solution Explorer by clicking the Show All Files button). You might have to hit the Refresh button in Solution Explorer to see them since they are not part of the actual project, but build artifacts.

To make the most sense of the process, it's helpful to provide names for your controls. Go ahead and provide names for the `Button` and `Calendar` controls, as follows:

```
<Button Name="ClickMe" Content="Button" HorizontalAlignment="Left" Margin="10,10,0,0"
       VerticalAlignment="Top" Width="75" Click="Button_Click">
//omitted for brevity
</Button>
<Calendar Name="MyCalendar" HorizontalAlignment="Left" Margin="10,41,0,0"
VerticalAlignment="Top"/>
```

WPF Controls, layouts, events and data binding

- Controllers
- Containers

WPF Control Library

- See table 25-1 page 1003-1004
  - Controls
    - Buttons
    - TextBox
    - ScrollBar
    - Sliders
    - Much more.
  - Frame Controllers
    - Menu
    - StatusBar
    - Much more
  - Media Controls
    - SoundPlayers
  - Layout controller
    - Border

    - canvas
    - DockPanel
    - Grid
    - GridView

- Dialog boxes

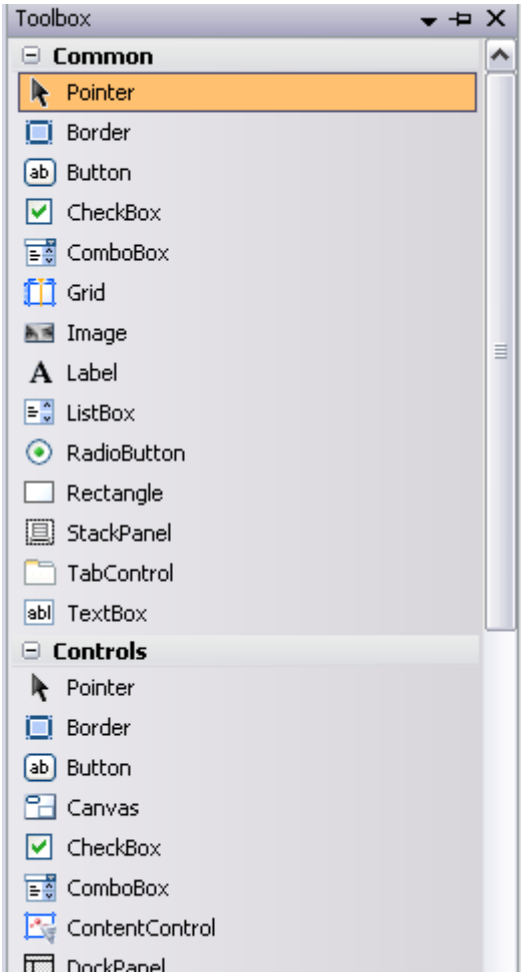
- Overview

Table 28-1. *The Core WPF Controls*

WPF Control Category	Example Members	Meaning in Life
Core user input controls	Button, RadioButton, ComboBox, CheckBox, Calendar, DatePicker, Expander, DataGrid, ListBox, ListView, ToggleButton, TreeView, ContextMenu, ScrollBar, Slider, TabControl, TextBlock, TextBox, RepeatButton, RichTextBox, Label	WPF provides an entire family of controls you can use to build the crux of a user interface.
Window and control adornments	Menu, ToolBar, StatusBar, ToolTip, ProgressBar	You use these UI elements to decorate the frame of a Window object with input devices (such as the Menu) and user informational elements (e.g., StatusBar and ToolTip).
Media controls	Image, MediaElement, SoundPlayerAction	These controls provide support for audio/video playback and image display.
Layout controls	Border, Canvas, DockPanel, Grid, GridView, GridSplitter, GroupBox, Panel, TabControl, StackPanel, Viewbox, WrapPanel	WPF provides numerous controls that allow you to group and organize other controls for the purpose of layout management.

- Since 4.0
    - Ink Controls (stylus)
    - Document Control (PDF style functionality)

Samples from the ToolBox



Grid (XAML)

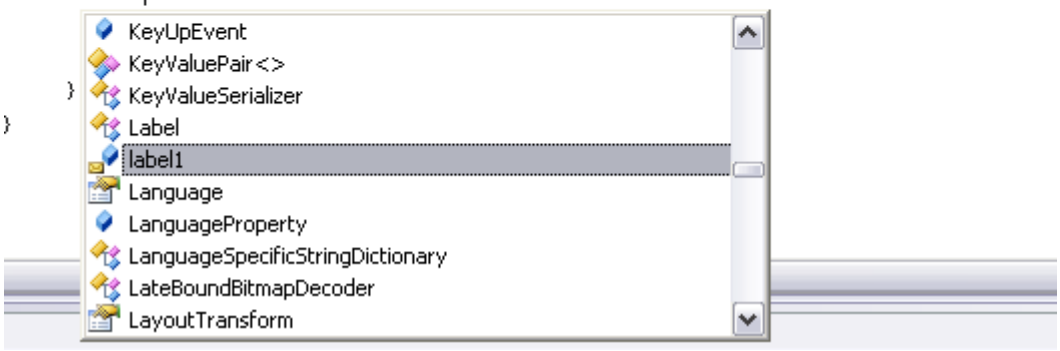
```
...
<Grid>

        <Button Height="23" Margin="126.116,47.678,76.9,0" Name="TrykMig"
VerticalAlignment="Top" Click="TrykMig_Click">Tryk mig</Button>
        <ListBox Margin="133.037,86.897,24.608,75.362" Name="listBox1"  />
        <Label Height="28" Margin="126.116,15.38,31.529,0" Name="label1"
VerticalAlignment="Top">Label</Label>
    </Grid>
...
```

- Controllers are automatically placed in the <Grid> area

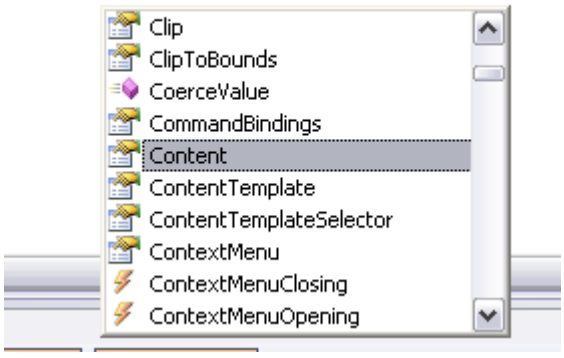
Interaction between the XAML Control and the cs. code

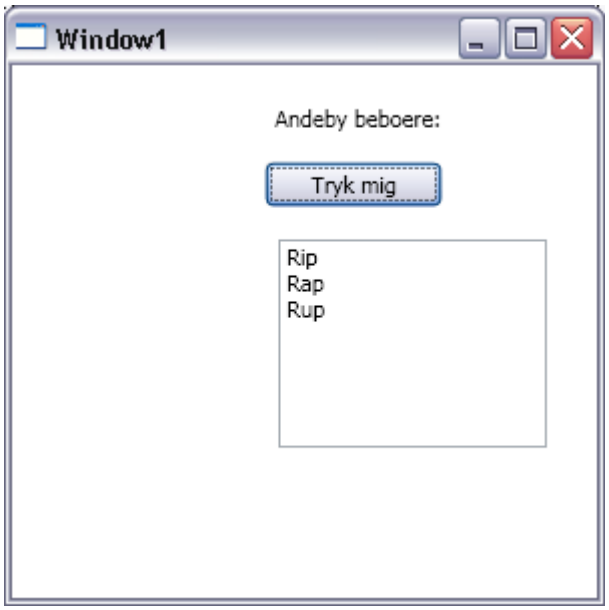
- A Label (label1) is added
- Full intellisense

```
private void TrykMig_Click(object sender, RoutedEventArgs e)
{
    Page1 page1 = new Page1();
    int antal = page1.Andeby.Items.Count;
    for( int i = 0; i<antal;i++)
        listBox1.Items.Add(page1.Andeby.Items[i]);

    l
}
}
```




```
label1.
```





Creating interaction on Buttons

- Just double-click on the control in the designer
  - Event is created automatically

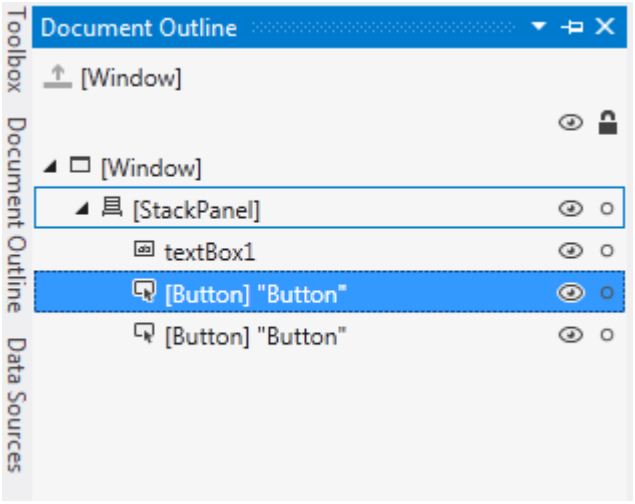
```
...
this.TrykMig.Click += new System.Windows.RoutedEventHandler(this.TrykMig_Click);

...
private void TrykMig_Click(object sender, RoutedEventArgs e)
        {
          // content
        }
...
```

Dialog boxes – are built-in

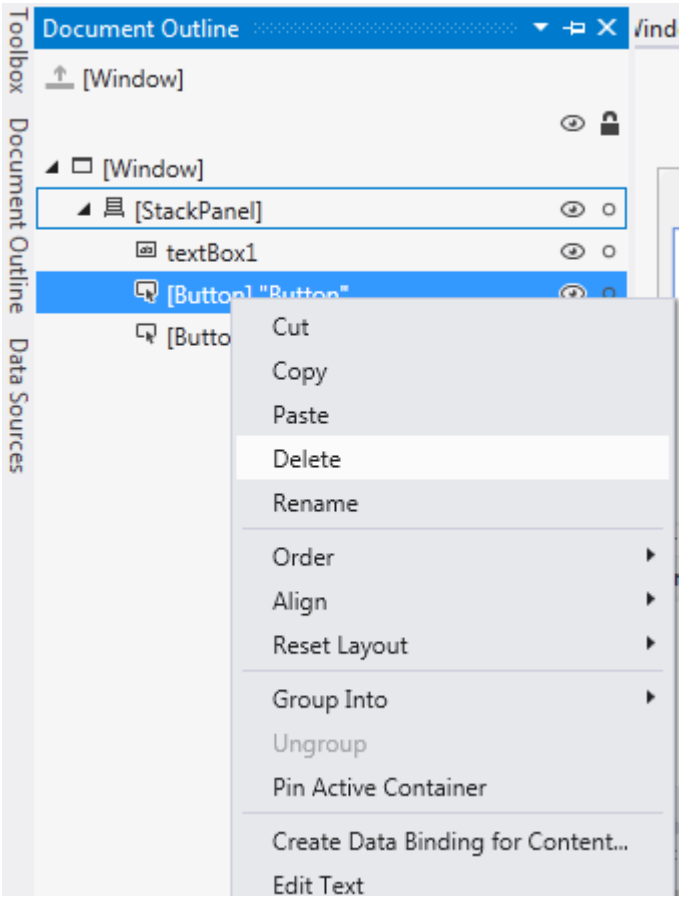
```
namespace WpfControls
{
public partial class MainWindow : Window
{
public MainWindow()
{
InitializeComponent();
}
```

```
private void btnShowDlg_Click(object sender, RoutedEventArgs e)
{
// Show a file save dialog.
SaveFileDialog saveDlg = new SaveFileDialog();
saveDlg.ShowDialog();
}
}
}
```

Use of the Document Outline Viewer



- Delete ??



Panels

- Organization of controllers in a window

Panel Controls

Table 25-2. *Core WPF Panel Controls*

Panel Control	Meaning in Life
`Canvas`	Provides a classic mode of content placement. Items stay exactly where you put them at design time.
`DockPanel`	Locks content to a specified side of the panel (`Top`, `Bottom`, `Left`, or `Right`).
`Grid`	Arranges content within a series of cells, maintained within a tabular grid.
`StackPanel`	Stacks content in a vertical or horizontal manner, as dictated by the `Orientation` property.
`WrapPanel`	Positions content from left to right, breaking the content to the next line at the edge of the containing box. Subsequent ordering happens sequentially from top to bottom or from right to left, depending on the value of the `Orientation` property.

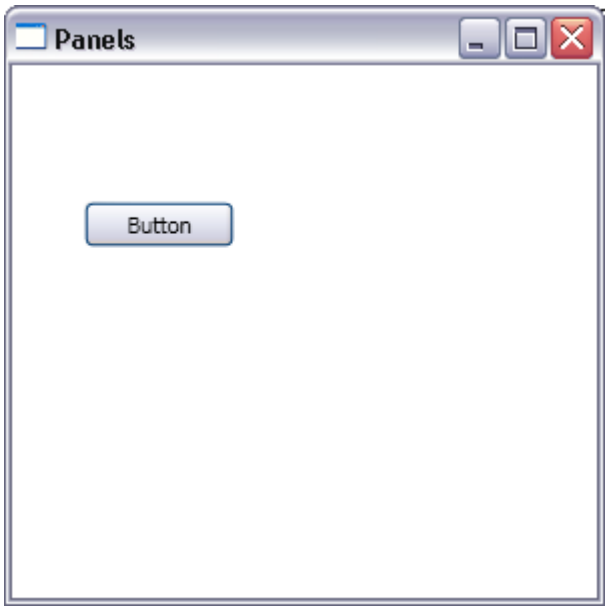
- Canvas
  - Traditional: Controller is placed where they are put
- DockPanel
  - Lock a control to one side: Top,Bottom, left,right
- Grid
  - Contains cells
- StackPanel
  - Content showed vertical or horizontal
- WrapPanel
  - Wraps from left to right uses a ”next line” if content is expanded)

Example: Canvas

```
<Window x:Class="WpfTest.Window2"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Panels" Height="300" Width="300">
    <Canvas>

    </Canvas>
</Window>
```

- Panels places controller on design time
- Example: Button

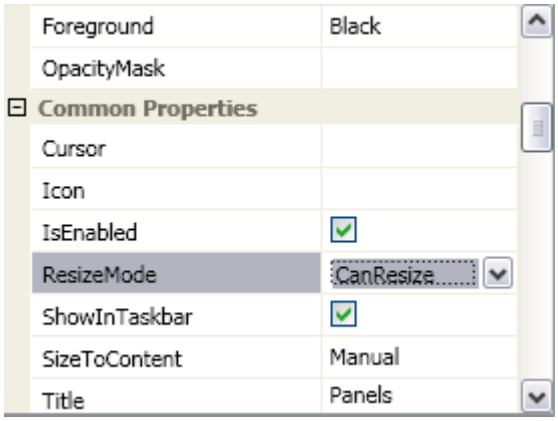
```
<Canvas>
        <Button Canvas.Left="36" Canvas.Top="68" Height="23" Name="button1"
Width="75">Button</Button>
    </Canvas>
```



Note If subelements within a `Canvas` do not define a specific location using attached property syntax (e.g., `Canvas.Left` and `Canvas.Top`), they automatically attach to the extreme upper-left corner of `Canvas`.

- Problem ?

- Can of course be solved with.:



- But not that smart!

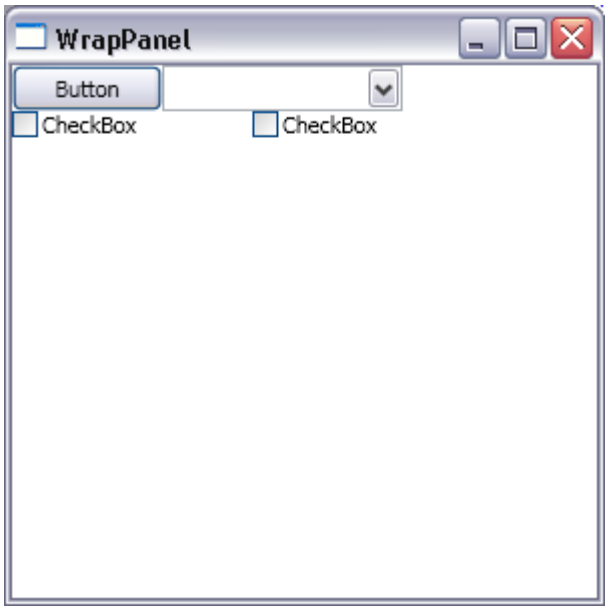
WrapPanel

- Wraps content to next line
  - Here example with a button and other content:

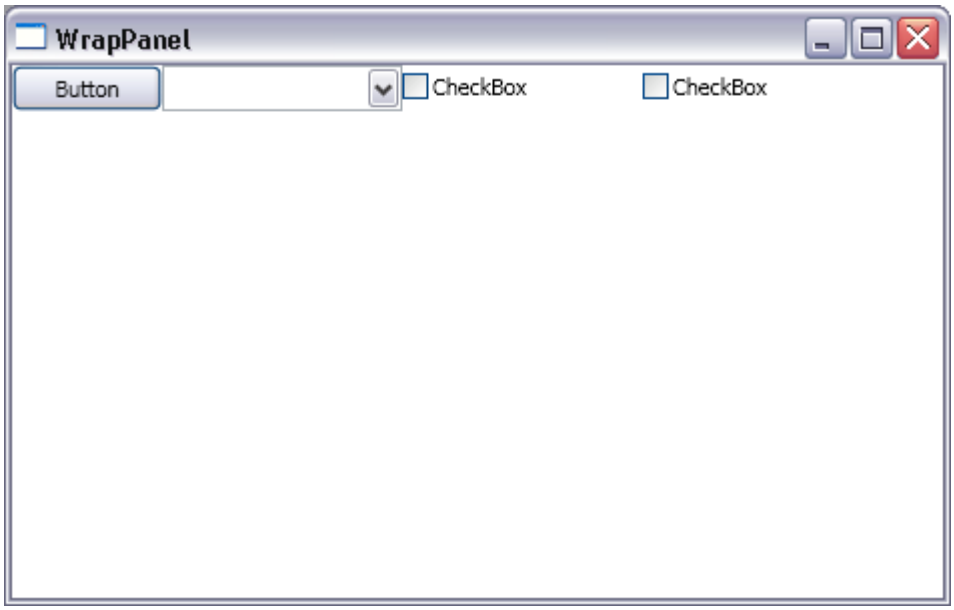
```
<Window x:Class="WpfTest.Window3"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="WrapPanel" Height="300" Width="300">
    <WrapPanel>
        <Button Height="23" Name="button1" Width="75">Button</Button>
        <ComboBox Height="23" Name="comboBox1" Width="120" />
        <CheckBox Height="16" Name="checkBox1" Width="120">CheckBox</CheckBox>
        <CheckBox Height="16" Name="checkBox2" Width="120">CheckBox</CheckBox>
    </WrapPanel>
</Window>
```

Behavior of the WrapPanel

- Normally

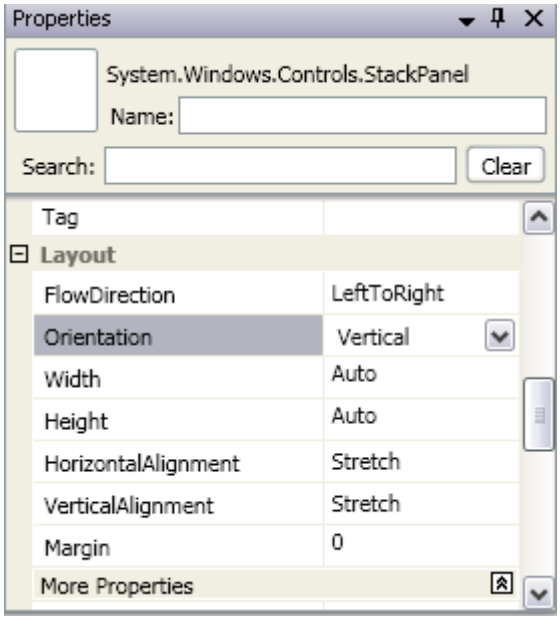


- when resizing

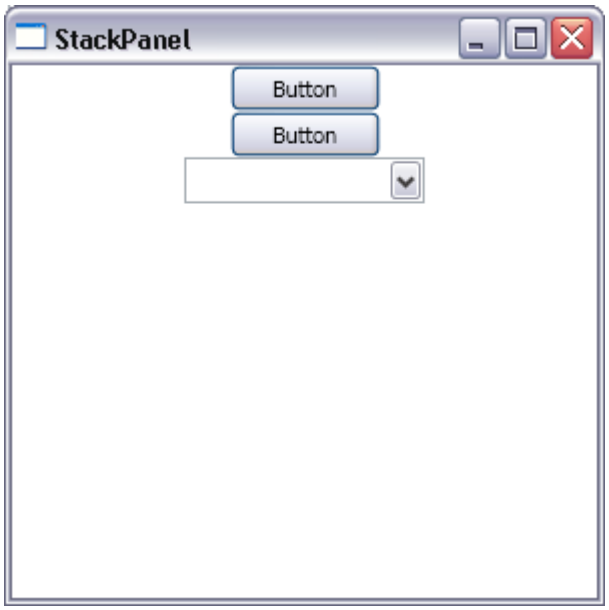


StackPanels

- Arranges content vertical or horizontal
- See properties




```
Window x:Class="WpfTest.Window4"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Window4" Height="300" Width="300">
    <StackPanel>
        <Button Height="23" Name="button1" Width="75">Button</Button>
        <Button Height="23" Name="button2" Width="75">Button</Button>
        <ComboBox Height="23" Name="comboBox1" Width="120" />
    </StackPanel>
</Window>
```



- resizing...

More about Grids (standard)

Sizing Grid Columns and Rows

Columns and rows in a grid can be sized in one of three ways.

- Absolute sizing (e.g., 100).
- Autosizing
- Relative sizing (e.g., 3x)

Absolute sizing is exactly what you would expect; the column (or row) is sized to a specific number of device-independent units. Autosizing sizes each column or row based on the controls contained with the column or row. Relative sizing is pretty much equivalent to percentage sizing in CSS. The total count of the numbers in relatively sized columns or rows gets divided into the total amount of available space.

In the following example, the first row gets 25 percent of the space, and the second row gets 75 percent of the space:

```
<Grid.ColumnDefinitions>
  <ColumnDefinition Width="1*" />
  <ColumnDefinition Width="3*" />
</Grid.ColumnDefinitions>
```

- Definition of rows / columns

```
<Grid ShowGridLines ="True" Background ="LightSteelBlue">
<!-- Define the rows/columns -->
<Grid.ColumnDefinitions>
<ColumnDefinition/>
<ColumnDefinition/>
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
<RowDefinition/>
<RowDefinition/>
</Grid.RowDefinitions>
<!-- Now add the elements to the grid's cells -->
<Label x:Name="lblInstruction" Grid.Column ="0" Grid.Row ="0"
FontSize="15" Content="Enter Car Information"/>
<Button x:Name="btnOK" Height ="30" Grid.Column ="0"
Grid.Row ="0" Content="OK"/>
<Label x:Name="lblMake" Grid.Column ="1"
Grid.Row ="0" Content="Make"/>
```

```
<TextBox x:Name="txtMake" Grid.Column ="1"
Grid.Row ="0" Width="193" Height="25"/>
<Label x:Name="lblColor" Grid.Column ="0"
Grid.Row ="1" Content="Color"/>
<TextBox x:Name="txtColor" Width="193" Height="25"
Grid.Column ="0" Grid.Row ="1" />
<!-- Just to keep things interesting, add some color to the pet name cell -->
<Rectangle Fill ="LightGreen" Grid.Column ="1" Grid.Row ="1" />
<Label x:Name="lblPetName" Grid.Column ="1" Grid.Row ="1" Content="Pet Name"/>
<TextBox x:Name="txtPetName" Grid.Column ="1" Grid.Row ="1"
Width="193" Height="25"/>
</Grid>
```

- Can be divided using a GridSplitter

```
<Grid Background ="LightSteelBlue">
<!-- Define columns -->
<Grid.ColumnDefinitions>
<ColumnDefinition Width ="Auto"/>
<ColumnDefinition/>
</Grid.ColumnDefinitions>
<!-- Add this label to cell 0 -->
<Label x:Name="lblLeft" Background ="GreenYellow"
Grid.Column="0" Content ="Left!"/>
<!-- Define the splitter -->
<GridSplitter Grid.Column ="0" Width ="5"/>
<!-- Add this label to cell 1 -->
<Label x:Name="lblRight" Grid.Column ="1" Content ="Right!"/>
</Grid>
```

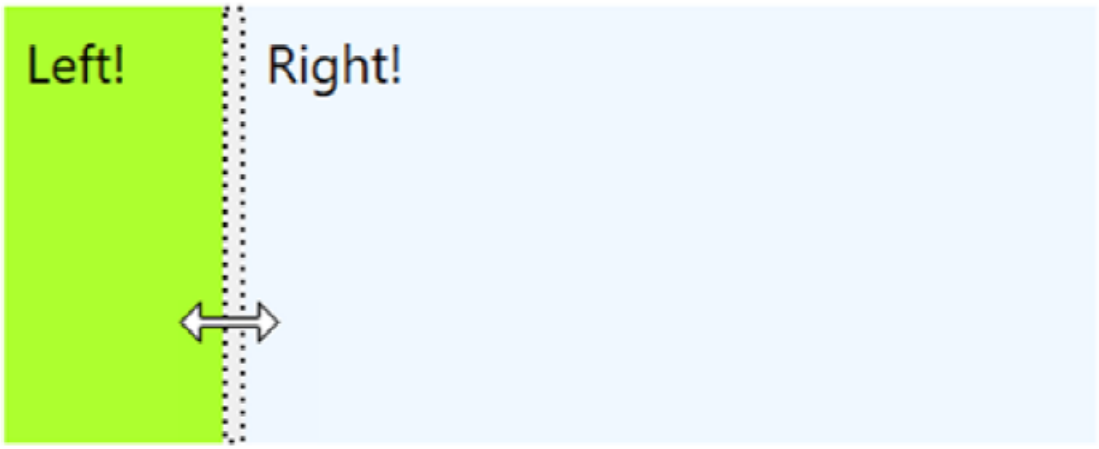


Figure 25-7. *Grid types containing splitters*

Other types of Panels

- See examples in the book p.1173 (DockPanel)

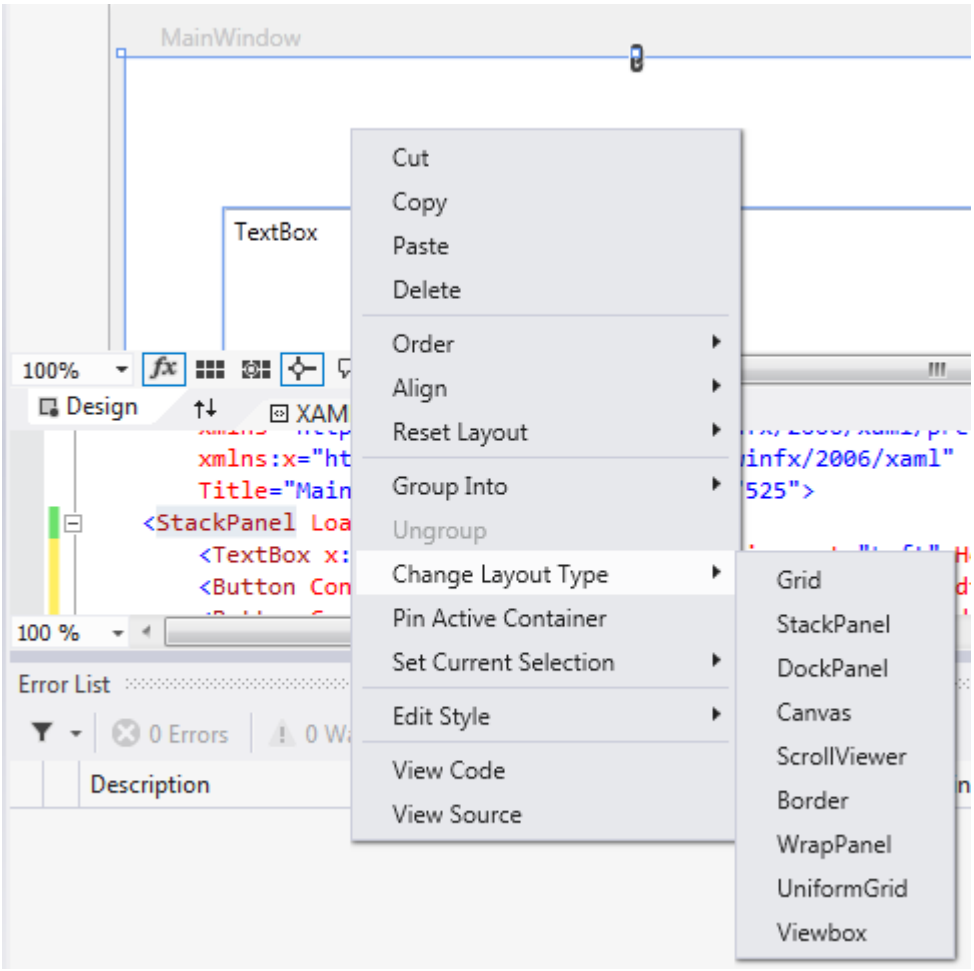
Scrolling

Enabling Scrolling for Panel Types

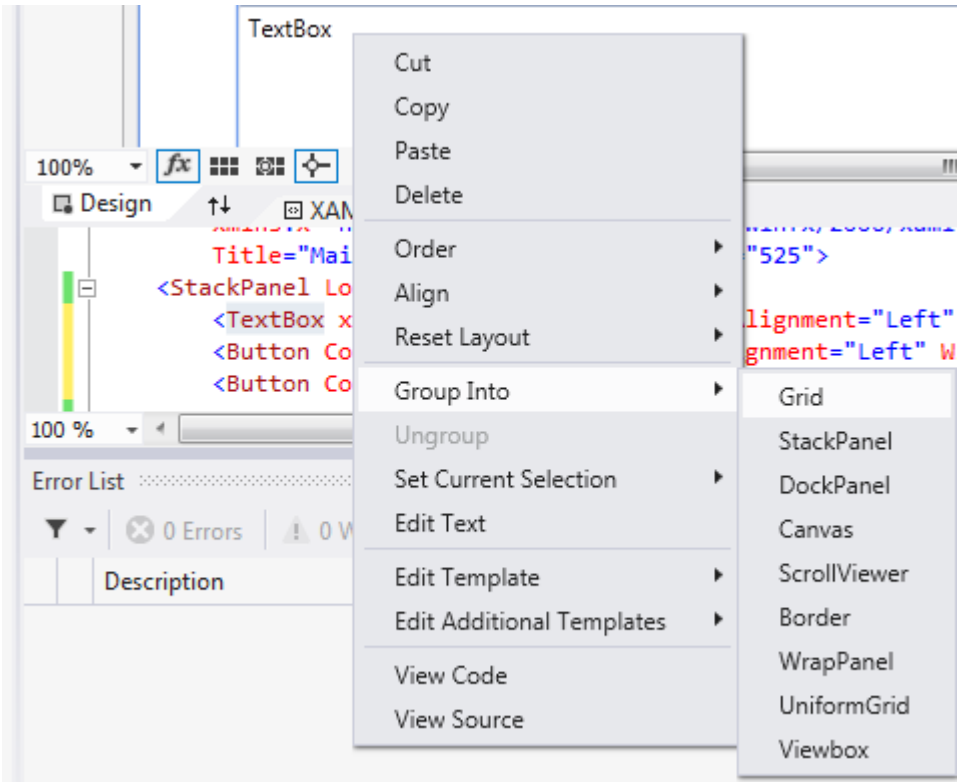
It is worth pointing out that WPF supplies a `ScrollViewer` class, which provides automatic scrolling behaviors for data within panel objects. The `SimpleScrollViewer.xaml` file defines the following:

```
<ScrollViewer>
  <StackPanel>
    <Button Content ="First" Background = "Green" Height ="40"/>
    <Button Content ="Second" Background = "Red" Height ="40"/>
    <Button Content ="Third" Background = "Pink" Height ="40"/>
    <Button Content ="Fourth" Background = "Yellow" Height ="40"/>
    <Button Content ="Fifth" Background = "Blue" Height ="40"/>
  </StackPanel>
</ScrollViewer>
```

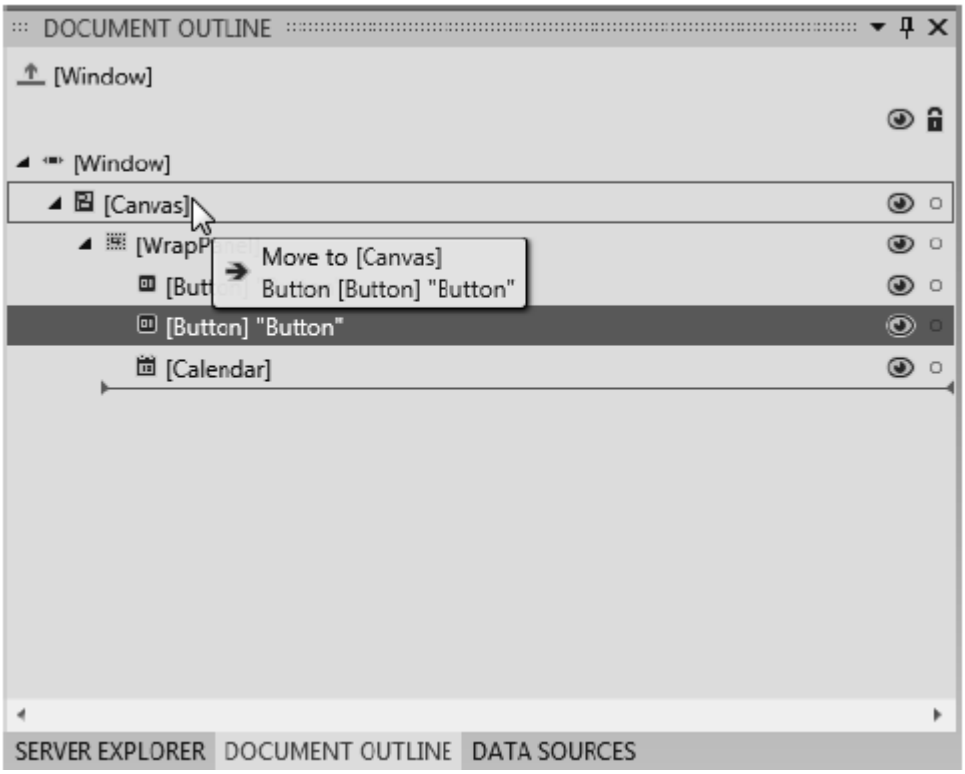
Use VS to work with panels



- can also group elements



- Can be moved via the Document Outline Viewer

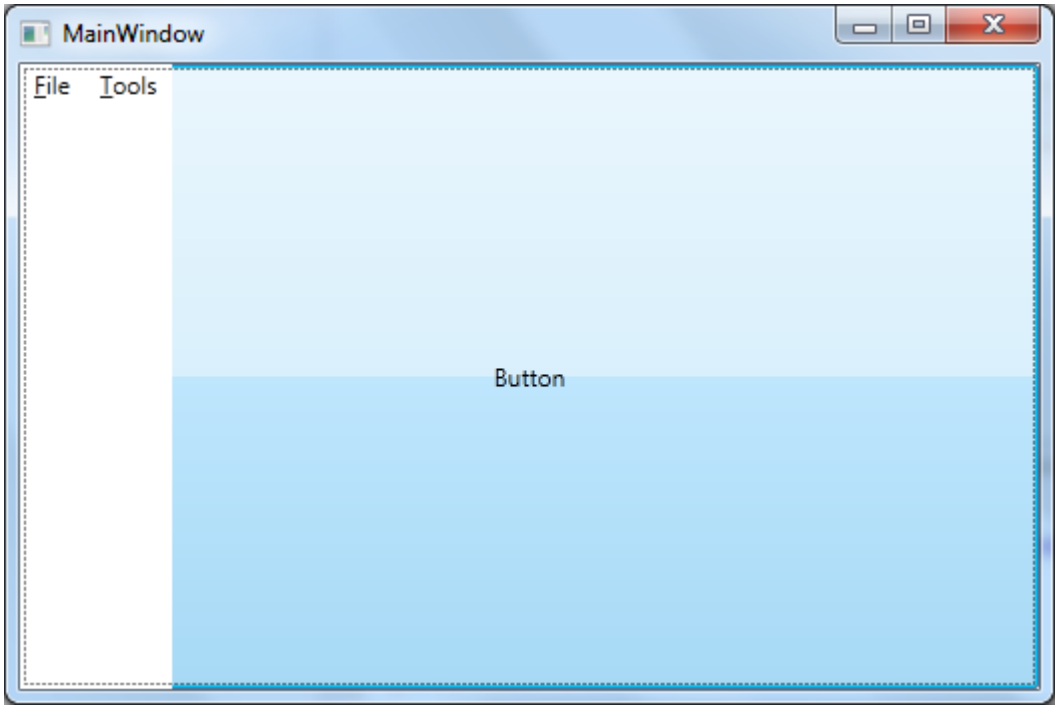


Window frames with nested panels

- See page 1020 -1021

Menu

- Find the ”Menu” item in the ToolBox
- Example




```
<Window x:Class="ShowXAML.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">
    <Grid Loaded=" Window_Loaded">
        <TextBox x:Name="textBox1" HorizontalAlignment="Left" Height="164"
Margin="49,75,0,0" TextWrapping="Wrap" Text="TextBox" VerticalAlignment="Top" Width="387"/>
        <Button Content="Button" HorizontalAlignment="Left" Width="509"/>
        <Button Content="Button" HorizontalAlignment="Left" Width="509"/>

<!-- Dock menu system on the top -->
```

```
        <Menu DockPanel.Dock ="Top"
HorizontalAlignment="Left" Background="White" BorderBrush ="Black">
            <MenuItem Header="_File">
                <Separator/>
                <MenuItem Header ="_Exit"/>
            </MenuItem>
            <MenuItem Header="_Tools">
                <MenuItem Header ="_Spelling Hints" />
            </MenuItem>
        </Menu>
    </Grid>
</Window>
```

- DB – Click on the menu line

```
private void MenuItem_Click_1(object sender, RoutedEventArgs e)
        {

        }
```

Toolbar (here dock to the left)


```
...
<ToolBar DockPanel.Dock ="Left" >
            <Button Content ="Exit" />
            <Separator/>
            <Button Content ="Check"/>

        </ToolBar>
...
```

- StatusBar – see page 1024

WPF Control Commands

- A WPF Control Command is INDEPENDENT of a special Control
  - Example:
    - Copy, Paste and Cut

- A WPF Control Command is an object that
  - implements ICommand

```
public interface ICommand
{
// Occurs when changes occur that affect whether
// or not the command should execute.
event EventHandler CanExecuteChanged;
// Defines the method that determines whether the command
// can execute in its current state.
bool CanExecute(object parameter);
// Defines the method to be called when the command is invoked.
void Execute(object parameter);
}
```

- Controls

Table 28-3. *The Intrinsic WPF Control Command Objects*

WPF Class	Command Objects	Meaning in Life
ApplicationCommands	Close, Copy, Cut, Delete, Find, Open, Paste, Save, SaveAs, Redo, Undo	Various application-level commands
ComponentCommands	MoveDown, MoveFocusBack, MoveLeft, MoveRight, ScrollToEnd, ScrollToHome	Various commands common to UI components
MediaCommands	BoostBase, ChannelUp, ChannelDown, FastForward, NextTrack, Play, Rewind, Select, Stop	Various media-centric commands
NavigationCommands	BrowseBack, BrowseForward, Favorites, LastPage, NextPage, Zoom	Various commands relating to the WPF navigation model
EditingCommands	AlignCenter, CorrectSpellingError, DecreaseFontSize, EnterLineBreak, EnterParagraphBreak, MoveDownByLine, MoveRightByWord	Various commands relating to the WPF Documents API

Examples:

```
<Menu DockPanel.Dock ="Top"
HorizontalAlignment="Left"
Background="White" BorderBrush ="Black">
<MenuItem Header="_File" Click ="FileExit_Click" >
<MenuItem Header ="_Exit" MouseEnter ="MouseEnterExitArea"
MouseLeave ="MouseLeaveArea" Click ="FileExit_Click"/>
</MenuItem>
<!-- New menu item with commands! -->
<MenuItem Header="_Edit">
<MenuItem Command ="ApplicationCommands.Copy"/>
<MenuItem Command ="ApplicationCommands.Cut"/>
<MenuItem Command ="ApplicationCommands.Paste"/>
</MenuItem>
<MenuItem Header="_Tools">
<MenuItem Header ="_Spelling Hints"
MouseEnter ="MouseEnterToolsHintsArea"
MouseLeave ="MouseLeaveArea"
Click ="ToolsSpellingHints_Click"/>
</MenuItem>
</Menu>
```

CommandBinding

```
private void SetF1CommandBinding()
{
CommandBinding helpBinding = new CommandBinding(ApplicationCommands.Help);
helpBinding.CanExecute += CanHelpExecute;
helpBinding.Executed += HelpExecuted;
CommandBindings.Add(helpBinding);
}

…

private void CanHelpExecute(object sender, CanExecuteRoutedEventArgs e)
{
// Here, you can set CanExecute to false if you want to prevent the
// command from executing.
e.CanExecute = true;
}
private void HelpExecuted(object sender, ExecutedRoutedEventArgs e)
{
MessageBox.Show("Look, it is not that difficult. Just type something!",
"Help!");
}
```

Open / save commands

- ICommand interfacet defines two events
- (CanExecute and Executed).

```
<Window x:Class="MyWordPad.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="MySpellChecker" Height="331" Width="508"
WindowStartupLocation ="CenterScreen" >
<!-- This will inform the Window which handlers to call,
when testing for the Open and Save commands. -->
<Window.CommandBindings>
<CommandBinding Command="ApplicationCommands.Open"
Executed="OpenCmdExecuted"
CanExecute="OpenCmdCanExecute"/>
<CommandBinding Command="ApplicationCommands.Save"
Executed="SaveCmdExecuted"
CanExecute="SaveCmdCanExecute"/>
</Window.CommandBindings>
```

```
<!-- This panel establishes the content for the window -->
<DockPanel>
...
</DockPanel>

</Window>

…
private void OpenCmdCanExecute(object sender, CanExecuteRoutedEventArgs e)
{
e.CanExecute = true;
}
private void SaveCmdCanExecute(object sender, CanExecuteRoutedEventArgs e)
{
e.CanExecute = true;
}

//Med indførsel af er det ok at fyre disse events af som kommer her:


private void OpenCmdExecuted(object sender, ExecutedRoutedEventArgs e)
{
// Create an open file dialog box and only show XAML files.
OpenFileDialog openDlg = new OpenFileDialog();
openDlg.Filter = "Text Files |*.txt";
// Did they click on the OK button?
if (true == openDlg.ShowDialog())
{
// Load all text of selected file.
string dataFromFile = File.ReadAllText(openDlg.FileName);
// Show string in TextBox.
txtData.Text = dataFromFile;
}
}
private void SaveCmdExecuted(object sender, ExecutedRoutedEventArgs e)
{
SaveFileDialog saveDlg = new SaveFileDialog();
saveDlg.Filter = "Text Files |*.txt";
// Did they click on the OK button?
if (true == saveDlg.ShowDialog())
{
// Save data in the TextBox to the named file.
File.WriteAllText(saveDlg.FileName, txtData.Text);
}
}
```

Routed events

Understanding Routed Events

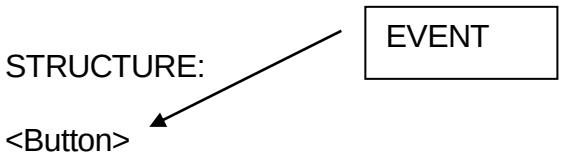
You might have noticed the `RoutedEventArgs` parameter instead of `EventArgs` in the previous code example. The routed events model is a refinement of the standard CLR event model designed to ensure that events can be processed in a manner that is fitting for XAML's description of a tree of objects. Assume you have a new WPF application project named WPFRoutedEvents. Now, update the XAML description of the initial window by adding the following `Button` control, which defines some complex content:

```
<Button Name="btnClickMe" Height="75" Width = "250" Click ="btnClickMe_Clicked">
  <StackPanel Orientation ="Horizontal">
    <Label Height="50" FontSize ="20">Fancy Button!</Label>
    <Canvas Height ="50" Width ="100" >
    <Ellipse Name = "outerEllipse" Fill ="Green" Height ="25"
             Width ="50" Cursor="Hand" Canvas.Left="25" Canvas.Top="12"/>
    <Ellipse Name = "innerEllipse" Fill ="Yellow" Height = "15" Width ="36"
             Canvas.Top="17" Canvas.Left="32"/>
    </Canvas>
  </StackPanel>
</Button>
```

Notice in the `Button`'s opening definition you have handled the `Click` event by specifying the name of a method to be called when the event is raised. The `Click` event works with the `RoutedEventHandler` delegate, which expects an event handler that takes an `object` as the first parameter and a `System.Windows.RoutedEventArgs` as the second. Implement this handler as so:

```
public void btnClickMe_Clicked(object sender, RoutedEventArgs e)
{
  // Do something when button is clicked.
  MessageBox.Show("Clicked the button");
}
```

If you run your application, you will see this message box display, regardless of which part of the button's content you click (the green `Ellipse`, the yellow `Ellipse`, the `Label`, or the `Button`'s surface). This is a good thing. Imagine how tedious WPF event handling would be if you were forced to handle a `Click` event for every one of these subelements. Not only would the creation of separate event handlers for each aspect of the `Button` be labor intensive, you would end up with some mighty nasty code to maintain down the road.




```
<Button>
<StackPanel>
  <Label></Label>
  <Canvas>
    <Ellipse></Ellipse>
  </Canvas>
 </StackPanel>
</Button>
```

- It would be awful to handle a Click event for all subelements

THEREFORE: Routed Events

- A routed event can propagate up or down within the object tree
- Is WPF specific indeed:
  - A direct event is when the WPF event result in a **CLR** event
  - A ”bubbling event” is when the event propagates **UP** in the object tree
  - A ”tunneling event” is when the event propagates **DOWN** in the object tree

Routed Bubbling events

The Role of Routed Bubbling Events

In the current example, if the user clicks the inner yellow oval, the `Click` event bubbles out to the next level of scope (the `Canvas`), then to the `StackPanel`, and finally to the `Button` where the `Click` event handler is handled. In a similar way, if the user clicks the `Label`, the event is bubbled to the `StackPanel` and then finally to the `Button` element.

Given this bubbling routed event pattern, you have no need to worry about registering specific `Click` event handlers for all members of a composite control. However, if you want to perform custom clicking logic for multiple elements within the same object tree, you can do so.

By way of illustration, assume you need to handle the clicking of the `outerEllipse` control in a unique manner. First, handle the `MouseDown` event for this subelement (graphically rendered types such as the `Ellipse` do not support a `Click` event; however, they can monitor mouse button activity via `MouseDown`, `MouseUp`, etc.).

```
<Button Name="btnClickMe" Height="75" Width = "250"
        Click ="btnClickMe_Clicked">
  <StackPanel Orientation ="Horizontal">
    <Label Height="50" FontSize ="20">Fancy Button!</Label>
    <Canvas Height ="50" Width ="100" >
    <Ellipse Name = "outerEllipse" Fill ="Green"
             Height ="25" MouseDown ="outerEllipse_MouseDown"
             Width ="50" Cursor="Hand" Canvas.Left="25" Canvas.Top="12"/>
    <Ellipse Name = "innerEllipse" Fill ="Yellow" Height = "15" Width ="36"
             Canvas.Top="17" Canvas.Left="32"/>
    </Canvas>
```

Routed Tunneiing Events

The Role of Routed Tunneling Events

Strictly speaking, routed events can be *bubbling* (as just described) or *tunneling* in nature. Tunneling events (which all begin with the `Preview` suffix—e.g., `PreviewMouseDown`) drill down from the topmost element into the inner scopes of the object tree. By and large, each bubbling event in the WPF base class libraries is paired with a related tunneling event that fires *before* the bubbling counterpart. For example, before the bubbling `MouseDown` event fires, the tunneling `PreviewMouseDown` event fires first.

Handling a tunneling event looks just like the processing of handling any other events; simply assign the event handler name in XAML (or, if needed, use the corresponding C# event-handling syntax in your code file) and implement the handler in the code file. Just to illustrate the interplay of tunneling and bubbling events, begin by handling the `PreviewMouseDown` event for the `outerEllipse` object, like so:

```
<Ellipse Name = "outerEllipse" Fill ="Green" Height ="25"
         MouseDown ="outerEllipse_MouseDown"
         PreviewMouseDown ="outerEllipse_PreviewMouseDown"
         Width ="50" Cursor="Hand" Canvas.Left="25" Canvas.Top="12"/>
```

Adding a Tab Control

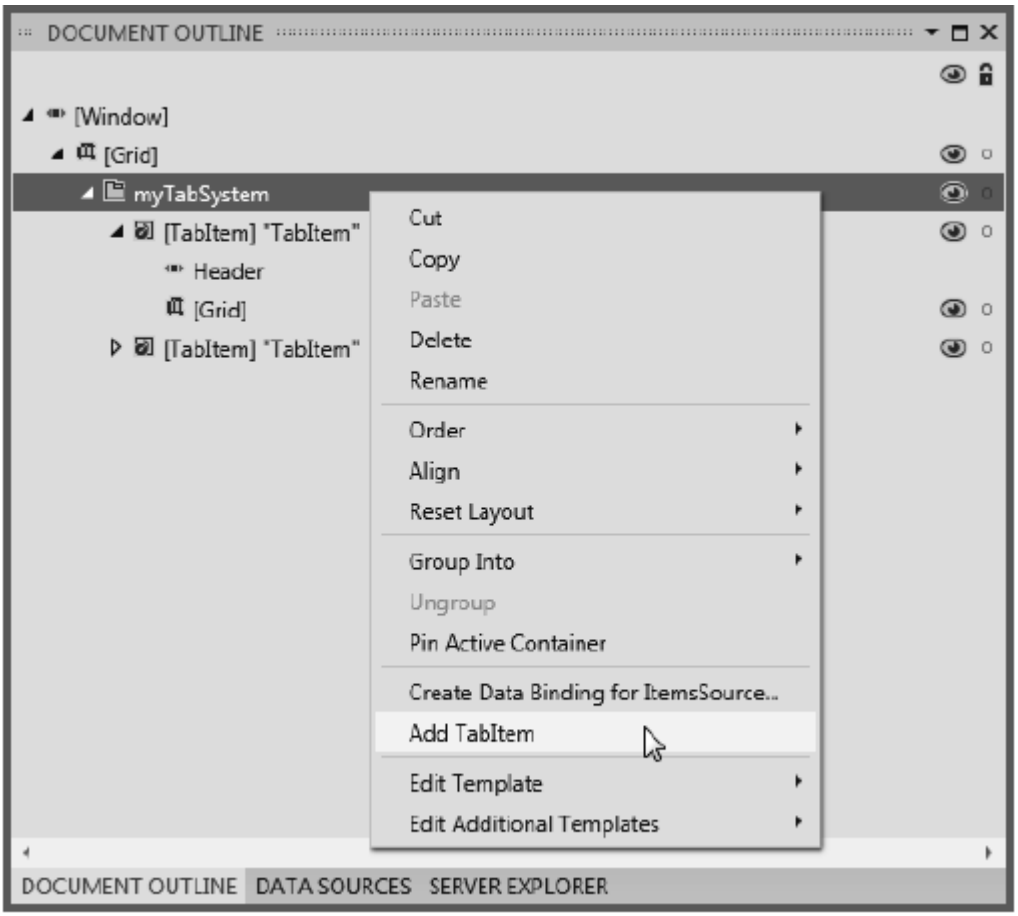


Figure 28-23. *Visually adding TabItems*

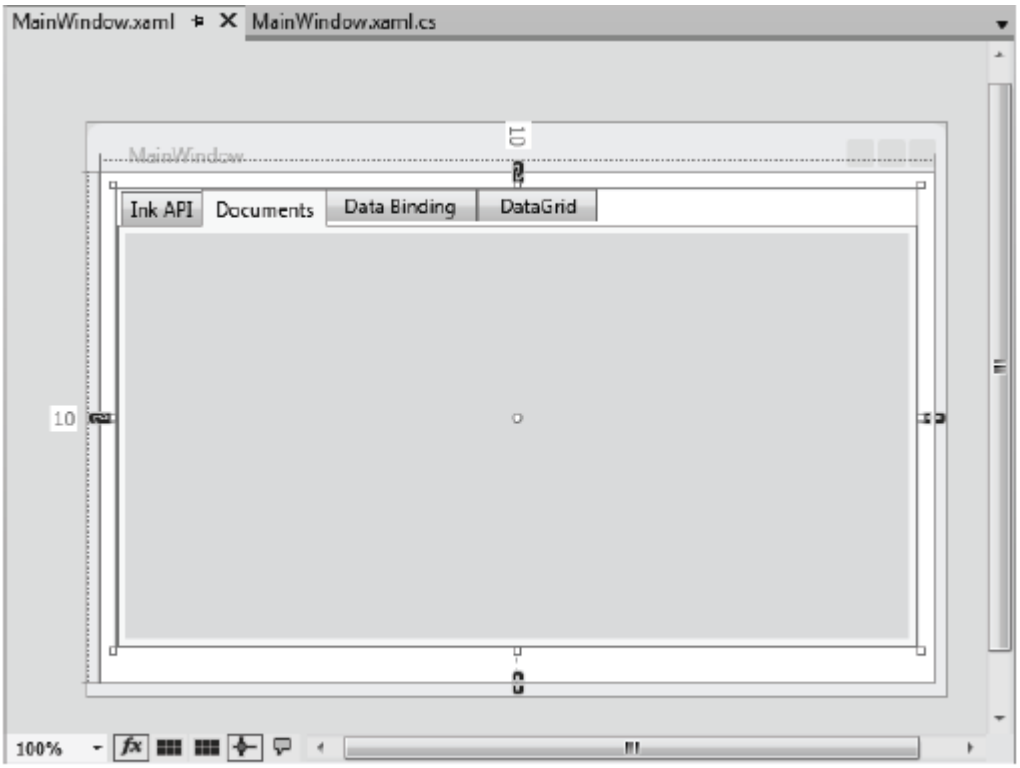
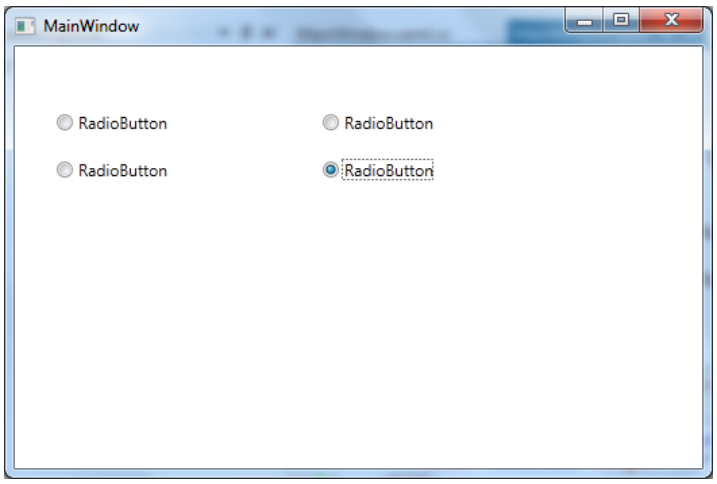


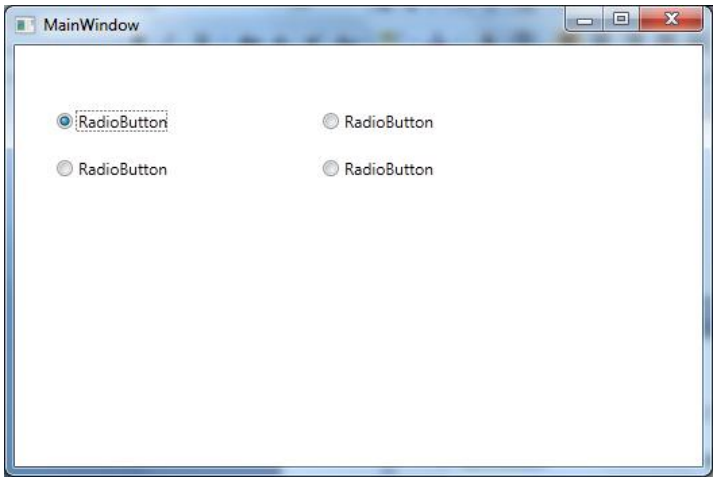
Figure 28-24. *The initial layout of the tab system*

- XAML

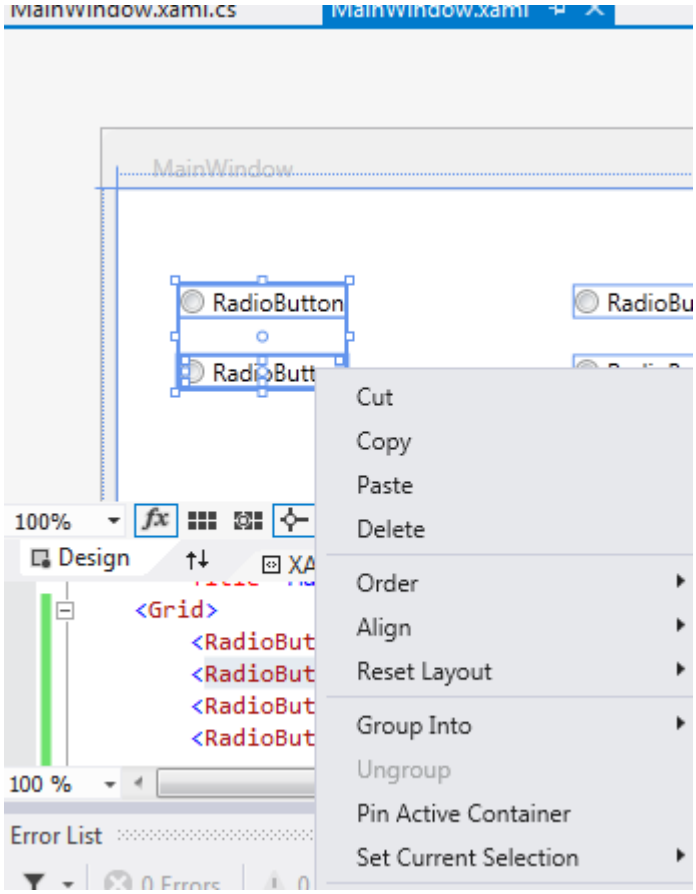
```
<TabControl x:Name="myTabSystem" HorizontalAlignment="Left" Height="280"
Margin="10,10,0,0" VerticalAlignment="Top" Width="489">
<TabItem Header="Ink API">
<Grid Background="#FFE5E5E5"/>
</TabItem>
<TabItem Header="Documents">
<Grid Background="#FFE5E5E5"/>
</TabItem>
<TabItem Header="Data Binding" HorizontalAlignment="Left" Height="20"
VerticalAlignment="Top" Width="95" Margin="-2,-2,-36,0">
<Grid Background="#FFE5E5E5"/>
</TabItem>
<TabItem Header="DataGrid" HorizontalAlignment="Left" Height="20"
VerticalAlignment="Top" Width="74" Margin="-2,-2,-15,0">
<Grid Background="#FFE5E5E5"/>
</TabItem>
</TabControl>
```

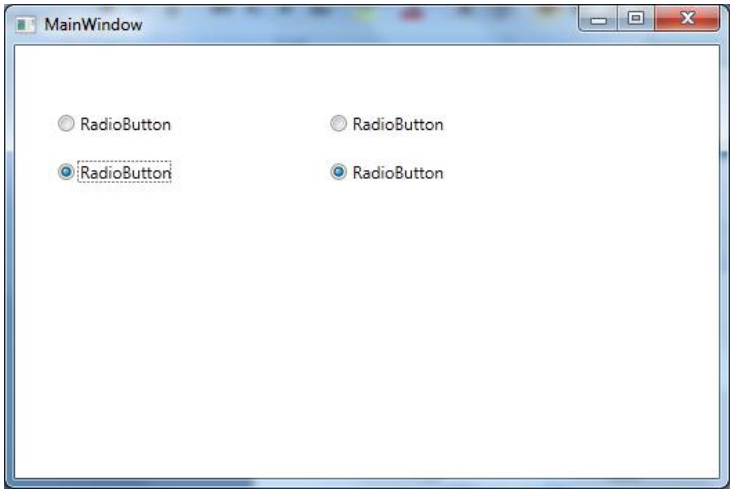
RadioButton



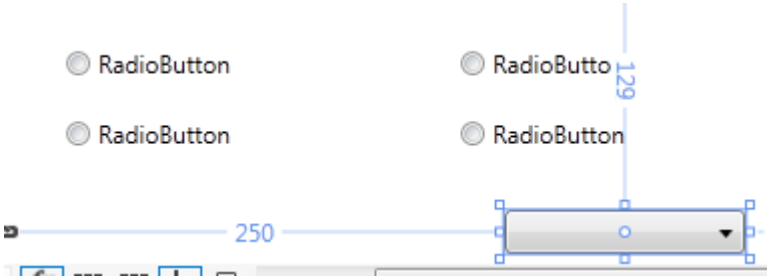


- Groupng (into groupes) ??
- Solution: Group Into – Border

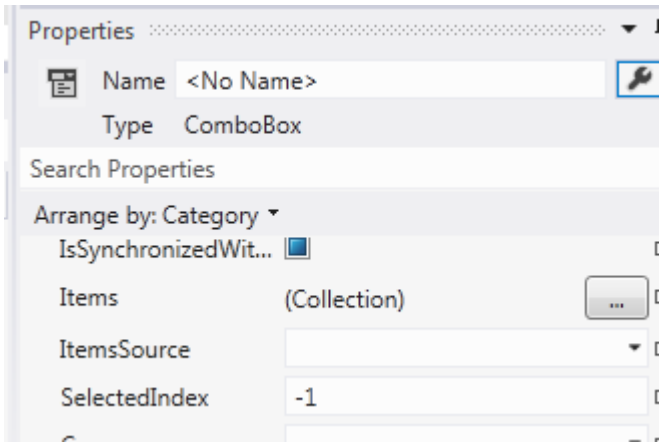

MainWindow.xaml.cs
MainWindow.xaml
MainWindow
RadioButton
RadioBu
Cut
Copy
Paste
Delete
Order
Align
Reset Layout
Group Into
Ungroup
Pin Active Container
Set Current Selection
100%
Design
<Grid>
100 %
Error List

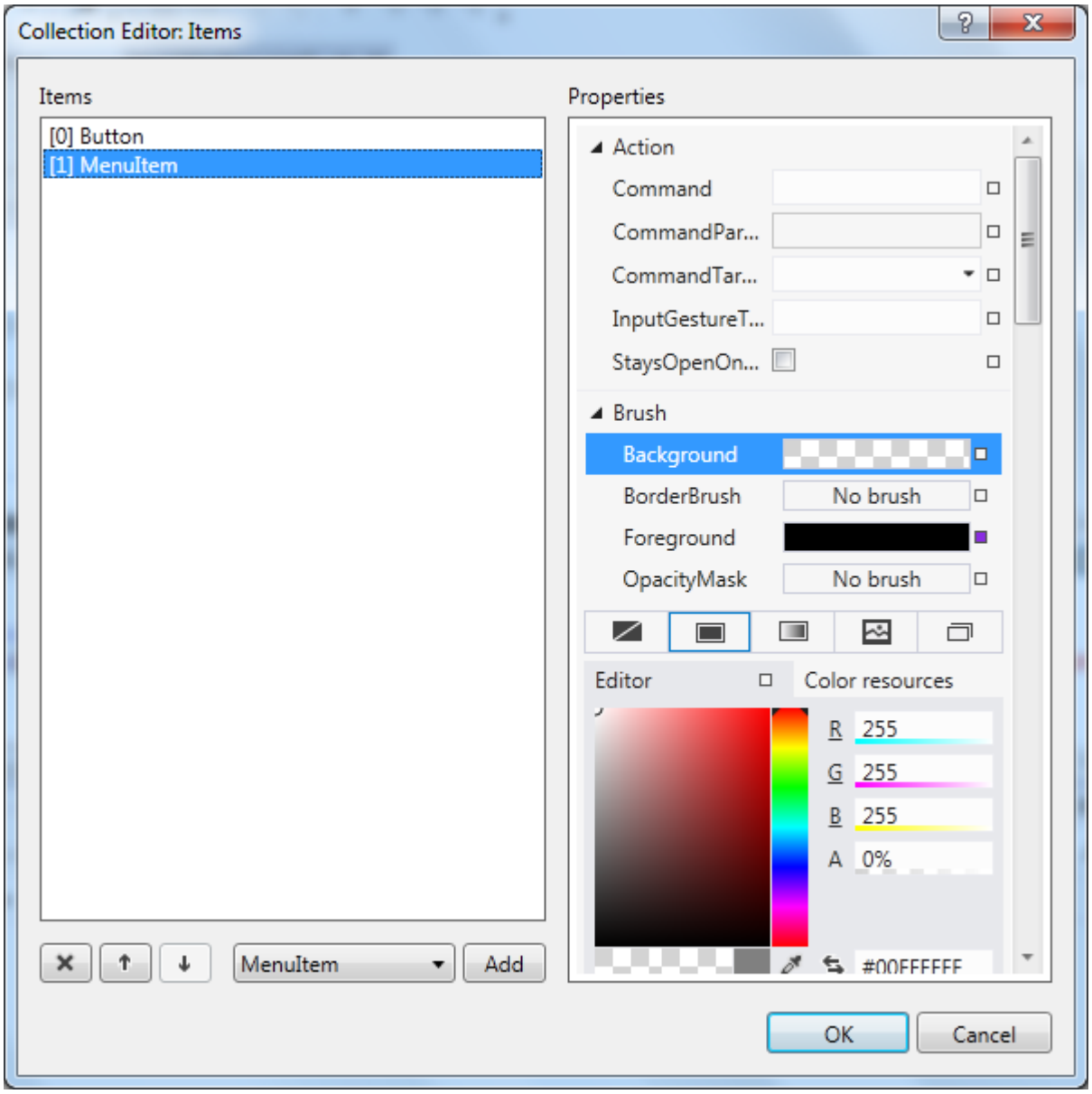

MainWindow
RadioButton
RadioButton
RadioButton
RadioButton

ComboBox


RadioButton
RadioButto
RadioButton
RadioButton
129
250

- Has Items


Properties
Name <No Name>
Type ComboBox
Search Properties
Arrange by: Category
IsSynchronizedWit...
Items (Collection)
ItemsSource
SelectedIndex -1

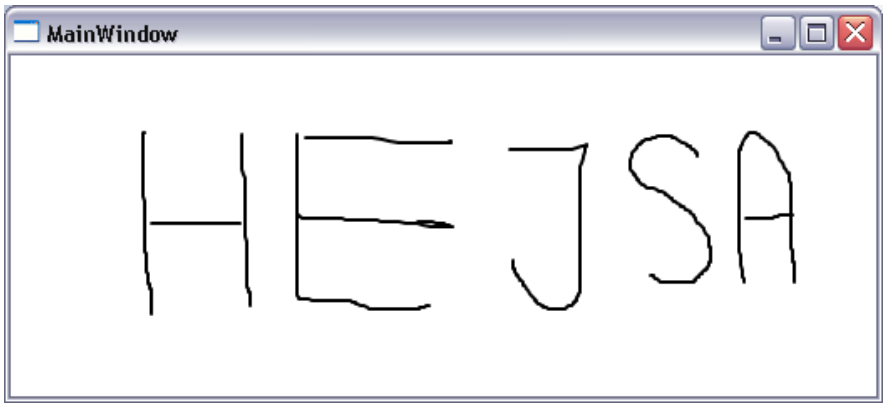


InkCanvas

- Just change <Canvas> to <InkCanvas>
- New functionality

```
<Window x:Class="WpfTestApp.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">
    <InkCanvas>

    </InkCanvas>
```

```
</Window>
```



XPS Document viewer

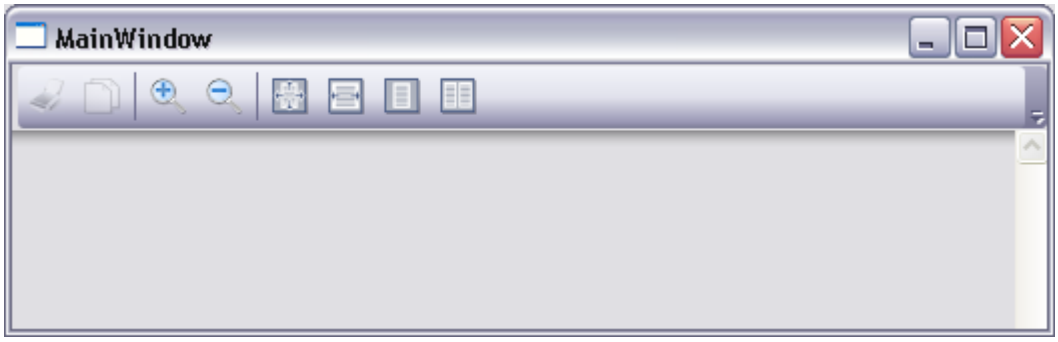
Table 28-4. *XPS Control Layout Managers*

Panel Control	Meaning in Life
FlowDocumentReader	Displays data in a FlowDocument and adds support for zooming, searching, and content layout in various forms.
FlowDocumentScrollViewer	Displays data in a FlowDocument; however, the data is presented as a single document viewed with scrollbars. This container does not support zooming, searching, or alternative layout modes.
RichTextBox	Displays data in a FlowDocument and adds support for user editing.
FlowDocumentPageViewer	Displays the document page-by-page, one page at a time. Data can also be zoomed, but not searched.

- Built-In <DocumentViewer>

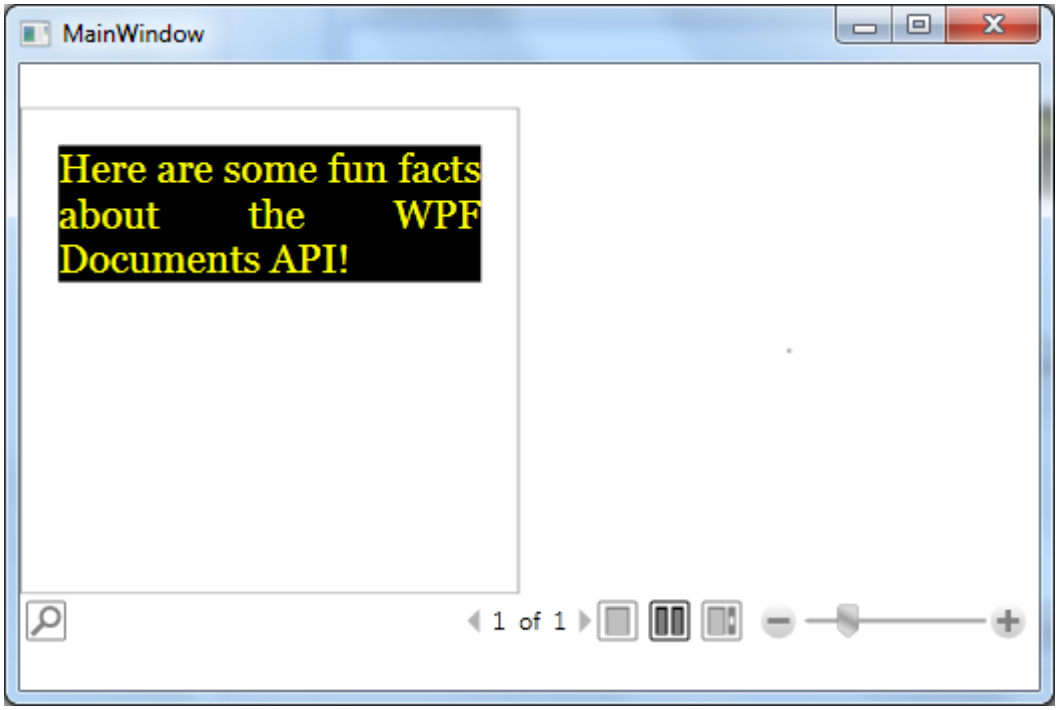
```
<Window x:Class="WpfTestApp.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">
    <Canvas>
        <DocumentViewer  Height="310" Canvas.Top="0"  Width="518"></DocumentViewer>
    </Canvas>
</Window>
```

- Has many properties !



FlowDocumentReader

```
<Window x:Class="Ink.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">
    <Grid>


<FlowDocumentReader x:Name="myDocumentReader" Height="269.4">
            <FlowDocument>
                <Section Foreground = "Yellow" Background = "Black">
                    <Paragraph FontSize = "20">
                        Here are some fun facts about the WPF Documents API!
                    </Paragraph>
                </Section>
                <List/>
                <Paragraph/>
            </FlowDocument>
        </FlowDocumentReader>

    </Grid>
</Window>
```

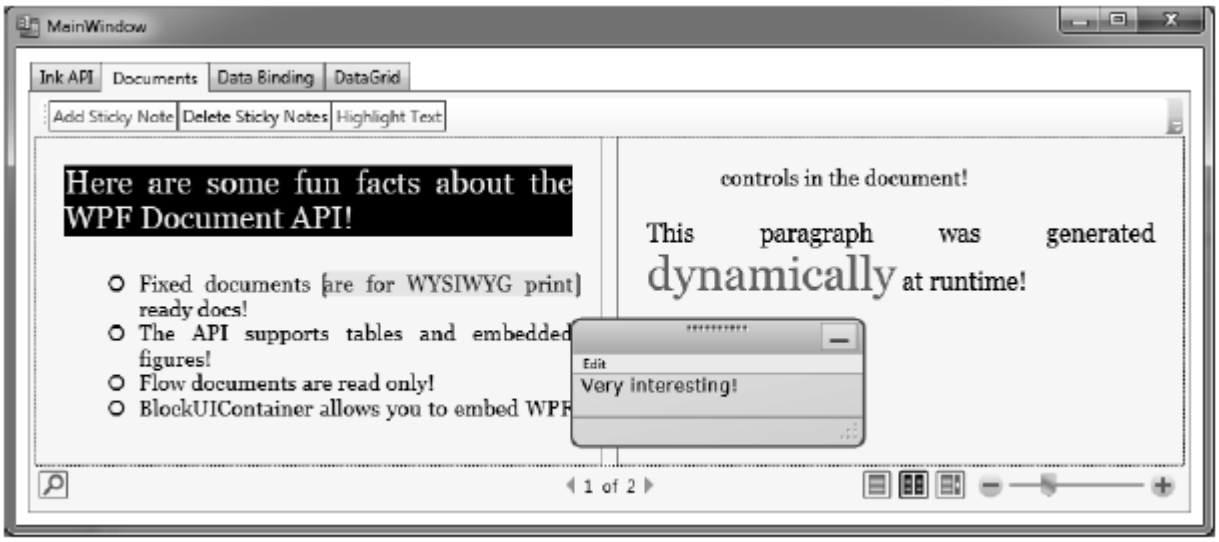


Comments (annotation) can also be used

```
<Window
...
xmlns:a=
"clr-namespace:System.Windows.Annotations;assembly=PresentationFramework"
x:Class="WpfControlsAndAPIs.MainWindow"
x:Name="Window"
Title="MainWindow"
Width="856" Height="383" mc:Ignorable="d"
WindowStartupLocation="CenterScreen" >
...
</Window>
```

```
<ToolBar>
<Button BorderBrush="Green" Content="Add Sticky Note"
Command="a:AnnotationService.CreateTextStickyNoteCommand"/>
<Button BorderBrush="Green" Content="Delete Sticky Notes"
Command="a:AnnotationService.DeleteStickyNotesCommand"/>
<Button BorderBrush="Green" Content="Highlight Text"
Command="a:AnnotationService.CreateHighlightCommand"/>
</ToolBar>
```

```
using System.Windows.Annotations;
using System.Windows.Annotations.Storage;
Next, implement this method:
private void EnableAnnotations()
{
// Create the AnnotationService object that works
// with our FlowDocumentReader.
AnnotationService anoService = new AnnotationService(myDocumentReader);
// Create a MemoryStream that will hold the annotations.
MemoryStream anoStream = new MemoryStream();
// Now, create an XML-based store based on the MemoryStream.
// You could use this object to programmatically add, delete,
// or find annotations.
AnnotationStore store = new XmlStreamStore(anoStream);
// Enable the annotation services.
anoService.Enable(store);
}
```



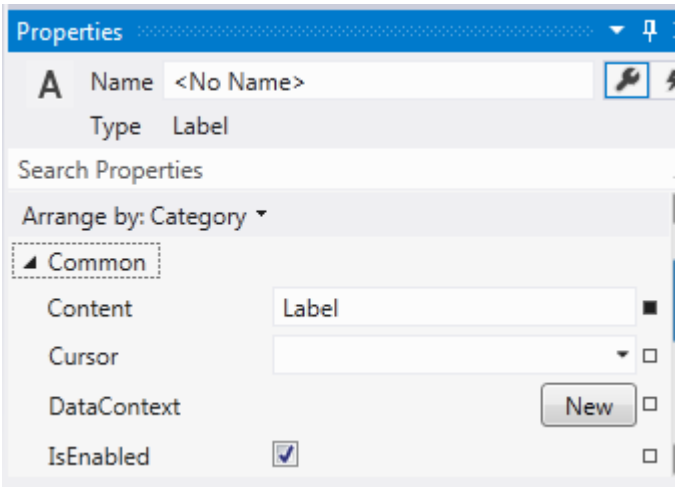
Save and load of documents

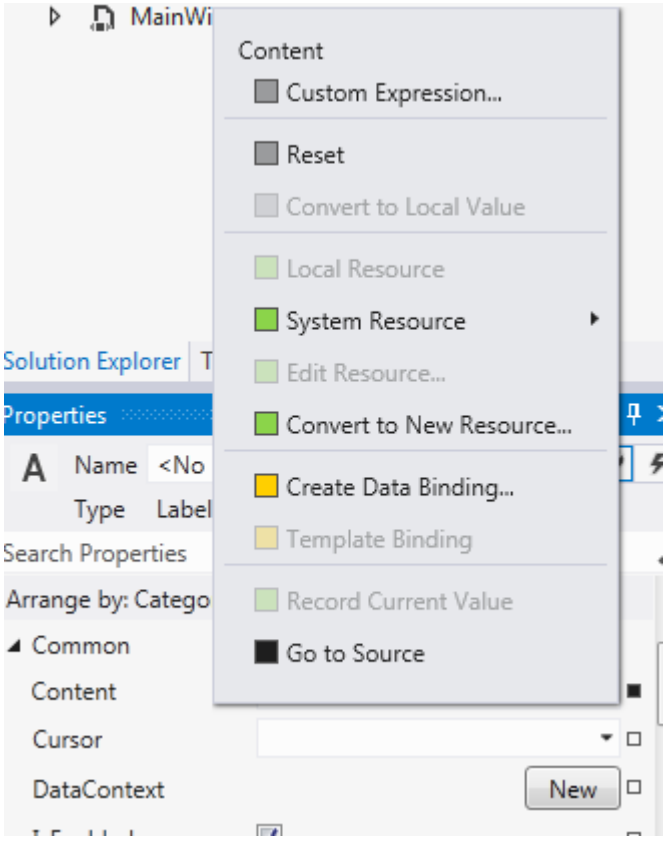
```
<Button x:Name="btnSaveDoc" HorizontalAlignment="Stretch"
VerticalAlignment="Stretch" Width="75" Content="Save Doc"/>
<Button x:Name="btnLoadDoc" HorizontalAlignment="Stretch"
VerticalAlignment="Stretch" Width="75" Content="Load Doc"/>
…

public MainWindow()
{
```

```
...
// Rig up some Click handlers for the save/load of the flow doc.
btnSaveDoc.Click += (o, s) =>
{
using(FileStream fStream = File.Open(
"documentData.xaml", FileMode.Create))
{
XamlWriter.Save(this.myDocumentReader.Document, fStream);
}
};
btnLoadDoc.Click += (o, s) =>
{
using(FileStream fStream = File.Open("documentData.xaml", FileMode.Open))
{
try
{
FlowDocument doc = XamlReader.Load(fStream) as FlowDocument;
this.myDocumentReader.Document = doc;
}
catch(Exception ex) {MessageBox.Show(ex.Message, "Error Loading Doc!");}
}
};
}
```

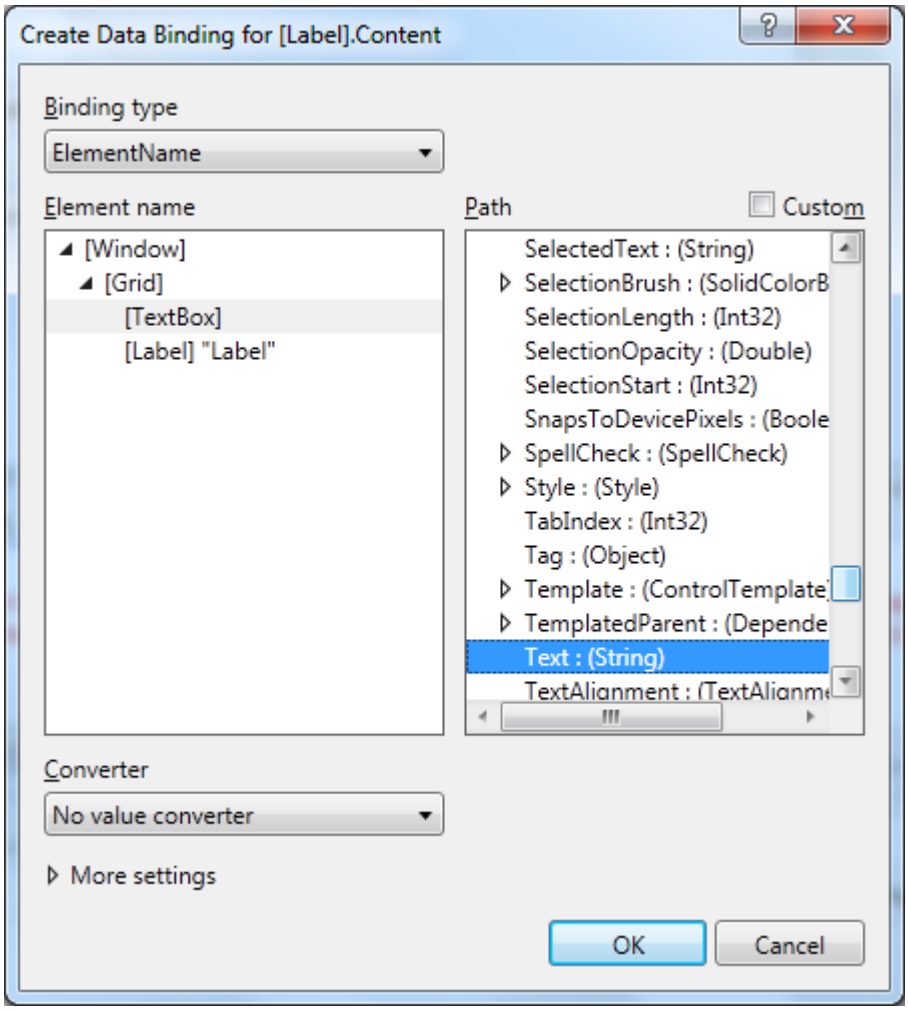
Databinding

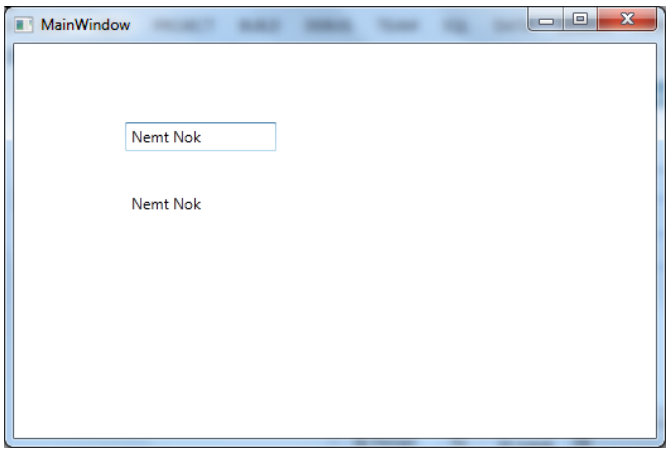
- Label
- Find Content (under Common): Click the rectangle





- Create Data binding


Create Data Binding for [Label].Content
Binding type
ElementName
Element name
Path
Custom
[Window]
[Grid]
[TextBox]
[Label] "Label"
SelectedText : (String)
SelectionBrush : (SolidColorB
SelectionLength : (Int32)
SelectionOpacity : (Double)
SelectionStart : (Int32)
SnapsToDevicePixels : (Boole
SpellCheck : (SpellCheck)
Style : (Style)
TabIndex : (Int32)
Tag : (Object)
Template : (ControlTemplate
TemplatedParent : (Depende
Text : (String)
Converter
No value converter
More settings
OK
Cancel


MainWindow
Nemt Nok
Nemt Nok

- XAML:

```
<Window x:Class="WPFDataBindExample.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">
    <Grid>
        <TextBox x:Name="textBox" HorizontalAlignment="Left" Height="23" Margin="88,62,0,0"
TextWrapping="Wrap" Text="TextBox" VerticalAlignment="Top" Width="120"/>
        <Label Content="{Binding Text, ElementName=textBox}" HorizontalAlignment="Left"
Margin="88,113,0,0" VerticalAlignment="Top" Width="120"/>

    </Grid>
</Window>
```

Example with a DataGrid

```
public class Author
    {
        public int ID { get; set; }
        public string Name { get; set; }
        public DateTime DOB { get; set; }
        public string BookTitle { get; set; }
        public bool IsMVP { get; set; }
    }
```

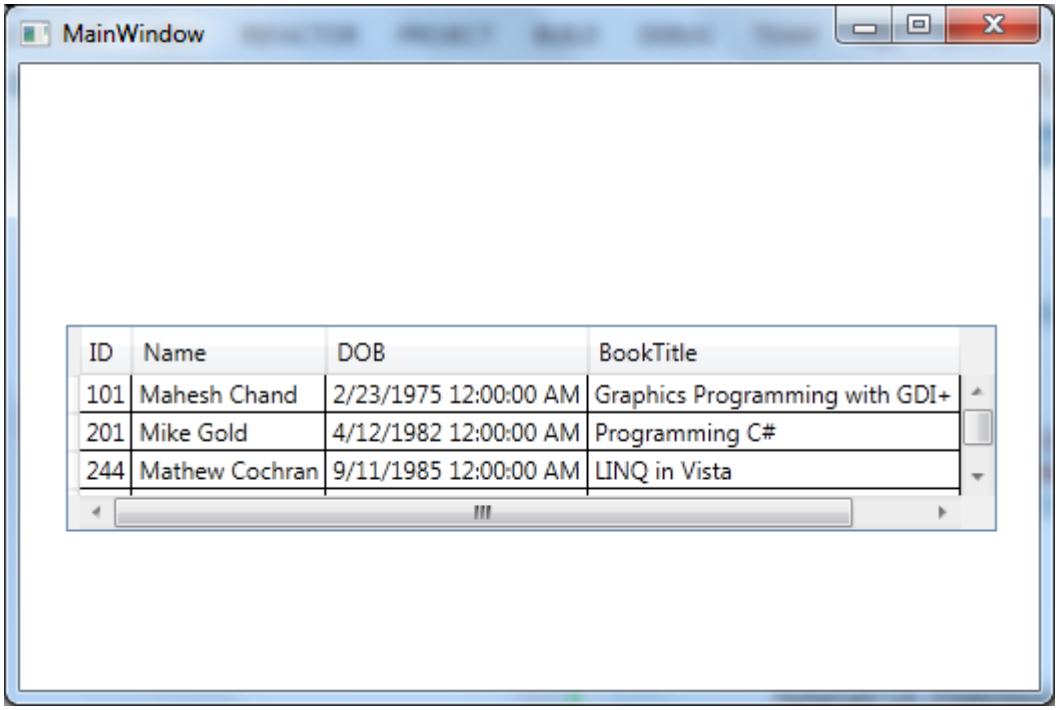
```
<Window
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:local="clr-namespace:WPFDataBindExample"
x:Class="WPFDataBindExample.MainWindow"
        Title="MainWindow" Height="350" Width="525">
    <Grid>
        <DataGrid Name="MyDataGrid" HorizontalAlignment="Left" Margin="23,130,0,0"
VerticalAlignment="Top" Width="465" Height="103">


        </DataGrid>


    </Grid>

</Window>
```

- Code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace WPFDataBindExample
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            SetGridData();
        }

        public void SetGridData()
        {
            List<Author> authors = new List<Author>();
            authors.Add(new Author()
            {
                ID = 101,
                Name = "Mahesh Chand",
                BookTitle = "Graphics Programming with GDI+",
                DOB = new DateTime(1975, 2, 23),
                IsMVP = false
            });
            authors.Add(new Author()

            {
                ID = 201,
                Name = "Mike Gold",
                BookTitle = "Programming C#",
                DOB = new DateTime(1982, 4, 12),
                IsMVP = true

            });

            authors.Add(new Author()

            {
                ID = 244,
                Name = "Mathew Cochran",
                BookTitle = "LINQ in Vista",
                DOB = new DateTime(1985, 9, 11),
                IsMVP = true

            });


            MyDataGrid.ItemsSource = authors;
```

```
        }
    }
}
```



Binding XML Documents to UI elements

- If an XML file with data is used
- Content is showed in an UI element

```
        XmlDataProvider provider = new XmlDataProvider();


          if (provider != null)

           {

             System.Xml.XmlDocument doc = new System.Xml.XmlDocument();

             doc.Load(fileName);

             provider.Document = doc;
```

```
            provider.XPath = "/opml/body/outline";

            myTreeView.DataContext = provider;
```

Dependency Properties

- A depency property can be set in XAML or via programming
- Encapsulates data and can be
  - Read-only
  - Write-only
  - Read-Write

- Example

```
…
<!-- Set three dependency properties! -->
<Button x:Name = "btnMyButton" Height = "50" Width = "100" Content = "OK"/>
…
```

- What is new here?
  - A Dependency Property must have a DependencyObject in inheritance chain
  - It is public, static and read-only
  - It is registred via Dependency.Register()
  - A CLR Property calls the DependencyObject to get and set values

- Advantages:
- Can inherit values from parent element
- Allows attached properties (can set values in other elements)
- Can calculate values based on multiple external values (example: animation)
- Supports triggers

- Example:

```
private void SetBindings()
{
Binding b = new Binding();
b.Converter = new MyDoubleConverter();
b.Source = this.mySB;
b.Path = new PropertyPath("Value");
// Specify the dependency property!
this.labelSBThumb.SetBinding(Label.ContentProperty, b);
}
```

Example

```
// FrameworkElement is-a DependencyObject.
public class FrameworkElement : UIElement, IFrameworkInputElement,
IInputElement, ISupportInitialize, IHaveResources, IQueryAmbient
{
...
// A static read-only field of type DependencyProperty.
public static readonly DependencyProperty HeightProperty;
// The DependencyProperty field is often registered
// in the static constructor of the class.
static FrameworkElement()
{
...
HeightProperty = DependencyProperty.Register(
"Height",
typeof(double),
typeof(FrameworkElement),
new FrameworkPropertyMetadata((double) 1.0 / (double) 0.0,
FrameworkPropertyMetadataOptions.AffectsMeasure,
new PropertyChangedCallback(FrameworkElement.OnTransformDirty)),
// The CLR wrapper, which is implemented using
// the inherited GetValue()/SetValue() methods.
public double Height
{
get { return (double) base.GetValue(HeightProperty); }
set { base.SetValue(HeightProperty, value); }
```

```
}
}
```

- Demands extra code compared to normal properties
- FrameworkElement is-a DependencyObject
- Notice FrameworkPropertyMetaData
    - Describes callbacks
    - Describes also FrameworkPropertyMetaDataOptions
        - For example: Can it use data-binding, can it be inherited?
- Height usesGetValue og SetValue
- Is registred

- FrameworkPropertyMetaData
    - Look like this:

```
new FrameworkPropertyMetadata(
// Default value of property.
(double)0.0,
// Metadata options.
FrameworkPropertyMetadataOptions.AffectsMeasure,
// Delegate pointing to method called when property changes.
new PropertyChangedCallback(FrameworkElement.OnTransformDirty)
)
```

- Last parameter to Register is validation
- Example:

```
private static bool IsWidthHeightValid(object value)
{
double num = (double) value;
return ((!DoubleUtil.IsNaN(num) && (num >= 0.0))
&& !double.IsPositiveInfinity(num));
}
```

- Is wrapped to a CLR property

```
public double Height
{
```

```
get { return (double) base.GetValue(HeightProperty); }
set { base.SetValue(HeightProperty, value); }
}
```

- Normal only GetValue and SetValue in wrapper – not validation

XAML

- WPF runtime can nu use the property

```
…
<Button x:Name="myButton" Height="100" .../>
…
```

- Functionality CAN NOT be called in XAML WPF

```
<!-- Nope! Can't call methods in WPF XAML! -->
<Button x:Name="myButton" this.SetValue("100") .../>
```

Building your own Dependency Property

- Create a User Control

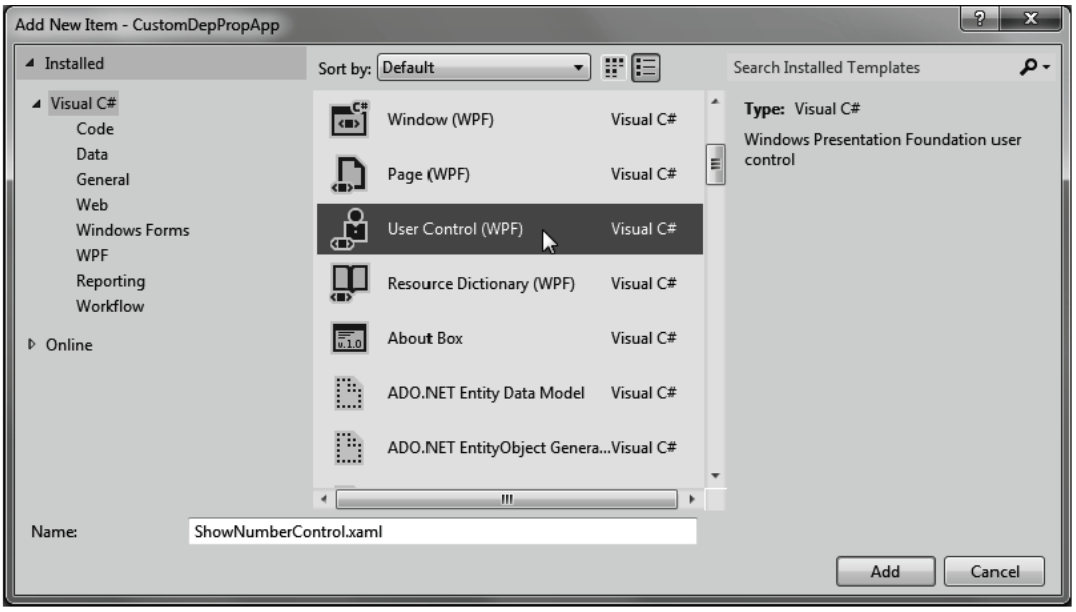


Figure 31-1. *Inserting a new custom UserControl*

- XAML:

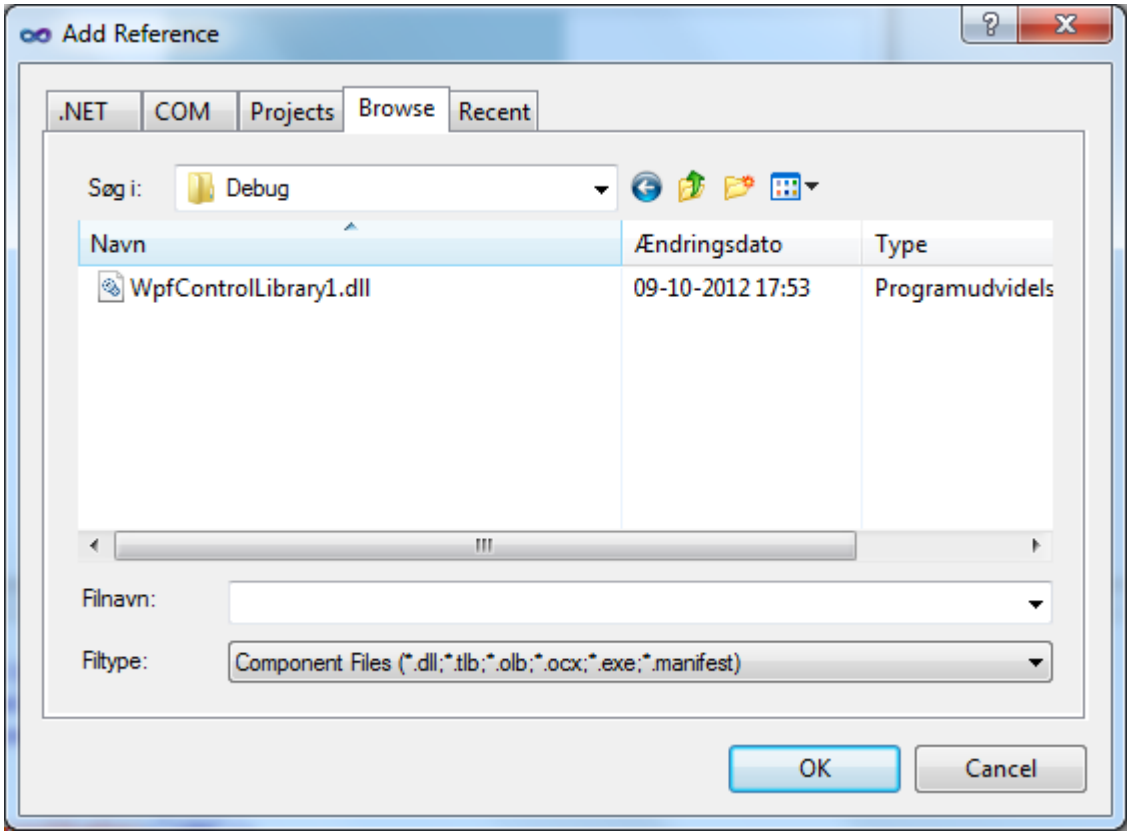
```
<UserControl x:Class="WpfControlLibrary1.UserControl1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
mc:Ignorable="d"
d:DesignHeight="300" d:DesignWidth="300">
    <Grid>
        <Label x:Name="numberDisplay" Height="50" Width="200" Background="LightBlue"/>
    </Grid>
</UserControl>
```

- In the cs code; create a (normal) property:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace WpfControlLibrary1
{
    /// <summary>
    /// Interaction logic for UserControl1.xaml
    /// </summary>
    public partial class UserControl1 : UserControl
    {
        public UserControl1()
        {
            InitializeComponent();
        }

        private int currNumber = 0;
        public int CurrentNumber
        {
            get { return currNumber; }
            set
            {
                currNumber = value;
                numberDisplay.Content = CurrentNumber.ToString();
            }
        }
    }
}
```

- Compile

- Create a normal user WPF project:
- ”Add Reference” to the control DLL



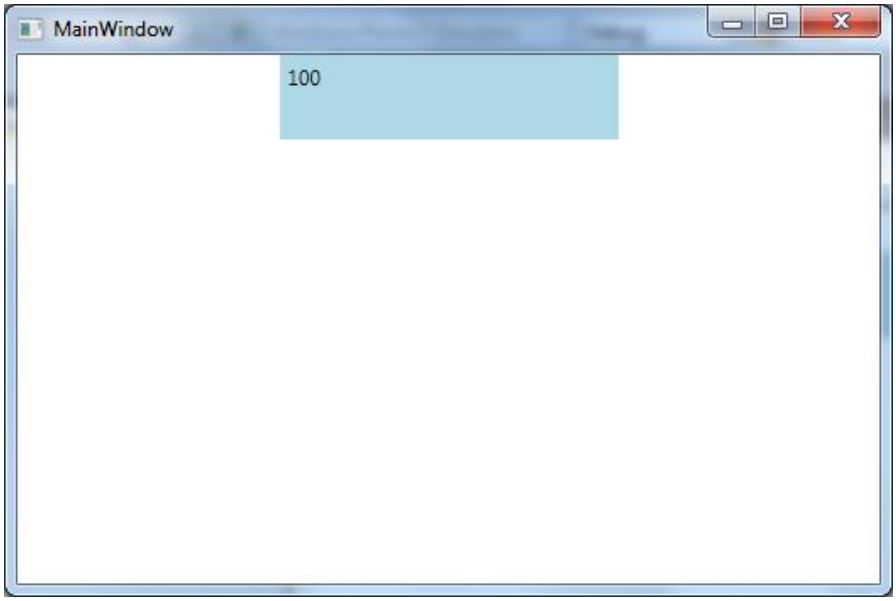
- Import xmls:Name = Namespace name; assembly=” ”

```
<Window x:Class="DependencyPropTest.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:myCtrls="clr-namespace:WpfControlLibrary1;assembly=WpfControlLibrary1"
        Title="MainWindow" Height="350" Width="525">

</Window>
```

- Place for instance a StackPanel containing myCtrls:UserControl1

```
<Window x:Class="DependencyPropTest.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:myCtrls="clr-namespace:WpfControlLibrary1;assembly=WpfControlLibrary1"
        Title="MainWindow" Height="350" Width="525">
    <StackPanel>
        <myCtrls:UserControl1 x:Name="myShowNumberCtrl" CurrentNumber="100"/>
    </StackPanel>
</Window>
```

- Output:



- Animation

- Change the UserControl1 application code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
```

```
namespace WpfControlLibrary1
{
    /// <summary>
    /// Interaction logic for UserControl1.xaml
    /// </summary>
    public partial class UserControl1 : UserControl
    {
        public UserControl1()
        {
            InitializeComponent();
        }

        private int currNumber = 0;
        public int CurrentNumber
        {
            get { return (int)GetValue(CurrentNumberProperty); }
            set { SetValue(CurrentNumberProperty, value); }
        }
        public static readonly DependencyProperty CurrentNumberProperty =
            DependencyProperty.Register("CurrentNumber",
                                        typeof(int),
                                        typeof(UserControl1),
                                        new UIPropertyMetadata(0));


    }
}
```

- Now animation can be added in the user project

```
<Window x:Class="DependencyPropTest.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:myCtrls="clr-namespace:WpfControlLibrary1;assembly=WpfControlLibrary1"
        Title="MainWindow" Height="350" Width="525">
    <StackPanel>
        <myCtrls:UserControl1 x:Name="myShowNumberCtrl" CurrentNumber="100">
            <myCtrls:UserControl1.Triggers>
                <EventTrigger RoutedEvent = "myCtrls:UserControl1.Loaded">
                    <EventTrigger.Actions>
                        <BeginStoryboard>
                            <Storyboard TargetProperty = "CurrentNumber">
                                <Int32Animation From = "100" To = "200" Duration = "0:0:10"/>
                            </Storyboard>
                        </BeginStoryboard>
                    </EventTrigger.Actions>
                </EventTrigger>
            </myCtrls:UserControl1.Triggers>
        </myCtrls:UserControl1>
    </StackPanel>
</Window>
```

- Adding data validation?
- See page 1057