ETH Zürich
Chair of Software Engineering

Java and C# in depth

Carlo A. Furia, Marco Piccioni, and Bertrand Meyer

C#: Graphical User Interfaces (GUI)

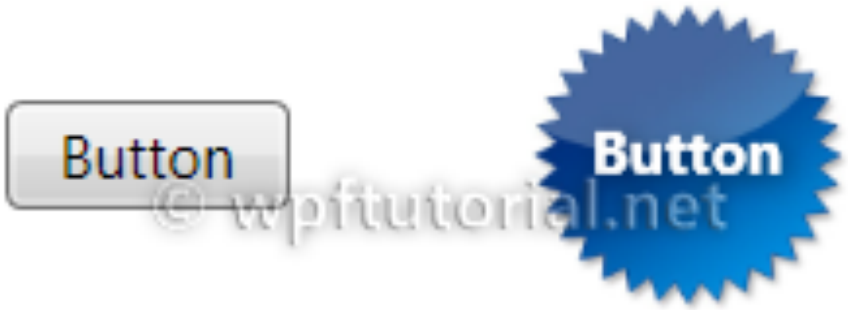
With material from Christoph Angerer

Windows Presentation Foundation (WPF)

- 2D/3D vector-based graphics, resolution independent, rendering using HW acceleration of graphic cards (Direct 3D)
- Text, typography, documents, multimedia
- Declarative UI with XAML
- Styles, templates for declarative customization
- Data binding
- Separate behavior with code-behind
- Needs .NET 3.0+

Controls

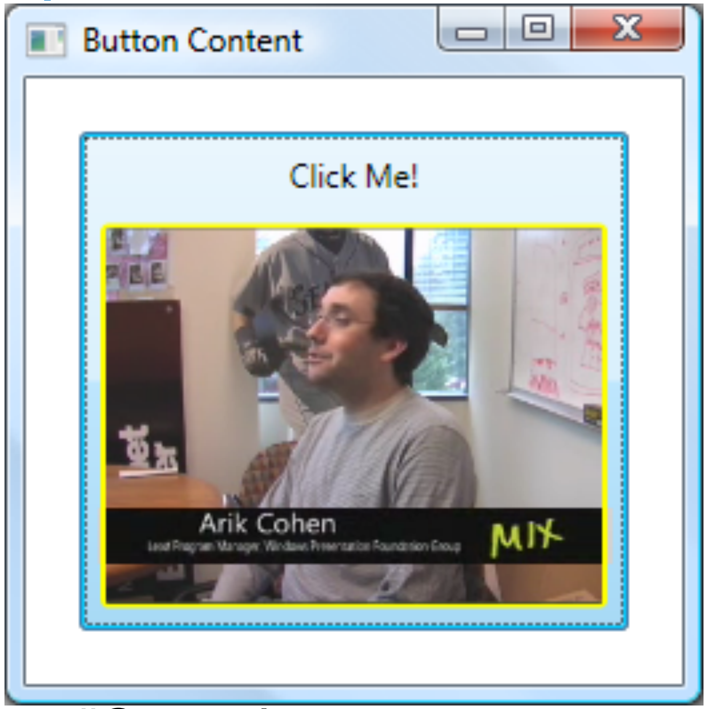
- WPF classes hosted by a window or document, having a UI and behavior
- Created using XAML or code
- Customizable using `ControlTemplate`



http://wpftutorial.net

Content Model

- The type and number of items that constitute the content of a control
- Some controls have just an item and type of content (e.g. `TextBox` has a string as `Text`)
- Other controls can contain multiple items of different types (e.g. `Button`)



http://msdn.microsoft.com/en-US/library/aa970268#Controls

XAML

- XML file that allows creating GUIs declaratively
  - XML elements map to objects
  - XML attributes map to properties and events
- Used to generate code connected to the code-behind file

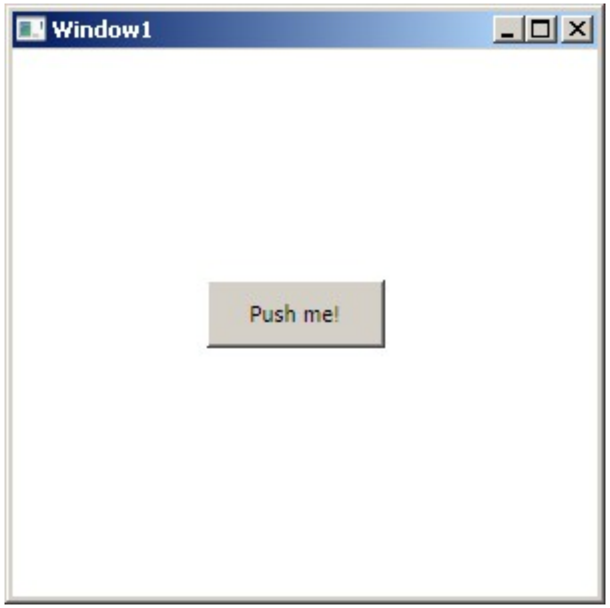
XAML file for sample app (VS 2012)

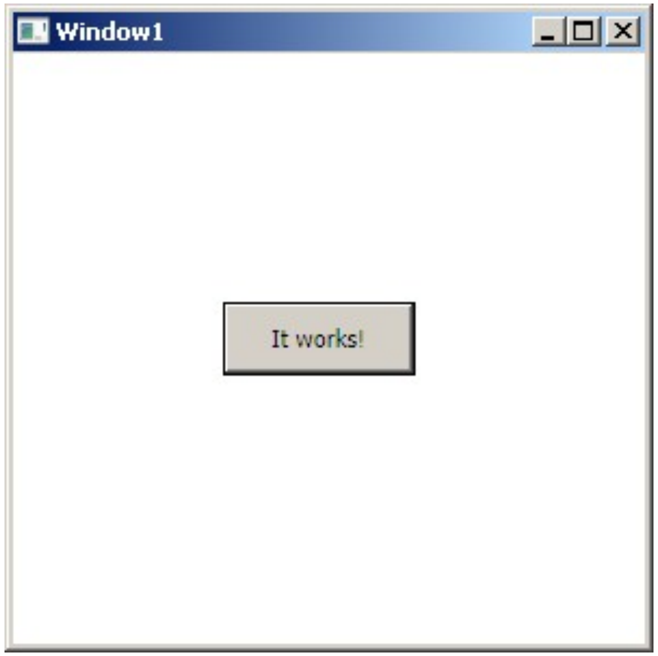
File MainWindow.xaml

```
<Window x:Class="WpfApplication1.MainWindow"
xmlns="
http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title=″MainWindow" Height=″350" Width=“400">
<Grid>
  <Button x:Name=“button1″ Content=“Push me!″
HorizontalAlignment=“Left″ Margin=″159,271,0,0″
VerticalAlignment=“Top″ Width=“75″ Click=“ButtonClick_1″/>
</Grid>
</Window>
```

Code-behind file for sample app

```
using System.Windows
namespace WpfApplication1
// Interaction logic for MainWindow.xaml
public partial class MainWindow : Window{
     public MainWindow(){
// Merges UI markup with code in this class,
//sets properties and registers event handlers
          InitializeComponent();
     }
     private void Button_Click_1 (object sender,
                                      RoutedEventArgs e){
          button1.Content = "It works!";
     }}
```


Window1
Push me!


Window1
It works!

Windows Presentation Foundation (WPF)

- The **`Window`** class is used for standalone applications to create windows and dialogs
- The **`Application`** class encapsulates application-scoped services:
  - startup
  - lifetime management
  - shared properties
  - shared resources

XAML Application file (VS 2012)

File App.xaml

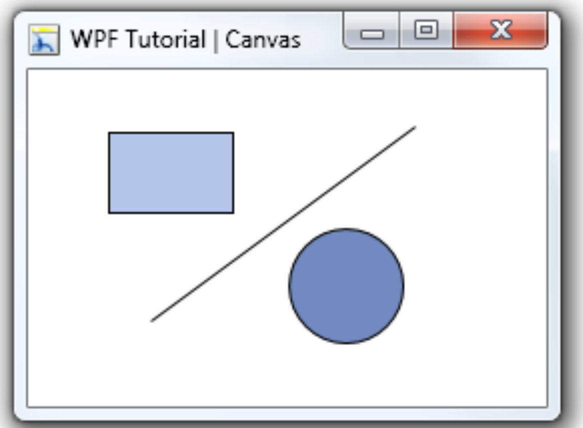
```
<Application x:Class="WpfApplication1.App"
xmlns="
http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:x="
http://schemas.microsoft.com/winfx/2006/xaml"
StartupUri="MainWindow.xaml">
     <Application.Resources>

     </Application.Resources>
</Application>
```

Layout

- Recursive system to size, position and draw a GUI element
- Measures and arranges a panel's children
- Uses component negotiation
  1. Control tells its parent required size/loc
  2. Parent tells control what space it can have
- WPF provides built-in layout panels

Sample Layout Panels: Canvas

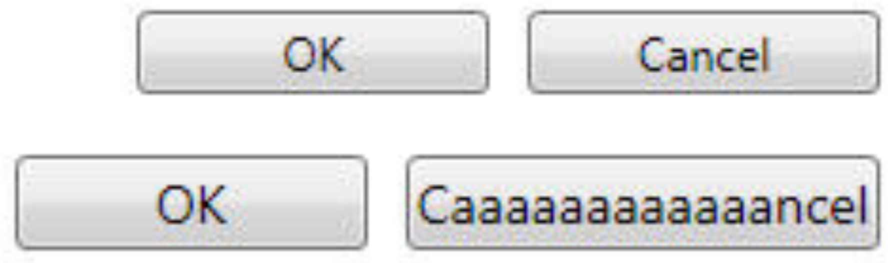
- Area within which you typically position 2D graphic elements by explicit relative coordinates
- Coordinates are relative to panel sides
- Z-order default of elements is as in XAML



http://wpftutorial.net

Sample Layout Panels: StackPanel

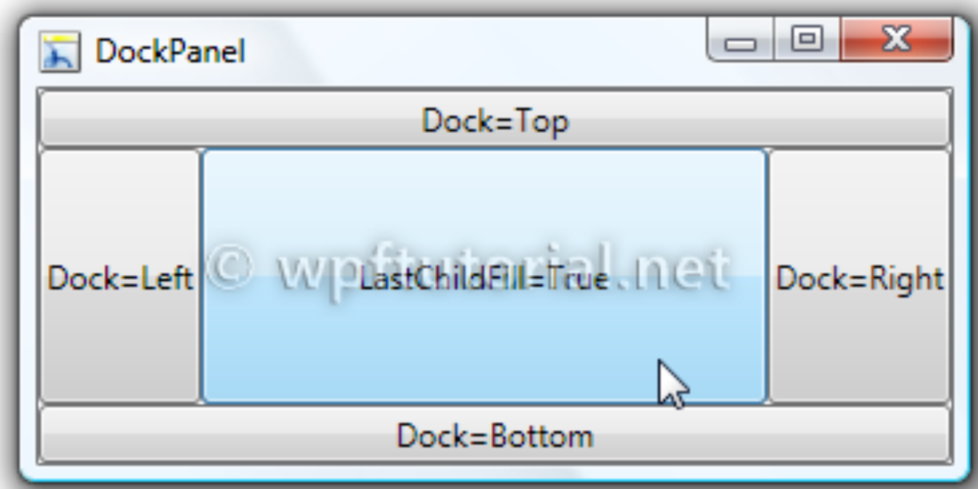
- Stacks child controls below or beside each other
- Useful for lists
- Used by ComboBox, ListBox, and Menu
- Controls automatically resize



http://wpftutorial.net

Sample Layout Panels: DockPanel

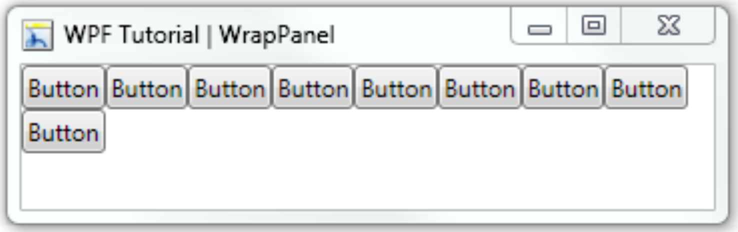
- Area within which you arrange children horizontally or vertically, relative to each other
- Child controls are aligned to the panel left, right, top, bottom and center (last control)

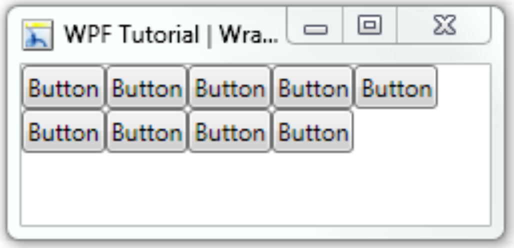


http://wpftutorial.net

Sample Layout Panels: WrapPanel

- Child controls are positioned sequentially from left to right
- Controls wrap to the next line when there is no more space in the line
- Similar to stackPanel but with wrapping

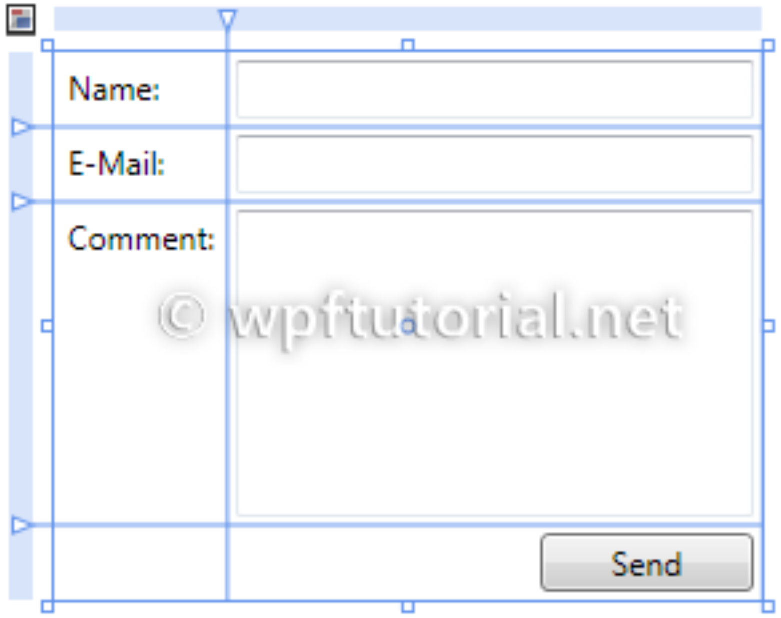



WPF Tutorial | Wra...
Button Button Button Button Button
Button Button Button Button

http://wpftutorial.net

Sample Layout Panels: Grid

- Child controls are positioned by rows and columns
- A cell can contain multiple controls
- A control can span over multiple cells
- Controls can overlap



http://wpftutorial.net

Dependency Properties

Dependency Properties (DPs) in a nutshell

- Provide a functionality extension to .NET properties
- Allow computing the property value using the values of other inputs (e.g. themes, user prefs, data binding, animations)
- Can implement validation, defaults, callbacks, and in general allow dynamic behavior
- From the user point of view they feel like .NET props

Dependency Properties abstractions

- DPs are backed by type `DependencyProperty`
  - enables registration of DPs
  - provides identification and info about the DP
  - as a base class enables objects to use DPs
- `DependencyObject` enables WFS's props system
  - base class that hosts the property
  - stores the property returned by `DependencyProperty.Register`
  - provides get, set, clear utility methods
  - handles prop changed notifications and callbacks

Setting and getting DPs

- While .NET properties read from private members, DPs are resolved dynamically when calling `GetValue()` inherited from `DependencyObject`

- DPs are set locally in a dictionary of keys and values in a `DependencyObject`
  - the key of an entry is the name of the property
  - the value is the value to set

Dependency Property example

...in class `DependencyObject...`

```
public static readonly DependencyProperty
IsRotatingProperty =
    DependencyProperty.Register(
    "IsRotating", typeof(Boolean),
//resource refs, callbacks, styles,
animations…
    );
public bool IsRotating{
    get { return
(bool)GetValue(IsRotatingProperty); }
    set { SetValue(IsRotatingProperty,
value); }}
```

Dynamic Resolution of DPs

DP values are resolved internally by following the precedence from top to bottom:

1. Animation
2. Binding expression
3. Local value
4. Custom style trigger
5. Custom template trigger
6. Custom style setter
7. Default style trigger
8. Default style setter
9. Inherited value
10. Default value

Dependency Property value precedence

- The value you get from a DP was potentially set by any other property-based input participating in the property system
- The value precedence (see previous slide) helps to have predictable interactions
- E.g. apply a style to all buttons' background props, but use locally set background for just one button (b1)
  - b1: property set twice, but only the locally set value counts because has precedence over style setter
  - all other buttons: style setter applies

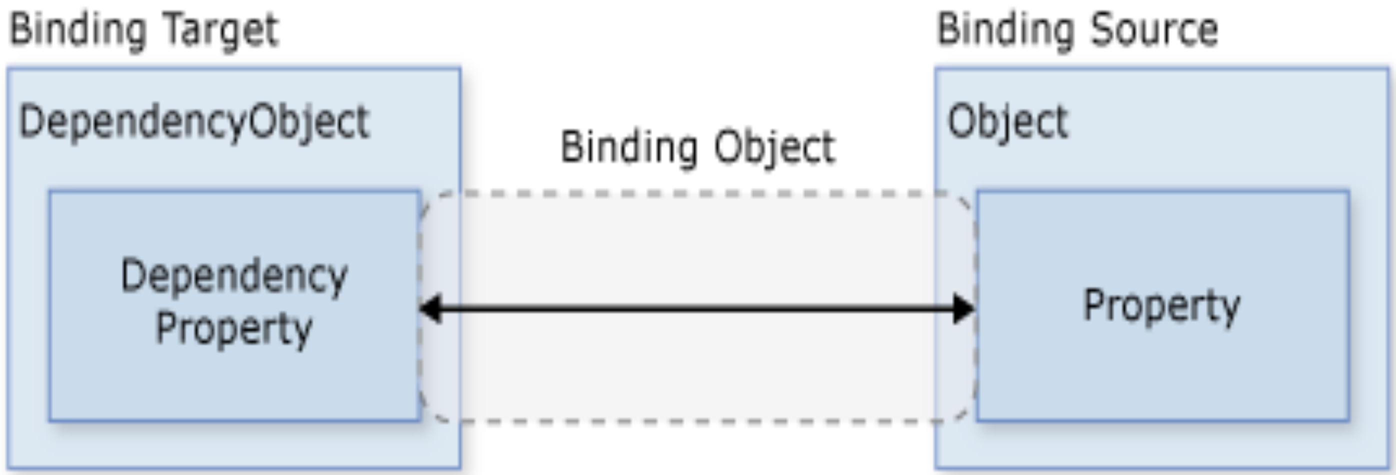
Advantages of Dependency Properties

- Reduced memory footprint
  Over 90% of the properties of a UI control typically stay at their initial values. DPs only store modified properties in the instance. The default values are stored once within the DP
- Value inheritance
  Provide the way to compute the value of a property based on the value of other inputs (see previous slide)
- Change notification
  DPs have a built-in change notification mechanism

Data Binding

- An usage scenario for DPs
- A way to automatically update data between GUI and business model using DPs
- It works in either direction, and in both as well
- It is the bridge between a binding target and a binding source
- The `Binding` class is the core element
- The `BindingExpression` class maintains the connection between the source and the target

Data Binding components

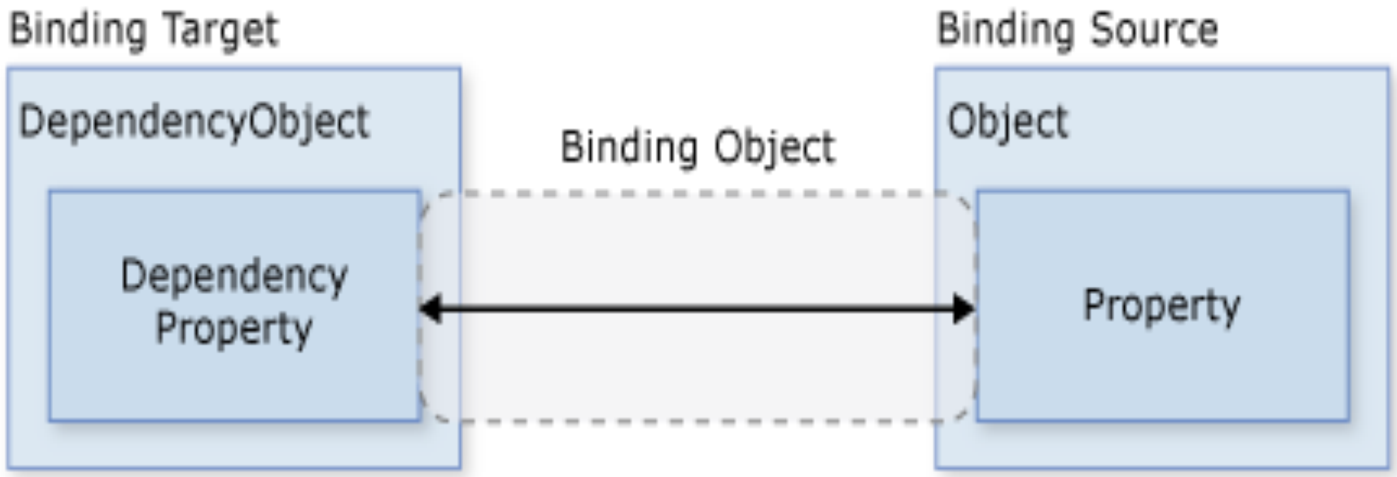


http://msdn.microsoft.com/en-US/library/aa970268#Data_Binding

Main components of the binding

- Binding target object
- Target property (must be a DP)
- Binding source object
- Path to value in the binding source to use

Data Binding example



Target object binding: TextBox

Target object DP: TextBox.Text

Source object binding Person

Path: Person.Name

http://msdn.microsoft.com/en-US/library/aa970268#Data_Binding

Data Binding example XAML

Typically done in XAML using the `{Binding}` markup

```
. . .
<!-- Bind the TextBox to the data source
 (TextBox.Text to Person.Name) -->
<TextBox Name="personNameTextBox"
Text="{Binding Path=Name}" />
. . .
```

Data Binding example code behind

```
public partial class DataBindingWindow :
Window {
 public DataBindingWindow()
 {
     InitializeComponent();
   // Create Person data source
   // Assuming Person has property Name
     Person person = new Person();
   // Make data source available for binding
    this.DataContext = person;
 }}}
```

Some Mono GUI toolkits

Gtk# 2.0 http://www.mono-project.com/GtkSharp

multi-platform, binds Gtk+ and GNOME libs, written in C with OO API, visual designer (Mono Develop)

Winforms http://www.mono-project.com/WinForms

compatible with Windows.Forms

Xamarin.Mac http://xamarin.com/mac

to build native Cocoa apps in C#