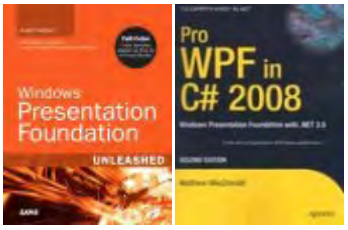
Lectures on WPF
Commands



This lecture is based on the books of Nathan&Lehenbauer and MacDonald:

Why Commands ?

Small programs call their functionality directly via their event handlers, but sophisticated programs should code it separately from the event handlers in functional tasks and insert an intermediary level between the event handlers and the functional tasks.
Commands or more precisely **Routed Commands** serve as such an intermediary layer separating design from logic what means separating the work of user interface designers from the work of software developers.
It frees the designer to give user friendly names to as many UIElements as he wants and to call the same software from many different places of the user interface, the keyboard and from the internet.
The independent software developer can give his preferred names to the event handlers and to its tasks independently to how the user interface looks.
Routed Commands bind the user interface to the software and help to resolve the usual conflicts, when the user isn't allowed to click all buttons at once.
The benefits of the intermediary routed command level are:
1. It keeps track of state of the task (executable or not?)
2. It informs any involved UI element to enable, disable, appear or hide to stay synchronized with the state of the task by firing and managing `CanExecuteChanged`-events = Command Routing.
3. It bundles and delegates several UI elements and keyboard inputs to a task.

All prefabricated commands that WPF offers have no functionality of their own.
They are empty hulls that can be filled with arbitrary event handlers.
They just provide a name and some short-cut-keys but nothing else.
There are 4 libraries containing about 100 ready to use command objects:

ApplicationCommands	`Copy, Cut, Paste, New, Open, Save, SaveAs, Print, ...`
ComponentCommands	`MoveDown, MoveUp, MoveLeft, MoveRight, ScrollPageDown, ...`
EditingCommands	`AlignCenter, Backspace, Delete, EnterLineBreak, ...`
MediaCommands	`Play, Stop, FastForward, ChannelUp, ChannelDown, BoostBass, ...`

Sample 1

This tiny sample demonstrates how a `MenuItem` and a `Button` and the keys Ctrl+N trigger
the same `ApplicationCommands.New`.
The following "`New`"-Command event handler `NewOnExecute` starts nothing but a `MessageBox`.

```
<Window x:Class="CommandsDemo"
 xmlns=  "http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Commands Sample 1" Height="128" Width="256">
 <Window.CommandBindings>
  <CommandBinding Command="New" Executed="NewOnExecute"/>
 </Window.CommandBindings>
 <StackPanel >
  <Menu>
   <MenuItem Header="File">
    <MenuItem Command="New"/>
   </MenuItem>
  </Menu>
  <Button Margin="5" Padding="5" Content="New" Command="New"/>
 </StackPanel>
</Window>
/*************************************************************************/
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
public partial class CommandsDemo : Window
{ [STAThread] public static void Main() { new Application().Run( new CommandsDemo() ); }
  public CommandsDemo() { InitializeComponent(); }
  private void NewOnExecute( object sender, ExecutedRoutedEventArgs args )
  { MessageBox.Show( "New command comes from: " + args.OriginalSource.ToString() ); }
}
```

Sample 2

This sample demonstrates how a `MenuItem` and a `Button` and the keys Ctrl+N trigger
the `ApplicationCommands.New`, whereas a second `Button` switches
the `ApplicationCommands.New`-sources on and off via an `ApplicationCommands.NotACommand`.

```
<Window x:Class="CommandsDemo"
 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Commands" Height="120" Width="300">
 <Window.CommandBindings>
  <CommandBinding Command="New"         Executed  ="doSomethingOnExecute"
                                        CanExecute="doSomethingCanExecute"/>
  <CommandBinding Command="NotACommand" Executed  ="switchOnExecute"/>
  <CommandBinding Command="Close"       Executed  ="CloseOnExecute"/>
 </Window.CommandBindings>
 <StackPanel>
  <Menu>
   <MenuItem Header="File">
    <MenuItem Command="New"   Header="Do Something"/>
    <Separator/>
    <MenuItem Command="Close" Header="Close"/>
   </MenuItem>
  </Menu>
  <Button Command="New"         Width="200" Height="25">Do Something           </Button>
  <Button Command="NotACommand" Width="200" Height="25">Switch Something On/Off</Button>
 </StackPanel>
</Window>
```

```
/*****************************************************************************/
using System;
using System.Windows;
using System.Windows.Input;
public partial class CommandsDemo : System.Windows.Window
{ [STAThread] public static void Main() { new Application().Run( new CommandsDemo() ); }
  Boolean enableFlag = true;
  public CommandsDemo() { InitializeComponent(); }
  private void doSomethingOnExecute( object sender, ExecutedRoutedEventArgs args )
  { MessageBox.Show( args.OriginalSource.ToString() + " fired !" ); }
  private void doSomethingCanExecute( object sender, CanExecuteRoutedEventArgs args )
  { args.CanExecute = enableFlag; }
  private void switchOnExecute( object sender, ExecutedRoutedEventArgs args )
  { enableFlag = !enableFlag; } //toggle flag
  private void CloseOnExecute( object sender, ExecutedRoutedEventArgs args )
  { string s = args.OriginalSource.ToString() + " wants to close ?";
    MessageBoxResult res = MessageBox.Show( s, "Test", MessageBoxButton.YesNo );
    if (res == MessageBoxResult.Yes) this.Close();
  }
}
```

Sample 3

This sample lets you copy images and texts into the clipboard and reload them from the clipboard.
Its most interesting part is the `PasteCanExecute`-function. It automatically switches off those `Paste`-buttons that cannot handle the current clipboard content.
Experiment: Try out this program by loading text and images from arbitrary other programs into the clipboard. Whenever you will return to this program, only those `Paste`-buttons will work that fit to the foreign clipboard content.

```
<Window x:Class="CommandsDemo"
 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 Title="Commands Sample 3" Height="400" Width="528">
 <Window.Resources>
  <Style TargetType="Button">
   <Setter Property="Width"  Value="128"/>
   <Setter Property="Height" Value= "24"/>
  </Style>
 </Window.Resources>
 <Window.CommandBindings>
  <CommandBinding Command="Copy"  Executed="CopyOnExecute"  CanExecute="CopyCanExecute" />
  <CommandBinding Command="Paste" Executed="PasteOnExecute" CanExecute="PasteCanExecute"/>
 </Window.CommandBindings>
 <StackPanel Orientation="Vertical">
  <StackPanel Orientation="Horizontal">
   <Image x:Name="image1" Source="http://www.miszalok.de/Images/Modonna.bmp"  Width="128"/>
   <Image x:Name="image2" Source="http://www.miszalok.de/Images/tinyFoto.jpg" Width="128"/>
   <TextBox x:Name="textbox1" Text="Plato"       Width="128"/>
   <TextBox x:Name="textbox2" Text="Aristoteles" Width="128"/>
  </StackPanel>
  <StackPanel Orientation="Horizontal">
   <Button x:Name="Copy1"  Content="Copy to Clipboard" Command="Copy"/>
   <Button x:Name="Copy2"  Content="Copy to Clipboard" Command="Copy"/>
   <Button x:Name="Copy3"  Content="Copy to Clipboard" Command="Copy"/>
   <Button x:Name="Copy4"  Content="Copy to Clipboard" Command="Copy"/>
  </StackPanel>
  <Label Height="20"/><!--empty space-->
  <StackPanel Orientation="Horizontal">
   <Button x:Name="Paste1" Content="Clear Clipboard"   Command="Paste"/>
   <Button x:Name="Paste2" Content="Paste to TextBox"  Command="Paste"/>
   <Button x:Name="Paste3" Content="Paste to Image"    Command="Paste"/>
   <Button x:Name="Paste4" Content="Paste to TitleBar" Command="Paste"/>
  </StackPanel>
  <StackPanel Orientation="Horizontal">
   <Label                  Width="128"/><!--empty space-->
   <TextBox x:Name="textbox3" Width="128"/>
   <Image   x:Name="image3"   Width="128"/>
   <Label                  Width="128"/><!--empty space-->
  </StackPanel>
 </StackPanel>
</Window>
```

```
/*****************************************************************************/
using System;
using System.Windows;
using System.Windows.Controls;

using System.Windows.Media.Imaging;
public partial class CommandsDemo : Window
{ [STAThread] public static void Main() { new Application().Run( new CommandsDemo() ); }
  public CommandsDemo() { InitializeComponent(); }
  private void CopyOnExecute(object sender, ExecutedRoutedEventArgs args)
  { Button b = (Button)args.OriginalSource;
    switch ( b.Name )
    { case "Copy1": Clipboard.SetImage( (BitmapSource)image1.Source ); break;
      case "Copy2": Clipboard.SetImage( (BitmapSource)image2.Source ); break;
      case "Copy3": textbox1.SelectAll(); textbox1.Copy();             break;
      case "Copy4": textbox2.SelectAll(); textbox2.Copy();             break;
    }
  }
  private void PasteOnExecute(object sender, ExecutedRoutedEventArgs args)
  { Button b = (Button)args.OriginalSource;
    switch ( b.Name )
    { case "Paste1": Clipboard.Clear();
                     textbox3.Clear(); image3.Source = null; break;
      case "Paste2": textbox3.Text = Clipboard.GetText();    break;
      case "Paste3": image3.Source = Clipboard.GetImage();   break;
      case "Paste4": Title         = Clipboard.GetText();    break;
    }
  }
  private void CopyCanExecute(object sender, CanExecuteRoutedEventArgs args)
  { args.CanExecute = true; //All Copy-buttons are always enabled
  }
  private void PasteCanExecute(object sender, CanExecuteRoutedEventArgs args)
  { Button b = (Button)args.OriginalSource;
    switch ( b.Name )
    { case "Paste1": args.CanExecute = Clipboard.ContainsImage() |
                                       Clipboard.ContainsText();  break;
      case "Paste2": args.CanExecute = Clipboard.ContainsText();  break;
      case "Paste3": args.CanExecute = Clipboard.ContainsImage(); break;
      case "Paste4": args.CanExecute = Clipboard.ContainsText();  break;
    }
  }
}
```