Module 4

”Events and Commands”

TEKNOLOGISK
INSTITUT

Agenda

- **Events**
- Commands

TEKNOLOGISK INSTITUT

WPF Trees

- Logical Tree
  - View in Visual Studio with
    - `View → Other Windows → Document Outline`
    - Bottom-left corner icon ☺
  - Essential for eventing

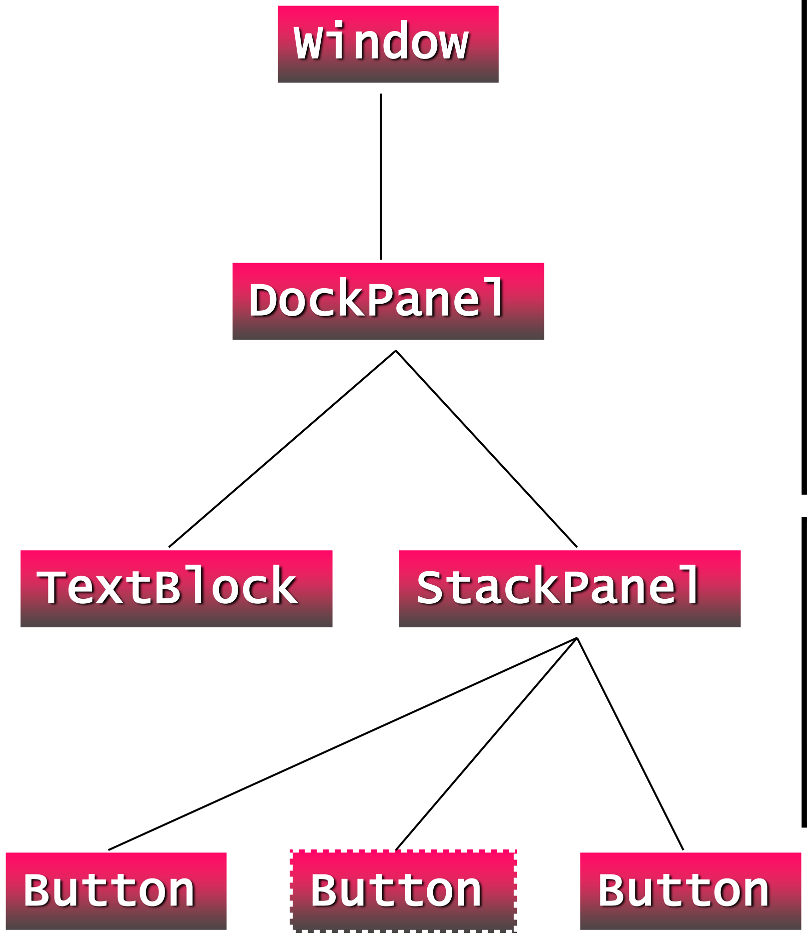
- Visual Tree
  - Elements deriving from `Visual` and `Visual3D`
  - View in Visual Studio with "WPF Tree Visualizer"
    - Access from Locals, Autos, or Watch window
  - Essential for styling and templating

Introducing Routed Events

- `RoutedEventArgs` and `RoutedEventHandler`
- Attached events
- Three types of routed events
  - Direct
  - Bubbling
  - Tunneling

TEKNOLOGISK
INSTITUT

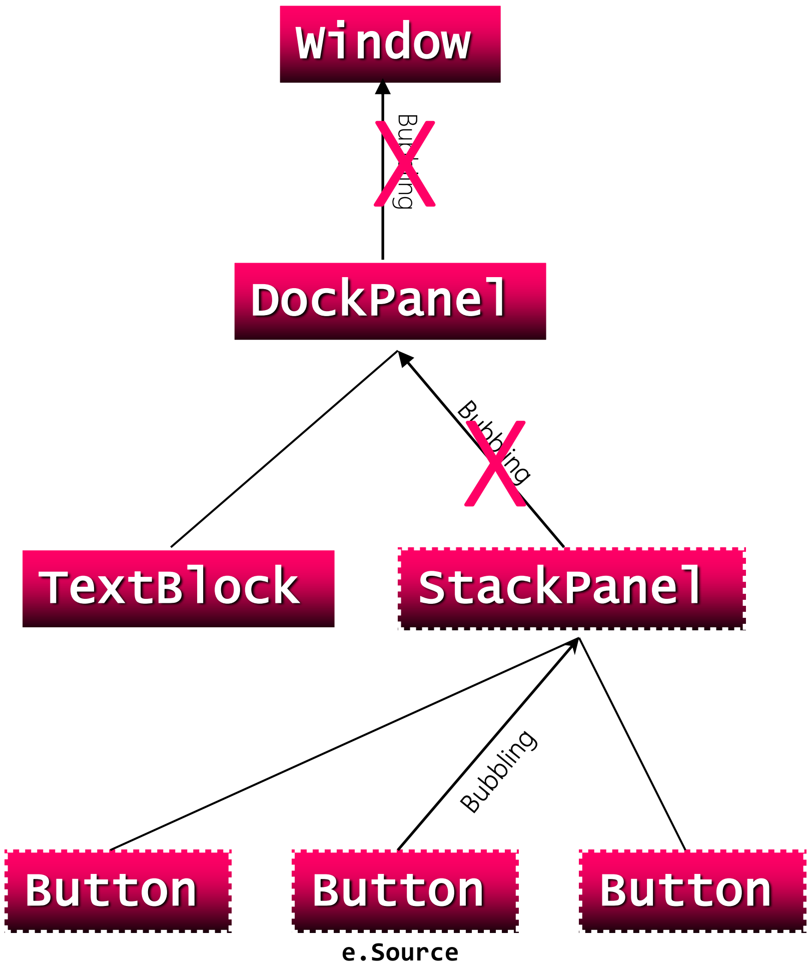
Direct Events



```
<Window Title="Routed Events">
   <DockPanel>
      <TextBlock>Event Routing</TextBlock>

      <StackPanel>
         <Button>One</Button>
         <Button MouseLeave="OnButtonLeave">
            Two
         </Button>
         <Button>Three</Button>
      </StackPanel>
   </DockPanel>
</Window>
```

```
private void OnButtonLeave(
   object sender, MouseEventArgs e )
{
   // Handle event...
}
```

Bubbling Events

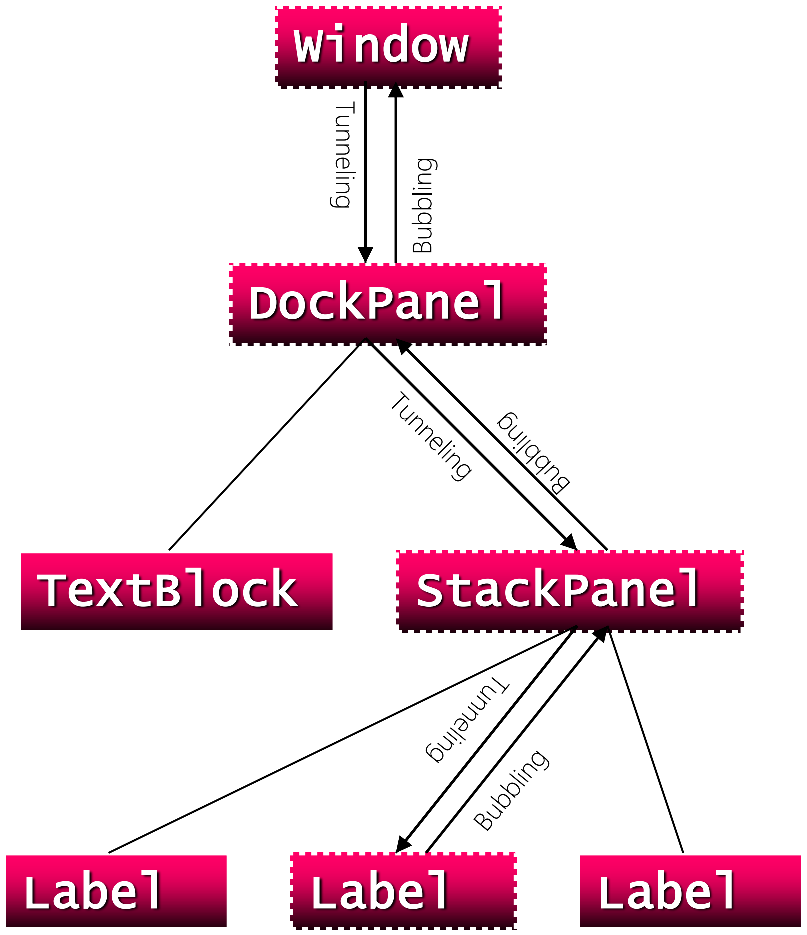


```
<Window Title="Routed Events">
   <DockPanel>
      <TextBlock>Bubbling Events</TextBlock>

      <StackPanel
         Button.Click="OnButtonClicked">
         <Button>One</Button>
         <Button>Two</Button>
         <Button>Three</Button>
      </StackPanel>
   </DockPanel>
</Window>
```

```
private void OnButtonClicked(
   object sender, RoutedEventArgs e )
{
   // Handle event
   ... e.Source ...

   e.Handled = true;
}
```

TEKNOLOGISK INSTITUT

Tunneling Events



- Events are paired
  - Tunneling ("**`Preview`***Event*")
  - Bubbling ("*Event*")
- Example:
  - `PreviewMouseDown`
  - `MouseDown`

```
PreviewMouseDown @ Window
PreviewMouseDown @ DockPanel
PreviewMouseDown @ StackPanel
PreviewMouseDown @ Label
MouseDown @ Label
MouseDown @ StackPanel
MouseDown @ DockPanel
MouseDown @ Window
```

A Few Words of Caution

- `RoutedEventArgs` properties
  - `Handled`
  - `Source` Control object raising event
  - `OriginalSource` Visual Tree object entailing event

- Some events interfere with each other
  - Click event interferes with (`Preview`)`MouseDown`

- Argh! Already handled events can still be handled…! ☺
  - But only programmatically
  - Bubbling and tunnelling continue
  - `UIElement.AddHandler()`
    - `handledEventsToo == true` in code!

`EventManager` Class

- `EventManager` class
  - `RegisterRoutedEvent()`
    - Creates new routed events
  - `RegisterClassHandler()`
    - Adds class-level event handlers

- `UIElement.RaiseEvent()`
  - Raises routed events

- Class-level event handling occurs <u>before</u> instance-level event handling

Application-Level Events

- `Application` events
  - `Startup`
  - `Exit`
  - `SessionEnding`
  - `Activated`
  - `Deactivated`
  - `DispatcherUnhandledException`
    - Not WPF-specific, but important in practice: `AppDomain.CurrentDomain.UnhandledException` event
- Added in `App.xaml`

Agenda

- Events
- **Commands**

Introducing Commands

- Commands are abstract, high-level event-style classes implementing **`ICommand`**
  - `Execute()` method
  - `CanExecute()` boolean method
  - `CanExecuteChanged` event
- Some controls implement **`ICommandSource`** to interact with commands
  - `Button`, `CheckBox`, `MenuItem`, …
- Built-in commands in five classes
  - `ApplicationCommands`, `ComponentCommands`, `MediaCommands`, `NavigationCommands`, `EditingCommands`
- Commanding is an essential ingredient in the MVVM pattern

Commands and Command Bindings

- `Command`
  - Can be invoked declaratively
  - Can be invoked programmatically
  - Can be invoked through input gestures
    - `MouseGesture`
    - `KeyGesture`
  - But nothing happens until command is bound

- `CommandBinding`
  - Binds commands to command handler
    - `Command`
    - `CanExecute`
    - `Executed`

- Commands bubble up the logical tree!

- Note: Parameters can be supplied to commands, if needed

Built-in Command Bindings

- Some controls have built-in command bindings
  - `TextBox`, …

- Bind via
  - `Command`
  - `ICommandSource.CommandTarget`
    - E.g. `Button`

- Note that command targets must be set with binding-syntax, i.e.

```
CommandTarget = "{Binding ElementName = textbox1}"
```

Custom Commands

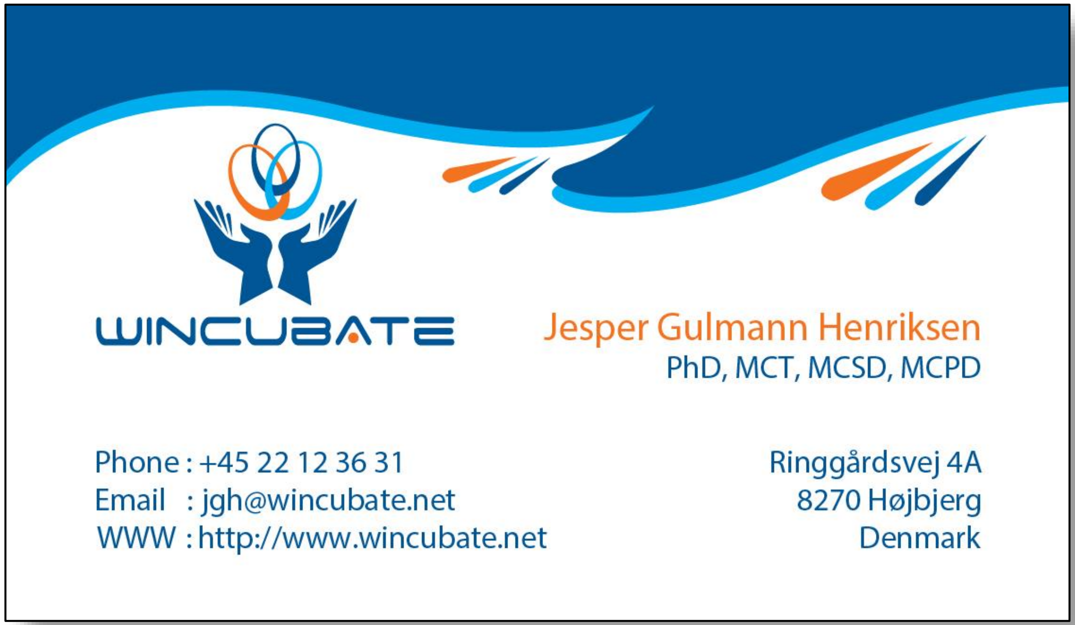
- `ICommand`
  - `RoutedCommand`
    - `RoutedUICommand` (adds localized **`Text`** property)

- Do custom commands in a **static** class by either
  - Implementing **`ICommand`** by hand, or
  - Using a **`Routed(UI)Command`**

Summary

- Events
- Commands

TEKNOLOGISK
INSTITUT



WINCUBATE

Jesper Gulmann Henriksen
PhD, MCT, MCSD, MCPD

Phone : +45 22 12 36 31
Email : jgh@wincubate.net
WWW : http://www.wincubate.net

Ringgårdsvej 4A
8270 Højbjerg
Denmark