

Praktische Informatik

Vorlesung 06

Fenster und Layouts

Zuletzt haben wir gelernt...

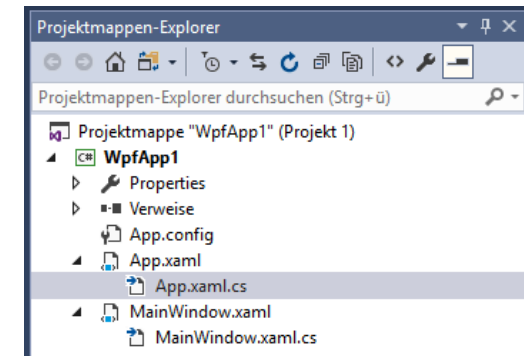
- Was Steuerelemente sind.
- Welche Basisklassen die WPF für Steuerelemente anbietet und welche Eigenschaften sich daraus für die meisten Controls ergeben.
- Was ContentControls sind.
- Der Umgang mit den ContentControls Label, Button, RadioButton und CheckBox.
- Wie man Text mit Hilfe des Controls TextBox einliest.
- Der Umgang mit dem Slider.

Inhalt heute

- Fenster und Dialoge
- Positionierung von Elementen
- Layoutcontainer
- StackPanel, WrapPanel
- DockPanel, Grid
- Canvas
- Verschachteln von Layoutcontainern

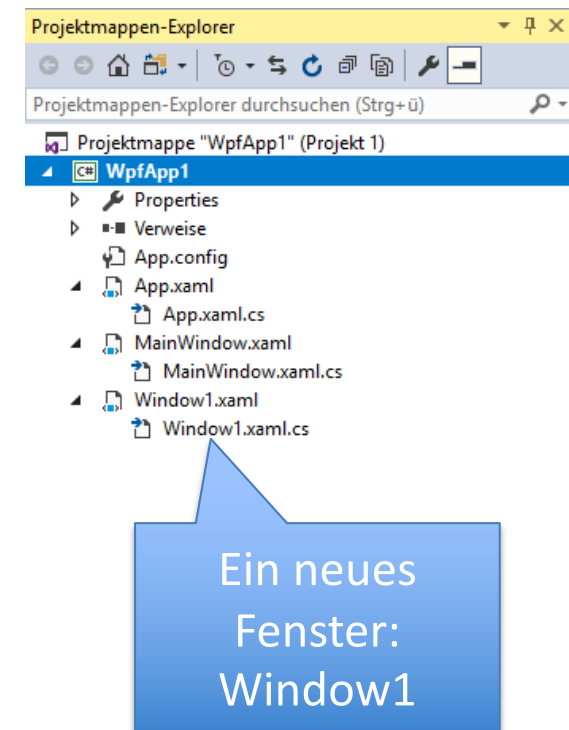
App.xaml

- Ein neues WPF-Projekt besteht direkt aus mehreren Dateien.
 - Bislang haben wir hauptsächlich in der XAML-Datei MainWindow.xaml und der zugehörigen Code Behind Datei MainWindow.xaml.cs gearbeitet.
- Zusätzlich finden wir aber auch die Dateien App.xaml und App.xaml.cs.
 - Diese beiden Dateien bilden den eigentlichen Startpunkt der Anwendung.
- In App.xaml wird z.B. festgelegt, welches Fenster als erstes angezeigt werden soll.
 - `StartupUri="MainWindow.xaml"`
- Zudem können hier anwendungsweite Ressourcen eingeführt werden.
 - Z.B. Icons, Styles, ...



Fenster hinzufügen

- Das erste Fenster unserer Anwendung wird in den Dateien `MainWindow`. [...] definiert.
 - Wir können aber beliebig viele Fenster zu unserer Anwendung hinzufügen.
 - Dazu können wir im Visual Studio mit der rechten Maustaste auf den Projektnamen klicken und den Menüpunkt Hinzufügen und Fenster auswählen.
- Es werden wieder zwei Dateien zum Projekt hinzugefügt.
 - Eine XAML-Datei und eine Code Behind Datei.



Window

- Für das neue Fenster wird eine eigene Klasse Window1 definiert, die von der Basisklasse Window ableitet:

```
<Window x:Class="WpfApp1.Window1"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WpfApp1"
        mc:Ignorable="d"
        Title="Window1" Height="300" Width="300">
    <Grid>

    </Grid>
</Window>
```

- Im XAML werden die notwendigen Namensbereiche eingebunden, die Größe des Fensters definiert und ein Titel vergeben.
 - Das Fenster dient als Wurzelement für den gesamten Inhalt.
 - Es kann **genau ein** Kindelement aufnehmen.
 - Meist ist dies ein sog. Layoutcontainer in diesem Fall ein Grid (später).

Fenster anzeigen

- Wir können dieses neue Fenster leicht benutzen.
 - Wir erzeugen dazu im ersten Fenster einen Button mit einem Click-Handler.
 - Dort können wir ein Objekt aus der neuen Klasse erzeugen und anzeigen.

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    var window = new Window1();
    window.Show();
}
```

- Die Methode Show() erbt die Klasse von der Basisklasse Window.
 - Das Fenster wird damit nicht-Modal angezeigt.

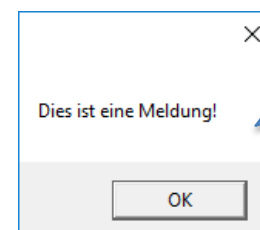
Modal vs nicht modal

- Ein **nicht modales** Fenster existiert gleichberechtigt zu allen anderen Fenstern.
 - Dieses wird mit der Methode `Show()` aus der Klasse `Window` erzeugt.
 - Es können gleichzeitig beliebig viele nicht modale Fenster angezeigt und benutzt werden.
- Ein **modales Fenster** zwingt den Benutzer hingegen dazu, erst dieses Fenster zu schließen, bevor er mit anderen Fenstern weiter arbeiten kann.
 - Ein solcher Dialog wird mit der Methode `ShowDialog()` aus der Klasse `Window` erzeugt.
 - Es kann immer nur ein modales Fenster aktiv sein.
- Die Anwendung wird in jedem Fall dann beendet, wenn das letzte Fenster geschlossen wird.

MessageBox

- Das WPF-Framework besitzt für einige Anwendungsfälle bereits fertige Dialoge.
 - Benachrichtigungen, Datei öffnen, Datei speichern, ...
 - Diese müssen also nicht selbst erstellt werden.
- Eine einfache Benachrichtigung kann z.B. mit der Klasse **MessageBox** angezeigt werden.
 - Dadurch wird eine modale Meldung für den Benutzer angezeigt, z.B. eine Fehlermeldung.

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    MessageBox.Show("Dies ist eine Meldung!");
}
```



Eine MessageBox ist auch sehr gut geeignet, um beim Programmieren mal ein paar Daten anzuzeigen, sog. printf-Debugging.

Positionierung

- In der letzten Vorlesung haben wir eine ganze Reihe von Steuerelementen kennen gelernt.
 - Button, TextBox, ...
- Um eine Benutzeroberfläche zu gestalten, müssen diese Steuerelemente sinnvoll in der Oberfläche angeordnet werden.
 - Dies haben wir auch schon getan, aber nicht wirklich gewusst wie das eigentlich funktioniert.
- In vielen Frameworks (z.B. WinForms) werden die Elemente absolut positioniert.
 - Die x/y-Position der Elemente im Fenster werden fest vorgegeben.
 - Dies hat aber Nachteile!
 - Was passiert mit der Oberfläche, wenn sich die Bildschirmgröße ändert?

Layoutcontainer

- In der WPF werden die Steuerelemente (meist) nicht absolut positioniert.
 - Es werden sog. Layoutcontainer genutzt.
- **Layoutcontainer (engl. panels)** nehmen mehrere Steuerelemente in sich auf und sorgen für ihre Ausrichtung.
 - Für diese Ausrichtung existieren unterschiedliche Strategien und daher auch unterschiedliche Layoutcontainer.
 - Viele Layoutcontainer reorganisieren ihre Steuerelemente auch automatisch, wenn sich der zur Verfügung stehende Platz ändert.
- Layoutcontainer gehören zu den wichtigsten Elementen der WPF.
 - Es ist wichtig die richtigen Layoutcontainer auswählen zu können.

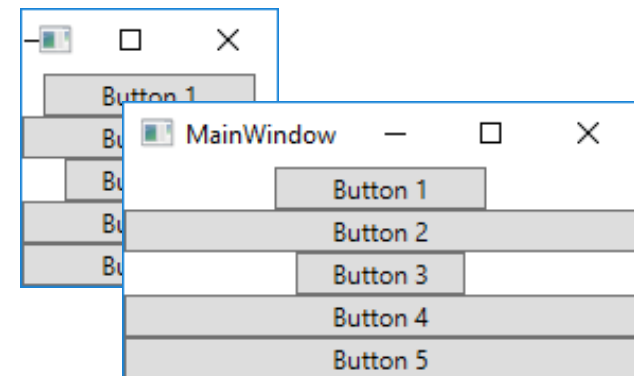
Gemeinsame Eigenschaften

- Alle Layoutcontainer erben von der Basisklasse `Panel`, die wiederum von `FrameworkElement` ableitet.
 - Alle Panels besitzen demnach die Eigenschaften, die wir in der letzten Vorlesung bereits gesehen haben.
 - Background, Foreground, FontSize, Width, Height, ...
- Alle Layoutcontainer besitzt zudem noch die Aufzählung `Children`.
 - Diese haben wir in der letzten Vorlesung genutzt, um den Zustand von mehreren Checkboxes auszulesen.
 - Man kann auch im Code Behind diese Auflistung nutzen, um dynamisch neue Elemente in ein Panel einzufügen.

StackPanel

- Der erste Layoutcontainer, den wir uns ansehen wollen, ist das StackPanel.
 - Das StackPanel ordnet standardmäßig seine Kindelemente übereinander an.
- Wenn die Elemente keine feste Größe vorgegeben haben, werden sie horizontal so lange vergrößert, bis der maximale Platz des übergeordneten Elements eingenommen wird.
 - Vergrößert sich der Platz (Fenster wird vergrößert), so werden auch die Elemente vergrößert.

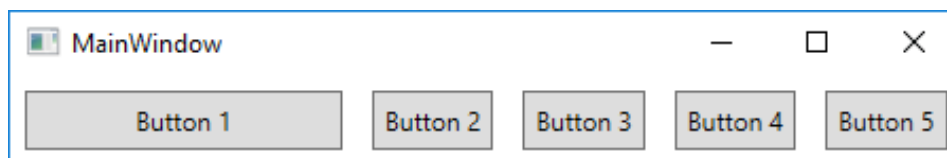
```
<StackPanel>
  <Button Width="100">Button 1</Button>
  <Button>Button 2</Button>
  <Button Width="80">Button 3</Button>
  <Button>Button 4</Button>
  <Button>Button 5</Button>
</StackPanel>
```



Horizontales StackPanel

- Ein StackPanel kann auch dazu gebracht werden, seine Elemente horizontal anzuordnen.
 - Dazu muss die Eigenschaft „Orientation“ auf den Wert „Horizontal“ gesetzt werden.
- Beispiel:

```
<StackPanel Orientation="Horizontal">
    <Button Margin="7" Padding="5" Width="150">Button 1</Button>
    <Button Margin="7" Padding="5">Button 2</Button>
    <Button Margin="7" Padding="5">Button 3</Button>
    <Button Margin="7" Padding="5">Button 4</Button>
    <Button Margin="7" Padding="5">Button 5</Button>
</StackPanel>
```



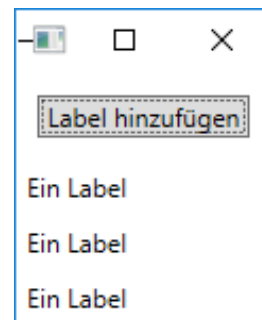
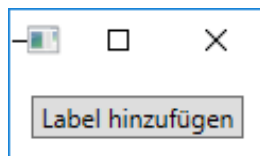
Dynamisch Elemente einfügen

- Wir nutzen ein StackPanel in XAML:

```
<StackPanel x:Name="panel">
    <Button Margin="10" Click="Button_Click">Label hinzufügen</Button>
</StackPanel>
```

- Im Code Behind können wir auf die Children Eigenschaft zugreifen:

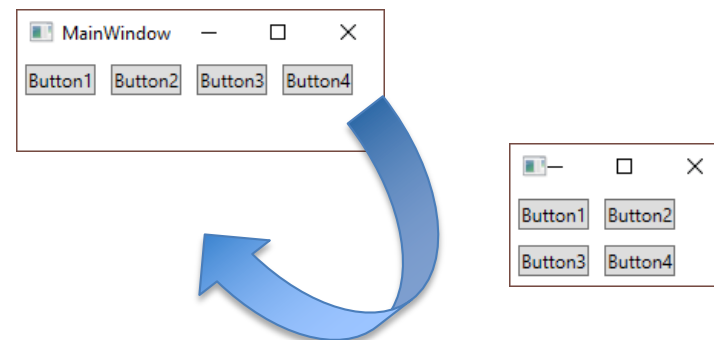
```
private void Button_Click(object sender, RoutedEventArgs e)
{
    var label = new Label();
    label.Content = "Ein Label";
    panel.Children.Add(label);
}
```



WrapPanel

- Ein WrapPanel ist sehr ähnlich zum StackPanel.
 - Es ist allerdings in der Lage, den Inhalt in eine neue Zeile umzuberechnen, wenn der vorhandene Platz nicht ausreicht.
 - Auch beim WrapPanel kann über die Abhängigkeitseigenschaft "Orientation" eine vertikale Ausrichtung erzeugt werden.

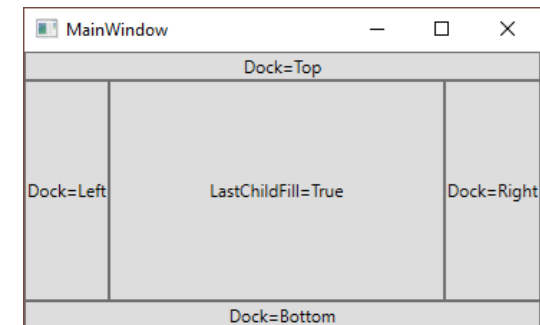
```
<WrapPanel>
  <Button Margin="5">Button1</Button>
  <Button Margin="5">Button2</Button>
  <Button Margin="5">Button3</Button>
  <Button Margin="5">Button4</Button>
</WrapPanel>
```



DockPanel

- Ein **DockPanel** ist bereits etwas komplizierter.
 - Mit dem DockPanel können Kindelemente an bestimmten Positionen (Top, Bottom, Left, Right, Center) verankert werden.

```
<DockPanel>
  <Button DockPanel.Dock="Top">Dock=Top</Button>
  <Button DockPanel.Dock="Bottom">Dock=Bottom</Button>
  <Button DockPanel.Dock="Left">Dock=Left</Button>
  <Button DockPanel.Dock="Right">Dock=Right</Button>
  <Button>LastChildFill=True</Button>
</DockPanel>
```



- Man kann auch mehrere Elemente an einer Position verankern.
 - Dabei ist dann die Reihenfolge der Elemente relevant.
- In der Standardeinstellung (LastChildFill=True) wird das letzte Element in der Mitte verankert.

Attached Properties

- Beim `DockPanel` haben wir eine Besonderheit der WPF gesehen.
 - Die Position der Kindelemente im `DockPanel` wird über die Eigenschaft `DockPanel.Dock` festgelegt, z.B. `DockPanel.Dock="Top"`
 - Diese Eigenschaft wird den Kindelementen vom `DockPanel` quasi vererbt.
- Solche Eigenschaften **werden angehängte Eigenschaften (engl. `attached properties`)** genannt.
 - Angehängte Eigenschaften sind besondere **Abhängigkeitseigenschaften**.
- Dadurch können Eigenschaften festgelegt werden, die Elemente nur dadurch besitzen, da sie z.B. in ein Panel eingefügt wurden.
 - Dies ist eine sehr wichtige Technik in WPF, um den XAML-Code übersichtlicher zu machen.

Grid

- Das **Grid** ist ein sehr mächtiger Layoutcontainer.
 - Er ordnet seine Elemente in einer tabellenartigen Struktur aus Zeilen und Spalten an.
- Das Grid muss zunächst konfiguriert werden.
 - Mit Hilfe von ColumnDefinitions und RowDefinitions wird im XAML festgelegt, wie viele Spalten und Zeilen die Tabelle besitzt.
 - Dabei wird für jede Spalte individuell die Breite und für Zeilen die Höhe festgelegt.

```
<Grid.ColumnDefinitions>
  <ColumnDefinition Width="100" />
  <ColumnDefinition Width="*" MinWidth="200" />
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
  <RowDefinition Height="Auto" />
  <RowDefinition Height="Auto" />
</Grid.RowDefinitions>
```

Die Tabelle erhält zwei Spalten und zwei Zeilen.

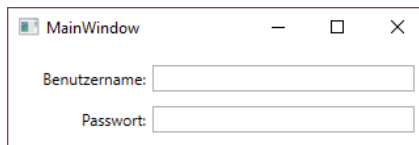
Elemente im Grid

- Die Breiten und Höhen von Zeilen/Spalten eines Grids können drei unterschiedliche Arten von Werten annehmen:
 - Ein exakter Wert, z.B. 200.
 - Der Wert "Auto": Die Größe wird automatisch aus dem Inhalt der jeweiligen Tabellenzelle bestimmt.
 - Der Wert "*": Die Größe wird aus dem restlichen Platz definiert, nachdem alle anderen Elemente gerendert wurden.
- Um die Elemente im Grid einzelnen Zellen zuzuordnen, existieren die angehängten Eigenschaften `Grid.Row` und `Grid.Column`.
 - `<Label Grid.Column="0" Grid.Row="1">Label:</Label>`
- Mit Hilfe der angehängten Eigenschaften `Grid.ColumnSpan` und `Grid.RowSpan` können auch mehrere Zellen genutzt werden.

Beispiel Benutzeranmeldung

- Es soll ein Formular erstellt werden, so dass sich Benutzer mit Benutzernamen und Passwort anmelden können:

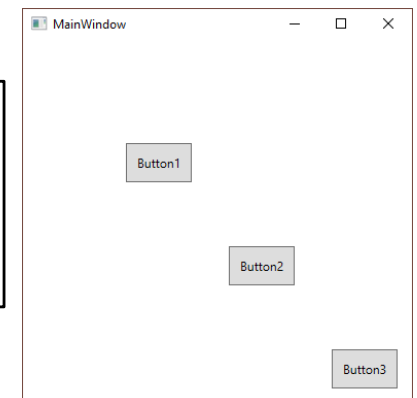
```
<Grid Margin="10">
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="100" />
    <ColumnDefinition Width="*" MinWidth="200" />
  </Grid.ColumnDefinitions>
  <Grid.RowDefinitions>
    <RowDefinition Height="Auto" />
    <RowDefinition Height="Auto" />
  </Grid.RowDefinitions>
  <Label Grid.Row="0" Grid.Column="0" HorizontalContentAlignment="Right">Benutzername:</Label>
  <TextBox Grid.Row="0" Grid.Column="1" Height="20"></TextBox>
  <Label Grid.Row="1" Grid.Column="0" HorizontalContentAlignment="Right" Margin="0 5 0 0">Passwort:</Label>
  <PasswordBox Grid.Row="1" Grid.Column="1" Margin="0 5 0 0" Height="20"></PasswordBox>
</Grid>
```




Canvas

- Der Layoutcontainer Canvas ist wieder recht einfach.
 - Er ist in der Lage, seine Kindelemente absolut zu positionieren.
 - Das hilft uns vor allem bei den kleinen Computerspielen, die wir in der nächsten Woche programmieren wollen.
- Für jedes Kindelement kann die Position durch der Abstand zu den Seitengrenzen des Canvas festgelegt werden.
 - Es existieren dazu die Eigenschaften Canvas.Top, Canvas.Bottom, Canvas.Left und Canvas.Right.
 - Man kann dabei entweder den Wert von Top oder Bottom, bzw. Left oder Right festlegen.

```
<Canvas>
  <Button Canvas.Top="100" Canvas.Left="100" Padding="10">Button1</Button>
  <Button Canvas.Top="200" Canvas.Left="200" Padding="10">Button2</Button>
  <Button Canvas.Top="300" Canvas.Left="300" Padding="10">Button3</Button>
</Canvas>
```



Springender Button

- Wir wollen ein kleines Computerspiel schreiben.
- Das Spiel wird gewonnen, sobald man einen einzelnen Button klicken konnte.
- Der Button bewegt sich aber an eine zufällige Stelle, sobald sich die Maus auf den Button bewegt.
- Dies können wir leicht mit Hilfe von Canvas und einigen Ereignissen programmieren.

Springender Button

```
<Canvas x:Name="canvas">
  <Button
    x:Name="button" Padding="10"
    Canvas.Top="100" Canvas.Left="100"
    MouseEnter="Button_MouseEnter"
    Click="button_Click">Klick mich</Button>
</Canvas>
```

```
private void Button_MouseEnter(object sender, MouseEventArgs e)
{
    var top = rnd.Next((int)(canvas.ActualHeight - button.ActualHeight));
    var left = rnd.Next((int)(canvas.ActualWidth - button.ActualWidth));

    Canvas.SetTop(button, top);
    Canvas.SetLeft(button, left);
}

private void button_Click(object sender, RoutedEventArgs e)
{
    MessageBox.Show("Gewonnen!");
}
```


ActualHeight und ActualWidth

- Wenn die Größe eines Layoutcontainers und Steuerelements nicht fest definiert ist, kümmert sich WPF darum, die richtige Größe zu bestimmen.
 - Z.B. über den Inhalt des Elements oder nach der Strategie des Layoutcontainers.
- Der zur Verfügung stehende Platz kann sich zur Laufzeit ändern.
 - Das Fenster wird vergrößert/verkleinert.
- Alle Klassen, die von FrameworkElement ableiten verfügen über die Eigenschaften ActualHeight und ActualWidth .
 - Also alle Steuerelemente und Layoutcontainer.
- Damit kann die aktuelle Größe des Elements in der Oberfläche gelesen werden.
 - Dies haben wir im letzten Beispiel genutzt, um die Größe des Fensters und des Buttons auszulesen.

Layoutcontainer schachteln

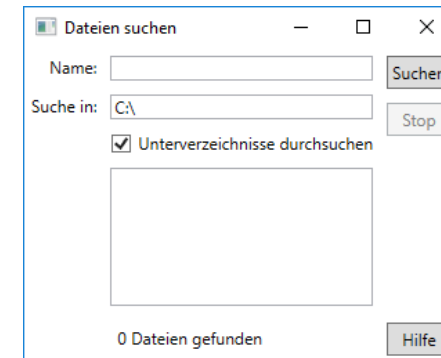
- Einzelne Layoutcontainer sind schon recht hilfreich.
 - Die richtige Stärke spielen sie aber erst aus, wenn man sie ineinander schachtelt.
- Ein Layoutcontainer kann „beliebige“ Elemente in sich aufnehmen.
 - Der Inhalt kann auch wieder ein anderer Layoutcontainer sein.

```
<DockPanel>  
  <StackPanel DockPanel.Dock="Top">  
    <StackPanel Orientation="Horizontal">  
      <Button>Button1</Button>  
      <Button>Button1</Button>  
      <Button>Button1</Button>  
    </StackPanel>  
  </StackPanel>  
</DockPanel>
```

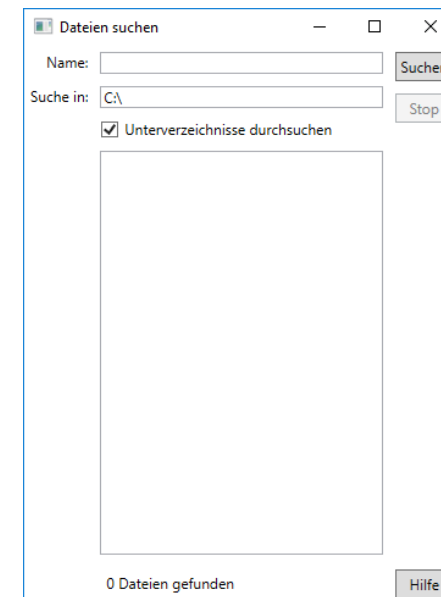
Drei Buttons werden wie eine Werkzeugleiste am oberen Rand des Fensters angeordnet.

Beispiel Dateisuche

- Es soll ein Programm erstellt werden, um Dateien in einem Verzeichnissystem zu suchen.
 - Dazu muss eine Benutzeroberfläche erstellt werden.
- In der Oberfläche muss ein Teil eines Dateinamens und ein Startverzeichnis eingegeben werden können.
 - Zudem sollen die Suchtreffer in einer Liste angezeigt werden können.
- Am besten, man fängt mit einer Skizze der Benutzeroberfläche auf Papier an.

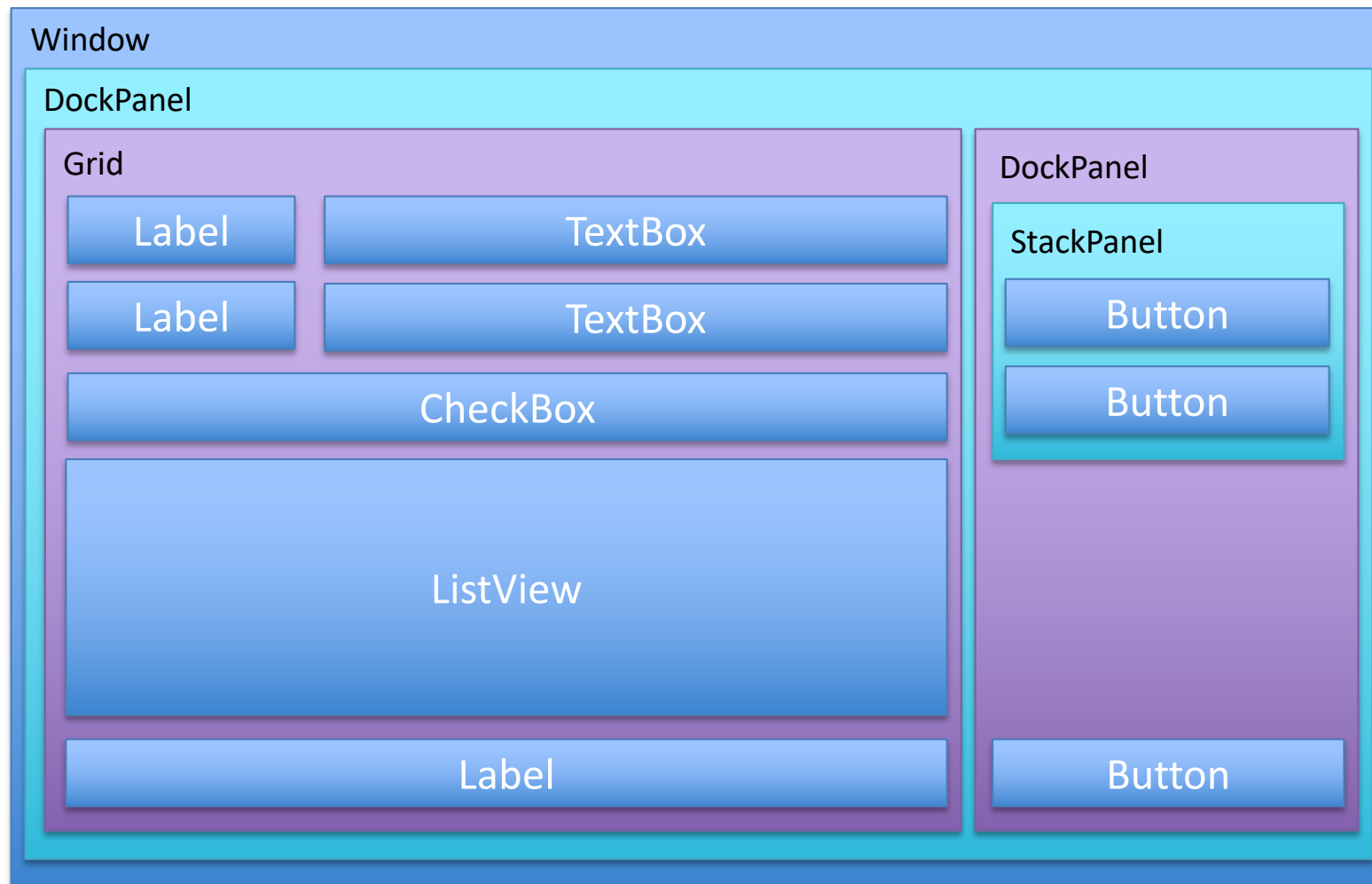


Sketch of a file search window titled "Dateien suchen". It features a "Name:" input field with a "Suchen" button to its right. Below this is a "Suche in:" input field with a "Stop" button to its right. A checked checkbox labeled "Unterverzeichnisse durchsuchen" is positioned below the "Suche in:" field. A large empty rectangular box is intended for search results. At the bottom, it displays "0 Dateien gefunden" and a "Hilfe" button.



Sketch of a file search window titled "Dateien suchen". It features a "Name:" input field with a "Suchen" button to its right. Below this is a "Suche in:" input field with a "Stop" button to its right. A checked checkbox labeled "Unterverzeichnisse durchsuchen" is positioned below the "Suche in:" field. A large empty rectangular box is intended for search results. At the bottom, it displays "0 Dateien gefunden" and a "Hilfe" button.

Prototyp der Benutzeroberfläche



XAML der Dateisuche

```
<DockPanel>
  <DockPanel DockPanel.Dock="Right" LastChildFill="False">
    <StackPanel DockPanel.Dock="Top">
      <Button Margin="5" Padding="3">Suchen</Button>
      <Button Margin="5" Padding="3" IsEnabled="False">Stop</Button>
    </StackPanel>
    <Button Margin="5" Padding="3" DockPanel.Dock="Bottom">Hilfe</Button>
  </DockPanel>
  <Grid>
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="Auto"/>
      <ColumnDefinition Width="*/>
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
      <RowDefinition Height="Auto"/>
      <RowDefinition Height="Auto"/>
      <RowDefinition Height="Auto"/>
      <RowDefinition Height="*/>
      <RowDefinition Height="Auto"/>
    </Grid.RowDefinitions>
    <Label Grid.Column="0" Grid.Row="0" HorizontalContentAlignment="Right">Name:</Label>
    <TextBox Margin="5" Grid.Column="1" Grid.Row="0" />
    <Label Grid.Column="0" Grid.Row="1" HorizontalContentAlignment="Right">Suche in:</Label>
    <TextBox Margin="5" Grid.Column="1" Grid.Row="1" Text="C:\"/>
    <CheckBox Grid.Column="1" Grid.Row="2" Margin="5" IsChecked="True">Unterverzeichnisse durchsuchen</CheckBox>
    <ListView Grid.Column="1" Grid.Row="3" Margin="5" MinHeight="100"></ListView>
    <Label Grid.Column="1" Grid.Row="4" Margin="5">0 Dateien gefunden</Label>
  </Grid>
</DockPanel>
```

Wir haben heute gelernt...

- Wie man Fenster erstellt und auf verschiedene Arten anzeigt.
- Das bereits fertige Dialogklassen existieren.
- Wie man Steuerelemente mit Hilfe von Layoutcontainern positioniert.
- Wie das StackPanel und das WrapPanel funktionieren.
- Wie man mit dem DockPanel und dem Grid umgeht.
- Wozu das Canvas benutzt werden kann.
- Wie man Layoutcontainern ineinander verschachtelt, um komplexe Layouts zu erzeugen.