

Kapitel 13

Layout

In diesem Kapitel:

Das StackPanel	276
Das DockPanel	278
Das Grid	282
Das GridSplitter-Element	288
Das UniformGrid	291
Das Canvas-Element	292
Das Viewbox-Element	293
Text-Layout	295
Das WrapPanel	299
Standard-Layout-Eigenschaften	300
Zusammenfassung	303

Layout ist ein wichtiges und sehr umfangreiches Feature von Windows Presentation Foundation (WPF). Ohne die Möglichkeiten des Layouts wäre es nur sehr schwer möglich, Applikationen mit Dialogfeldern oder Fenstern, die in der Größe änderbar sind, zu programmieren. In herkömmlichen Windows-Frameworks (z.B. MFC, Windows Forms) werden die einzelnen Steuerelemente eines Dialogfeldes oft mit fest vorgegebenen Positionen und Größen angelegt. Dieser Weg wird in WPF selten verwendet, da es die Möglichkeit gibt, alle Steuerelemente eines Fensters mithilfe von Layout-Steuerelementen so zu verwalten, dass eine variable Position und Größe ohne wesentlichen Programmieraufwand erzielbar sind.

In Windows Presentation Foundation wird das Layout der Steuerelemente in einem Fenster über so genannte Panels gesteuert. Je nach Art des verwendeten Panels werden die enthaltenen Steuerelemente angeordnet. Die Position des jeweiligen Elements wird durch das Panel bestimmt, welches das Element enthält. Einige Panel-Elemente verwalten auch die Größe ihrer Kindelemente. In WPF stehen diverse Steuerelemente zur Verfügung, die das Layout beeinflussen (Tabelle 13.1):

Panel	Anwendung
Canvas	Einfaches Element für die genaue Positionierung von Kindelementen.
StackPanel	Ordnet die Kindelemente vertikal oder horizontal an.
DockPanel	Ordnet die Kindelemente am Rand des Elternelements an.
TabPanel	Ordnet die Kindelemente als einzelne umschaltbare Seiten an.
WrapPanel	Ordnet die Kindelemente in einer Zeile an. Wenn der Platz in der Zeile nicht reicht, findet ein Zeilenumbruch statt.
Viewbox	Darstellung von Grafiken.
Grid	Ordnet die Kindelemente in einer Tabelle mit Zeilen und Spalten an.

Tabelle 13.1 Die vorhandenen Panel-Typen

Wir wollen die Möglichkeiten dieser Layout-Steuerelemente nun im Einzelnen betrachten.

BEGLEITDATEIEN

Unter `.\Samples\Chapter13\` finden Sie die Beispieldateien für dieses Kapitel.

Das StackPanel

Das Panel, welches am einfachsten zu benutzen ist, heißt `StackPanel`. Mit einem `StackPanel` können Sie die Kindelemente entweder nebeneinander oder untereinander anordnen. Bei einer vertikalen Anordnung der Elemente im `StackPanel` werden die Standardhöhen der Kindelemente für die Positionierung verwendet. Dies bedeutet, dass die Höhe eines Elements aus der Höhe des Inhalts bestimmt wird, es sei denn, Sie geben explizit einen Wert für die Höhe des Elements an. In einem horizontalen `StackPanel` werden die Standardbreiten der Kindelemente angewendet. Somit ergibt sich die Breite aus der Breite des Inhaltselements. Natürlich können Sie auch hier die Breite selbst angeben. In diesem Fall werden die Eigenschaften innerhalb der Hierarchie nicht weitergegeben. Die Kindelemente, die in einem Layout-Panel angelegt werden, müssen nicht vom gleichen Typ sein. Sie können also unterschiedliche Steuerelemente im Panel beliebig mischen.

Im folgenden Beispiel werden verschiedene Steuerelemente (TextBlock, Button, CheckBox,...) untereinander angeordnet.

```
<Window x:Class="StackPanel1.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="Vertikales StackPanel" Height="300" Width="300"
  >
  <StackPanel>
    <TextBlock>Hier steht ein Text!</TextBlock>
    <Button>Die erste Schaltfläche</Button>
    <CheckBox>Wählen Sie hier aus</CheckBox>
    <Button>Klicken Sie hier!</Button>
    <TextBlock>Auch hier steht was...</TextBlock>
    <TextBox>Eingabe hier!</TextBox>
  </StackPanel>
</Window>
```

Listing 13.1 Ein StackPanel mit vertikaler Anordnung der Elemente

Das StackPanel in Listing 13.1 ordnet die Steuerelemente folgendermaßen an:

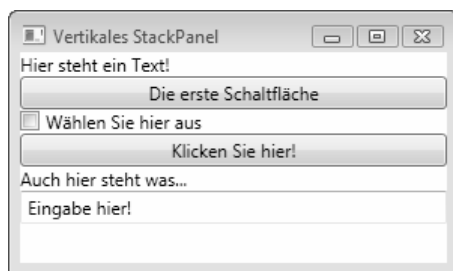


Abbildung 13.1 Vertikale Anordnung im StackPanel

Im Beispiel aus Listing 13.1 müssen wir beachten, dass das StackPanel die Breite und Höhe des Anwendungsfensters annimmt. Weiterhin nehmen die Kindelemente im StackPanel wiederum genau diese Breite an. Für die jeweilige Höhe der einzelnen Elemente wird die Standardhöhe verwendet, die sich aus der Höhe des Inhalts eines Kindelementes ergibt. Die Standardhöhe kann für die verschiedenen Elemente durchaus einen unterschiedlichen Wert annehmen.

Wird nun das Fenster in der Breite verändert, dann passt sich das StackPanel an die neue Breite an. Die Kindelemente des StackPanel passen sich dann ebenfalls an diese Breite an. Verkleinern Sie dagegen die Höhe des Fensters, so werden die Kindelemente des StackPanel ebenfalls verkleinert und irgendwann einfach abgeschnitten bzw. nicht mehr dargestellt.

Mit einer kleinen Änderung im Listing 13.1 können Sie das StackPanel dazu bewegen, seine Kindelemente in horizontaler Reihenfolge anzuordnen:

```
<!-- Wie in Listing 1.1 -->
<StackPanel Orientation="Horizontal">
<!-- Weiter wie in Listing 1.1 -->
```

Listing 13.2 Horizontale Anordnung im StackPanel

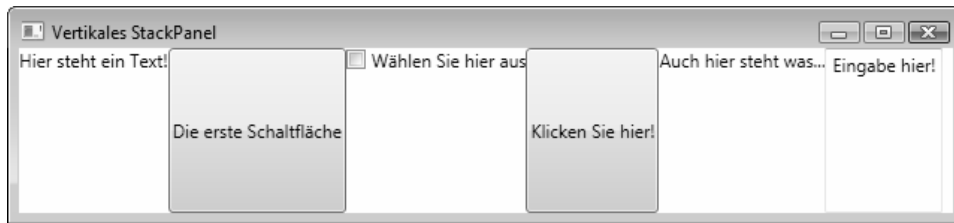


Abbildung 13.2 Ein horizontales StackPanel

In Abbildung 13.2 ist deutlich zu sehen, dass die Höhen der Kindelemente nun durch die Höhe des StackPanel (welches das Fenster vollständig ausfüllt) bestimmt werden. Die Breite der einzelnen Elemente ist durch ihre Standardbreite gegeben. Sie können gut erkennen, dass sich diese Breite aus der Breite des Inhalts des Elements ergibt. So ist die erste Schaltfläche ganz links etwas breiter als die zweite Schaltfläche, da die Texte unterschiedliche Breiten haben. Wird die Breite des Fensters und damit des StackPanel verkleinert, werden die Elemente rechts nach und nach abgeschnitten und nicht mehr dargestellt. Dieses Verhalten können Sie jedoch durch eine explizite Angabe von Breiten bzw. Höhen für die Kindelemente beeinflussen.

Das DockPanel

Ein DockPanel-Element ermöglicht es, das Gesamt-Layout eines Teils der Benutzerschnittstelle zu definieren. Sie können angeben, an welchem Rand (oben, unten, rechts oder links) ein Kindelement angeordnet werden soll. Werden mehrere Elemente am gleichen Rand angelegt, so werden diese Elemente neben- oder untereinander, ähnlich wie in einem StackPanel, angeordnet.

HINWEIS Wenn nicht anders angegeben, werden beim Andocken an das Elternelement die jeweiligen Standardbreiten und -höhen verwendet. Sie können natürlich explizit Breiten und Höhen für die Kindelemente angeben, die dann auch entsprechend verwendet werden.

```
<Window x:Class="DockPanel1.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="DockPanel #1" Height="300" Width="300"
  >
  <DockPanel>
    <Button DockPanel.Dock="Top">Oben</Button>
    <Button DockPanel.Dock="Bottom">Unten</Button>
    <Button DockPanel.Dock="Left">Links</Button>
    <Button DockPanel.Dock="Right">Rechts</Button>
  </DockPanel>
</Window>
```

Listing 13.3 Einfaches Docking mit dem DockPanel

Wenn sie das Beispiel aus Listing 13.3 ausführen, werden Sie sehen, dass die Schaltfläche, die zuletzt erzeugt wird (Schaltfläche mit dem Inhalt *Rechts*), den Bereich des DockPanel vollständig auffüllt.



Abbildung 13.3 Das DockPanel aus Listing 13.3

Für jede Schaltfläche wird angegeben, an welchem Rand des DockPanel sie andockt werden soll. Hierzu wird die spezielle Eigenschaft Dock aus dem DockPanel-Element verwendet:

```
DockPanel.Dock="Right"
```

Im Beispiel aus Listing 13.1 werden die Schaltflächen, die oben und unten andockt werden, in der Breite des DockPanel dargestellt. Die beiden Schaltflächen rechts und links werden in die verbleibende Höhe eingepasst. Die linke Schaltfläche hat die Standardbreite und die rechte Schaltfläche füllt schließlich den verbleibenden Rest des DockPanel-Elements auf.

HINWEIS Wenn Sie die Größe des Fensters aus Listing 13.1 ändern, bleibt die Anordnung der Schaltflächen im DockPanel erhalten. Nur die jeweiligen Breiten und Höhen, die nicht fest vorgegeben wurden, werden entsprechend der Fenstergröße neu berechnet und dargestellt.

Wenn Sie das Auffüllen durch das letzte Kindelement verhindern, müssen Sie das Attribut LastChildFill auf False setzen (Listing 13.4). Die letzte Schaltfläche, die im DockPanel dargestellt wird, erhält dann einfach die Standardbreite. Ein Teil des DockPanel-Elements bleibt dann frei.

```
<Window x:Class="DockPanel2.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="DockPanel #2" Height="300" Width="300"
  >
  <DockPanel LastChildFill="False">
    <Button DockPanel.Dock="Top">Oben</Button>
    <Button DockPanel.Dock="Bottom">Unten</Button>
    <Button DockPanel.Dock="Left">Links</Button>
    <Button DockPanel.Dock="Right">Rechts</Button>
  </DockPanel>
</Window>
```

Listing 13.4 DockPanel ohne »Auffüllen«



Abbildung 13.4 Ein nicht vollständig gefülltes DockPanel

Im nächsten Beispiel soll die Schaltfläche *Links* nun nicht mehr die gesamte Höhe des DockPanel-Elements ausfüllen. In diesem Fall (Listing 13.5) müssen wir die Höhe für diese Schaltfläche explizit angeben. Die Schaltfläche wird dann in der Mitte des zur Verfügung stehenden Bereiches dargestellt.

```
<Window x:Class="DockPanel3.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="DockPanel #3" Height="300" Width="300"
  >
  <DockPanel LastChildFill="False">
    <Button DockPanel.Dock="Top">Oben</Button>
    <Button DockPanel.Dock="Bottom">Unten</Button>
    <Button DockPanel.Dock="Left" Height="22">Links</Button>
    <Button DockPanel.Dock="Right">Rechts</Button>
  </DockPanel>
</Window>
```

Listing 13.5 Eine Schaltfläche mit fest definierter Höhe im DockPanel

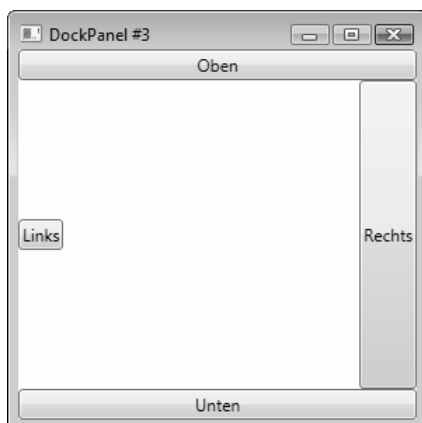


Abbildung 13.5 Die Schaltfläche steht in der Mitte des Bereiches, der im DockPanel zur Verfügung steht

Um die Position der Schaltfläche zu modifizieren, können wir die Attribute `HorizontalAlignment` und `VerticalAlignment` entsprechend einsetzen. Im folgenden Code-Beispiel soll die Schaltfläche mit dem Text *Links* nun weiter unten, aber direkt oberhalb der Schaltfläche mit dem Text *Unten* angeordnet werden (Listing 13.6 und Abbildung 13.6).

```
<Window x:Class="DockPanel4.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="DockPanel #4" Height="300" Width="300"
  >
  <DockPanel LastChildFill="False">
    <Button DockPanel.Dock="Top">Oben</Button>
    <Button DockPanel.Dock="Bottom">Unten</Button>
    <Button DockPanel.Dock="Left" Height="22" VerticalAlignment="Bottom">Links</Button>
    <Button DockPanel.Dock="Right">Rechts</Button>
  </DockPanel>
</Window>
```

Listing 13.6 Die Schaltfläche ist jetzt unten angeordnet

Sie erkennen sicherlich schon, wie leistungsfähig selbst einfache Layout-Elemente sind, wenn die Darstellung der Kindelemente über die verschiedenen Attribute angepasst wird.



Abbildung 13.6 Die kleine Schaltfläche ist unten links angeordnet

Die Darstellung der Kindelemente im `DockPanel` ist natürlich von der Reihenfolge abhängig, in der die Elemente erzeugt werden. Gehen Sie vom Beispiel in Listing 13.4 aus. Nun sollen zuerst die beiden Schaltflächen auf der rechten und der linken Seite erzeugt werden. Wir erhalten dann den in Abbildung 13.5 gezeigten neuen Aufbau des Fensters.

```
<Window x:Class="DockPanel5.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="DockPanel #5" Height="300" Width="300"
  >
  <DockPanel LastChildFill="False">
    <Button DockPanel.Dock="Left">Links</Button>
```

```
<Button DockPanel.Dock="Right">Rechts</Button>
<Button DockPanel.Dock="Top">Oben</Button>
<Button DockPanel.Dock="Bottom">Unten</Button>
</DockPanel>
</Window>
```

Listing 13.7 Die Reihenfolge der Definition ist entscheidend für den Aufbau

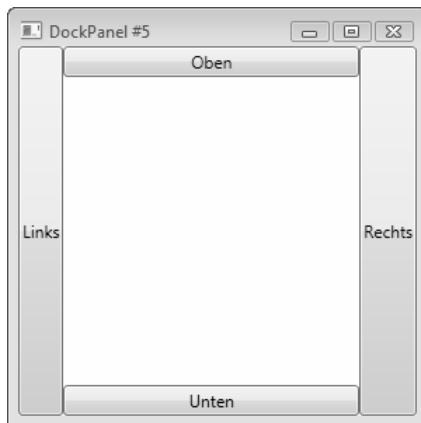


Abbildung 13.7 Zuerst werden die beiden Schaltflächen rechts und links erzeugt

HINWEIS Die Elemente in einem DockPanel überlappen sich niemals. Wenn einzelne Elemente in der geforderten Größe nicht dargestellt werden können, wird dieser Teil einfach abgeschnitten.

Die Kindelemente im DockPanel entscheiden teilweise selbst, in welcher Größe sie sich darstellen müssen. Wenn eine Schaltfläche oben oder unten im DockPanel platziert wird, so entscheidet das DockPanel über die Breite der Schaltfläche oder Sie müssen die Breite des Kindelements extra angeben. Das Kindelement, im Beispiel die Schaltfläche, entscheidet aber ggf. selbst, wie hoch das Element sein soll. Wird die Schaltfläche rechts oder links angedockt, so verhält es sich umgekehrt. Die Höhe wird vom DockPanel bestimmt und die Breite wird vom Kindelement vorgegeben. Hierbei spielt dann der Text auf der Schaltfläche eine entscheidende Rolle. Wenn möglich, wird die Breite so gesetzt, dass der gesamte Text auf der Schaltfläche sichtbar ist. Wenn dies nicht möglich ist, wird ein Teil des Kindelementes abgeschnitten.

Das Grid

Das Grid ist ein sehr mächtiges Layout-Steuerelement. Es bietet die Möglichkeit, die Kindelemente in einer oder mehreren Zeilen und Spalten anzuordnen. Sie werden sehen, dass die Höhe der Zeilen und die Breite der Spalten im Grid sehr leicht konfigurierbar sind.

Das einfachste Grid-Element enthält nur eine Zelle, also eine Zeile und eine Spalte. In dieser Grid-Zelle kann ein Kindelement platziert werden. Dieses Element wird standardmäßig in der Mitte der Zelle dargestellt (Listing 13.8).

```
<Grid>
  <Button Width="100" Height="30">Test</Button>
</Grid>
```

Listing 13.8 Das einfachste Grid mit einer Zelle

Um nun mehrere Spalten und Zeilen im Grid darzustellen, müssen diese in den Bereichen `Grid.ColumnDefinitions` bzw. in den `Grid.RowDefinitions` angelegt werden. Im Beispiel aus Listing 13.9 wird ein Grid mit zwei Spalten und drei Zeilen erzeugt. In den einzelnen Zellen werden `TextBlock`-Elemente für die Ausgabe von Texten benutzt. Beachten Sie hierbei, dass die `TextBlock`-Elemente links oben mit der Ausgabe von Texten beginnen. Diese Standardeinstellung können Sie über die Attribute `HorizontalAlignment` und `VerticalAlignment` des `TextBlock`-Elements ändern. In diesem Beispiel wollen wir mithilfe des Grid-Attributs `ShowGridLines="True"` die Trennlinien zwischen den Zellen im Grid darstellen (Abbildung 13.8). Für jedes `TextBlock`-Element wird über die Attribute `Grid.Column` und `Grid.Row` (aus der Grid-Klasse) festgelegt, in welcher Zeile und Spalte es dargestellt werden soll.

```
<Window x:Class="Grid.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="Grid #1"
  Height="300" Width="300">

  <Grid ShowGridLines="True">
    <Grid.ColumnDefinitions>
      <ColumnDefinition />
      <ColumnDefinition />
    </Grid.ColumnDefinitions>

    <Grid.RowDefinitions>
      <RowDefinition />
      <RowDefinition />
      <RowDefinition />
    </Grid.RowDefinitions>

    <TextBlock Grid.Column="0" Grid.Row="0">Name:</TextBlock>
    <TextBlock Grid.Column="1" Grid.Row="0">Vorname:</TextBlock>
    <TextBlock Grid.Column="0" Grid.Row="1">Heckhuis</TextBlock>
    <TextBlock Grid.Column="1" Grid.Row="1">Jürgen</TextBlock>
    <TextBlock Grid.Column="0" Grid.Row="2">Löffelmann</TextBlock>
    <TextBlock Grid.Column="1" Grid.Row="2">Klaus</TextBlock>
  </Grid>
</Window>
```

Listing 13.9 Ein Grid mit mehreren Spalten und Zeilen

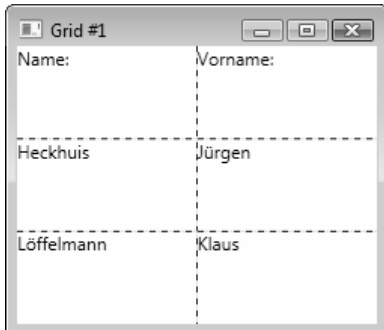


Abbildung 13.8 Ein Grid mit mehreren Spalten und Zeilen

Beachten Sie bitte auch das Verhalten des Grid-Elements beim Ändern der Größe des Fensters. Das Grid füllt in diesem Beispiel immer das gesamte Fenster aus. Zeilen und Spalten werden entsprechend in der Höhe und der Breite angepasst, so dass die Höhen- und Breitenverhältnisse im Grid-Element erhalten bleiben.

Nun ist es im Normalfall nicht damit getan, einfach nur einige Zeilen und Spalten zu definieren. Im nächsten Beispiel werden darum die Breiten und Höhen der Spalten und Zeilen im Grid-Element konfiguriert. Dies wird ebenfalls in den jeweiligen `RowDefinition`- und `ColumnDefinition`-Elementen durchgeführt.

Es gibt drei Möglichkeiten, die Höhe und Breite der Zellen zu definieren. Um Spalten (und Zeilen) mit einer festen Breite (Höhe) zu erzeugen, wird im `ColumnDefinition`-Element (`RowDefinition`) einfach die gewünschte Pixelanzahl angegeben:

```
<Grid.ColumnDefinitions>
  <ColumnDefinition Width="35" />
</Grid.ColumnDefinitions>
```

Wird in einer `ColumnDefinition` der Wert `Auto` angegeben, so wird die Breite der Spalte aus der maximalen Breite der Kindelemente bestimmt, die in der Spalte enthalten sind:

```
<Grid.ColumnDefinitions>
  <ColumnDefinition Width="Auto" />
</Grid.ColumnDefinition>
```

Die dritte Variante der Breitenangabe in einer `ColumnDefinition` erstellt Spalten, deren Breite sich am vorhandenen Platz im Elternfenster orientiert. In diesem Fall wird als Parameter ein `»*«` verwendet:

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

Im letzten Beispiel wird ein Grid mit einer Spalte erzeugt, welche die gesamte Breite des Elternelements bekommt. Wird das Elternelement in der Breite verändert, so wird die Spaltenbreite entsprechend angepasst. Das ist aber noch nicht alles. Betrachten Sie folgendes Beispiel:

```
<Grid.ColumnDefinitions>
  <ColumnDefinition Width="*" />
  <ColumnDefinition Width="2*" />
  <ColumnDefinition Width="4*" />
  <ColumnDefinition Width="0.5*" />
</Grid.ColumnDefinitions>
```

Hier erhalten Sie ein Grid mit vier Spalten, die sich an die Breite des Elternelements anpassen. Das besondere ist jedoch, dass die Breiten der einzelnen Spalten in dem Verhältnis angelegt werden, die in den Width-Attributen angegeben sind. Die Summe aller Breiten ist »7.5*« und entspricht der Gesamtbreite des Grid-Elements. Wenn die Gesamtbreite im Grid nun zum Beispiel 300 Pixel beträgt, dann haben die einzelnen Spalten folgende Breiten:

Spalte-Nr.	Breitenangabe	Breite in Pixel
1	*	40
2	2*	80
3	4*	160
4	0.5*	20
Summe:	7.5*	300

Wird die Breite dieses Grid-Elements geändert, z.B. durch eine Änderung der Fenstergröße, so bleibt das Breitenverhältnis der einzelnen Grid-Spalten jedoch erhalten.

Alles was hier über die Definition von Spaltenbreiten geschrieben wurde, gilt ebenfalls für die Definition von Zeilenhöhen im Grid. Das entsprechende Definitionselement heißt dann RowDefinition. Auch hier ist es möglich, feste, automatisch angepasste und im Verhältnis angepasste Höhen für eine Zeile anzugeben. In Listing 13.10 wird ein vollständiges Beispiel mit einem konfigurierten Grid gezeigt. Interessant ist hier die Breitenermittlung für die erste Spalte ganz links. Für diese Spalte wird die Einstellung Auto verwendet. Somit wird die Breite des größten Elements dieser Spalte benutzt. Hier ist das die Breite der Schaltfläche in der zweiten Zeile.

```
<Window x:Class="Grid2.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="Grid #2" Height="300" Width="300"
  >
  <Grid ShowGridLines="True">
    <Grid.RowDefinitions>
      <RowDefinition Height="Auto" />
      <RowDefinition Height="*" />
      <RowDefinition Height="2*" />
      <RowDefinition Height="50" />
      <RowDefinition />
    </Grid.RowDefinitions>
```

```

<Grid.ColumnDefinitions>
  <ColumnDefinition Width="Auto" />
  <ColumnDefinition Width="*" />
  <ColumnDefinition Width="0.5*" />
  <ColumnDefinition Width="50" />
  <ColumnDefinition />
</Grid.ColumnDefinitions>

<Button Grid.Column="0" Grid.Row="0">Test</Button>
<Button Grid.Column="0" Grid.Row="1">Ein großer Button</Button>
<TextBlock Grid.Column="0" Grid.Row="2">Auto</TextBlock>
<TextBlock Grid.Column="1" Grid.Row="2">*</TextBlock>
<TextBlock Grid.Column="2" Grid.Row="2">0.5*</TextBlock>
<TextBlock Grid.Column="3" Grid.Row="2">Fest 50</TextBlock>
<TextBlock Grid.Column="4" Grid.Row="2">
  Keine<LineBreak />Angabe
</TextBlock>
</Grid>
</Window>

```

Listing 13.10 Ein Grid-Element mit diversen Breiten- und Höhendefinitionen

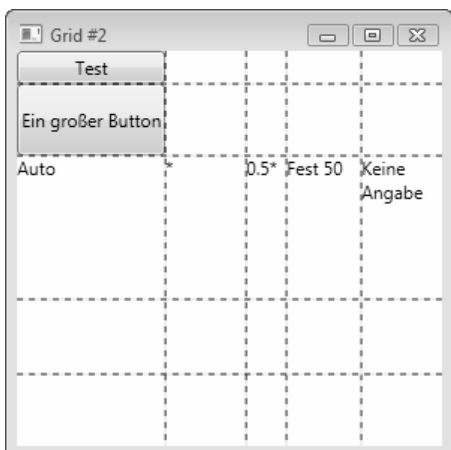


Abbildung 13.9 Das Grid aus Listing 13.10

Oft ist es erforderlich, dass ein Element im Grid mehrere Zeilen oder Spalten belegt. Hierfür stehen in der Grid-Klasse die beiden Eigenschaften `Grid.RowSpan` und `Grid.ColumnSpan` zur Verfügung. Mit diesen beiden Eigenschaften ist es sehr leicht möglich, ein Grid-Element zur beliebigen Positionierung von anderen Elementen zu benutzen und über mehrere Zellen zu verteilen. Die linken Ränder der Grid-Spalten haben dann in etwa das Verhalten von definierten Tabulatorpositionen.

Das Beispiel in Listing 13.11 zeigt die Positionierung von mehreren `TextBlock`-Elementen in einem Grid. Für die einzelnen `TextBlock`-Elemente werden an verschiedenen Stellen die Attribute `Grid.ColumnSpan` und `Grid.RowSpan` verwendet, damit die längeren Textinhalte auch vollständig ausgegeben werden. **Abbildung 13.10** zeigt die Ausgabe von Listing 13.11 in einem Fenster. Die Attribute `Grid.RowSpan` und `Grid.ColumnSpan` können natürlich bei beliebigen Kindelementen angewendet werden.

HINWEIS Beachten Sie, dass bei einem TextBlock-Element der Text nur dann über mehrere Zeilen (Grid.RowSpan) verteilt wird, wenn das Attribut TextWrapping="Wrap" benutzt wird.

```
<Window x:Class="Grid3.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="Grid #3" Height="300" Width="300"
  >
  <Grid ShowGridLines="true">
    <Grid.RowDefinitions>
      <RowDefinition Height="Auto" />
      <RowDefinition Height="Auto" />
      <RowDefinition Height="Auto" />
      <RowDefinition Height="Auto" />
      <RowDefinition Height="Auto" />
      <RowDefinition Height="30" />
      <RowDefinition Height="30" />
      <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="Auto" />
      <ColumnDefinition Width="20" />
      <ColumnDefinition Width="20" />
      <ColumnDefinition Width="20" />
      <ColumnDefinition Width="20" />
      <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>
    <TextBlock Grid.Column="0" Grid.Row="0">Name:</TextBlock>
    <TextBlock Grid.Column="0" Grid.Row="1">Vorname:</TextBlock>
    <TextBlock Grid.Column="0" Grid.Row="2">PLZ und Ort:</TextBlock>
    <TextBlock Grid.Column="0" Grid.Row="3">Strasse und Nr.:</TextBlock>
    <TextBlock Grid.Column="0" Grid.Row="4">Telefon:</TextBlock>
    <TextBlock Grid.Column="0" Grid.Row="5">Angaben:</TextBlock>

    <TextBlock Grid.Column="1" Grid.Row="0" Grid.ColumnSpan="5">Heckhuis</TextBlock>
    <TextBlock Grid.Column="2" Grid.Row="1" Grid.ColumnSpan="5">Jürgen</TextBlock>
    <TextBlock Grid.Column="3" Grid.Row="2" Grid.ColumnSpan="5">12345 Wpfstadt</TextBlock>
    <TextBlock Grid.Column="4" Grid.Row="3" Grid.ColumnSpan="5">Avalonstrasse 22</TextBlock>
    <TextBlock Grid.Column="5" Grid.Row="4">01234-567890</TextBlock>
    <TextBlock Grid.Column="1" Grid.Row="5" Grid.ColumnSpan="4" Grid.RowSpan="3" TextWrapping="Wrap">
      Hier können weitere Angaben zur Person eingetragen werden. Diese Angaben sind natürlich
      freiwillig.
    </TextBlock>
  </Grid>
</Window>
```

Listing 13.11 Ein Grid zur Positionierung von Texten, die mehrere Spalten und Zeilen überspannen

WICHTIG Bedenken Sie, dass die vorgestellten Layout-Elemente in einer beliebigen Hierarchie verwendet werden können. Es ist möglich, in der Zelle eines komplexen Grid-Elementes ein DockPanel oder StackPanel zu platzieren. Ebenso kann man in eine Grid-Zelle ein weiteres Grid einfügen und konfigurieren.

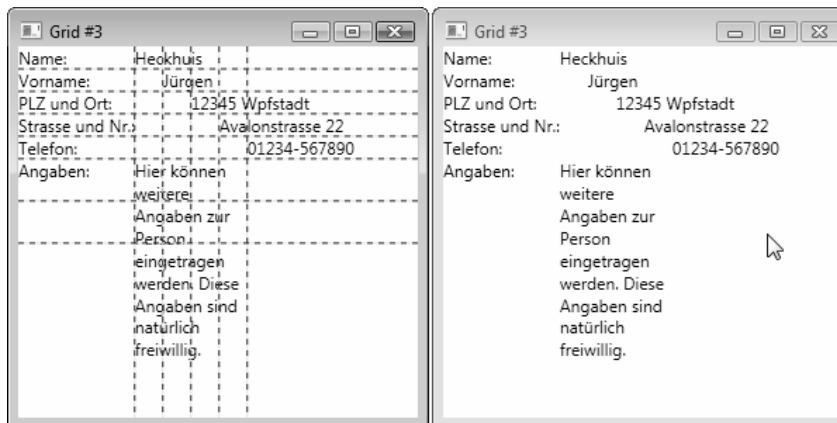


Abbildung 13.10
Das Grid aus Listing 13.11 – mit und ohne Trennlinien

Das GridSplitter-Element

Beim Einsatz eines Grid-Elementes taucht natürlich sofort die Frage auf, ob die Breite der Spalten bzw. die Höhe der Zeilen zur Laufzeit einfach änderbar sind. Um das zu ermöglichen, können Sie ein oder mehrere GridSplitter-Elemente einsetzen. Eine einfache Anwendung des GridSplitter-Elementes zeigt Listing 13.12:

```
<Window x:Class="GridSplitter.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="GridSplitter" Height="120" Width="300"
  >
  <Grid>
    <Grid.ColumnDefinitions>
      <ColumnDefinition />
      <ColumnDefinition />
    </Grid.ColumnDefinitions>
    <Button Grid.Column="0">Button Nr. 1</Button>
    <Button Grid.Column="1">Button Nr. 2</Button>
    <GridSplitter Grid.Column="1" Width="5" Background="Blue" HorizontalAlignment="Left"/>
  </Grid>
</Window>
```

Listing 13.12 Grid mit einem GridSplitter-Element

Der Bereich des GridSplitter-Elements ist in Abbildung 13.11 gut erkennbar in blau sichtbar. Wenn Sie die Maus über dieses Element bewegen, ändert sich auch der Mauszeiger entsprechend. Bei einer Verschiebung des GridSplitter-Elements wird in diesem Fall eine Schaltfläche größer und die andere kleiner. Im Beispiel haben wir ein vertikales Trennelement, welches nach der Grid-Spalte mit dem Index »0« eingefügt werden soll.

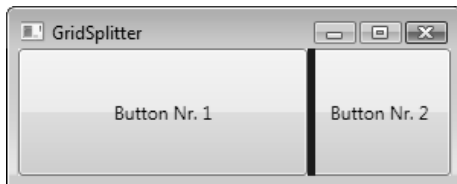


Abbildung 13.11 Das Grid mit nach rechts verschobenem GridSplitter-Element

Welcher Wert für die Eigenschaft Grid.Column angegeben werden muss, hängt allerdings auch noch von der Eigenschaft HorizontalAlignment des GridSplitter-Elements ab. Der Standardwert für diese Eigenschaft ist Right. In diesem Fall wird Grid.Column auf den Wert 0 gesetzt, wie in Listing 13.12 gezeigt wurde. Das GridSplitter-Element ist dann ein Teil der linken Grid-Zelle. Wird die Eigenschaft HorizontalAlignment auf den Wert Left eingestellt, dann ist das Trennelement ein Teil der linken Grid-Zelle und die Eigenschaft Grid.Column muss auf den Wert 1 gesetzt werden, damit die Trennung zwischen den beiden Schaltflächen erfolgt.

```
<GridSplitter Grid.Column="1" Width="5" Background="Blue" HorizontalAlignment="Left"/>
```

HINWEIS Beachten Sie bitte, dass ein GridSplitter-Element das Element in der Zelle (hier die Schaltfläche) teilweise überdeckt. Das ist besser erkennbar, wenn man die Breite des Trennelements vergrößert. Die Ereignisbearbeitung für die Schaltflächen wird aber nur dann ausgelöst, wenn Sie in den Bereich klicken, der als Schaltfläche auch tatsächlich sichtbar ist.

Das nächste Beispiel (Listing 13.13) zeigt den Einsatz mehrerer GridSplitter-Elemente, um alle Zeilen und Spalten eines Grid-Steuerelements in der Breite und in der Höhe zu modifizieren.

```
<Window x:Class="GridSplitter2.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="GridSplitter #2" Height="250" Width="300"
  >
  <Grid>
    <Grid.ColumnDefinitions>
      <ColumnDefinition />
      <ColumnDefinition />
      <ColumnDefinition />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
      <RowDefinition />
      <RowDefinition />
      <RowDefinition />
    </Grid.RowDefinitions>
```

```

<Button Grid.Column="0" Grid.Row="0">1 - 1</Button>
<Button Grid.Column="0" Grid.Row="1">1 - 2</Button>
<Button Grid.Column="0" Grid.Row="2">1 - 3</Button>
<Button Grid.Column="1" Grid.Row="0">2 - 1</Button>
<Button Grid.Column="1" Grid.Row="1">2 - 2</Button>
<Button Grid.Column="1" Grid.Row="2">2 - 3</Button>
<Button Grid.Column="2" Grid.Row="0">3 - 1</Button>
<Button Grid.Column="2" Grid.Row="1">3 - 2</Button>
<Button Grid.Column="2" Grid.Row="2">3 - 3</Button>
<GridSplitter Width="6" Grid.Column="0" Grid.Row="0" Grid.RowSpan="3" Background="Blue"
  ResizeDirection="Columns" />
<GridSplitter Width="6" Grid.Column="1" Grid.Row="0" Grid.RowSpan="3" Background="Blue"
  ResizeDirection="Columns" />
<GridSplitter Height="6" Grid.Column="0" Grid.Row="0" Grid.ColumnSpan="3" Background="Red"
  ResizeDirection="Rows" HorizontalAlignment="Stretch" VerticalAlignment="Bottom" />
<GridSplitter Height="6" Grid.Column="0" Grid.Row="1" Grid.ColumnSpan="3" Background="Red"
  ResizeDirection="Rows" HorizontalAlignment="Stretch" VerticalAlignment="Bottom" />
</Grid>
</Window>

```

Listing 13.13 Ein Grid mit horizontalen und vertikalen GridSplitter-Elementen

Nach der Definition von Spalten und Zeilen für das Grid-Element wird für jede Zelle eine Schaltfläche angelegt. Danach werden zuerst zwei vertikale und dann zwei horizontale GridSplitter-Elemente definiert. Für die vertikalen Trennelemente wird die Eigenschaft `ResizeDirection` auf den Wert `Columns` gesetzt. Dies ist die Standardeinstellung für diese Eigenschaft. Die horizontalen Trennelemente verwenden den Wert `Rows` für diese Eigenschaft. Hier müssen außerdem die Eigenschaften `HorizontalAlignment` und `VerticalAlignment` angepasst werden. Für die vertikalen Trennelemente können dagegen die Standardeinstellungen benutzt werden. Da die Trennelemente über das gesamte Grid hinweg sichtbar und anwendbar sein sollen, müssen die Eigenschaften `Grid.RowSpan` und `Grid.ColumnSpan` entsprechend auf den Wert 3 gesetzt werden. Das Resultat zeigt Abbildung 13.12. Sie können nun Breite und Höhe aller Zellen im Grid mit der Maus beeinflussen.

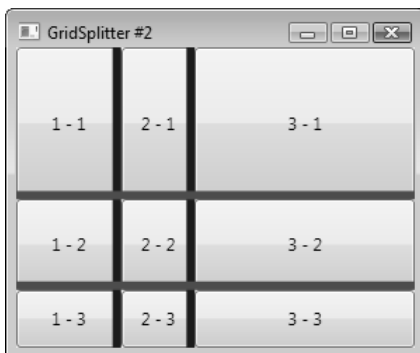


Abbildung 13.12 Vertikale und horizontale GridSplitter-Elemente im Einsatz

Das UniformGrid

Im UniformGrid-Element haben alle Spalten die gleiche Breite und alle Zeilen die gleiche Höhe. Die Kindelemente werden der Reihe nach in die Grid-Zellen eingefügt. Es beginnt mit Zeile 1, Spalte 1. Danach wird das Element in Zeile 1, Spalte 2 eingefügt. Das geht so weiter, bis die erste Zeile komplett mit den Kindelementen belegt ist. Nun wird mit der zweiten Zeile im UniformGrid fortgefahren. Im UniformGrid-Element gibt es keine Eigenschaften, die eine direkte Adressierung einer Zelle (wie beim normalen Grid-Element) ermöglichen.

```
<Window x:Class="UniformGrid.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="UniformGrid" Height="300" Width="300"
  >
  <UniformGrid Columns="4" Rows="3">
    <Button>1, 1</Button>
    <Button>2, 1</Button>
    <Button>3, 1</Button>
    <Button>4, 1</Button>
    <Button>1, 2</Button>
    <Button>2, 2</Button>
    <Button>3, 2</Button>
    <Button>4, 2</Button>
    <Button>1, 3</Button>
    <Button>2, 3</Button>
    <Button>3, 3</Button>
    <Button>4, 3</Button>
  </UniformGrid>
</Window>
```

Listing 13.14 Ein UniformGrid mit vier Spalten und drei Zeilen

Listing 13.14 zeigt ein UniformGrid-Element mit vier Spalten und drei Zeilen aus Abbildung 13.13. Fügt man im Beispiel aus Listing 13.14 weitere Kindelemente in das Grid ein, so werden diese nicht dargestellt, da alle Zeilen und Spalten schon belegt sind. Das UniformGrid verhält sich jedoch anders, wenn eine feste Breite und Höhe für das Element vorgegeben wird. In diesem Fall werden auch die Kindelemente dargestellt, welche die vorgegebene Spalten- bzw. Zeilenanzahl überschreiten.



Abbildung 13.13 Ein UniformGrid mit gleicher Zeilenhöhe und Spaltenbreite für alle Zellen

Das Canvas-Element

Viele Layout-Anforderungen können mit den bereits vorgestellten Panels erfüllt werden. Was nun noch fehlt, ist ein Element, welches eine präzise Positionsvorgabe für seine Kindelemente ermöglicht. Oft ist es erforderlich, einzelne Grafikelemente genau an einer bestimmten Position auszugeben. Die Position der Grafiken soll nicht durch das automatische Layout beeinflusst werden. In diesem Fall müssen Sie ein Canvas-Element benutzen.

Eigentlich verhält sich das Canvas-Element sehr einfach. Es ermöglicht die genaue Positionierung von Elementen in Abhängigkeit von den Ecken des Elements. Ein Canvas-Element funktioniert etwa so, wie man früher mit Visual Basic 6.0 oder den Microsoft Foundation Classes (MFC) die Steuerelemente in einem Dialogfeld fest positioniert hat. Bei der Positionierung im Canvas müssen wir also die Koordinaten relativ zu den Ecken des Canvas-Elements angeben. Hierzu gibt es vier »angehängte« Eigenschaften (attached properties) in der Canvas-Klasse: `Top`, `Left`, `Bottom` und `Right`. Das folgende Beispiel (Listing 13.15) zeigt die Anwendung des Canvas bei der Positionierung von vier Schaltflächen.

```
<Window x:Class="Canvas.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="Canvas" Height="300" Width="300"
>
  <Canvas>
    <Button Canvas.Left="20" Canvas.Top="20">Button 1</Button>
    <Button Canvas.Right="20" Canvas.Top="20">Button 2</Button>
    <Button Canvas.Left="20" Canvas.Bottom="20">Button 3</Button>
    <Button Canvas.Left="200" Canvas.Top="200">Button 4</Button>
  </Canvas>
</Window>
```

Listing 13.15 Vier Schaltflächen im Canvas-Element

Beachten Sie bei diesem Beispiel die Bewegung der Schaltflächen beim Verändern der Fenstergröße. Je nachdem, an welche Eigenschaft im Canvas die Position des Kindelements gebunden wird, bleibt die Position relativ zur linken, oberen Fensterecke gleich (*Button 1* und *Button 4*) oder sie verschiebt sich (*Button 2* und *Button 3*).

Hier drängt sich sofort ein Vergleich zur `Anchor`-Eigenschaft bei `WindowsForms`-Steuerelementen auf. Das Canvas-Element verhält sich jedoch etwas anders: Wenn man z. B. `Canvas.Left` und `Canvas.Right` an die Schaltfläche bindet, so wird nicht – wie in `WindowsForms` – das Steuerelement beim Aufziehen des Fensters verbreitert, sondern es wird eine der beiden Eigenschaften einfach ignoriert und die Größe der Schaltfläche bleibt unverändert. Um das `Anchor`-Verhalten nachzubauen, müssen Sie zusätzlich ein `Grid`-Element verwenden.

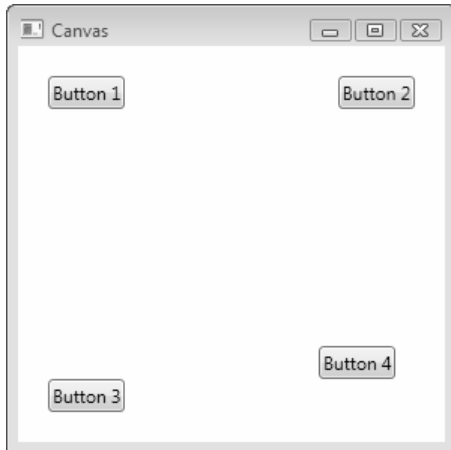


Abbildung 13.14 Vier Schaltflächen im Canvas-Element

Das Canvas-Element dient in erster Linie zur exakten Positionierung von Grafiken. Wenn ein Kindelement im Canvas zu groß ist, so wird der überstehende Teil normalerweise nicht abgeschnitten. Dies geschieht erst, wenn die Eigenschaft `ClipToBounds` des Canvas auf `True` gesetzt wird.

Das Viewbox-Element

Um Grafiken in der Größe anzupassen, gibt es das Viewbox-Element. Es ist möglich, eine beliebige Grafik automatisch an die Größe der Viewbox anzupassen. Dabei können verschiedene Varianten gewählt werden. Einerseits kann das Seitenverhältnis der Grafik beibehalten werden oder die Grafik so skaliert werden, dass sie das gesamte Elternelement ausfüllt. Dieses Verhalten der Viewbox wird mit der Eigenschaft `Stretch` gesteuert.

```
<Window x:Class="ViewBox1.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="Viewbox" Height="300" Width="300"
  >
  <Viewbox Stretch="None">
    <!-- Stretch="Fill" oder Stretch="Uniform" oder Stretch="UniformToFill" -->
    <Canvas Width="30" Height="30">
      <Ellipse Canvas.Left="1" Canvas.Top="1"
        Width="14" Height="14"
        Fill="Red" Stroke="Black" />
      <Ellipse Canvas.Left="1" Canvas.Bottom="1"
        Width="14" Height="14"
        Fill="Green" Stroke="Black" />
      <Ellipse Canvas.Right="1" Canvas.Top="1"
        Width="14" Height="14"
        Fill="Blue" Stroke="Black" />
      <Ellipse Canvas.Right="1" Canvas.Bottom="1"
        Width="14" Height="14"
        Fill="Yellow" Stroke="Black" />
    </Canvas>
  </Viewbox>
</Window>
```

```

</Canvas>
</Viewbox>
</Window>

```

Listing 13.16 Benutzung der Viewbox

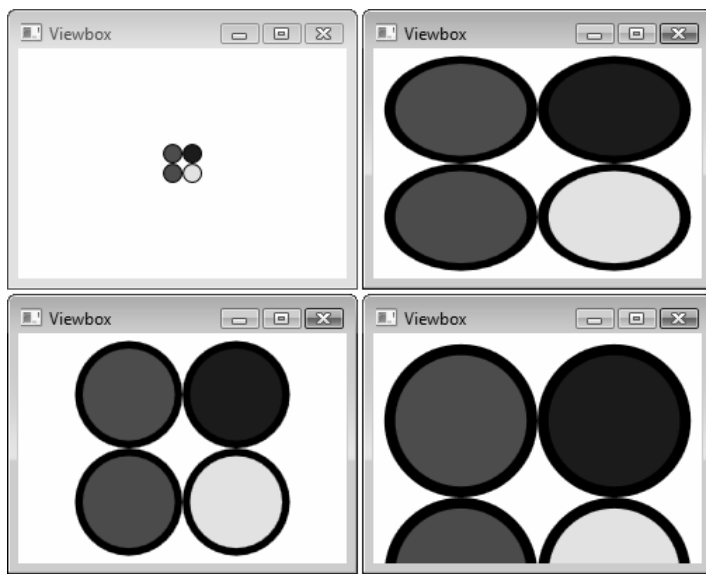


Abbildung 13.15 Grafikausgabe mit der Viewbox

Abbildung 13.15 zeigt die verschiedenen Möglichkeiten mit dem Viewbox-Element. Für das Fenster oben links wurde das Stretch-Attribut auf None gesetzt. In diesem Fall wird die Grafik in der Originalgröße ausgegeben. Im Fenster rechts oben wurde das Attribut Stretch auf Fill eingestellt. In der unteren Reihe wurde links der Wert Uniform und für das rechte Fenster der Wert UniformToFill verwendet. Beim Wert Uniform wird das Seitenverhältnis der Grafik beibehalten und diese wird vollständig ausgegeben. Wird die Größe des Fensters geändert, dann wird, je nach eingestelltem Stretch-Attribut, die Ausgabe der Grafik angepasst.

In einer Viewbox kann man nicht nur ein Canvas-Element skalieren, sondern auch alle anderen WPF-Elemente. Die Qualität beim Vergrößern oder Verkleinern von beliebigen Elementen ist dabei sehr gut, da alles als Vektorgrafik dargestellt wird (Abbildung 13.16). Bei der Skalierung von Pixel-Grafiken können jedoch Qualitätsverluste auftreten, wenn die Vergrößerungswerte sehr groß werden oder die Grafik eine geringe Auflösung hat.



Abbildung 13.16 Ein Button-Element in der Viewbox, links: Stretch="None", rechts: Stretch="Fill"

Text-Layout

Beim Layout von Texten müssen verschiedene Eigenschaften berücksichtigt werden. Einerseits muss der Umbruch der Texte korrekt sein. Andererseits gibt es viele Parameter bei einer Schriftart, welche die Größe und die Darstellung eines Textes stark beeinflussen. Das einfachste Element, welches Texte darstellen kann, ist das `TextBlock`-Element.

Um einfache, kurze Texte darzustellen, kann das `TextBlock`-Element folgendermaßen benutzt werden:

```
<TextBlock>Hallo, Welt!</TextBlock>
```

Der Text wird im oben gezeigten Beispiel ohne weitere Formatierung dargestellt. Dennoch bietet das `TextBlock`-Element viele Formatierungsmöglichkeiten an. Zunächst kann man eine Schriftart für die Textdarstellung bestimmen. Dabei ist es natürlich auch möglich, die Größe der Schrift und den Schriftschnitt (fett, kursiv,...) festzulegen:

```
<TextBlock FontFamily="Courier New" FontSize="24" FontWeight="Bold">Guten Morgen!</TextBlock>
```

Eine weitere wichtige Darstellungsart des `TextBlock`-Elements erlaubt den Umbruch von längeren Textzeilen. Es gibt zwei Varianten des Zeilenumbruchs. Mit dem Wert `Wrap` für das Attribut `TextWrapping` erreicht man einen »vollständigen« Umbruch. D.h., wenn ein Wort nicht mehr in die Zeile passt, wird an einer Buchstabengrenze umgebrochen. Wird dem Attribut dagegen der Wert `WrapWithOverflow` zugewiesen, so wird ein nicht in die Zeile passendes Wort einfach abgeschnitten. Die überstehenden Buchstaben werden nicht dargestellt. Die beiden Varianten werden im Listing 13.17 vorgestellt.

```
<StackPanel>  
  <TextBlock FontSize="24" FontStyle="Italic" FontFamily="Courier New" TextWrapping="WrapWithOverflow">  
    Guten Morgen, alle miteinander hier im Raum!  
  </TextBlock>  
  
  <TextBlock FontSize="24" FontWeight="Bold" FontFamily="Arial" TextWrapping="Wrap">  
    Auf Wiedersehen, meine Damen und Herren!  
  </TextBlock>  
</StackPanel>
```

Listing 13.17 Das `TextBlock`-Element mit `TextWrapping`-Möglichkeiten



Abbildung 13.17 TextBlock mit TextWrapping-Möglichkeiten

In Abbildung 13.17 können Sie die Wirkungsweise der verschiedenen Zeilenumbruchsvarianten sehen. Das erste TextBlock-Element schneidet das Wort »miteinander« einfach ab und stellt den Rest des Wortes nicht dar (`TextWrapping="WrapWithOverflow"`). Im zweiten TextBlock wird das längere Wort »Wiedersehen« einfach im Wort selbst umgebrochen (`TextWrapping="Wrap"`). Auch wenn ein Zeilenumbruch aktiviert ist, versucht ein TextBlock-Element den Text zunächst in einer Zeile auszugeben.

Ein TextBlock-Element kann jedoch nicht nur Texte umbrechen, sondern auch andere Steuerelemente, wie z.B. Schaltflächen oder Eingabefelder. In Listing 13.18 werden mehrere Elemente mithilfe eines TextBlocks zeilenweise angeordnet. Die einzelnen Elemente im TextBlock werden im Prinzip genauso angeordnet, als wären es normale Wörter.

```
<TextBlock FontSize="24" TextWrapping="Wrap">
  <Button>Test</Button>
  <Button>Noch ein Test</Button>
  <CheckBox>Ein neues Control</CheckBox>
  <TextBox>Hier Eingabe</TextBox>
</TextBlock>
```

Listing 13.18 Steuerelemente im TextBlock-Element mit Zeilenumbruch

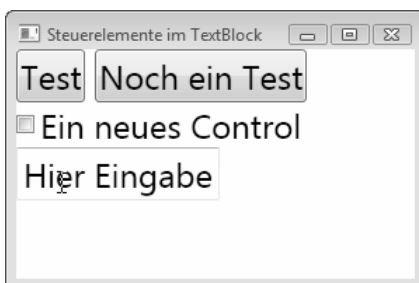


Abbildung 13.18 Steuerelemente im TextBlock mit Zeilenumbruch

Innerhalb eines Textes können sie im TextBlock-Element verschiedene Effekte in beliebigen Kombinationen anwenden. Dazu zählen fette und kursive Schrift, sowie unterstrichener Text und ein Text als Hyperlink. Die verschiedenen Möglichkeiten können Sie im Listing 13.19 sehen.

```
<TextBlock FontSize="20" FontFamily="Arial" TextWrapping="Wrap">
  <Italic>Ein schöner Text</Italic>
  <Bold>in schöner Schrift</Bold>
  <Underline>mit einem Strich darunter.</Underline>
  <LineBreak /><LineBreak />
  <Hyperlink>Klicken Sie bitte hier!</Hyperlink>
</TextBlock>
```

Listing 13.19 Verschiedene Effekte im TextBlock-Element

An dieser Stelle soll auch erwähnt werden, dass mehrfache Leerzeichen hintereinander vom XAML-Compiler entfernt werden. Leerzeichen werden im Allgemeinen für die Formatierung des XAML-Codes benutzt. Um dies zu zeigen, ändern wir die dritte Zeile in Listing 13.19 durch Einfügen vieler Leerzeichen:

```
<Bold>  in    schöner    Schrift  </Bold>
```

Trotz dieser Modifikation im XAML-Code wird sich die Ausgabe dieser Zeile nicht ändern, wenn das Programm ausgeführt wird. Die »überzähligen« Leerzeichen werden vom XAML-Compiler einfach entfernt. Natürlich gibt es viele Fälle, in denen der Programmierer dies verhindern möchte. Hierzu muss das `xml:space`-Attribut auf den Wert `preserve` gesetzt werden, wie in Listing 13.20 gezeigt wird.

```
<TextBlock FontSize="20" FontFamily="Arial" TextWrapping="Wrap">
  <Italic>Ein schöner Text</Italic>
  <Bold xml:space="preserve">  in    schöner    Schrift    </Bold>
  <Underline>mit einem Strich darunter.</Underline>
  <LineBreak /><LineBreak />
  <Hyperlink xml:space="preserve">Klicken Sie    bitte    hier!</Hyperlink>
</TextBlock>
```

Listing 13.20 Das Attribut `xml:space` wird benutzt

In Listing 13.20 kann man sehen, dass das Attribut `xml:space` für jedes Element im TextBlock erneut gesetzt werden muss. Wenden Sie dieses Attribut im TextBlock-Element selbst an, werden auch die Leerzeichen, die an den Zeilenanfängen zum Formatieren des XAML-Codes stehen, mit in die Ausgabe einbezogen.

HINWEIS Beachten Sie, dass bei Anwendung des Attributs `xml:space="preserve"` auch Zeilenumbrüche im XAML-Code in der Ausgabe umgesetzt werden. Um einen Zeilenumbruch ohne das Attribut `xml:space="preserve"` zu erzielen, können Sie das Element `<LineBreak />` beliebig oft in den Text einsetzen.

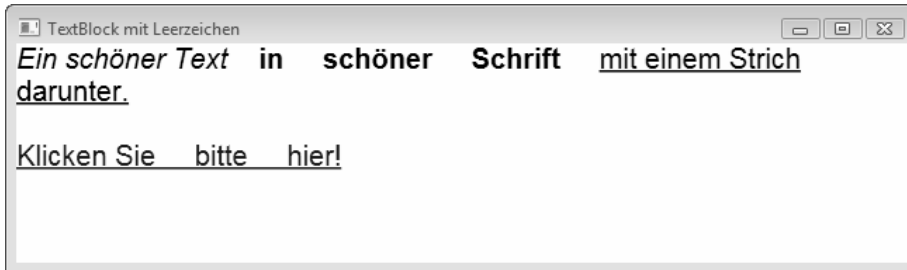


Abbildung 13.19 Mehrere Leerzeichen mit dem Attribut `xml:space`

Im `TextBlock`-Element ist es ebenfalls möglich, die Anordnung des Textes mit dem Attribut `TextAlignment` zu beeinflussen. Es sind vier Varianten vorhanden: `Left`, `Right`, `Center` und `Justify`. Listing 13.21 zeigt die Auswirkungen beim `TextAlignment`. Bei der Darstellung des Textes wird mit der gewählten Anordnung auch der Zeilenumbruch (`TextWrapping`) im `TextBlock`-Element berücksichtigt.

```
<Window x:Class="TextBlock4.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="TextBlock mit TextAlignment" Height="300" Width="300"
  >
  <StackPanel Orientation="Vertical">
    <StackPanel Orientation="Horizontal">
      <Border BorderBrush="Blue" BorderThickness="1">
        <TextBlock Width="140" TextAlignment="Left" TextWrapping="Wrap">
          Das ist ein Test mit dem TextAlignment-Attribut, welches den Text <Bold>links</Bold>
          anordnet.
        </TextBlock>
      </Border>
      <Border BorderBrush="Blue" BorderThickness="1">
        <TextBlock Width="140" TextAlignment="Right" TextWrapping="Wrap">
          Das ist ein Test mit dem TextAlignment-Attribut, welches den Text <Bold>rechts</Bold>
          anordnet.
        </TextBlock>
      </Border>
    </StackPanel>
    <StackPanel Orientation="Horizontal">
      <Border BorderBrush="Blue" BorderThickness="1">
        <TextBlock Width="140" TextAlignment="Center" TextWrapping="Wrap">
          Das ist ein Test mit dem TextAlignment-Attribut, welches den Text <Bold>zentriert</Bold>
          anordnet.
        </TextBlock>
      </Border>
      <Border BorderBrush="Blue" BorderThickness="1">
        <TextBlock Width="140" TextAlignment="Justify" TextWrapping="Wrap">
          Das ist ein Test mit dem TextAlignment- Attribut, welches den Text im <Bold>Blocksatz</Bold>
          anordnet.
        </TextBlock>
      </Border>
    </StackPanel>
  </StackPanel>
</Window>
```

```

    </TextBlock>
  </Border>
</StackPanel>
</StackPanel>
</Window>

```

Listing 13.21 Die Benutzung des Attributs TextAlignment

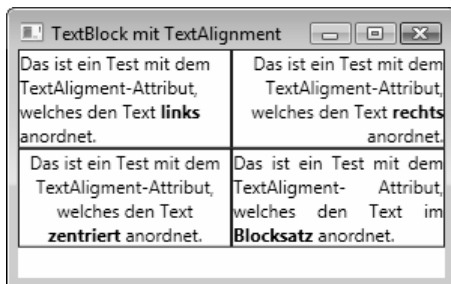


Abbildung 13.20 Die Ausgabe mit verschiedenen Werten für das TextAlignment

Das WrapPanel

Ein weiteres Panel, welches WPF-Elemente in einer ganz bestimmten Art und Weise anordnet, ist das WrapPanel. In diesem Steuerelement werden die Kindelemente zunächst horizontal nebeneinander angeordnet, bis die maximal erlaubte Breite erreicht wird. Dann findet, wie bei Texten, ein Zeilenumbruch statt. Das bedeutet, dass alle weiteren Elemente in der nächsten Zeile ausgegeben werden.

Im Listing 13.22 wird ein WrapPanel mit mehreren nebeneinander angeordneten Schaltflächen vorgestellt. Wird das linke Fenster in Abbildung 13.21 verkleinert, reicht der vorhandene Platz für die Schaltflächen nicht mehr aus. Nach und nach werden nun die Schaltflächen in weiteren Steuerelementzeilen umgebrochen und ausgegeben. Alle Kindelemente bleiben sichtbar, bis die Breite oder Höhe des Fensters generell für die vollständige Darstellung des Inhalts zu klein wird.

```

<WrapPanel>
  <Button>Button 1</Button>
  <Button>Hier klicken</Button>
  <Button>Test 3</Button>
</WrapPanel>

```

Listing 13.22 WrapPanel in horizontaler Ausrichtung



Abbildung 13.21 Das WrapPanel wird Schritt für Schritt in der Breite verkleinert

Auch das `WrapPanel` erlaubt zwei Orientierungen. Wird die Eigenschaft `Orientation` auf `Horizontal` gesetzt, so bekommen sie das oben gezeigte Verhalten. Wird dagegen `Orientation` auf `Vertical` eingestellt, so werden die einzelnen Kindelemente untereinander angeordnet, bis der untere Rand des Elternelements erreicht ist. Dann werden die weiteren Elemente in einer neuen Spalte rechts daneben ausgegeben.

Standard-Layout-Eigenschaften

Alle Steuerelemente in Windows Presentation Foundation haben einige Standardeigenschaften, d.h., Eigenschaften, die Sie mit jedem Steuerelement anwenden können. Hierbei handelt es sich um grundlegende Eigenschaften, welche die einzelnen Elemente aus der Basisklasse `FrameworkElement` erben.

Width- und Height-Eigenschaft

Die beiden Eigenschaften `Width` und `Height` geben eine Breite und eine Höhe für ein Steuerelement vor. Verwenden Sie eines oder beide dieser Eigenschaften, so fixieren Sie die Größe dieses Elements. Dies hat verschiedene Auswirkungen. Es ist z. B. schwieriger, eine Benutzerschnittstelle mit vielen in Breite und Höhe fixierten Elementen zu lokalisieren, da die verschiedenen Landessprachen oft unterschiedliche Textbreiten benötigen. Außerdem ist es schwieriger, Dialogfelder zu erstellen, die in der Größe veränderbar sind. Wenn es geht, sollten Sie also die Definition von festen Breiten und Höhen vermeiden.

Bei der Benutzung der Eigenschaften `Width` und `Height` wird das Layout-System von WPF jedoch immer versuchen, Ihre Wünsche auszuführen. Wenn ein Element nicht vollständig darstellbar ist, z. B. weil das Fenster oder der Bildschirm zu klein ist, wird es an der entsprechenden Stelle einfach abgeschnitten.

Eine vorgegebene Breite oder Höhe eines Elements wird ggf. an die Kindelemente weitergegeben.

MinWidth-, MaxWidth-, MinHeight- und MaxHeight-Eigenschaft

Mit diesen vier Eigenschaften können Sie die minimal erlaubte Breite und Höhe und die maximal erlaubte Breite und Höhe für ein Steuerelement definieren. Die Benutzung dieser Eigenschaften macht eigentlich nur dann einen Sinn, wenn Sie kein fest definiertes Layout mit `Width` und `Height` benutzen.

Mit den Min- und Max-Eigenschaften für Breite und Höhe ist es möglich, die Größenänderungen, welche das Layout-System von WPF vornehmen kann, einzuschränken. Dies ist eine einfache Möglichkeit, die Größe von Steuerelementen zu kontrollieren.

Wenn Sie für eine der Eigenschaften einen »unmöglichen« Wert angeben, z. B. `MinHeight="1000000"`, so wird das Element zwar dargestellt, aber an der passenden Stelle abgeschnitten. Das Programm wird in solchen Fällen nicht mit einer Fehlermeldung abgebrochen. Daran ändert auch die zusätzliche Definition einer sehr kleinen maximalen Höhe nichts (z. B. `MaxHeight="50"`). In diesem Fall wird die zu kleine maximale Höhe ignoriert und die `MinHeight`-Eigenschaft für die Darstellung des Elements benutzt.

HorizontalAlignment- und VerticalAlignment-Eigenschaft

Die Eigenschaft `TextAlignment` haben Sie schon beim `TextBlock`-Element kennen gelernt. Ganz ähnlich funktionieren die Eigenschaften `HorizontalAlignment` und `VerticalAlignment`. Hierbei wird aber nicht der Text rechts-, linksbündig oder zentriert angeordnet, sondern die Steuerelemente werden entsprechend rechts, links oder mittig angeordnet. Es ist also sehr einfach, eine Schaltfläche ohne viele komplizierte Berechnungen mitten in einem Elternelement zu platzieren.

Die vier Möglichkeiten bei der Eigenschaft `HorizontalAlignment` sind `Left`, `Right`, `Center` und `Stretch`. Die Eigenschaft `VerticalAlignment` erlaubt `Top`, `Bottom`, `Center` und `Stretch`.

In einem `StackPanel` oder `WrapPanel` mit der Einstellung `Orientation="Vertical"` werden alle Kindelemente von oben nach unten angelegt. Das erste Element wird am oberen Rand des Elternelementes ausgegeben. Nun können Sie für das Panel zusätzlich die Eigenschaft `VerticalAlignment="Bottom"` setzen. Die Kindelemente werden jetzt so angeordnet, dass das letzte Element mit dem unteren Rand des Elternelementes abschließt.

HINWEIS Wenn die Eigenschaften `VerticalAlignment="Bottom"` bzw. `HorizontalAlignment="Right"` gesetzt werden, bleibt so die Reihenfolge der Elemente erhalten. Das letzte Kindelement schließt nur mit dem unteren bzw. rechten Rand ab.

Die Auswahl `Stretch` ist für beide Eigenschaften der Standardwert. In diesem Fall versucht das Element, den gesamten Platz auszunutzen, der zur Verfügung steht.

Margin-Eigenschaft

Die Eigenschaft `Margin` bestimmt die Breite des Randes, der um das Element herum frei gelassen werden soll. Diese Eigenschaft kann benutzt werden, wenn ein Element sein Elternelement nicht vollständig ausfüllen soll. `Margin`-Werte können in unterschiedlicher Weise angegeben werden. Wenn Sie eine Zahl angeben (z. B. `Margin="10"`), so wird dieser Wert für alle vier Ränder (oben, unten, rechts und links) des Kindelements herangezogen. Mit der Angabe von zwei Zahlen für die Eigenschaft (z. B. `Margin="10 20"`) definieren Sie mit der ersten Zahl die Breite des rechten und linken Randes. Die zweite Zahl in der `Margin`-Eigenschaft gibt an, wie breit die Ränder oben und unten sein sollen. Wenn Sie vier Werte angeben, können Sie vier unterschiedliche Werte für alle Ränder definieren. Die Reihenfolge der Zahlen ist dann: Linker, oberer, rechter und unterer Rand.

Das Beispiel in Listing 13.23 zeigt die Benutzung der verschiedenen `Margin`-Arten. Dort wird ein `Grid` mit vier Spalten definiert. Die Zellen werden durch gestrichelte Linien dargestellt (`ShowGridLines="True"`).

HINWEIS Die Angabe von drei Zahlen in der Margin-Eigenschaft ist nicht erlaubt und führt zum Abbruch des Programms.

```
<Window x:Class="ShowMargin.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="Margin" Height="300" Width="500"
  >
  <Grid ShowGridLines="True">
    <Grid.ColumnDefinitions>
      <ColumnDefinition />
      <ColumnDefinition />
      <ColumnDefinition />
      <ColumnDefinition />
    </Grid.ColumnDefinitions>

    <Button Grid.Column="0">Ohne Margin</Button>
    <Button Grid.Column="1" Margin="15">Margin 1</Button>
    <Button Grid.Column="2" Margin="10,30">Margin 2</Button>
    <Button Grid.Column="3" Margin="10,20,30,40">Margin 4</Button>
  </Grid>
</Window>
```

Listing 13.23 Verwendung der Eigenschaft Margin

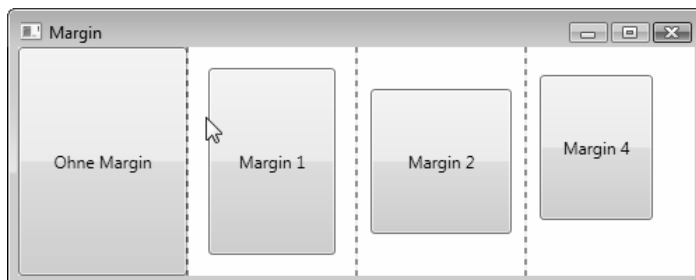
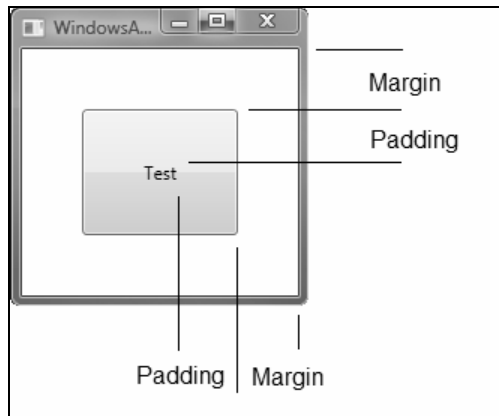


Abbildung 13.22 Verwendung unterschiedlicher Margin-Definitionen

Wird das Fenster mit dem in Listing 13.23 gezeigten Programm in der Größe geändert, so werden die Breiten der Ränder der einzelnen Schaltflächen unverändert bleiben. Die Größen der Schaltflächen werden aber entsprechend angepasst. Verkleinern Sie das Fenster nun immer weiter, so wird irgendwann ein Punkt erreicht werden, an dem der definierte Rand und die Standardgröße des Kindelements nicht mehr in den vorhandenen Bereich passen. Dann wird das jeweilige Kindelement (hier die Schaltflächen) nicht mehr vollständig dargestellt. Es kann sogar passieren, dass ein Element gar nicht mehr dargestellt wird. D.h., die Darstellung des Randes hat immer Vorrang vor der Ausgabe des Kindelementes.

Padding-Eigenschaft

Die Eigenschaft `Padding` beeinflusst den Abstand zwischen dem Rand eines Steuerelements und den darin enthaltenen Daten (Content). In Abbildung 13.23 wird der Unterschied zwischen den beiden Eigenschaften `Margin` und `Padding` dargestellt.

**Abbildung 13.23** Margin und Padding

Die `Padding`-Eigenschaft finden Sie nur in solchen WPF-Elementen, die einen Inhalt darstellen können. Wenn Sie die Werte für das `Padding` zu groß wählen, ist es möglich, dass der Inhalt des Elements gar nicht oder nur teilweise dargestellt wird. Auch bei der `Padding`-Eigenschaft können Sie eine Zahl, zwei oder vier Zahlen als Parameter angeben. Das Verhalten der `Padding`-Parameter ist identisch zu dem der `Margin`-Angaben.

Zusammenfassung

In diesem Kapitel haben Sie die verschiedenen Layout-Panels von WPF kennen gelernt. Mit WPF werden die Positionen und Größen der einzelnen Elemente in einem Fenster normalerweise nicht mehr statisch festgelegt, sondern es werden Panels verwendet, die eine dynamische Anordnung der Elemente beim Vergrößern oder Verkleinern des Fensters ermöglichen. WPF enthält viele leistungsfähige Panels: `StackPanel`, `DockPanel` und `WrapPanel`. Weiterhin steht das `Grid`-Element für ein sehr flexibles Layout zur Verfügung.

In den Layout-Steuerelementen gibt es bestimmte Standardeigenschaften, die es ermöglichen, die Art des Layouts zu steuern.

Außerdem gibt es ein WPF-Element, welches das genaue Positionieren von Elementen auf absoluter Basis ermöglicht. In diesem `Canvas`-Element können einzelne Grafikelemente exakt arrangiert werden.

Für das Layout von Texten wurde in WPF natürlich auch gesorgt. Das `TextBlock`-Element lässt viele Formatierungsmöglichkeiten zu und stellt auch einen Zeilenumbruch zur Verfügung.

Sie können alle Layout-Elemente hierarchisch ineinander verschachteln, so dass auch komplexe Anordnungen von Steuerelementen in einem Fenster mit änderbarer Größe automatisch angeordnet werden können.

